# Demo Scripts - SQL Source Control (SOC)

1. Using F5 from Visual Studio, show the Demo Simple Talk website and show the list of articles and the RSS feeds.

**Link SimpleTalkDev\_Steve to source control.**

1. Run SimpleTalkDev.sql to create the SimpleTalkDev\_Stevedatabase
2. Show that there are objects.
3. Link it to SQL Source Control
4. Explain the difference between shared and dedicated (shared is simpler as there is a single DB, but developers can make breaking changes and impact others, and overwrite each other.
5. Choose Dedicated.
6. In the commit tab, show the list of objects.
7. Link the following static data tables: CountryCodes and RSSFeeds
8. Show that the data now appears in the commit tab just as the schema did.
9. Do the initial commit.
10. Open TortoiseSVN and show the folder structure. Open a table in notepad to show that it’s just the creation SQL.

**Create a second development database called SimpleTalkDev\_Grant**

1. It’s a team tool after all, so create a DB using a snippet *create\_database\_rgdemo* to create a database and enable it with the TRUSTWORTHY so it will work with tSQLt. An explanation is here: <http://stackoverflow.com/questions/10530190/tsqlt-trustworthy-and-clr-requirement>
2. Rgdemo\_create\_database contains:

|  |
| --- |
| CREATE DATABASE <DBName,, SimpleTalkDev\_>  GO  USE <DBName,, SimpleTalkDev\_>  GO  ALTER DATABASE <DBName, ,SimpleTalkDev\_> SET TRUSTWORTHY ON |

1. Useful tip for those who use SQL Prompt snippets: Use Ctrl-Shift-M to fill in the Template Parameters, and set DBName to **SimpleTalkDev\_Grant** and press Enter then F5 to create the database.
2. Link to the same repo as SimpleTalkDev\_Steve (right-click to the right of “linked to” in the Setup tab to copy the repo URL)
3. In Get Latest, show the schema creation scripts, including the linked static data tables.
4. Expand the stored procedures folder in the Object Explorer to prove that there’s nothing there yet. It also means that it’s more impressive when the objects appear after the ‘get latest’ is run.
5. Do a Get Latest, which should populate the database. Explain that the SQL Compare engine is being used under the hood to compare source control to the database.
6. **Now we make some changes to SimpleTalkDev\_Steve**:
   1. Add a new table (we should decide on a specific table so the subsequent demos can have the same one).
   2. Add a new RSS feed record. This shows how static data changes are shared.
   3. Modify prc\_getContacts and change something simple in the Select statement. Also add a comment that is relevant to the audience (eg, the location of the show)
   4. Change the colour of the Simple Talk font (Andrew Denty – TBD)
7. Now build the web app again so we can see the effect of the above local changes.
8. Go to the commit tab, type in a comment, and commit.
9. Go to **SimpleTalkDev\_Grant** and demonstrate how easy it is to take these changes using Get Latest. Open the procedure to prove that it worked.

That’s it for the SOC part.

### SQL Test / tSQLt

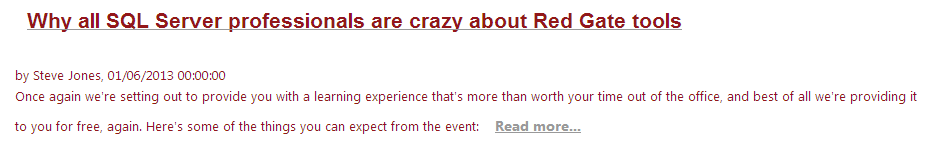
SQL Test is a Red Gate tool that provides IDE integration for tSQLt.

1. Describe SQL Test and tSQLt
   * NUnit for databases. Like NUnit, tSQLt is an open source tool.
   * SQL Test is a Red Gate tool, providing integration into SSMS.
   * Easy to share tests via SOC as they are implemented as stored procedures.
   * Deployment scripts don’t contain tests as they can be excluded using a built-in SQL Compare filter.
   * A few SQL Cop tests are included by default. These ‘best practice’ tests are useful because the same test code can be used on any database, whereas unit tests need to be custom written depending on the business logic of the application
2. Open the file prcAddArticle\_and\_Test.sql in SSMS. This has five steps to follow to demonstrate SQL Test. The existing articles we can see on the website were added manually. We need a stored procedure to do this. We will start with a requirement to test that we can add an Article Title.
   1. Start by adding a Test to check that the procedure (that we haven’t written yet) prcAddArticle can take a @Title parameter and save this to the Articles table. Use SQL Test to add the new test to the Unit Tests test class.
   2. Run the test in SQL Test. This fails, as the Test is trying to call a non-existent procedure. We do it this way to avoid false positives.
   3. Now add the procedure. This adds the completed procedure. True TDD dictates that only the minimum functionality should be implemented each time to get the test passing.
   4. Now let’s just execute the procedure manually (use the commented out SQL). Refresh the website. It should have added a new article.
   5. Run SQL Test again to show that the test now passes.
3. Now that we have a working unit test, we can commit the change to source control. This adds not only the new procedure, but also the test, as it’s just another stored procedure. This means that all tests written by all developers can be easily shared amongst the team, ensuring that developers can make changes and check that they aren’t causing others’ code to break.
4. Do a Get Latest from **SimpleTalkDev\_Grant** and run the tests on this database using SQL Test to demonstrate that everything is working on Grant’s database.
5. Now Grant “the cowboy coder” Fritchey is going to make a breaking change...
6. Acting on behalf of Grant, quickly modify the prcAddArticle procedure on **SimpleTalkDev\_Grant** and change the following line from 100 to 42 characters:

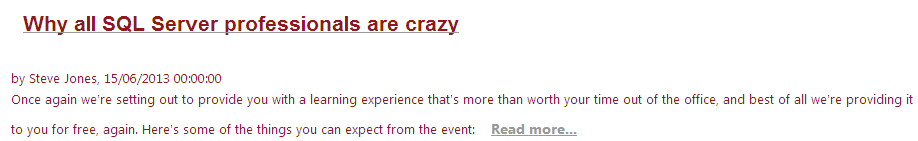
@Title VARCHAR(42) = NULL,

1. Because Grant isn’t a good developer, commit the change, putting in an arrogant comment, such as “I’m too good to need tests”.
2. From **SimpleTalkDev\_Steve** go to the ‘Get Latest’ tab, inspecting the nature of Grant’s change by opening the History by right clicking on the change. Despite the comment being poor, pull the modified prcAddArticle to **SimpleTalkDev\_Steve**.
3. Steve is understandably skeptical about Grant’s change, so is going to manually run the stored procedure (again, run the SQL in the comment of the procedure). Open the web site and this will have truncated the title, which had the change been deployed would have been very embarrassing:

This:



Truncates to:



1. Of course Grant should have run the SQL Test tests right after making his change.
2. Now find Grant’s database in SQL Test and demonstrate the failing test.
3. As Grant, edit the procedure again, changing the parameter length back to 142.
4. Run the tests in SQL Test and will now all pass.
5. The End…

### Segue to the CI session:

1. Open up TeamCity, which will have been quietly running in the background. A Test will have failed. This can be used as a cliffhanger to tease the audience into coming to the CI session. What this failed test is has yet to be decided. Ideas welcome.
   * Ideas: Tests that run on local changes may not work in combination with changes committed my by other developers. This is why there is a benefit to run integration tests as part of an automated build process.
   * Maybe the deployment script failed? The creation script test might be simplest? Or an upgrade test? The trouble with the latter is that “migrations” might be raised, and that still doesn’t work that well, and isn’t supported with the TeamCity plugin.
   * Maybe a number of things will have gone wrong, demonstrating that there is value to CI beyond running unit tests locally?

# FAQ

Here are some questions that might be asked.

**Q: Which source control systems are supported?**

A: All of them. As TFS, SVN and Vault are centralised source control systems (rather than distributed), SOC will perform a single stage check in, committing the changes directly to the centralised system.

If you are using DVCS, such as GIT or Mercurial, you will save the changes to your local clone of the repository and push the changes when you’re ready.

You can also use your own version control client to do the “check-in” and “get latest”. You do this using the “link to working folder” option in the link dialog. This method means that VSS users, and users of the more obscure version control tools, can benefit from SOC.

**Q: How do I do atomic commits?**

A: If you are using GIT or Mercurial, then SOC will save the changes to your working folder, allowing you to do the commit and push atomically with your app code. To do atomic commits for TFS and SVN, for example, use the “Link to working folder” option in the SOC link dialog, and use the standalone source control client tool to do the atomic commit.

**Q: Does this work with SSDT?**

A: Yes. You simply link SOC to the database project folder, allowing those who prefer offline development in Visual Studio to collaborate with SSMS users.

**Q: Is the custom migrations feature supported on GIT/Mercurial**

A: We’re working on it – please add your votes on UserVoice.

**Q: How does licensing work?**

A: To automate Red Gate command line tools on a server, such as a build server, an Automation License is needed. This isn’t part of the SQL Developer Bundle, as this is a client-licensed offering. The Automation License is priced at $595/agent, $1495 for 5 agents, $2495 for 10 agents.

**Q: Why haven’t you implemented <insert feature here> yet?**

A: There is a UserVoice suggestions forum for SQL Source Control. Please vote for your requests here, or speak to a Redgater at the booth, who will log your request.