Integrating SQL Server with Visual Sourcesafe: Best Practices

A Case Study in Making SQL Server development safe and manageable

Bill Allison
May 2003

The Scenario:
Let's say, for argument's sake that you are faced with any type of database-driven application development project. The database is SQL Server 2000, and perhaps it's a web project. As a starting point for this case study, we will assume you're using Visual Studio Enterprise 6.0, SQL Server 2000, running on a Windows 2000 platform. We're in mid-development and we discover that the database development is happening on the development SQL Server via the Enterprise Manager (EM). Many inexperienced database programmers get drawn to using the EM because of it's ease of use -- it is a powerful tool. The problem is that the EM is not designed as a development environment and is not integrated with source control. If you change stored procedures or make DDL changes, the previous state is lost. While it's possible to script out a backup before making changes, then you will face either a sea of backup scripts or you will need to deal with source control issues.

No Current Acceptable Out-Of-the Box Solution for VSS/SQL integration:
What about Visual Interdev integration? While it's possible to do some integration with Interdev, as far as I can tell, the only VSS integration is for stored procedures -- so if you are doing user defined functions (UDF) development, those aren't integrated. I have also heard anecdotal evidence about some very weird behavior with regard to how VSS uses some system stored procedures for management. (You will notice that Interdev presents SQL that is unique to Interdev - for example the ALTER syntax in the sproc.) I talked with another developer who is in a tech lead/DB architect role on another project and they got burned when they cleared a database and regenerated from the DDL because they lost some state information pertinant to VSS/Interdev integration. Finally, what of table/view/udf/udt development? In all these cases, using Interdev amounts to making changes directly to the development database - with no source control history. [Disclaimer: To be 100% honest here, I have examined VSS/Interdev integration and discarded it rather quickly - at some point I plan to write a more thorough investigation -- so I hereby disclose that I am biased against it. If you know something I don't, please email me! -- (looks like someday we will get better integration...)]

A Solution

Needless to say, we worked out a solution for our project. We went down some dead-ends, and made several fine-tunings -- which is why I am writing this up: hopefully you'll save some time. In changing the way our developers interacted with the database, the goals were to:

1. eliminate the "database gatekeeper" role, and allow all developers to write and implement SQL, getting mentoring where necessary
2. create accountability amongst database developers by maintaining change-history
3. allow consistent release versioning between our codebase and the database to allow us to rollback (e.g. from version 1.3 to 1.1)

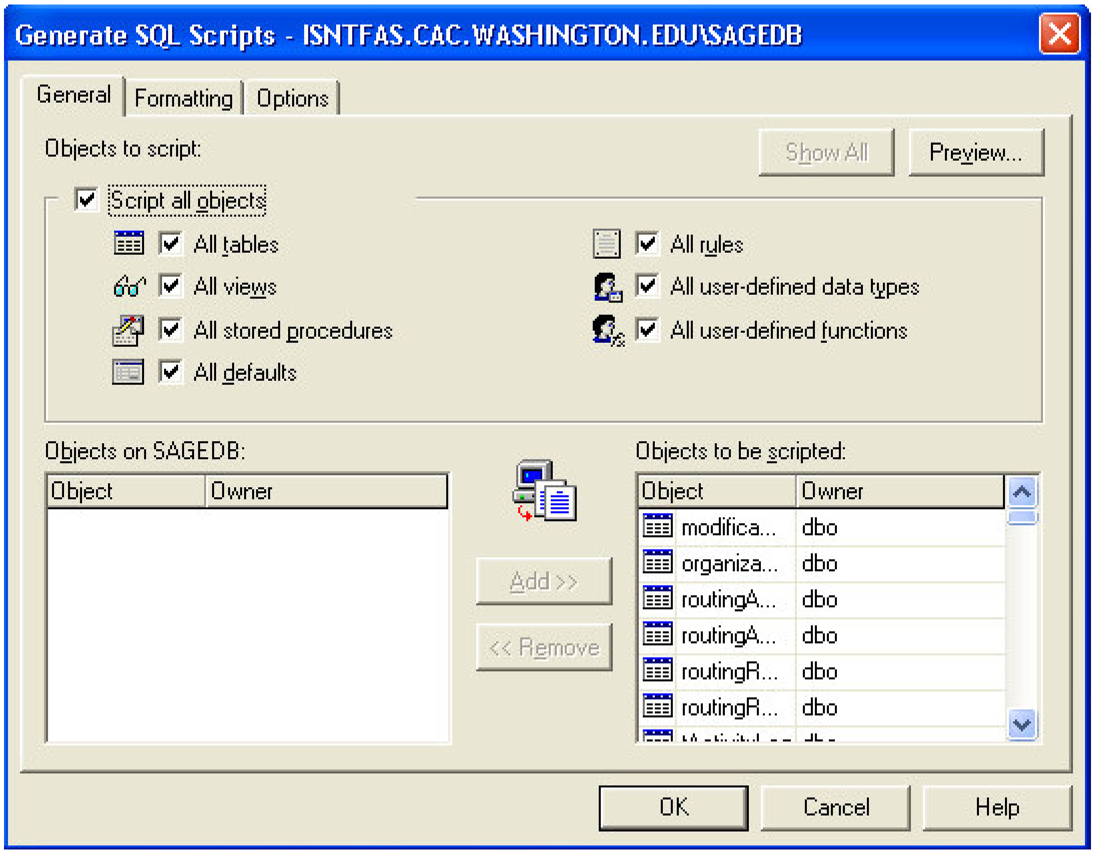
4. make it easy for developers to work on the database so they won't be tempted to use "shortcuts" (ie, working in EM and scripting out later if they remember)
5. allow complete database generation from scripts alone, at any point in a development cycle

We experimented with several different approaches, and I have to credit Ken Henderson, who wrote an excellent book, The Guru's Guide to SQL Server™ Stored Procedures, XML, and HTML. I found the chapter on source control very useful. Where I part ways with Mr. Henderson is in his automation of the process -- he wrote a utility, and took advantage of some of the capabilities of the Query Analyzer to use command-shell type scripting to automate checkout. I wanted to keep things even simpler.

Step-by-Step Howto - Implementing Our Solution:

Step 1: Script out the database

Using the EM, right-click on your database and select: All Tasks \ Generate SQL. Make sure to include ALL the objects (sprocs, udfs, udts, views and tables - if you have DTS packages, export them too):


Generate SQL Scripts - ISNTFAS.CAC.WASHINGTON.EDU\SAGEDB

General | Formatting | Options

Objects to script:

[✓] Script all objects
[✓] All tables
[✓] All views
[✓] All stored procedures
[✓] All defaults
[✓] All rules
[✓] All user-defined data types
[✓] All user-defined functions

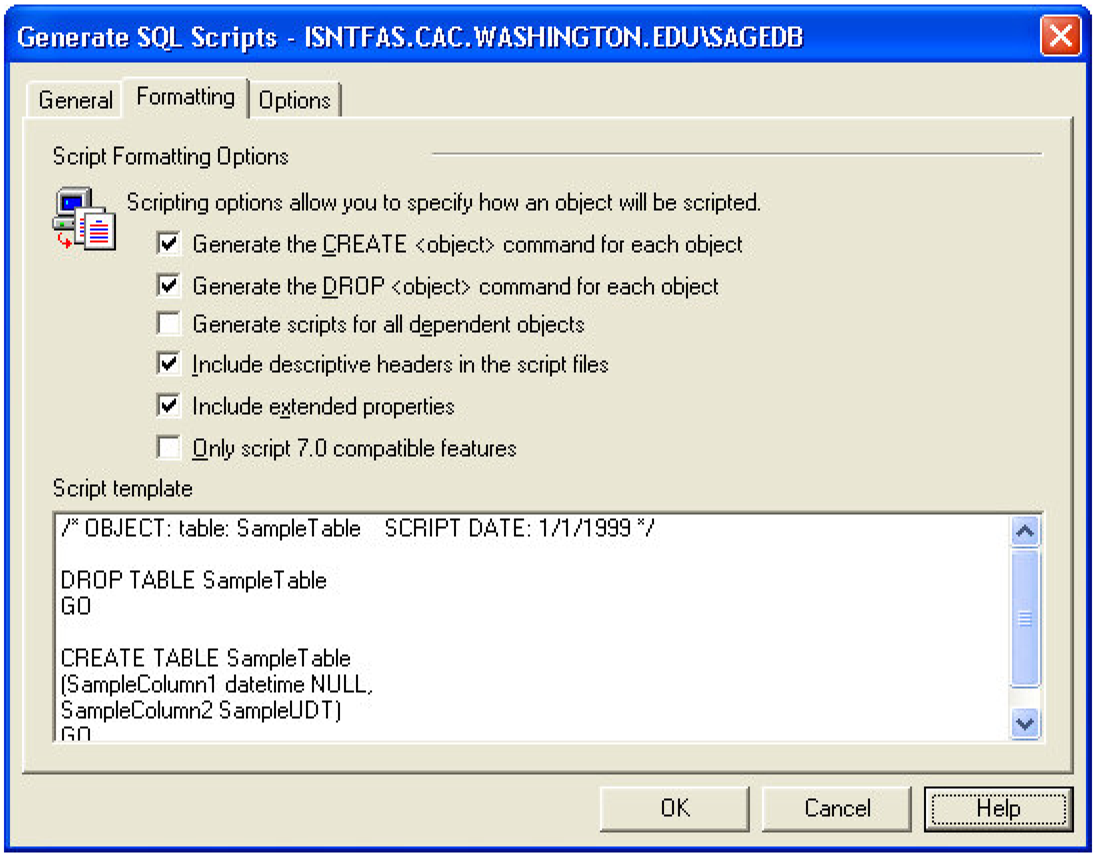
Objects on SAGEDB:

Object	Owner

Objects to be scripted:

Object	Owner
modifica...	dbo
organiza...	dbo
routingA...	dbo
routingA...	dbo
routingR...	dbo
routingR...	dbo

Then click over to the formatting tab, and make sure that you generate both CREATE and DROP statements, extended properties and descriptive headers if you want them. Make sure NOT to script SQL 7.0 compatibility unless you a) know what you're doing and b) have a reason to do this:



Script Formatting Options

Scripting options allow you to specify how an object will be scripted.

- [✓] Generate the CREATE <object> command for each object
- [✓] Generate the DROP <object> command for each object
- [ ] Generate scripts for all dependent objects
- [✓] Include descriptive headers in the script files
- [✓] Include extended properties
- [ ] Only script 7.0 compatible features

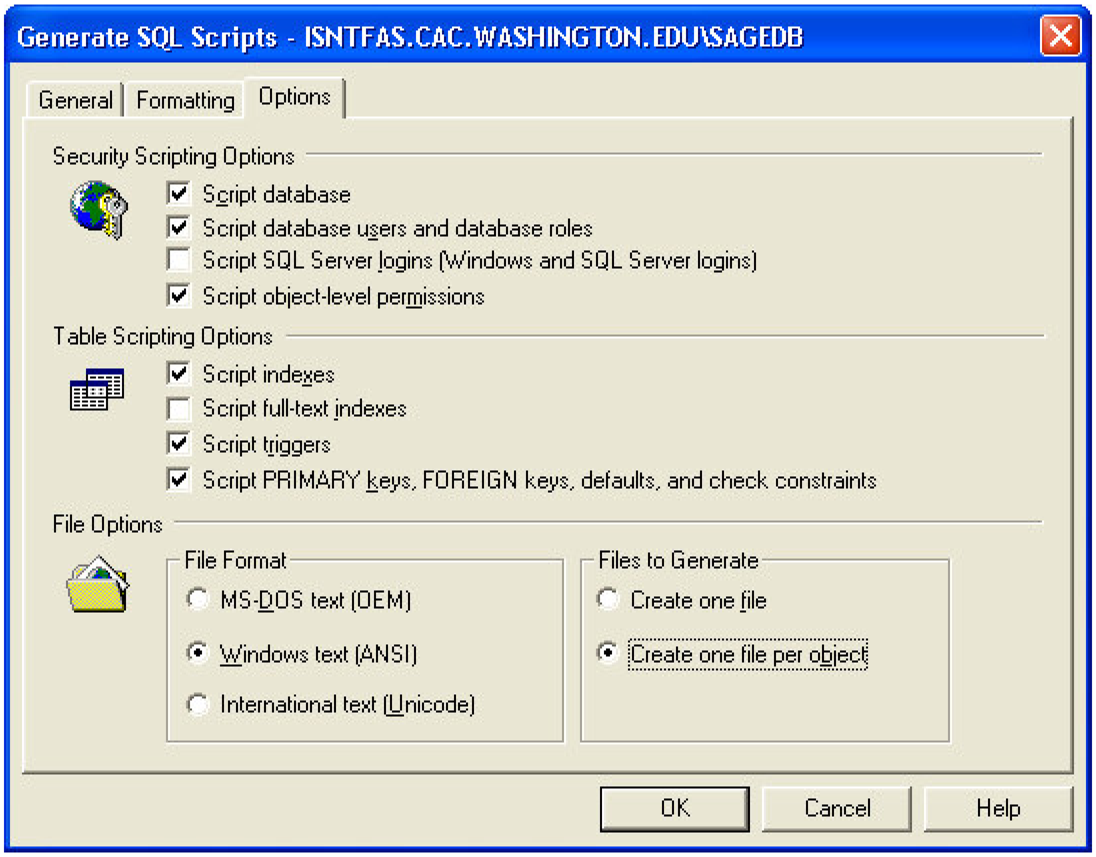
Script template

```
/* OBJECT: table: SampleTable   SCRIPT DATE: 1/1/1999 */

DROP TABLE SampleTable
GO

CREATE TABLE SampleTable
(SampleColumn1 datetime NULL,
SampleColumn2 SampleUDT)
GO
```

Then make sure that you have the correct options selected. You need to script the database, which will let you generate the whole DB from this script, you will script the database users and roles, and you will script object level permissions. If your project uses a separate DBA or DBA staff, some of these options might change, and you should confer with them (they might prefer to maintain users/roles/permissions). Conversely, if you maintain the whole database, then you may want to manage the SQL server windows logins here.

Also you will want to make sure that you are scripting out your primary keys, indexes, triggers etc. Henderson warns against using unicode - we have always selected ANSI / windows text.

Finally, make sure to create one file per object -- otherwise you will end up with one enormous file that does everything and that is of no benefit when you're implementing source control.



Generate SQL Scripts - ISNTFAS.CAC.WASHINGTON.EDU\SAGEDB

Security Scripting Options

- [✓] Script database
- [✓] Script database users and database roles
- [ ] Script SQL Server logins (Windows and SQL Server logins)
- [✓] Script object-level permissions

Table Scripting Options

- [✓] Script indexes
- [ ] Script full-text indexes
- [✓] Script triggers
- [✓] Script PRIMARY keys, FOREIGN keys, defaults, and check constraints

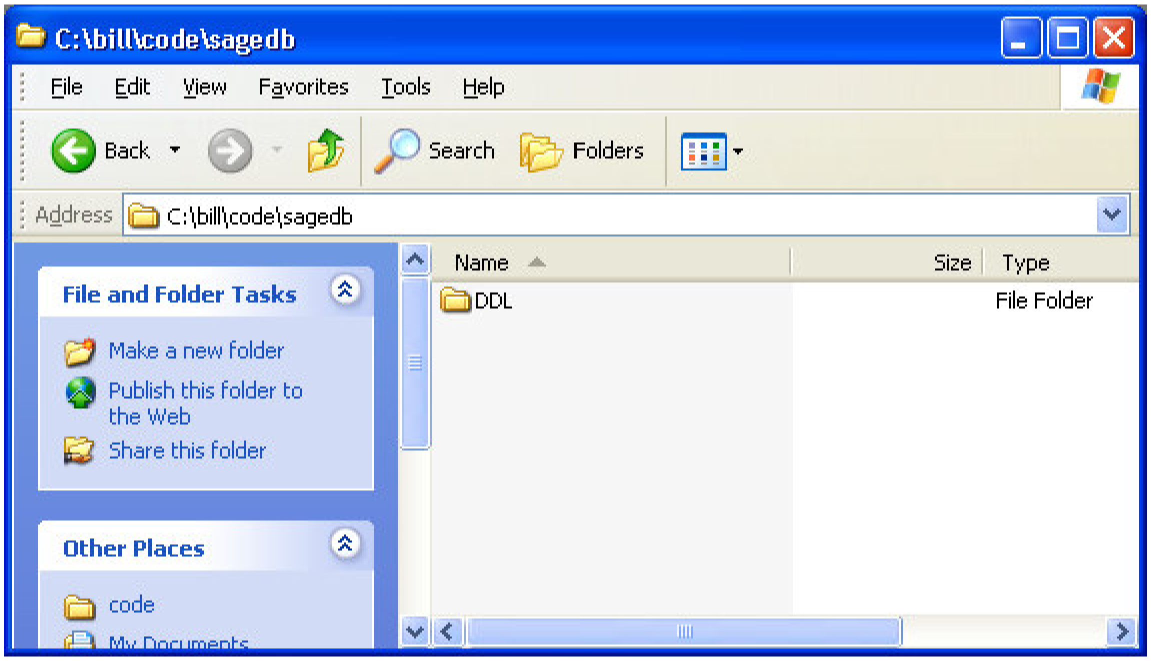
File Options

File Format: MS-DOS text (OEM) [✓ Windows text (ANSI)] International text (Unicode)

Files to Generate: Create one file [✓ Create one file per object]

Some people like to make a folder hierarchy (separate folders for UDF's, sprocs, etc.) we opted not to - mainly because SQL server scripts everything out to a single directory, and we're trying to keep things very simple. As you will see, we do use scripting from the enterprise manager from time to time, so we don't need to move files around. Similarly, you will notice that SQL server scripts out extensions like *.PRC for stored procs, and *.UDF for UDFs. We initially renamed everything to *.SQL using a script, but realized that we can use the extensions in Windows explorer to differentiate between file types, thus making working with one big folder more manageable. Unless you have a palm pilot - PRC is not associated with any other application (for example), so you can work with the suffixes.

What I would recommend, is if you have a parent tree for your overall project, make a folder named with your database name, and then perhaps a subfolder called something like "DDL." Ideally you will have scripted your database onto a sensible path like:

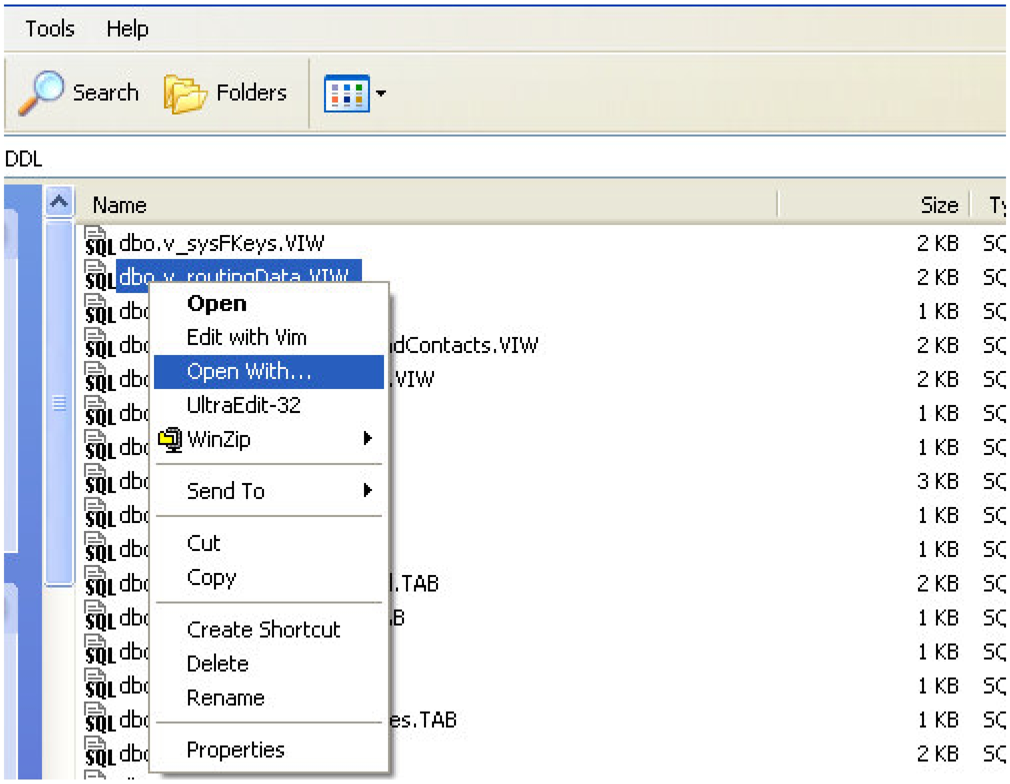


Step Two: Add the files to VSS
I am assuming you know how to use Visual Sourcesafe. I guess many people don't use the VSS client, but trust me, it's pretty user-friendly. If you don't know how - hit Google up for this, it's been written about pretty extensively. Add the folder(s) containing your SQL code to a logical node in VSS, by dragging them there and then write a comment if you want to.

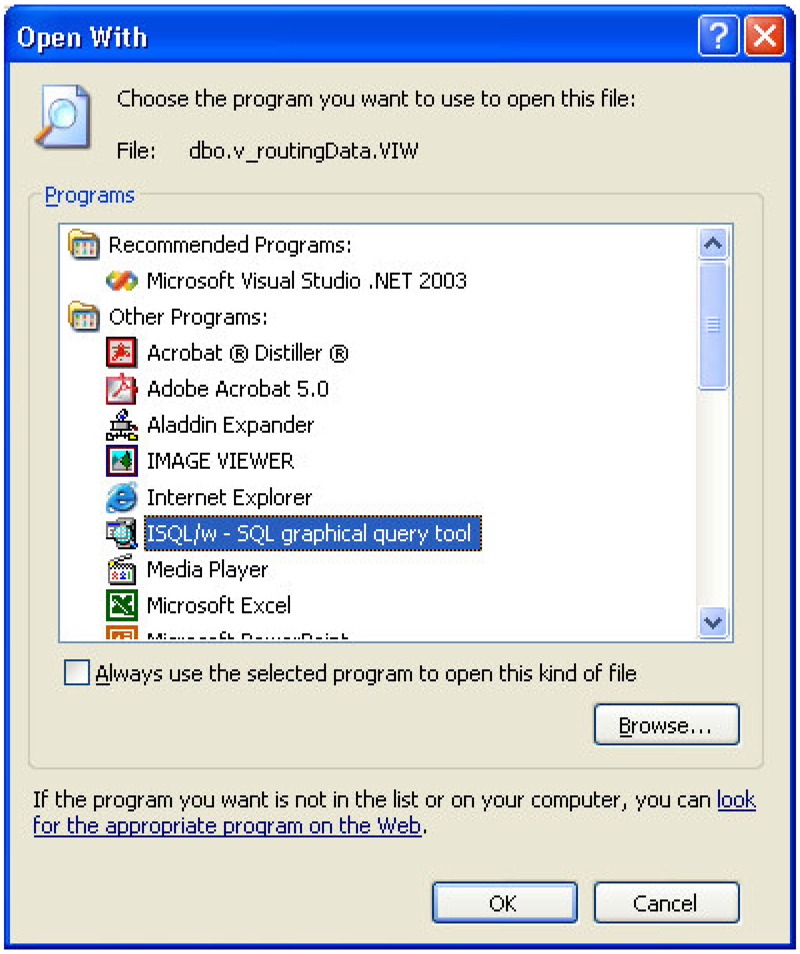
Now, you've done the easy part -- congratulations - your database is in SourceSafe. Like anything else, for this to work, you have to begin to use a different way of working that embraces the concept of source control in day-to-day practice.

Step Three: Associate the SQL Files With Query Analyzer
We explored some different ways of working on development day-to-day. For one thing - nobody uses Interdev for SQL development, period. The mesh between SQL Server and Interdev is too tenuous to support this. The basic methodology is this:

Associate the SQL file extensions in your DDL folder (PRC, UDF, UDT, TAB) with ISQLW (the Query Analyzer) by right-clicking on the file and choosing "Open With":



dbo.v_sysFKeys.VIW 2 KB
dbo.v_routingData.VIW 2 KB



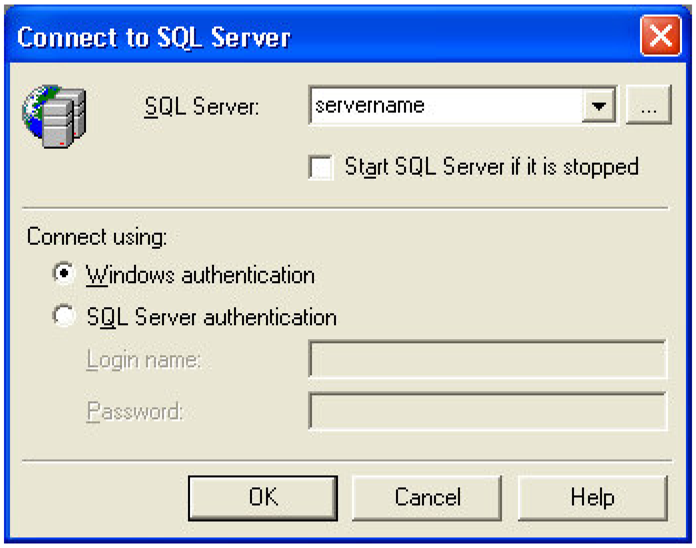
Choose the program you want to use to open this file:

File: dbo.v_routingData.VIW

☐ Always use the selected program to open this kind of file

If the program you want is not in the list or on your computer, you can look for the appropriate program on the Web.

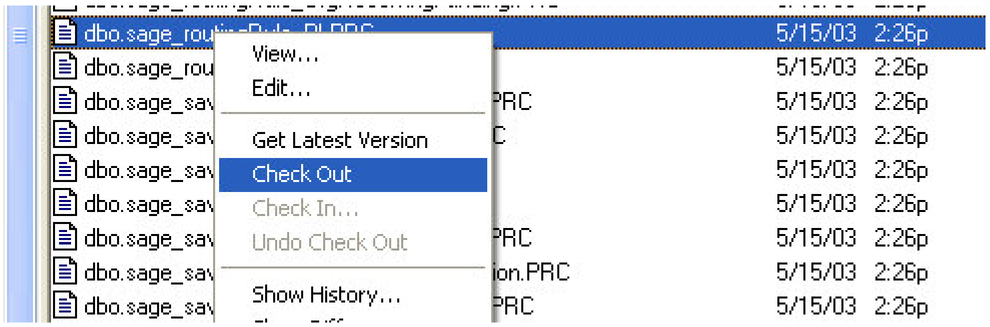
When you open a file using open with, the second time you come through, you can simply right-click and choose "ISQL/W - SQL Graphical Query Tool" to open the file. Each time you do this, an instance of the Query Analyzer opens, prompting you for login information:



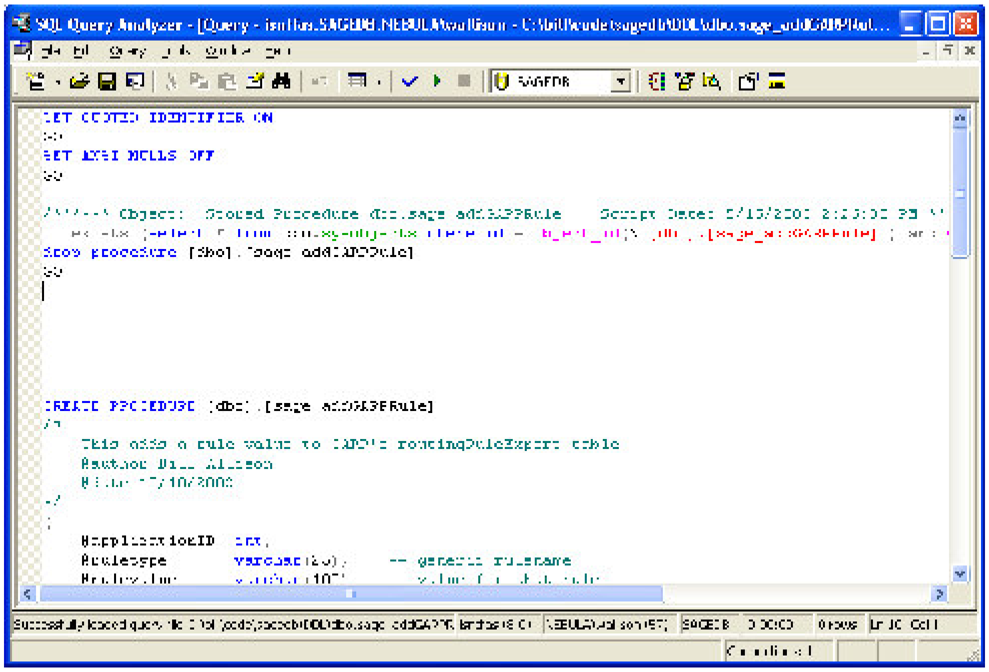
I would highly recommend that you use integrated Windows authentication if possible. You should also verify that you are working off of the proper servername - as it will just use the name of the last server you used.

Step Four: Check out an Object, Make Changes, Test, Then Check It Back In
Some of how you do this will depend on how your development environment is setup. If each developer has their own database instance, you might work differently than if everyone is working off a shared development database. MOST IMPORTANTLY: Don't break the build. In other words, if you will be working on (for example) a stored procedure you might do this:

1) Check out the file for the sproc you want to work on:



Now that you've checked it out, it will be writable in your DDL folder. Right-click the file and open it with the Query Analyzer, and it will look something like this:


SQL Query Analyzer - [Query - [illegible].SAGEDB.NEBULA\[illegible] - C:\bill\code\sagedb\DDL\dbo.sage_addGAPPRul...
SAGEDB

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

/****** Object:  Stored Procedure dbo.sage_addGAPPRule    Script Date: 5/13/2005 2:13:02 PM ******/
   [illegible] from dbo.sysobjects where id = object_id(N'[dbo].[sage_addGAPPRule]') and [illegible]
drop procedure [dbo].[sage_addGAPPRule]
GO




CREATE PROCEDURE [dbo].[sage_addGAPPRule]
/*
    This adds a rule value to GAPP's routingRuleExpert table
    @author Bill Allison
    @[illegible] 5/10/2005
*/
(
    @applicationID  int,
    @ruletype       varchar(20),    -- generic rulename
    @[illegible]    varchar(100)    [illegible]
```

Successfully loaded query file [illegible] sagedb\DDL\dbo.sage_addGAPPR... [illegible] NEBULA\[illegible] SAGEDB 0:00:00 0 rows Ln 10, Col 1

Key point: make sure the database you're using is the right one, at the top.

So, as you can see, in query analyzer you will get nice syntax highlighting, and everything. You could also open the file in your favorite editor - but I favor QA for a reason: when you want to implement your changes on the database, you simply press the green execute button, or Control-E. If the compile failed, you will get a detailed error message, and if it works, you will see "The command(s) completed successfully." At the point that you have made your changes, VSS still has the original, in case you goof. If you are happy with your changes, simply press the "Save" button, and right-click / check in on the VSS client.

As you can see, this is a fairly simple approach. This is the way we handle stored procedure, udf, udt and view development. See below for some exceptions.

Step Five: Promoting a Release

When we promote a release of our product, we script *everything* out of the database as we did the first time. We have found that this works as an added safeguard - it catches obsolete objects, and also finds the stray things that got into the database but bypassed source control. You would do this scripting from a stable database, for example a staging database that is used as a release candidate (to be copied by the DBAs to production). Here's how we handle a release:

Making a Release

1. check out all DB objects in directory (right-click the folder and check out everything)
2. script entire DB as you did originally - to the same folder (everything is writable, so we'll write right over the other files)
3. when scripting is complete, check all the files back in
4. anything still checked out *doesn't exist in the database* - check to make sure the items are not needed and then delete them from VSS (*not* using the "Destroy" option)
5. Now, look at the DDL folder -- sort the window by "Attributes" (might have to add this tab using windows explorer) -- all the checked in items will be "RA" - anything listed as "A" exists in the database but is not under source control -- verify that these things are good, and then drag them

onto the DDL folder in VSS (add to VSS)

6. LABEL your release. (right-click the DDL folder in VSS and choose label. You should do this in a way that is harmonious with the labelling you use for the associated code

Exception for TABLE DDL
On our project, we are not making very many table changes at this point. Further, many of the developers prefer using the graphical tools for table development. We reached a compromise - if a developer wants to use the standard method (ddl by hand), fine. Alternately, they may simply use the EM for table development. When we promote a new release, we will get the new table DDL when we script out and label.

Exception for New Objects, and Views
Most often, new objects are created using an existing one as a model. If a developer wants to use the EM to make a new object that's OK too - they can simply script it out the single object and add it to VSS. We also recognize people's fondness for using query-builder to make a complex view, so we accomodate this in the same manner.

Conclusions

We have been successfully using this method for VSS - SQL Server integration now for some time. We have achieved all of our goals, and have increased the skills of all the developers on the team by involving them in database development while maintaining the integrity of the system.

If you have any feedback, or other suggestions, I'd love to hear them. Please don't hesitate to contact me.

share feedback -

Return To Bill's Home

library / articles

Google Search
[✓ Search WWW] Search allisonfamily.org