

| | |
|---|-----|
| 1. SQL Source Control 3 documentation | 3 |
| 1.1 Requirements | 4 |
| 1.1.1 Permissions | 5 |
| 1.2 Supported source control systems | 6 |
| 1.2.1 Setting up a local SVN repository | 7 |
| 1.2.2 Setting up an SVN server | 8 |
| 1.3 Licensing | 10 |
| 1.3.1 Activating | 11 |
| 1.3.2 Deactivating | 17 |
| 1.3.3 Troubleshooting licensing and activation | 20 |
| 1.4 Installing | 23 |
| 1.5 Updating | 24 |
| 1.5.1 Using Check for Updates | 25 |
| 1.5.2 Troubleshooting Check for Updates errors | 27 |
| 1.5.3 Turning on Frequent Updates | 28 |
| 1.6 Linking to source control | 30 |
| 1.6.1 SVN | 31 |
| 1.6.2 TFS | 33 |
| 1.6.3 Vault | 35 |
| 1.6.4 Git, Mercurial, or other systems | 37 |
| 1.6.4.1 Linking to a working folder | 38 |
| 1.6.4.2 Linking to a custom setup | 40 |
| 1.6.4.2.1 Working with config files | 42 |
| 1.6.4.3 Example - source-controlling database schema and application code together using a working folder | 43 |
| 1.6.5 Evaluation repository | 51 |
| 1.6.5.1 Moving the evaluation repository to a SVN server | 52 |
| 1.6.6 Database development models | 54 |
| 1.6.7 Static data | 56 |
| 1.6.8 SSDT projects | 58 |
| 1.7 Using SQL Source Control | 59 |
| 1.7.1 Committing changes | 60 |
| 1.7.2 Getting the latest version | 62 |
| 1.7.3 Viewing source control history | 63 |
| 1.7.4 Getting a specific version | 65 |
| 1.7.5 Undoing changes | 67 |
| 1.7.6 Conflicts and merging | 68 |
| 1.7.7 Using filters to exclude objects | 70 |
| 1.7.8 Branching and merging | 73 |
| 1.7.9 Deploying a database from source control | 75 |
| 1.7.10 Working with migration scripts | 77 |
| 1.7.10.1 Example - deploying with migration scripts | 80 |
| 1.8 Options | 96 |
| 1.8.1 Comparison options | 97 |
| 1.8.2 Logging changes to shared databases | 99 |
| 1.8.3 Changing the location of the working base | 102 |
| 1.8.4 Disabling TFS policy checking | 103 |
| 1.8.5 Changing the number of revisions shown in the History dialog box | 104 |
| 1.8.6 Enabling multiple SSMS windows | 105 |
| 1.9 Migration scripts V2 beta | 106 |
| 1.9.1 What migration scripts do | 107 |
| 1.9.2 Enabling the Migrations V2 beta | 109 |
| 1.9.3 Disabling the Migrations V2 beta | 110 |
| 1.9.4 How V2 migration scripts are used in deployment | 112 |
| 1.9.5 Setting the location of the temporary database | 115 |
| 1.9.6 Example V2 migration scripts | 117 |
| 1.9.7 Example: renaming a table without data loss | 124 |
| 1.9.8 Example: writing a V2 migration script that affects objects not yet in the target database | 132 |
| 1.10 Troubleshooting | 138 |
| 1.10.1 How SQL Source Control works behind the scenes | 139 |
| 1.10.2 Can't link to TFS 2012 or Visual Studio Online | 141 |
| 1.10.3 Linking fails due to SVN pre-commit hooks | 142 |
| 1.10.4 Logging and log files | 143 |
| 1.10.5 Object changed by Unknown | 145 |
| 1.10.6 Error messages | 147 |
| 1.10.6.1 Failed to locate the [object name] for the [object name] | 148 |
| 1.10.6.2 Failed to resolve no-ops after 5 tries | 150 |
| 1.10.6.3 ICredentialsProvider is unset, therefore can't get | 152 |
| 1.10.6.4 Newer version of Proc found - please update your Source Control | 153 |
| 1.10.6.5 SQL Source Control can't access this database because the directory once contained more than one SQL Server Database Project (.sqlproj) file | 154 |
| 1.10.6.6 System.OutOfMemoryException | 155 |
| 1.10.6.7 The specified path, file name, or both are too long | 156 |

| | |
|--|-----|
| 1.10.6.8 The Team Foundation Server for this workspace does not support one or more of the checkin options you have selected | 157 |
| 1.10.6.9 This commit doesn't meet the server's policy requirements, or the policy isn't configured on your machine | 158 |
| 1.10.6.10 Unsaved documents cannot be cut or copied to the clipboard from the Miscellaneous Files project | 160 |
| 1.10.7 About the RedGateLocal schema | 161 |
| 1.10.8 Third-party software used by SQL Source Control | 162 |
| 1.11 Release notes and other versions | 163 |
| 1.11.1 Frequent Update release notes | 164 |
| 1.11.1.1 SQL Source Control 3.8 Frequent Updates release notes | 165 |
| 1.11.1.2 SQL Source Control 3.7 Frequent Updates release notes | 170 |
| 1.11.2 SQL Source Control 3.8 release notes | 171 |
| 1.11.3 SQL Source Control 3.7 release notes | 174 |
| 1.11.4 SQL Source Control 3.6 release notes | 176 |
| 1.11.5 SQL Source Control 3.5 release notes | 178 |
| 1.11.6 SQL Source Control 3.4 release notes | 180 |
| 1.11.7 SQL Source Control 3.3 release notes | 182 |
| 1.11.8 SQL Source Control 3.2 release notes | 183 |
| 1.11.9 SQL Source Control 3.1 release notes | 184 |
| 1.11.10 SQL Source Control 3.0 release notes | 187 |
| 1.11.11 SQL Source Control 2.2 release notes | 189 |
| 1.11.12 SQL Source Control 2.1 release notes | 190 |
| 1.11.13 SQL Source Control 1.1 release notes | 192 |
| 1.11.14 SQL Source Control 1.0 release notes | 193 |
| 2. Locking objects | 194 |
| 2.1 Setting up object locking | 197 |
| 2.2 Removing object locking | 203 |
| 3. Source-controlling static data | 206 |

SQL Source Control 3 documentation

About SQL Source Control

SQL Source Control is an add-in for Management Studio that links your database to your source control system. For more information, see the [SQL Source Control product page](#).

Loading Twitter Widget

Widget #449140573912330240

Quick links

★ [SQL Source Control 4 has been released](#)

[Supported source control systems](#)

[Linking to source control](#)

[Evaluation repository](#)

[Troubleshooting](#)

Requirements

To use SQL Source Control, you need:

- SQL Server Management Studio 2005 (RTM, SP1, SP2, SP3, and SP4), 2008 (RTM, SP1, SP2, R2, R2 SP1, and R2 SP2), 2012, or 2014 (CTP2)
- Windows Server 2008 R2, Windows Server 2012, Windows 7, or Windows 8
- .NET Framework 4 or later
- MDAC 2.8+

If you're using Team Foundation Server, you also need a compatible version of Team Explorer. For more information, see [this troubleshooting page](#).

Supported SQL Server versions

- SQL Server 2005 (RTM, SP1, SP2, SP3 and SP4)
- SQL Server 2008 (RTM, SP1, SP2, R2, R2 SP1, and R2 SP2)
- SQL Server 2012 (RTM)
- SQL Server 2014 (RTM)

SQL Source Control doesn't support SQL Server 2000, SQL Express, SQL Azure or SSMS Express.

Permissions

The permissions required to use SQL Source Control depend on your version of SQL Server and the objects in your database schema. If you don't have sufficient permissions, some objects may be missing from the change list. For example, user defined types won't be listed if you don't have permissions for the schema they belong to.

You need:

- `dbo` permissions for the database you want to link to source control. This is because SQL Source Control writes extended properties at the database level.
- permission to read the default trace
- permission to make all the listed changes in a commit or get latest

Committing or getting latest can fail if you don't have these permissions. When committing or retrieving fails, SQL Source Control uses transactions to try to roll the changes back; however, this won't always be successful.

- if you have encrypted stored procedures, you need `sysadmin` permissions to commit or retrieve them

Additionally, if you're using SQL Server 2008 or later, we recommend:

- `SELECT` permission for the system `viewsys.sql_expression_dependencies`. You may experience poor performance if you don't have this permission.
- `VIEW SERVER STATE` permissions to commit or retrieve some encrypted objects

For more information about permissions, see your SQL Server documentation.

Supported source control systems

SQL Source Control has full integration with:

- [Subversion](#) 1.5 and later
- [Team Foundation Server](#) 2005 and later (including [Visual Studio Online](#))
- [Vault Standard](#) and [Vault Professional](#) 5 and later

[Migration scripts](#) are only compatible with SVN, TFS and Vault. If you want to use migration scripts with other source control systems, including Git and Mercurial, try the [Migrations V2 beta](#).

Other source control systems

If you want to use a different source control system (eg Git or Mercurial), you can link to a working folder, or create a custom setup.

For more information about linking to different source control systems, see [Linking to source control](#).

Setting up a local SVN repository

This page describes how to set up a local Subversion (SVN) repository using TortoiseSVN, a free SVN client for Windows.

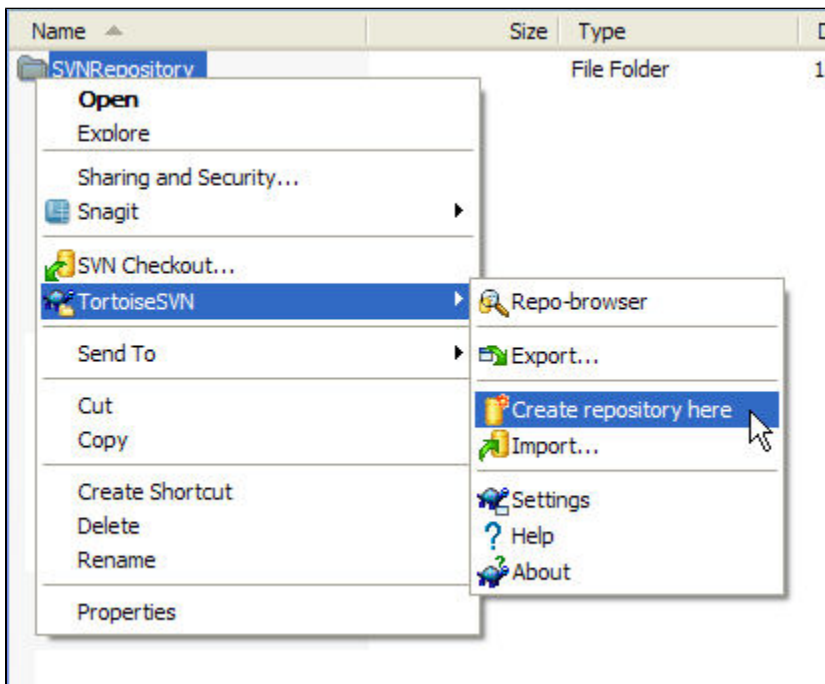
It's difficult to share changes and keep backups with a local repository, so we don't recommend it as a long-term solution. Instead, [set up a Subversion server](#).

Alternatively, you can use the SVN command line interface. For more information, see the [Subversion documentation](#).

Creating a repository

To create a local repository:

1. [Download and install TortoiseSVN](#). You might need to restart your computer after installation.
2. In Windows Explorer, browse to or create an empty folder where you want to create the repository; for example, `C:\SVNRepository`
3. Right-click the folder, and in the TortoiseSVN menu, select **Create repository here**:



The repository is created.

Using the repository

The URL for a local repository takes the form: `file:\\C:\<RepositoryFilePath>`

Use this URL to [link your database to SVN](#).

The URL is case sensitive.

Setting up an SVN server

This is a simple overview of setting up a Subversion (SVN) server using [VisualSVN Server](#), an installation and administration application for SVN on Microsoft Windows servers. This page doesn't cover manual installation and configuration of SVN, or installation on non-Windows servers.

For more detailed information about setting up an SVN server, see:

- [Chapter 6 of the Subversion documentation - Server Configuration](#) (svnbook.red-bean.com)
- [Chapter 3 of the TortoiseSVN documentation - The Repository](#) (tortoisesvn.net)

Installing with VisualSVN Server

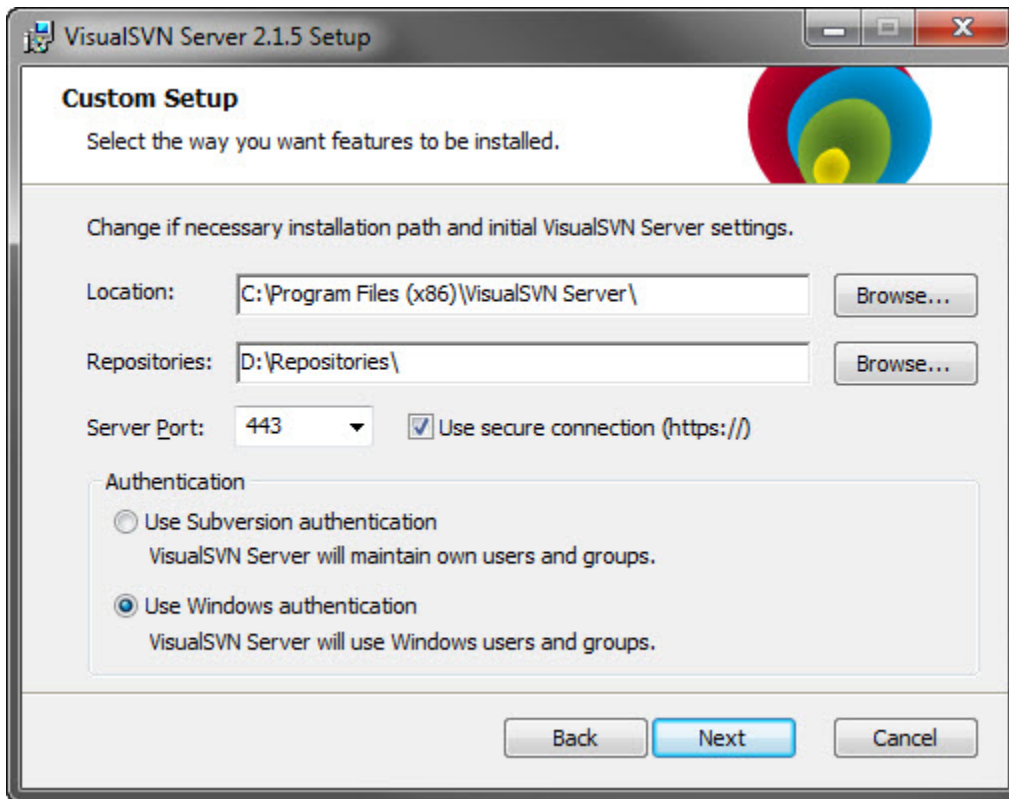
VisualSVN Server automates the setup of a SVN server, and is available both as a free tool (the Standard Edition), and as the paid Enterprise Edition. The Enterprise Edition includes Integrated Windows Authentication, as well as richer logging and administration tools.

This example uses the free version.

To set up SVN, download and run the VisualSVN Server installer on the server you want to use, then follow the wizard to complete the installation.

You can [download the VisualSVN Server installer from visualsvn.com](#) VisualSVN Server provides an [installation getting started guide](#).

Page 4 of the installation wizard lets you specify the location where the SVN repositories are created, and the type of authentication:

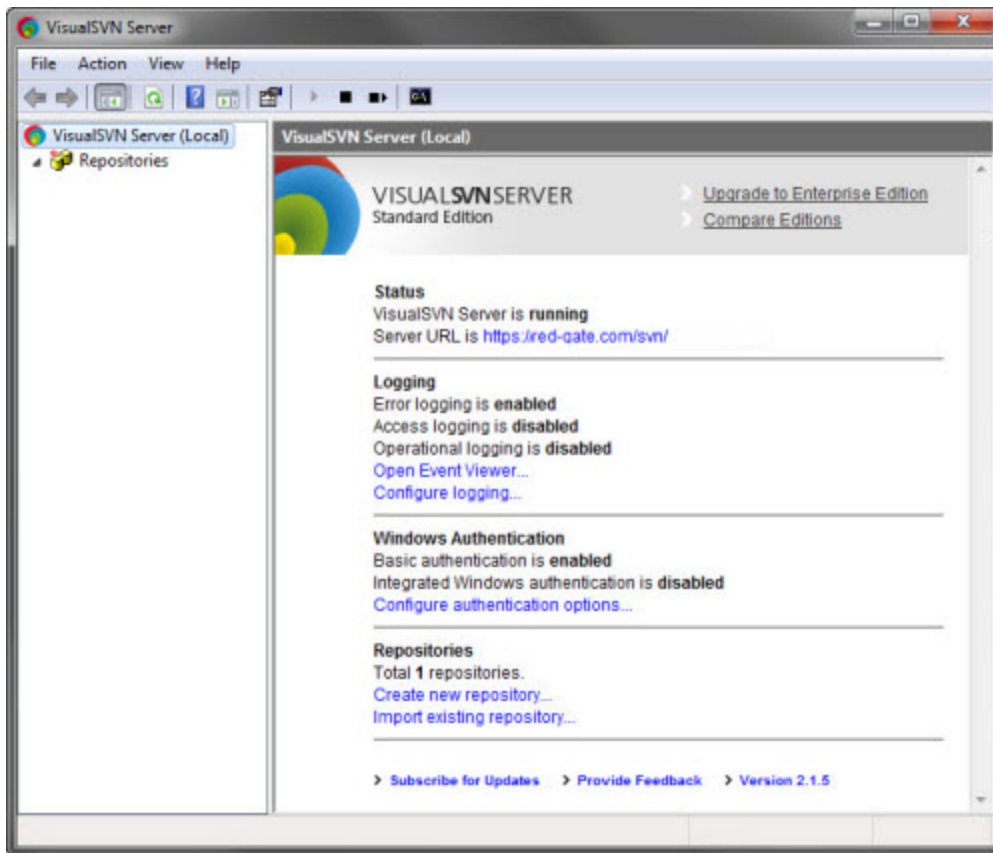


SVN authentication requires you to set up users and credentials on the SVN server.

Windows authentication allows you to use your existing Windows user accounts.

If you're using Windows authentication in VisualSVN Server Standard Edition (the free version), or SVN authentication in either edition, SQL Source Control may prompt you to enter your user name and password when linking a database source control.

At the end of the installation, run the VisualSVN Server Manager:



The Server Manager allows you to set up repositories and configure security.

To set up a repository to use with SQL Source Control:

1. In the Console Tree pane, to the left, right-click **Repositories**, and click **Create New Repository**.
The Create New Repository dialog box is displayed.
2. In Repository Name, type a name for the repository.
Optionally, to create the [recommended VisualSVN Server directory](#) structure in your repository, select the Create default structure check box.
3. Click **OK**.
The repository is created.

Using the repository with SQL Source Control

To use the repository with SQL Source Control, you need to create a folder for your database.

To create a folder in the repository:

1. Right click the repository, select **New**, and click **Folder**.
The Create Folder dialog box is displayed.
2. Specify a name for the folder, and click **OK**.

The folder is created.

To link a [database to SVN](#), you need the URL for the repository.

To find the URL of a repository in VisualSVN Server Manager, right-click the repository, and click **Copy URL to Clipboard**.

Licensing

When you install most Redgate products (apart from free ones), you have **14 days** to evaluate them without purchase.

For a few products, you have 28 days: DLM Automation Suite, DLM Automation Suite for Oracle, SQL Prompt, SQL Source Control, Source Control for Oracle.

If you need more time to evaluate a product, email licensing@red-gate.com.

Finding your serial number

When you buy a license for a product, we'll send you an invoice that contains your serial number to activate the product. Your invoice shows how many instances of a product the serial number can be used to activate. For information about how to activate, see [Activating](#).

If you can't find your invoice, you can view your serial numbers at red-gate.com/myserialnumbers. You'll need to log in to your Redgate account with the email address and password you provided when you bought the product.

If you need to reinstall products on the same computer (eg after installing a new operating system), you can reactivate them using the same serial number. This doesn't affect the number of distinct activations for the serial number. For information about moving a serial number to a different computer, see below.

Serial numbers for bundles and suites

If you've bought a bundle or suite of products, your serial number activates all the products in the bundle or suite. For bundles containing both server and client tools (such as the SQL DBA Bundle) you will have two serial numbers.

If you deactivate a bundle or suite serial number, all products using that serial number will be deactivated.

For information on which products are included in a bundle, see [Bundle history](#).

Changing the serial number used to activate a product

To change the serial number used to activate a product, on the **Help** menu, select **Enter Serial Number**. For some products, you will need to deactivate the old serial number first.

Moving a serial number to a different computer

To move a serial number to a different computer, deactivate the serial number on the old computer, then use it to activate the product on the new computer.

To deactivate a serial number, on the **Help** menu, select **Deactivate Serial Number**. If the Deactivate Serial Number menu item isn't available, use the [deactivation tool](#).

If you can't deactivate a serial number, use the [Request Extra Activations](#) page to request more activations for your serial number. You'll need to provide your serial number and the reason for the additional activations.

Activating

This page applies to a number of Redgate products, so the screenshots below may not match your product.

When you activate a product with your serial number, the licensing and activation program sends an activation request to the Redgate activation server, using checksums of attributes from your computer. The checksums sent to the activation server do not contain any details that might pose a security risk. The activation server returns an activation response and an encrypted key to unlock the software. The licensing and activation program should activate your product within a few seconds.

If you experience problems with activating your products, you'll be directed to [activate manually](#).


- [Activating using the GUI](#)
- [Activating using the command line](#)
- [Manual activation](#)

Activating using the GUI

These instructions apply to a number of Redgate products, so the screenshots below may not match your product.

To activate your products:

1. On the **Help** menu, click **Enter Serial Number**.
The product activation dialog box is displayed, for example:



Activate SQL Compare

Enter your SQL Compare serial number

Serial number

Your serial number is on your invoice or you can [find it online](#)


☒ **Track this activation**
Sends information about this activation (including your machine name) to Red Gate.
This is useful if you contact support about your activations. [More information](#)

If you purchased SQL Compare as part of a bundle, other products may be activated by this process. The products activated are listed when activation is completed.

E-mail (optional)
Please provide the email address you would like us to send update notifications to:

☒ **I'd also like to receive the Red Gate Newsletter.** [Read our privacy policy](#)

redgate Activate Cancel

2. Enter your serial number.
When you have entered a valid serial number,

is displayed next to the serial number box:

Activate SQL Compare

Enter your SQL Compare serial number

Serial number
000-000-123456-0000 ✓

Your serial number is on your invoice or you can [find it online](#)

☒ **Track this activation**
Sends information about this activation (including your machine name) to Red Gate.
This is useful if you contact support about your activations. [More information](#)

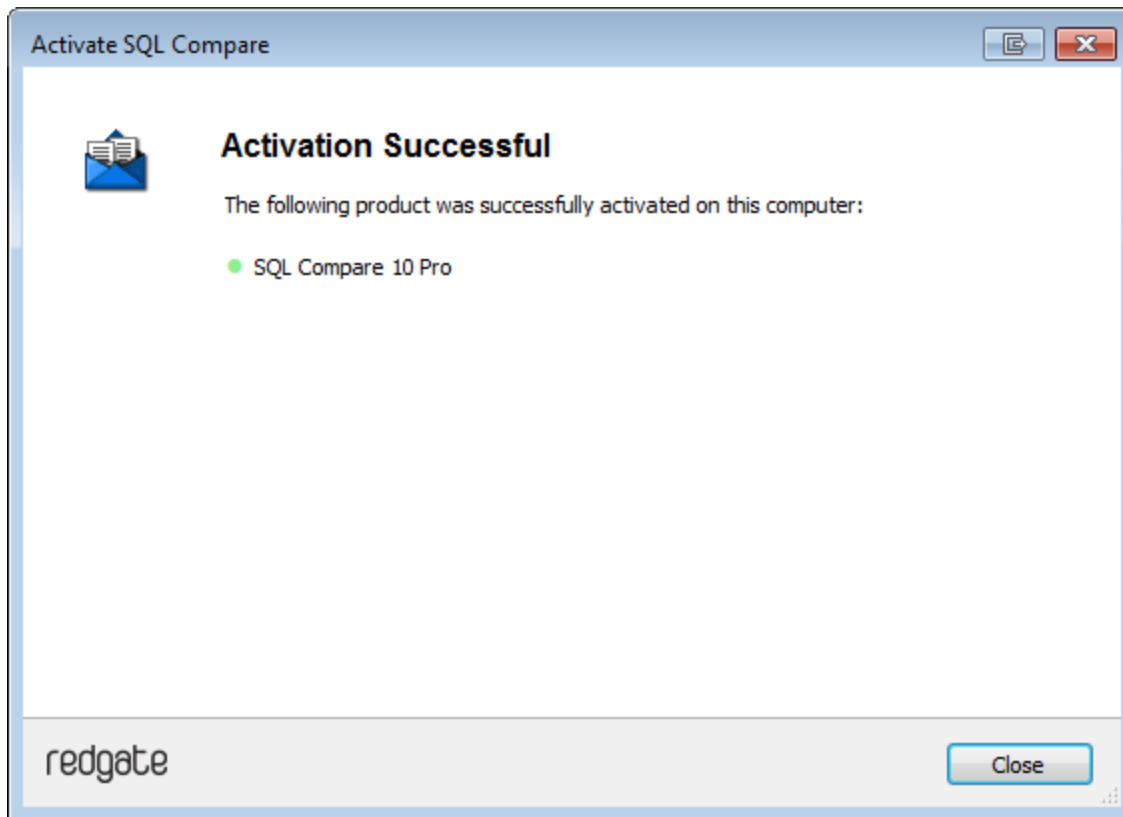
If you purchased SQL Compare as part of a bundle, other products may be activated by this process. The products activated are listed when activation is completed.

E-mail (optional)
Please provide the email address you would like us to send update notifications to:
user@example.com

☒ **I'd also like to receive the Red Gate Newsletter.** [Read our privacy policy](#)

redgate Activate Cancel

3. If you want to receive email updates from Redgate, enter your email address.
The list of identifiers and your email address may already be populated using information available to the licensing client from the Windows installation on your computer. No information is sent back to Redgate when the fields are populated.
When you activate your product, the optional information you entered is recorded by Redgate with your serial number. Your email address is not linked to the data collected should you consent to participate in the Quality Improvement Program provided with some Red Gate products.
4. Click **Activate**.
Your activation request is sent to the Red Gate activation server.
When your activation has been confirmed, the **Activation successful** page is displayed, for example:



If there is a problem with your activation request, an error dialog box is displayed. For information about activation errors and what you can do to resolve them, see [Troubleshooting licensing and activation errors](#). Depending on the error, you may want to try [manual activation](#).

5. Click **Close**.
You can now continue to use your product.

Activating using the command line

Open a command prompt, navigate to the folder where your product executable file is located and run a command with the following syntax:

```
<name of productEXE> /activateSerial:<serialNumber>
```

For example:

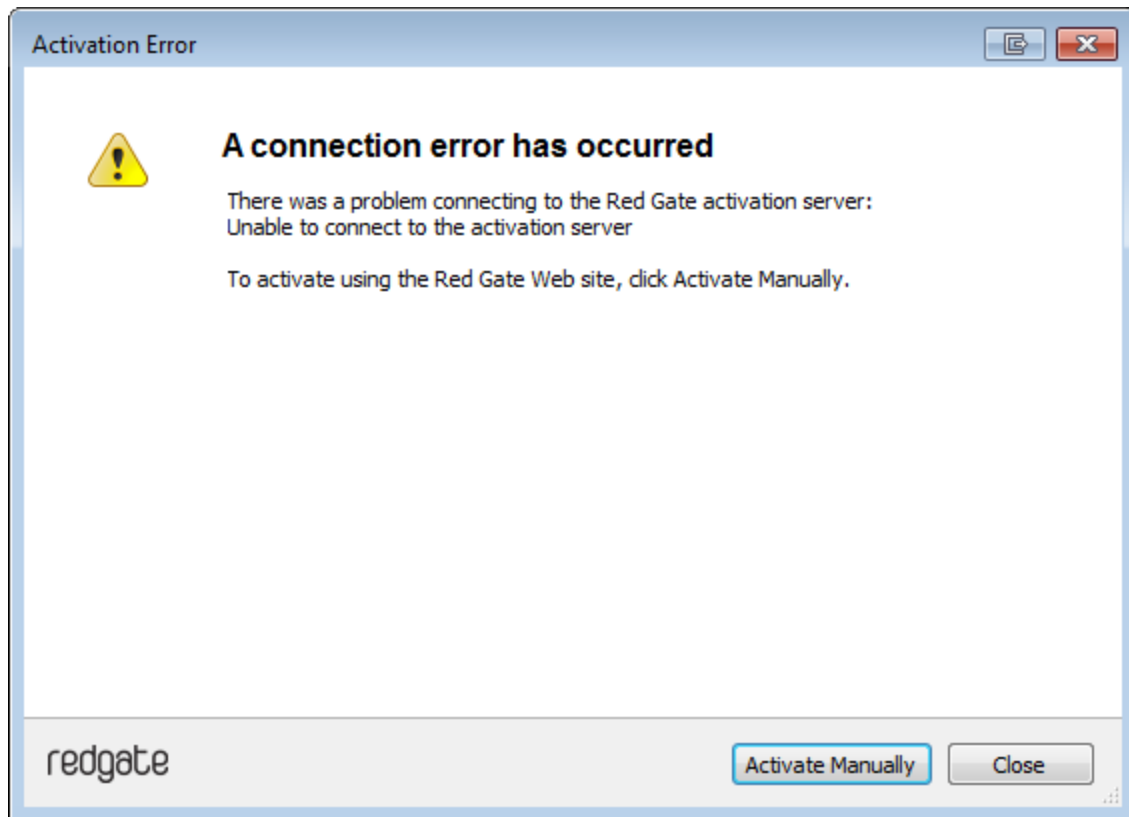
```
sqlcompare /activateSerial:123-456-789012-ABCD
```

The product activation dialog box is displayed. Follow the instructions below.

Manual activation

Manual activation enables you to activate products when your computer does not have an internet connection or your internet connection does not allow SOAP requests. You will need access to another computer that does have an internet connection.

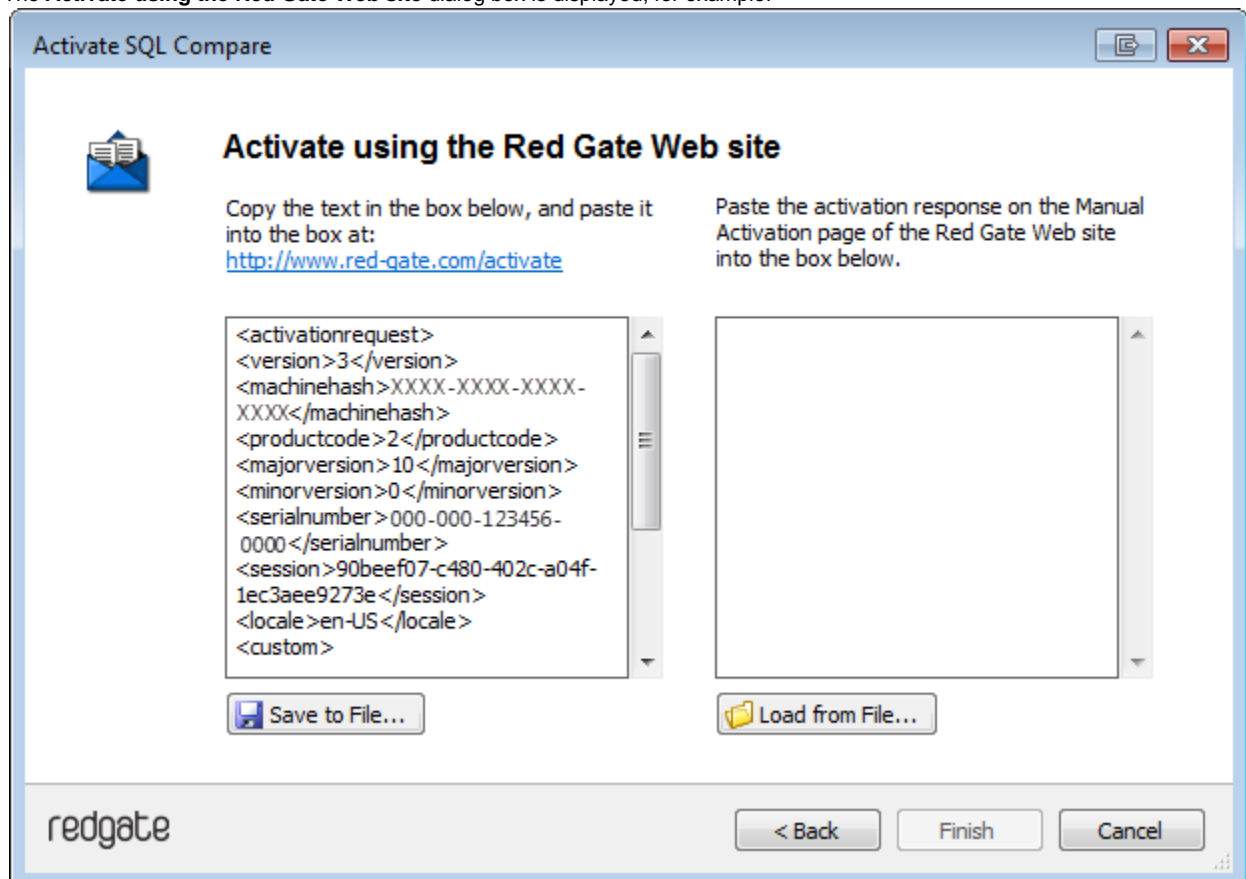
You can use manual activation whenever the **Activation Error** dialog box is displayed and the **Activate Manually** button is available, for example:



To activate manually:

1. On the error dialog box, click **Activate Manually**.

The **Activate using the Red Gate Web site** dialog box is displayed, for example:



2. Copy all of the activation request, and **leave this dialog box open** (if you close the dialog box, you may have to start again). Alternatively you can save the activation request, for example to a location on your network or to a USB device.
3. On a computer that has an Internet connection, go to the **Manual Activation** page at <http://www.red-gate.com/activate> and paste the activation request into the box under **Step 1**.

Account ▾ Quotes Shopping Cart

redgate
ingeniously simple tools

Home Products Store Community Support Our Company

I'm looking for... 🔍

Manual Activation

Use the activation request from the licensing program to generate an activation response so that you can activate products on your computer.

Step 1

Paste the activation request into the box below. Make sure you paste all of the text.

```
<activationrequest>
<version>3</version>
<machinehash>XXXX-XXXX-XXXX-XXXX</machinehash>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<serialnumber>000-000-123456-0000</serialnumber>
<session>90beef07-c480-402c-a04f-1ec3aee9273e</session>
<locale>en-US</locale>
<custom>
```

Get Activation Response

Step 2

Copy the contents of this box into your product activation dialog box.


Save to File...

Got a question?

0800 169 7433
shop@red-gate.com

4. Click **Get Activation Response**.
5. When the activation response is displayed under **Step 2**, copy all of it. Alternatively you can save the activation response to a .txt file.
6. On the computer where the licensing and activation program is running, paste the activation response or if you saved it, load it from the file.


Activate SQL Compare

 **Activate using the Red Gate Web site**


Copy the text in the box below, and paste it into the box at:
<http://www.red-gate.com/activate>

Paste the activation response on the Manual Activation page of the Red Gate Web site into the box below.

```
<activationrequest>
<version>3</version>
<machinehash>XXXX-XXXX-XXXX-XXXX</machinehash>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<serialnumber>000-000-123456-0000</serialnumber>
<session>90beef07-c480-402c-a04f-1ec3aee9273e</session>
<locale>en-US</locale>
<custom>
```

 Save to File...

```
<activationresponse>
<data>
<machinehash>XXXX-XXXX-XXXX-XXXX</machinehash>
<version>3</version>
<productcode>2</productcode>
<majorversion>10</majorversion>
<minorversion>0</minorversion>
<edition>professional</edition>
<userspurchased>1</userspurchased>
<serialnumber>000-000-123456-0000</serialnumber>
```

 Load from File...

redgate

< Back Finish Cancel

7. Click **Finish**.
The **Activation successful** page is displayed.
8. Click **Close**.
You can now continue to use your product.

Deactivating

This page applies to several Redgate products, so the screenshots below may not match your product.



Download deactivation tool

You can use the deactivation tool to deactivate a serial number so you can reuse it on another computer. You can also use it to deactivate serial numbers for products you've uninstalled.

When you deactivate a serial number for a bundle of products, all the products in the bundle are deactivated. For information about what products are in your bundle, see [Bundle history](#).

To deactivate a serial number, your computer must have an internet connection. If you can't deactivate a serial number, you can [request additional activations](#) for that serial number. You may need to do this if:

- your computer doesn't have an internet connection
- your network uses a proxy server that interrupts contact between the product and the Redgate activation server
- your serial numbers aren't displayed in the deactivation tool (eg if the product installation is corrupted)

Deactivating using the command line

Open a command prompt, navigate to the folder where your product executable file is located and run a command with the following syntax:

```
<productEXE> /deactivateSerial
```

For example:

```
sqlcompare /deactivateSerial
```

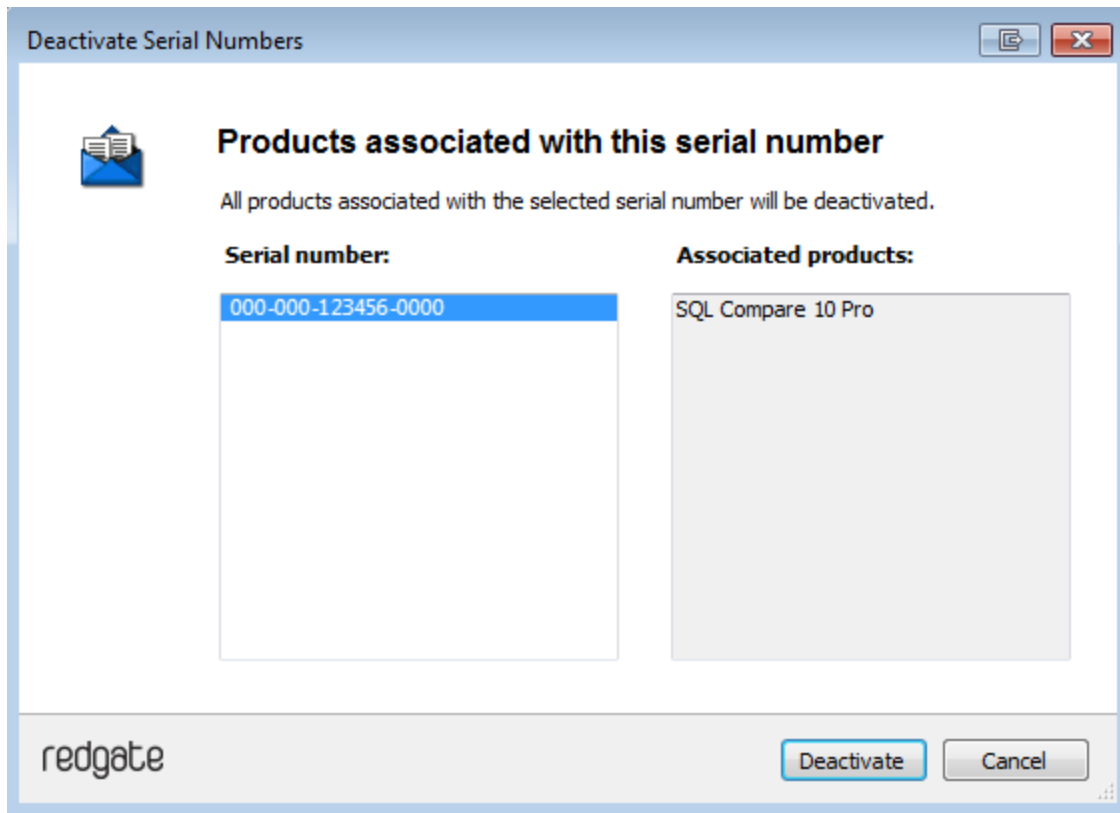
The **Deactivate Serial Numbers** dialog box is displayed. Follow the instructions below.

Deactivating using the GUI

To deactivate your products:

1. Start the deactivation tool. To do this, either [download](#) the tool and run the executable file, or on the **Help** menu of the product, click **Deactivate Serial Number**.

The **Deactivate Serial Numbers** dialog box is displayed. For example:



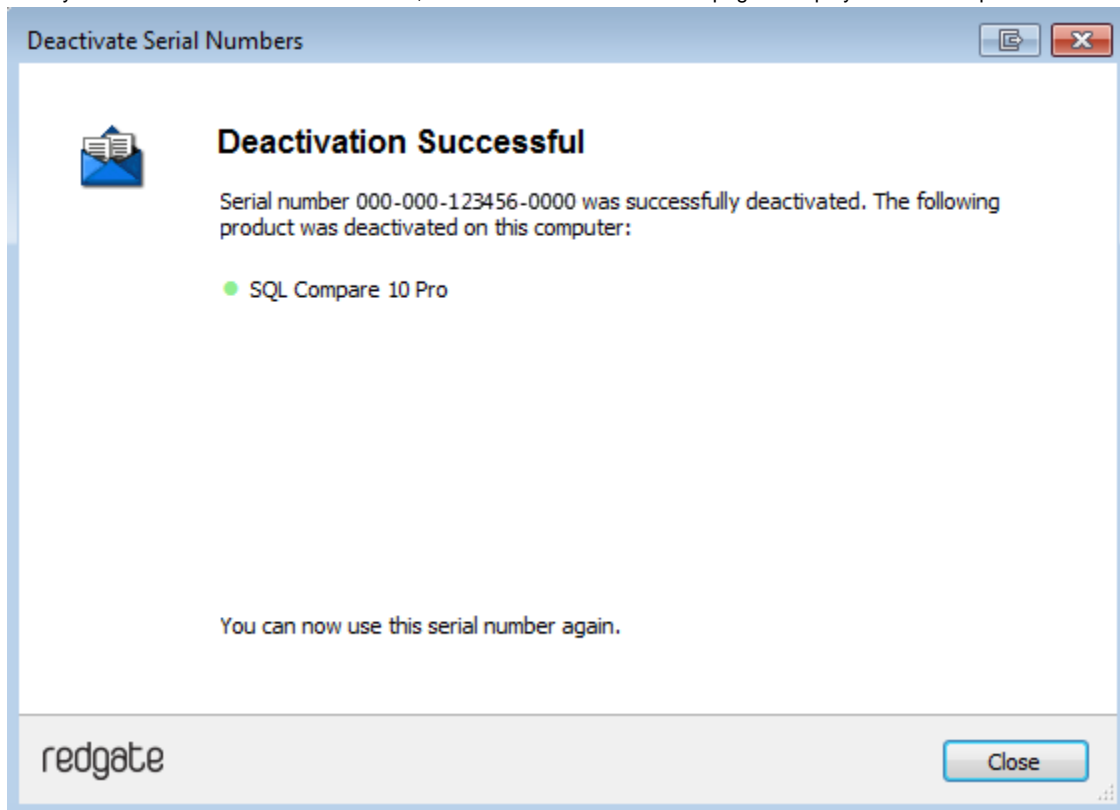
If you're running the executable file, the dialog box displays all the serial numbers for Red Gate products that have been activated on your computer.

If the serial number is for a bundle, all the products in the bundle are displayed under **Associated products**.

2. Select the serial number you want to deactivate and click **Deactivate**.

Your deactivation request is sent to the Red Gate activation server.

3. When your deactivation has been confirmed, the **Deactivation successful** page is displayed. For example:



If there's a problem with your deactivation request, an error dialog box is displayed. For information about deactivation errors and how to resolve them, see [Troubleshooting licensing and activation errors](#).

4. Click **Close**. You can now use this serial number on a different computer.

Troubleshooting licensing and activation

This page provides information about errors you may encounter when you activate Redgate products:

- The number of activations for this serial number has been exceeded
- This serial number has been disabled
- This serial number was for a trial extension
- This serial number is not registered with the activation server
- This serial number is not for <product name>
- This serial number is not for this version
- The activation request is in the wrong format
- The activation request contains an invalid machine hash
- The activation request contains an invalid session
- The activation request contains an invalid serial number
- The activation request contains an invalid product code or version number
- There's a problem deactivating your serial number
- This serial number is not activated on this computer
- Products not activated on this computer

The number of activations for this serial number has been exceeded

This error message is displayed when a serial number is activated on more computers than the number of licenses that were purchased for that serial number.

When you purchase products from Redgate, we send you an invoice that includes your serial numbers. The serial numbers enable you to activate the software a number of times, depending on how many licenses you purchased and the terms in the [license agreement](#). When this limit is reached, you will see this error message.

To fix the problem, you can:

- [deactivate](#) the product on another computer to free up a license
- [purchase](#) more licenses
- [request additional activations](#) for your serial number

This serial number has been disabled

This error message is displayed when you try to activate a product using a serial number that Redgate has disabled.

When you upgrade a product, your existing serial numbers will be disabled and we will issue new ones with your invoice. If you cannot find your new serial numbers, you can review them at <http://www.red-gate.com/myserialnumbers>

Redgate will also disable serial numbers for non-payment of invoices or breach of the terms in the [license agreement](#). If you think we have disabled your serial numbers in error, email licensing@red-gate.com

This serial number was for a trial extension

This error message is displayed when you have requested a trial extension and you try to reuse the serial number that was provided for the trial extension; trial extensions can be used one time only.

To continue using the product, you need to [purchase](#) it.

This serial number is not registered with the activation server

This error message is displayed when the serial number you entered does not exist on the Redgate activation server.

To find out your serial numbers, check your invoice or go to <http://www.red-gate.com/myserialnumbers>

This serial number is not for <product name>

This error message is displayed when the serial number you entered is not for the product you are trying to activate.

To find out your serial numbers, check your invoice or go to <http://www.red-gate.com/myserialnumbers>

This serial number is not for this version

This error message is displayed when the serial number you entered is for a different version of the product you are trying to activate.

If the serial number is for an older version of the product, and you don't have that version installed on your computer, you can download it from the [Release notes and other versions page](#).

If you want to upgrade to the latest version of the product, go to the [Upgrade center](#) to get a quote or purchase an upgrade, or email sales@red-gate.com.

The activation request is in the wrong format

This error message is displayed:

- if your internet connection does not allow SOAP requests.
Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed.
- if you are activating by email and there is a problem with the format of the activation request.
Check that you copied and pasted all of the activation request.
Alternatively, try using manual activation. Go to <http://www.red-gate.com/activate> and paste your activation request under **Step 1**.
- when you are using manual activation and there is a problem with the format of the activation request. If the format is incorrect, for example part of the request is missing, the Redgate activation server cannot process the request.
Check that you copied and pasted all of the activation request.

For more information about activating manually, see [Manual activation](#).

The activation request contains an invalid machine hash

This error message is displayed:

- if your internet connection does not allow SOAP requests.
Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see [Manual activation](#).
- when you are using manual activation and there is a problem with the format of the *machinehash* element in the activation request. The *machinehash* is a checksum of attributes from your computer. We use the *machinehash* to identify computers on which our products have been activated. If the format of the *machinehash* element is incorrect, the Redgate activation server cannot process the request.
Check that you copied and pasted the activation request correctly.

The activation request contains an invalid session

This error message is displayed:

- if your internet connection does not allow SOAP requests.
Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see [Manual activation](#).
- when you are using manual activation and there is a problem with the format of the activation request. If the format of the *session* element is incorrect, the Redgate activation server cannot process the request.
Check that you copied and pasted the activation request correctly.

The activation request contains an invalid serial number

This error message is displayed:

- if your internet connection does not allow SOAP requests.
Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see [Manual activation](#).
- when you are using manual activation and there is a problem with the format of the activation request. If the format of the serial number is incorrect, the Redgate activation server cannot process the request.
Check that you copied and pasted the activation request correctly.

The activation request contains an invalid product code or version number

This error message is displayed:

- if your internet connection does not allow SOAP requests.
Try using manual activation; on the error dialog box, click **Activate Manually**, and then follow the instructions that are displayed. For more information, see [Manual activation](#).
- when you are using manual activation and there is a problem with the format of the activation request. If the product code or version numbers are incorrect, the Redgate activation server cannot process the request.
Check that you copied and pasted the activation request correctly.

There's a problem deactivating your serial number

This error message is displayed if your computer is not connected to the internet or your internet connection does not allow SOAP requests. You cannot deactivate a serial number if your computer does not have an internet connection.

Try deactivating again later. If the problem persists, contact your system administrator.

If you require more activations because you cannot deactivate your serial number, you can request them on the [Request Extra Activations](#) page.

This serial number is not activated on this computer

This error message is displayed when you try to deactivate a serial number that has not been activated on your computer.

If you think the product installation on your computer is corrupt, you can try re-activating the product, and then deactivating the product again.

If you require more activations because you cannot deactivate your serial number, you can request them on the [Request Extra Activations](#) page.

Products not activated on this computer

This error message is displayed when you try to deactivate a serial number for a bundle of Redgate products and those products were not activated on your computer.

If you think the product installation on your computer is corrupt, you can try re-activating the product, and then deactivating the product again.

If you require more activations because you cannot deactivate your serial number, you can request them on the [Request Extra Activations](#) page.

Installing

Most Redgate products are available as part of a bundle. You can select which individual products to install when you run the installer.

When you install a non-free product, you have 14 days to evaluate the product. For the DLM Automation Suite, DLM Automation Suite for Oracle, SQL Source Control, Schema Compare for Oracle, Data Compare for Oracle, and Source Control for Oracle, you have 28 days. For more information, see [Licensing](#).

To install a Redgate product:

1. Download the product from the [website](#).
2. Run the installer and follow the instructions.

The product is listed on the **Start** menu under **Red Gate**.

Updating

Minor releases are free for all users. For example, if you have a license for version 7.0 of a product, you can upgrade to version 7.1 at no cost. When you download and install a minor release, the product is licensed with your existing serial number automatically.

Major releases are free for users with a current Support and Upgrades contract. For example, if you have a license for version 7 of a product, you can upgrade to version 8 at no cost. When you download and install a major release, the product is licensed with your existing serial number automatically.

If you don't have a current Support and Upgrades contract, installing a major release will start a free 14-day trial. You'll need to buy a new license and activate the product with your new serial number.

To check whether you have a current Support and Upgrades contract or see the cost of upgrading to the latest major version of a product:

- visit the [Upgrade Center](#)
- email sales@red-gate.com
- call:
 - 1 866 733 4283 (toll free USA and Canada)
 - 0800 169 7433 (UK freephone)
 - +44 (0)870 160 0037 (rest of world)

To check the latest version of a product, see [Current versions](#).

How to upgrade

You can download the latest version of a product using [Check for Updates](#), the [Upgrade Center](#), or the [Redgate website](#).

- If you download the latest version from the Upgrade Center or our website, you need to run the installer to upgrade the product.

Some Redgate products are available as part of bundle. You can select which products you want to upgrade when you run the installer.

- If you use Check for Updates, the installer runs automatically.

You can install the latest *major* version of any product (other than SQL Backup Pro) on the same machine as the previous version. For example, you can run version 9 and version 10 in parallel. However, installing a *minor* release will upgrade the existing installation.

To revert to an earlier version, uninstall the later version, then download and install the version you want from the Release notes and other versions page. You can use a serial number for a later version to activate an earlier version.

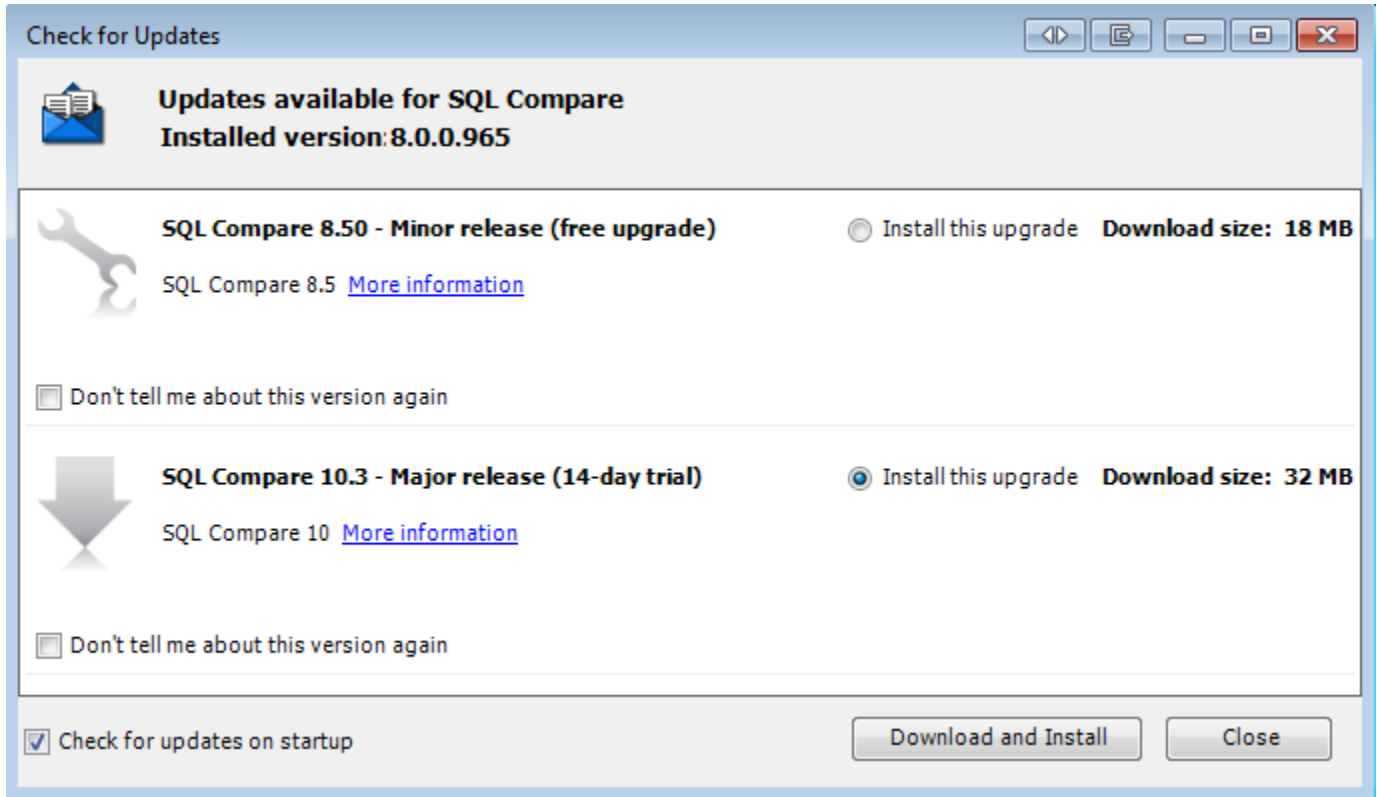
Using Check for Updates

This page applies to several Redgate products, so the screenshots below may not match your product.

The Check for Updates service checks whether a more recent version of the product is available to download. To use the service, your computer must have a connection to the internet. If your internet connection uses a proxy server, make sure your web browser connection settings are configured correctly.

The Check for Updates service doesn't work with automatic configuration scripts.

To check for updates for a Redgate product, on the **Help** menu, click **Check for Updates**. Any available updates are listed:



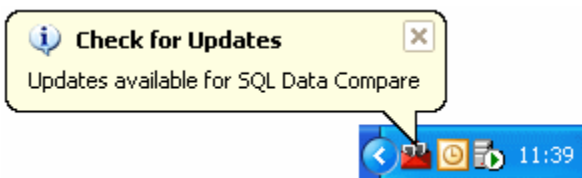
To view the full release details in your default web browser, click **More information**.

To get the update, click **Download and Install**. If you have a choice of updates, choose by selecting **Install this upgrade**, and then click **Download and Install**.

The installer will ask you to close the program. If you're upgrading an add-in, you'll also be asked to close the host program (SQL Server Management Studio, Visual Studio or Query Analyzer).

About the Check for Updates service

When you start the application, the Check for Updates service informs you automatically when there are updates available:



If you don't want to receive these notifications for the product, clear the **Check for updates on startup** check box.

If you don't want the Check for Updates service to inform you about a particular update again, select the **Don't tell me about this version again** check box. The Check for Updates service will still inform you of new updates when they become available.

Troubleshooting Check for Updates errors

For details about how to use the Check for Updates service, see [Using Check for Updates](#).

Error: There is a problem saving the download file to your computer

This error message is displayed if:

You don't have enough disk space

The Check for Updates service downloads the updates to the location defined by the *RGTEMP* environment variable, or the *TMP* variable if the *RGTEMP* variable doesn't exist.

If you don't have enough disk space, you can change the environment variable to a location with more space.

Changing the *RGTEMP* or the *TMP* variables will affect other programs that use those variables. The *RGTEMP* variable affects only Redgate programs. For information about environment variables, see your Windows documentation.

There's a problem with permissions on your computer

The Check for Updates service downloads the updates to the location defined by the *RGTEMP* environment variable, or the *TMP* variable if the *RGTEMP* variable does not exist. If your user account doesn't have permissions to write to the location specified by these environment variables, contact your system administrator.

There's a problem with the download file on the Redgate web server

Contact [Redgate support](#).

Error: There is a problem with the network connection

This error message is displayed if:

Your internet connection dropped while the Check for Updates service was downloading the updates

Try checking for updates again later.

Proxy authentication failed

Check your user name and password.

Your computer can't connect to the Check for Updates service.

Contact your system administrator. If you're using a proxy server, check it's configured correctly (see Control Panel > Internet Options > Connections).

The Check for Updates service doesn't work with automatic configuration scripts.

There's a problem with the download file on the Redgate web server

Contact [Redgate support](#).

Turning on Frequent Updates

This page covers several Redgate tools. The screenshots below might not match your product.

For some Redgate tools, if you want to get updates more often, you can turn on **Frequent Updates**. We recommend Frequent Updates if you want to:

- get new features, performance improvements and bug fixes as soon as they're available
- give feedback on new features and help improve Redgate tools

Installing an update is always optional.

Compatible tools

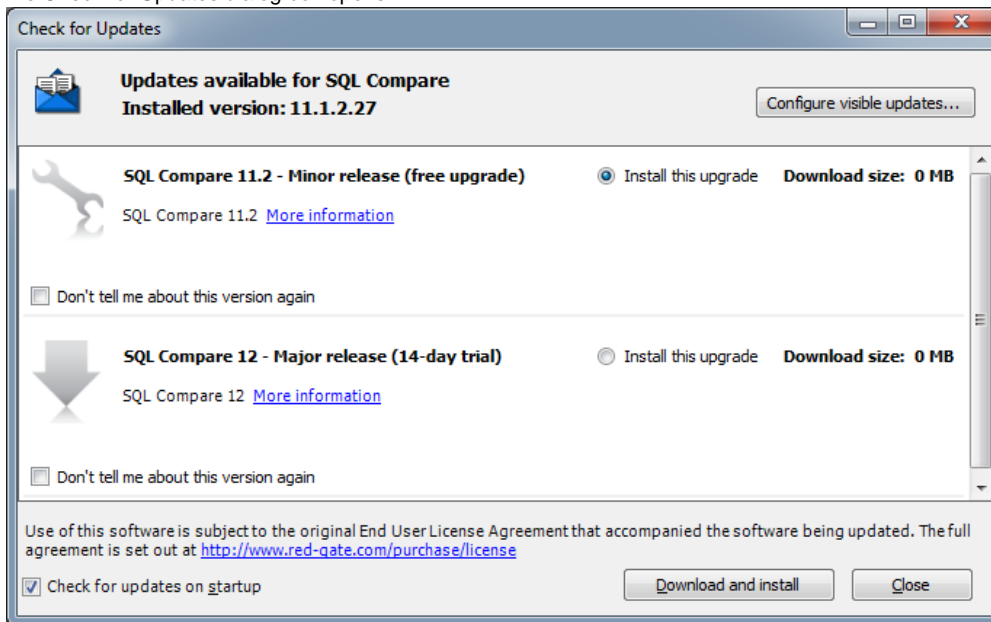
You can get Frequent Updates for:

- SQL Compare
- SQL Data Compare
- SQL Source Control

Turning on Frequent Updates

1. Under the Help menu, select **Check for updates**.

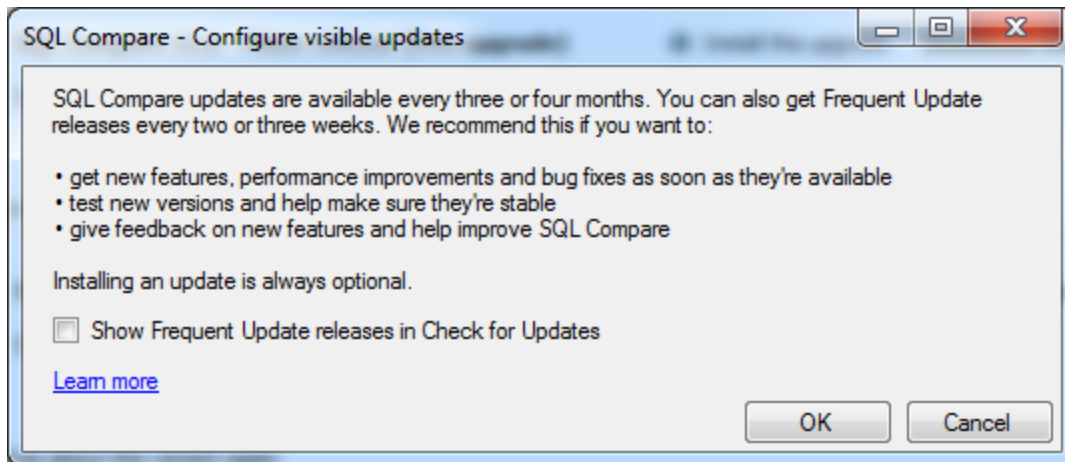
The Check for Updates dialog box opens:



This lists available updates.

2. Click **Configure visible updates**.

The **Configure visible updates** dialog opens:



3. In the dialog box, select the **Show Frequent Update releases in Check for Updates** checkbox and click **OK**.

If there are Frequent Update releases available, they're listed in the Check for Updates dialog box. To install a Frequent Update release, click **Download and install**.

Turning off Frequent Updates

To turn off Frequent Updates, open the **Configure visible updates** dialog (as in step 1 above) and clear the **Show Frequent Update releases in Check for Updates** checkbox. You'll no longer see Frequent Update releases in Check for Updates.

If you've already installed a Frequent Update release, you'll stay on that version until you either:

- install a newer standard update (when one is available)
- uninstall the tool, then download it from the product page and reinstall it
This returns you to the most recent standard release.

Linking to source control

These pages explain how to link your database to your source control system using SQL Source Control.

- [SVN](#)
- [TFS](#)
- [Vault](#)
- [Git, Mercurial, or other systems](#)

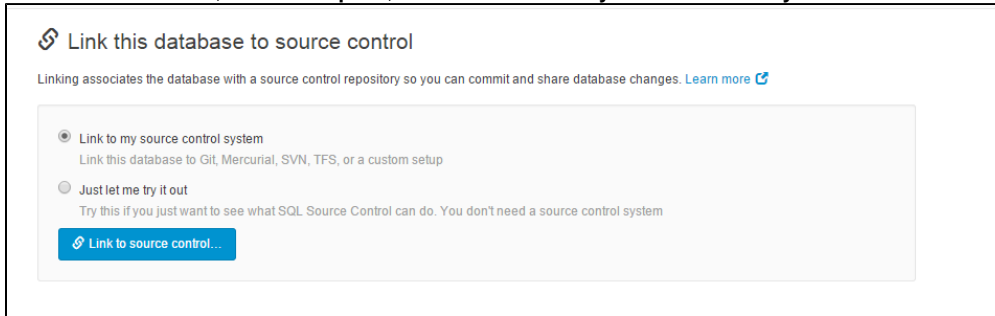
Don't have a source control system?

If you just want to experiment with SQL Source Control without setting up a source control system, try the [evaluation repository](#).

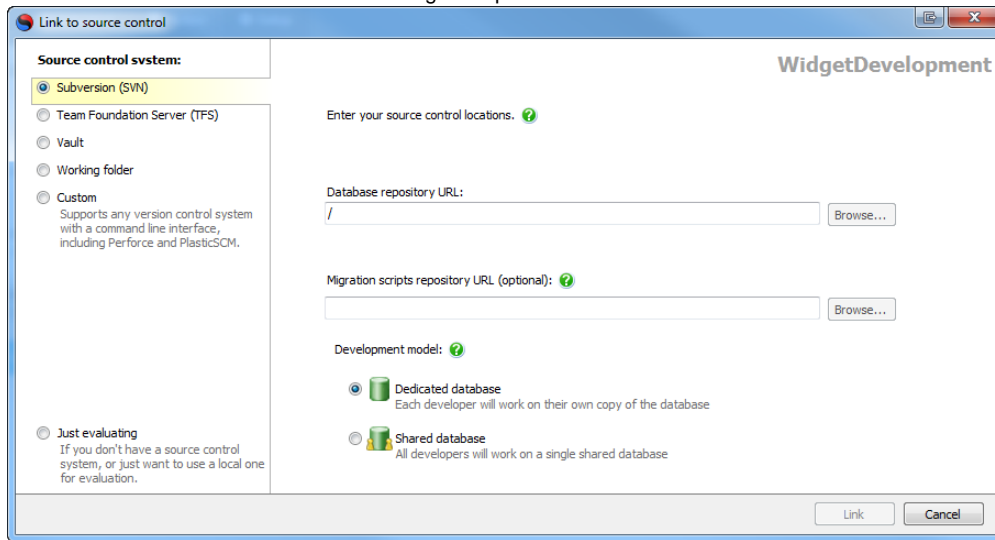
SVN

This page explains how to link your database to Subversion (SVN).

1. In the Object Explorer, select the database you want to link to source control.
2. In SQL Source Control, on the **Setup** tab, make sure **Link to my source control system** is selected and click **Link to source control**:

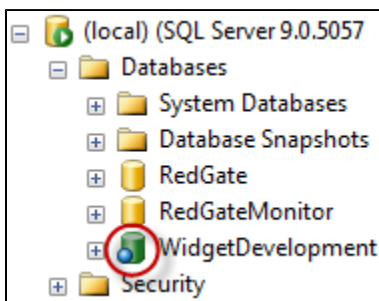


The **Link Database to Source Control** dialog box opens:



3. On the left, select **Subversion (SVN)**.
4. In the **Database repository URL** field, specify a folder in your SVN repository where SQL Source Control will save SQL scripts. For example: `http://Subversion.Example.com/Databases/AdventureWorks`
If you're the first person to link the database to source control, specify an empty folder. If someone has already linked this database to source control, specify the folder they used.
5. If you want to use **migration scripts**, under **Migration scripts repository URL**, specify an existing, empty folder in the repository. The folder can't be in the database folder. For example, you can use `Repository folderMigration scripts`, but not `Repository folderDatabase folderMigration scripts`.
6. If you're linking to a database that will be used by multiple developers, make sure **Shared database** is selected. For more information, see [Database development models](#).
7. Click **Link**.

The database is linked to source control. The database icon in the Object Explorer changes to show that the database is linked:



After you link

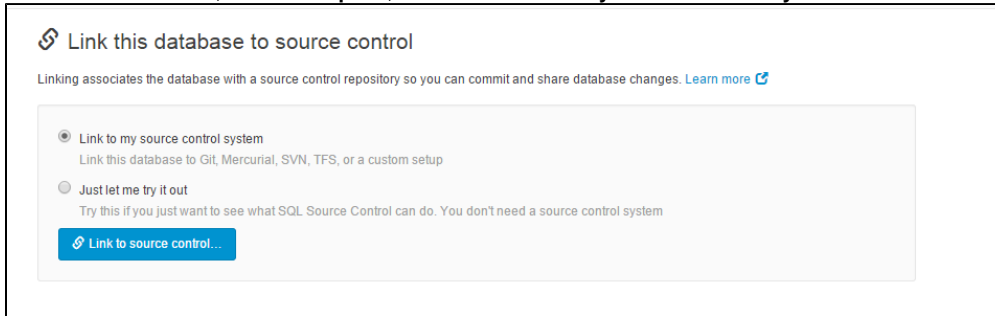
If you're the first person to link this database, add the database objects to source control. To do this, go to the **Commit changes** tab and commit the objects.

If you linked a database that's already in source control, update your database to the latest version. To do this, go to the **Get latest** tab and get the changes.

TFS

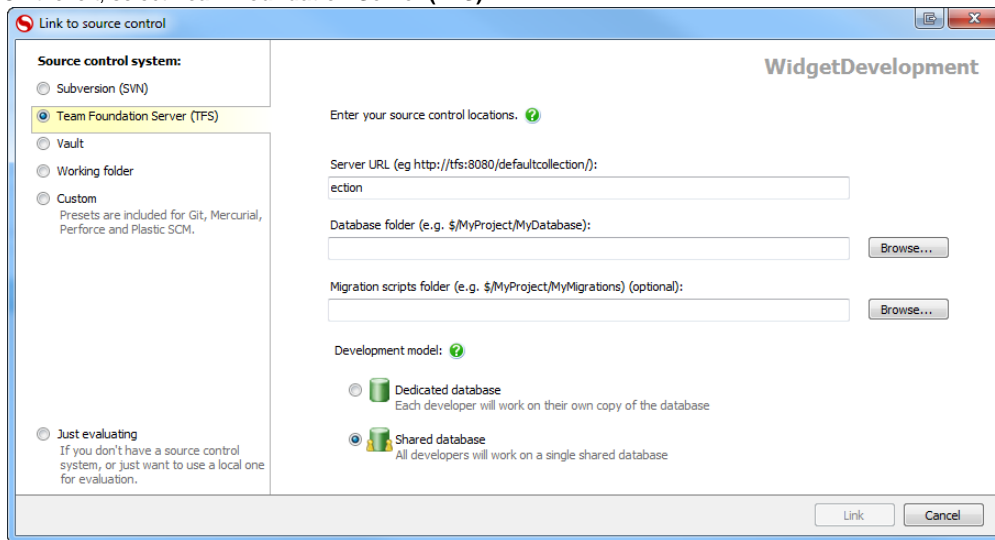
This page explains how to link your database to Team Foundation Server (TFS).

1. In the Object Explorer, select the database you want to link to source control.
2. In SQL Source Control, on the **Setup** tab, make sure **Link to my source control system** is selected and click **Link to source control**:



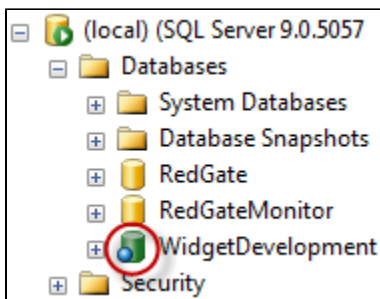
The **Link Database to Source Control** dialog box opens.

3. On the left, select **Team Foundation Server (TFS)**:



4. In the right-hand pane, in the **Server URL** field, specify the URL of your Team Foundation Server. If you use a non-standard port, specify this in the URL (eg `http://myurl:8080/tfs`).
5. In the **Database folder** field, enter the folder in TFS where SQL Source Control will store SQL scripts. If you're the first person to link the database to source control, specify an empty folder. If someone has already linked this database to source control, specify the folder they used.
6. If you want to use **migration scripts**, under **Migration scripts folder**, specify an existing, empty folder in the repository. The folder can't be in the database folder. For example, you can use *Repository folderMigration scripts*, but not *Repository folderDatabase folderMigration scripts*.
7. If you're linking to a database that will be used by multiple developers, make sure **Shared database** is selected. For more information, see [Database development models](#).
8. Click **Link**.

The database is linked to source control. The database icon in the Object Explorer changes to show that the database is linked:



After you link

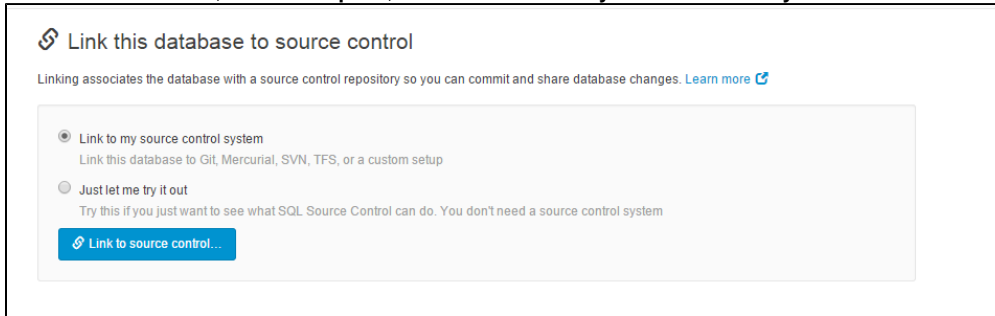
If you're the first person to link this database, add the database objects to source control. To do this, go to the **Commit changes** tab and commit the objects.

If you linked a database that's already in source control, update your database to the latest version. To do this, go to the **Get latest** tab and get the changes.

Vault

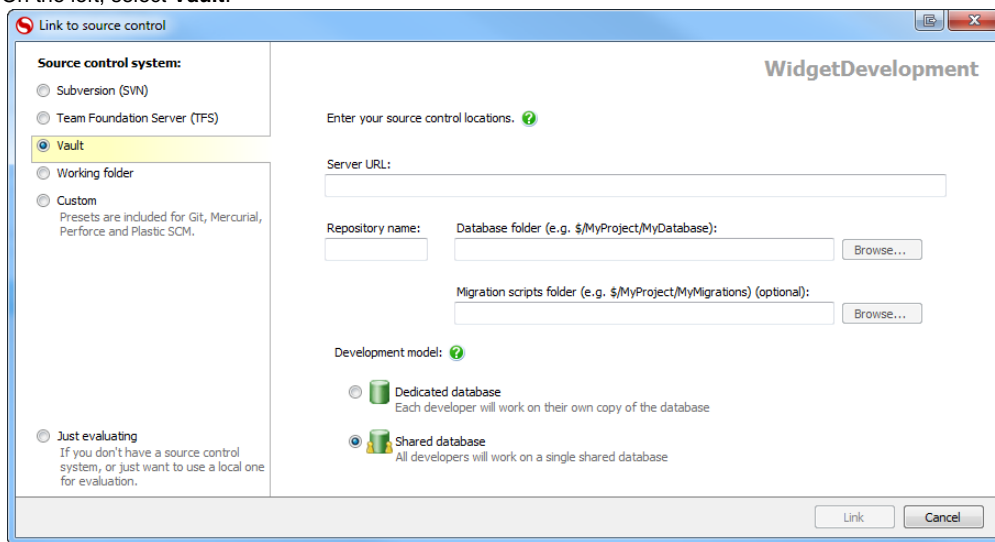
This page explains how to link your database to Vault.

1. In the Object Explorer, select the database you want to link to source control.
2. In SQL Source Control, on the **Setup** tab, make sure **Link to my source control system** is selected and click **Link to source control**:



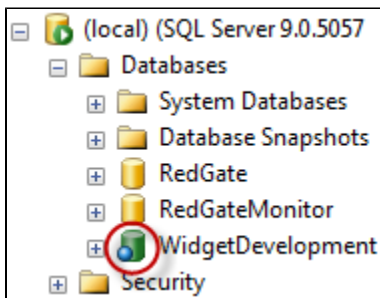
The **Link Database to Source Control** dialog box opens.

3. On the left, select **Vault**.



4. In the **Server URL** field, enter the location of your Vault server (hostname or IP address); for example, *my-vault-server.your-domain*.
5. In the **Repository name** field, enter the name of your repository.
6. In the **Database folder** field, enter the folder in the Vault repository where script files for database will be stored (eg *\$/my-projects/my-database-project*).
If you're the first person to link the database to source control, specify an empty folder. If someone has already linked this database to source control, specify the folder they used.
7. If you want to use migration scripts, under **Migration scripts repository URL**, specify an existing, empty folder in the repository.
The folder can't be in the database folder. For example, you can use *Repository folder\Migration scripts*, but not *Repository folder\Database folder\Migration scripts*.
8. If you're linking to a database that will be used by multiple developers, make sure **Shared database** is selected. For more information, see [Database development models](#).
9. Click **Link**.

The database is linked to source control. The database icon in the Object Explorer changes to show that the database is linked:



After you link

If you're the first person to link this database, add the database objects to source control. To do this, go to the **Commit changes** tab and commit the objects.

If you linked a database that's already in source control, update your database to the latest version. To do this, go to the **Get latest** tab and get the changes.

Git, Mercurial, or other systems

SQL Source Control only has full integration with SVN, TFS and Vault.

If you want to link to another source control system, including Git or Mercurial, you can do this by:

- linking to a **working folder**, or
- linking to a **custom setup** using config files

Linking to a working folder

When you link to a working folder, SQL Source Control doesn't automate any source control operations (eg add, push, edit etc). Instead:

- When you make changes to the database, the changes are listed in the Commit tab. When you click **Save changes**, the changes are scripted as SQL files to the working folder. You can then commit the script files manually with your source control client.
- When the SQL files in your working folder are changed (eg by pulling with your source control system), the changes are listed in the Get latest tab. When you click **Apply changes**, the changes are applied to your database.

For instructions for how to link to Git, Mercurial or other source control system using a working folder, see [Linking to a working folder](#).

If you want to do atomic commits of your database schema together with your application code, you can do this by linking to a working folder. For more information, see [Example - source-controlling database schema and application code together using a working folder](#).

Linking to a custom setup

When you link to a custom setup, you specify a config file to automate source control operations. Preset config files are provided for:

- Git
- Mercurial
- Perforce
- Plastic SCM

For instructions for how to link to Git, Mercurial or other source control system with a custom setup, see [Linking to a custom setup](#).

For more information about customizing config files, including how to create your own, see [Working with config files](#).

Linking to a working folder

A working folder contains SQL script files that represent your database. The working folder can be source-controlled with your source control system in the same way you source-control other files.

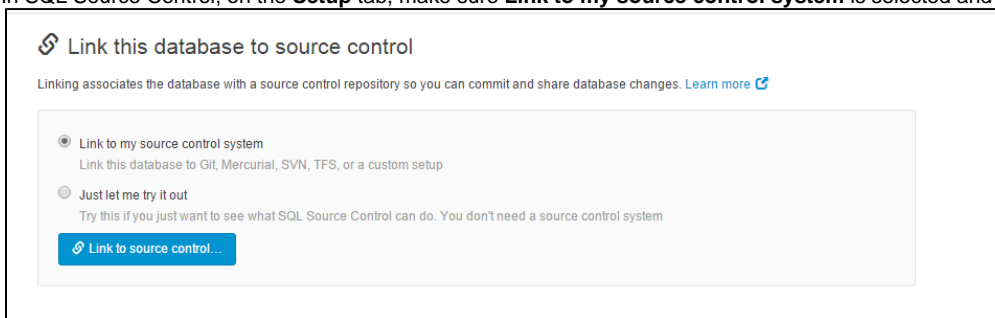
When you link to a working folder, SQL Source Control doesn't automate any source control operations (eg add, push, edit etc). Instead:

- When you make changes to the database, the changes are listed in the Commit tab. When you click **Save changes**, the changes are scripted as SQL files to the working folder. You can then commit the script files manually with your source control client.
- When the SQL files in your working folder are changed (eg by pulling with your source control system), the changes are listed in the Get latest tab. When you click **Apply changes**, the changes are applied to your database.

If you want to do atomic commits of your database schema together with your application code, you can do this by linking to a working folder. For more information, see [Example - source-controlling database schema and application code together using a working folder](#).

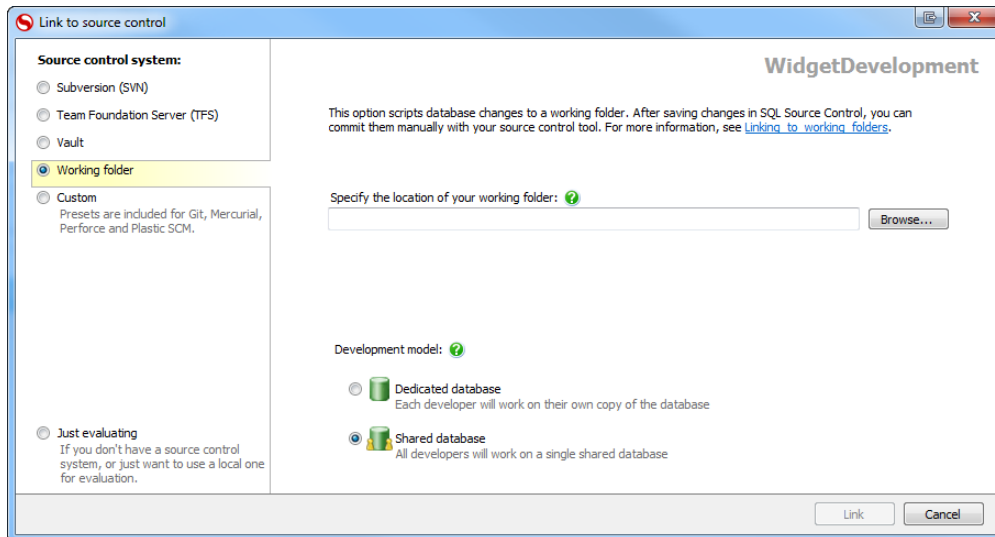
To link to a working folder

1. Create an empty folder in your source control repository. This will be the working folder you link to.
2. In the Object Explorer, select the database you want to link to source control.
3. In SQL Source Control, on the **Setup** tab, make sure **Link to my source control system** is selected and click **Link to source control**:



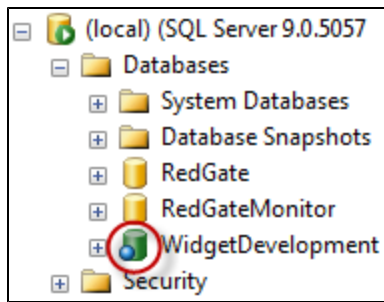
The **Link Database to Source Control** dialog box opens.

4. On the left, select **Working folder**:



5. Enter the location of the working folder.
6. If you're linking to a database that will be used by multiple developers, make sure **Shared database** is selected. For more information, see [Database development models](#).
7. Click **Link**.

The database is linked to source control. The database icon in the Object Explorer changes to show that the database is linked:



After you link

If you're the first person to link this database, add the database objects to source control. To do this, go to the **Commit changes** tab and save the objects to the script folder, then commit the SQL file changes using your source control client.

If you linked a database that's already in source control, update your database to the latest version. To do this, get the latest changes to the script folder using your source control client, then go to the **Get latest** tab and apply the changes to the database.

Linking to a custom setup

When link to a custom setup, you specify a config file to automate source control operations. Preset config files are provided for:

- Git
- Mercurial
- Perforce
- Plastic SCM

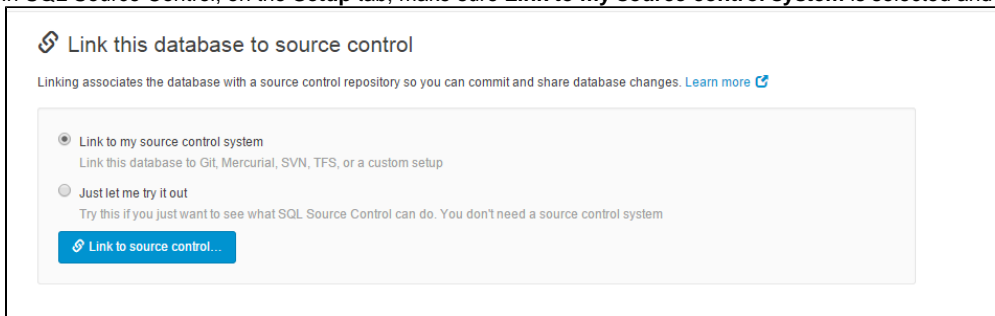
Before you link

The preset config files work well for most setups. However, if you want to use your own config file, create the file before you link.

For more information about how to create your own config file, see [Working with config files](#).

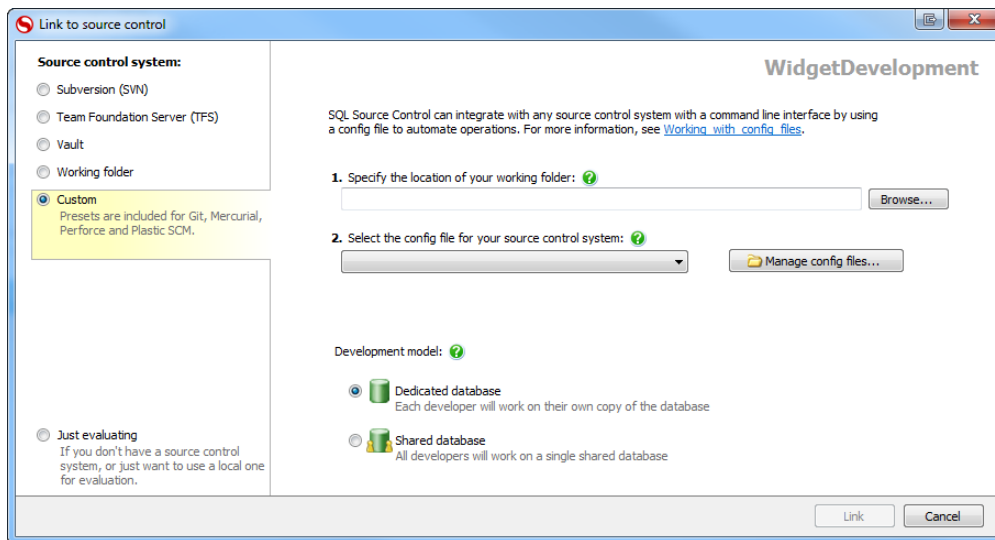
To link to a custom setup

1. Create an empty folder in your source control repository. This will be the working folder you link to.
2. In the Object Explorer, select the database you want to link to source control.
3. In SQL Source Control, on the **Setup** tab, make sure **Link to my source control system** is selected and click **Link to source control**:



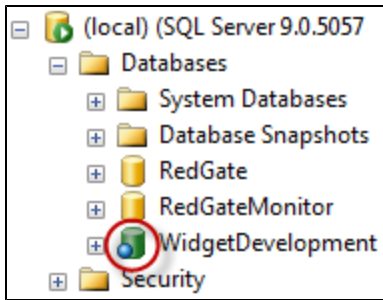
The **Link Database to Source Control** dialog box opens.

4. On the left, select **Custom**:



5. Enter the location of the working folder.
6. Select the config file for your source control system.
7. If you're linking to a database that will be used by multiple developers, make sure **Shared database** is selected. For more information, see [Database development models](#).
8. Click **Link**.

The database is linked to source control. The database icon in the Object Explorer changes to show that the database is linked:



After you link

If you're the first person to link this database, add the database objects to source control. To do this, go to the **Commit changes** tab and commit the objects.

If you linked a database that's already in source control, update your database to the latest version. To do this, go to the **Get latest** tab and get the changes.

Working with config files

SQL Source Control has full integration with SVN, TFS and Vault. If you want to customize how SQL Source Control works with these systems, or if you want to use a different source control system (eg Git or Mercurial), you can use a config file.

A config file is an XML file that contains command line hooks to automate source control operations (add, edit, delete etc). SQL Source Control can use config files to automate source control operations for any source control system with a command line interface. Preset config files are provided for:

- Git
- Mercurial
- Perforce
- Plastic SCM

You can modify the preset files or create your own.

Commands in the config files

The preset config files include the following commands:

Get latest

Updates the local working folder with latest version in source control.

Commit

Commits all changes in the local working folder to source control.

Add

Adds new files to the local working copy. Changes can then be committed to source control using the Commit command.

Edit

Makes the local working copy of the files available for editing. Changes can then be committed to source control using the Commit command.

Delete

Deletes the files from the local working copy. Changes can then be committed to source control using the Commit command.

Revert

Undoes changes if an error occurs during a commit.

Creating a custom config file

To create a custom config file for your source control system:

1. On the **Link to Source Control** dialog box, on the left, select **Custom**.
2. Click **Manage config files**.
The *CommandLineHooks* folder opens.
3. Open *Template.xml* in a text editor and specify commands and verification codes for the source control actions you want to automate. Help is provided inside *Template.xml*.

Make sure you specify the name you want to use for the config file in the `<Name>` tag. For example:

```
<Name>Darcs</Name>
```

To include multiple operations in a single command, separate them with `&&`. For example:

```
accurev add -c -d ($Files) && accurev keep -m && accurev promote -k
```

4. Save your changes as a new XML file in the default config files folder:
`%USERPROFILE%\AppData\Local\Red Gate\SQL Source Control 3\CommandLineHooks`

You can select the config file in the **Custom** tab in the **Link to source control** dialog box.

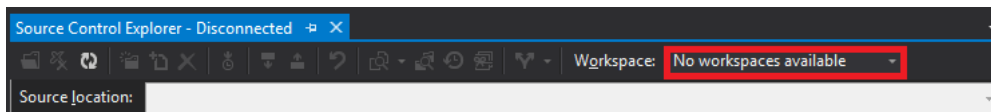
Example - source-controlling database schema and application code together using a working folder

If you want to do atomic commits of your database schema together with your application code, you can do this by [linking to a working folder](#).

In this example, we'll link a working folder to a TFS workspace. The principles can be applied to any other source control system.

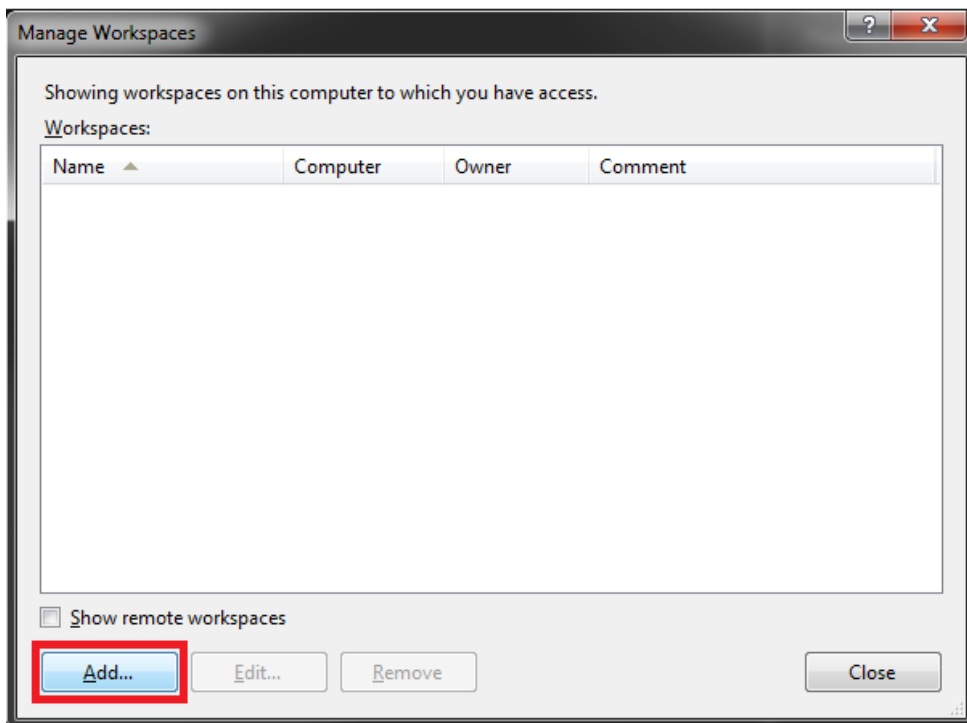
1. Create a new (server) workspace for the existing application.

To do this, in Visual Studio, in the Source Control Explorer tab, select the **Workspace** drop-down menu:



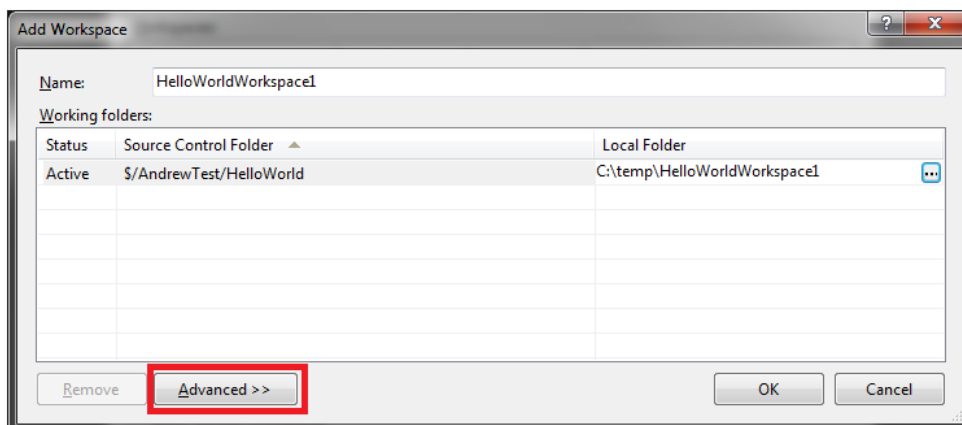
2. The **Manage Workspaces** dialogue box opens.

To add a new workspace, click **Add**:



The **Add Workspace** dialogue box opens.

3. Under **Source Control Folder**, specify the server folder that contains the application.
Under **Local Folder**, specify a local folder.
4. Click **Advanced**.



Under **Location**, choose whether you want to use a local or server workspace:

Add Workspace

Name: HelloWorldWorkspace1

Server: soc-tfs-2012rtm.testnet.red-gate.com\DefaultCollection

Owner: Andrew Farries

Computer: DEV-ANDREWF1

Location: Server

File Time: Local

Permissions: Private workspace

A private workspace can be used only by its owner.

Comment:

Working folders:

| Status | Source Control Folder | Local Folder |
|--------|--|------------------------------|
| Active | \$/AndrewTest/HelloWorld | C:\temp\HelloWorldWorkspace1 |
| | Click here to enter a new working folder | |
| | | |
| | | |
| | | |

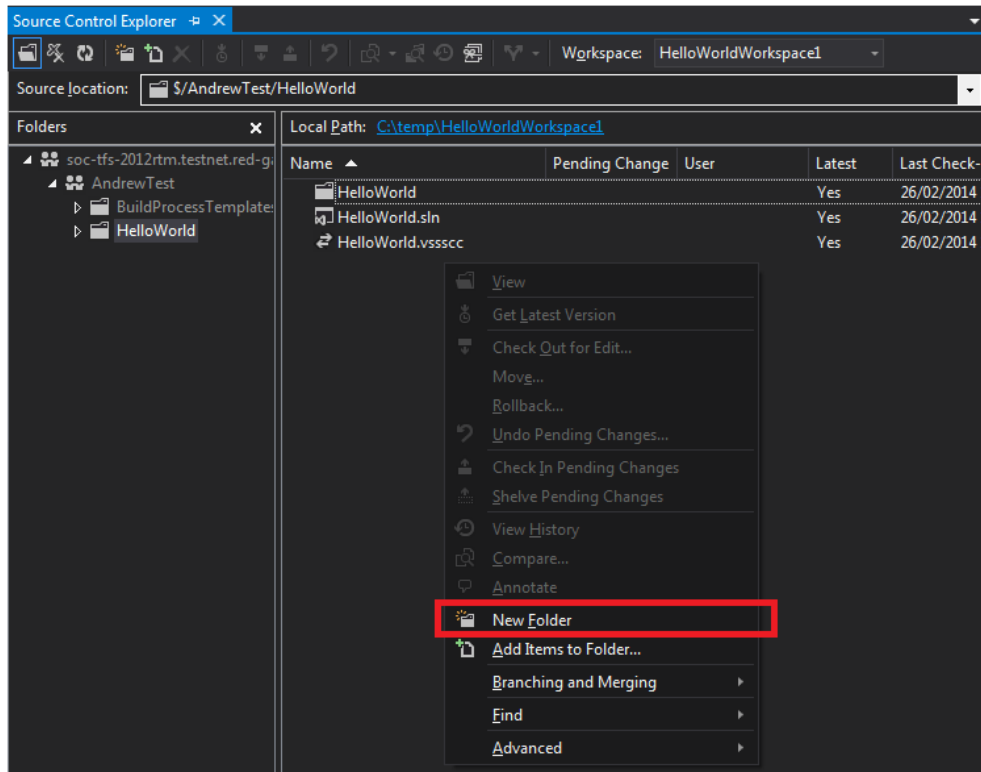
Remove << Advanced OK Cancel

TFS 2012 introduces a distinction between local and server workspaces. In earlier versions of TFS, all workspaces are server workspaces.

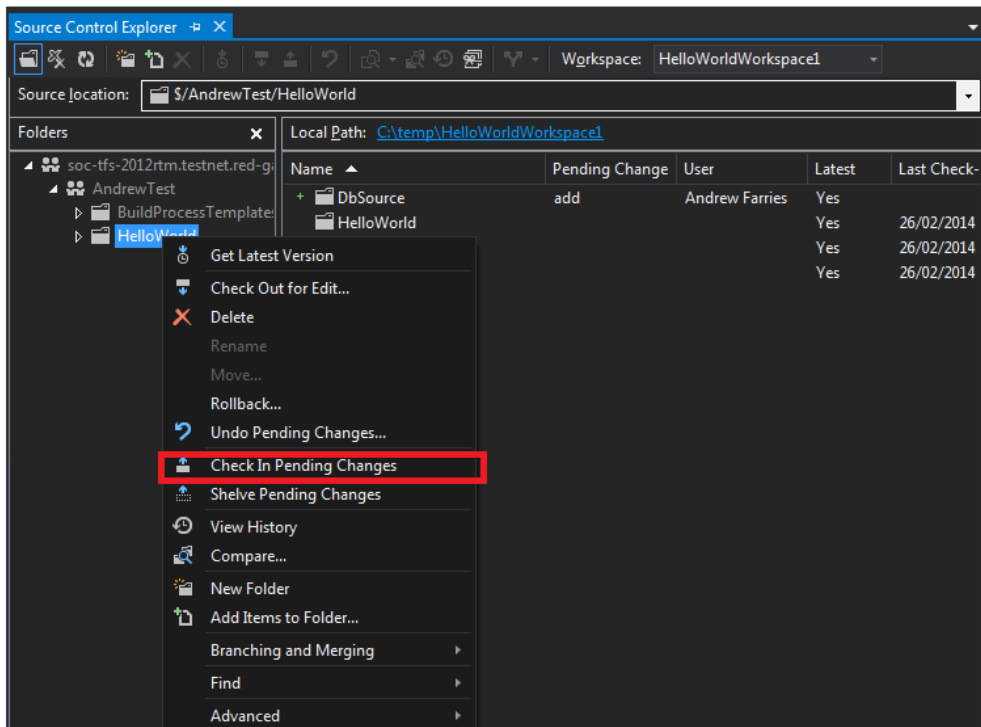
However, local workspaces support more operations offline without needing to communicate with the TFS server. Additionally, when you add new databases objects in SSMS, you need to manually add the new SQL files to the repository (see step 11 below). If you use a local workspace, Team Explorer will automatically detect and suggest new files for addition to the repository.

For more information, see [Server workspaces vs. local workspaces](#) on MSDN Blogs.

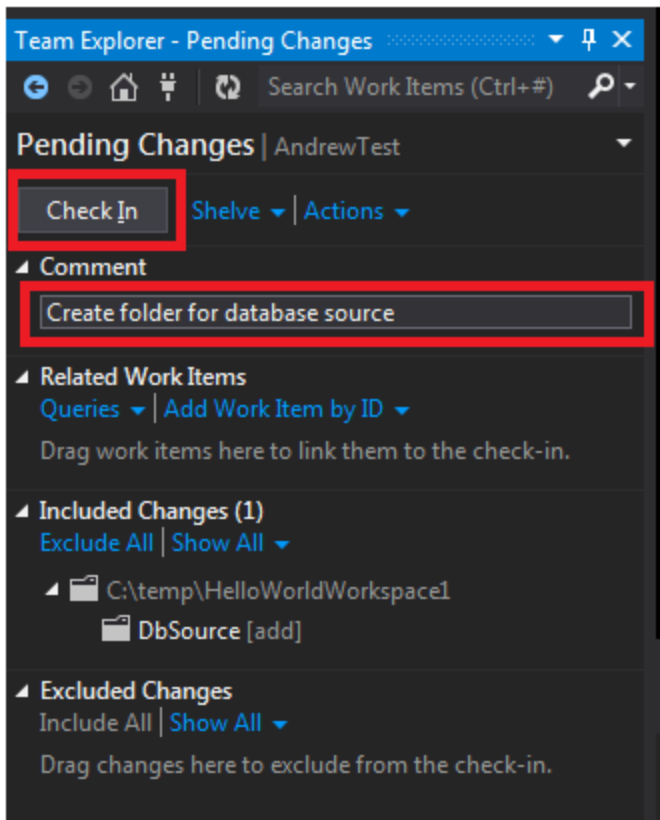
5. In Source Control Explorer, create an empty folder to contain the database source code:



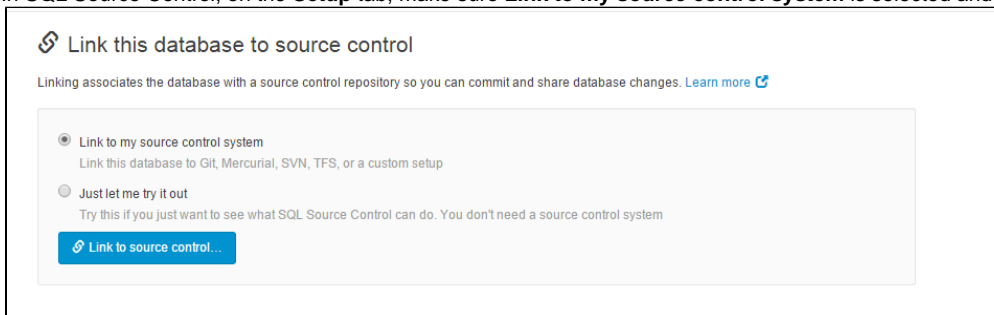
6. To check in the empty folder, right-click on the folder and select **Check In Pending Changes**. In this example, the folder is called named DbSource.



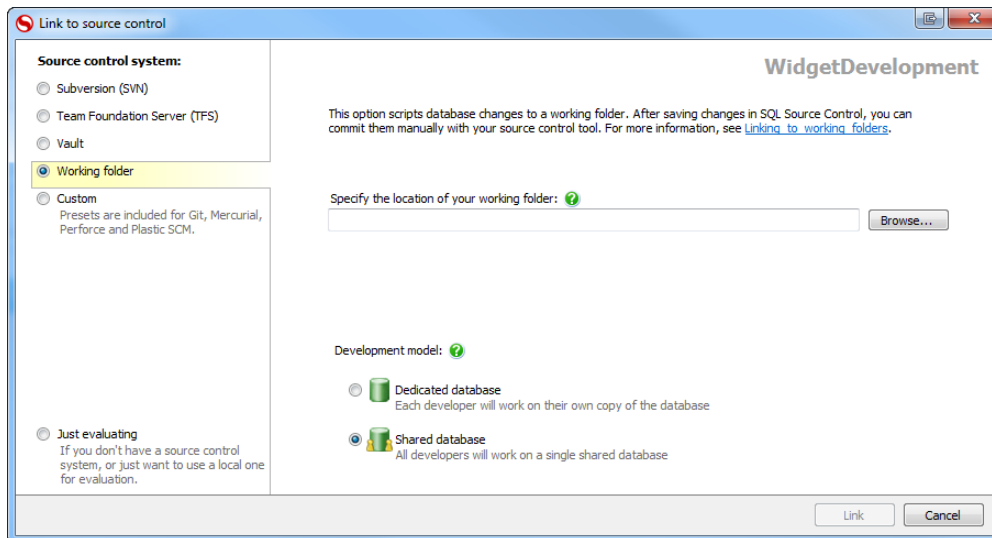
In the **Pending Changes** tab, write a check-in comment and click **Check In**:



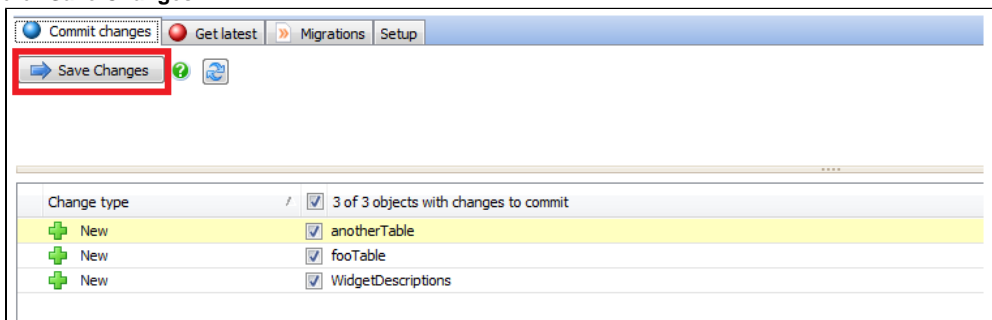
7. In SSMS, select the database in the Object Explorer.
8. In SQL Source Control, on the **Setup** tab, make sure **Link to my source control system** is selected and click **Link to source control**:



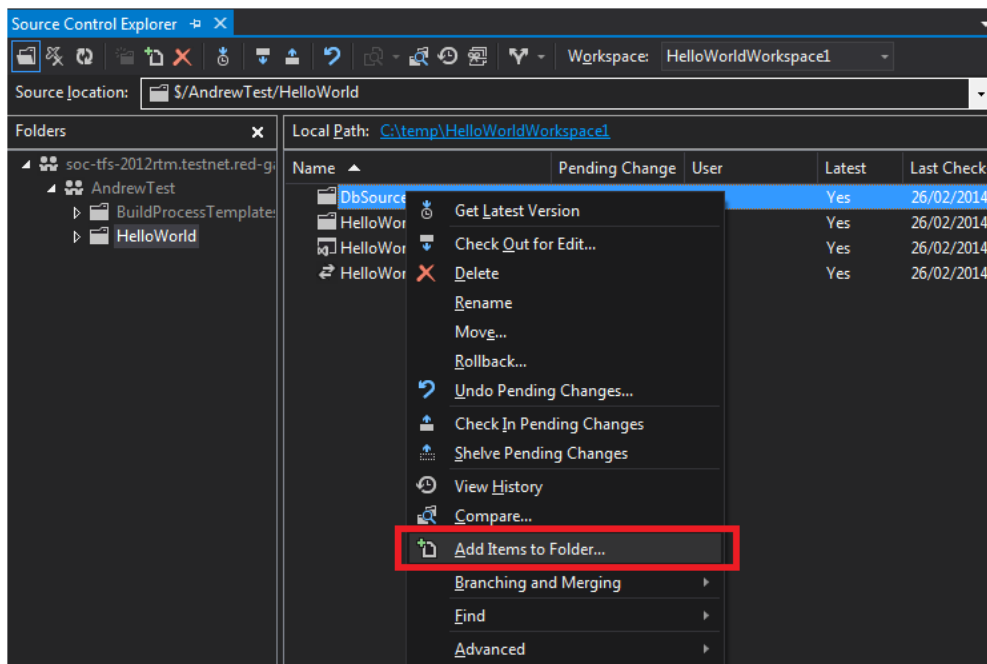
The **Link to source control** dialog box opens.



9. In the left, under Source control system, select **Working folder**.
10. Specify the location of the empty folder and click **Link**.
The database is linked to the working folder.
11. In the **Commit changes** tab, do an initial commit of all the objects in the database. To do this, make sure all the objects are selected and click **Save Changes**.



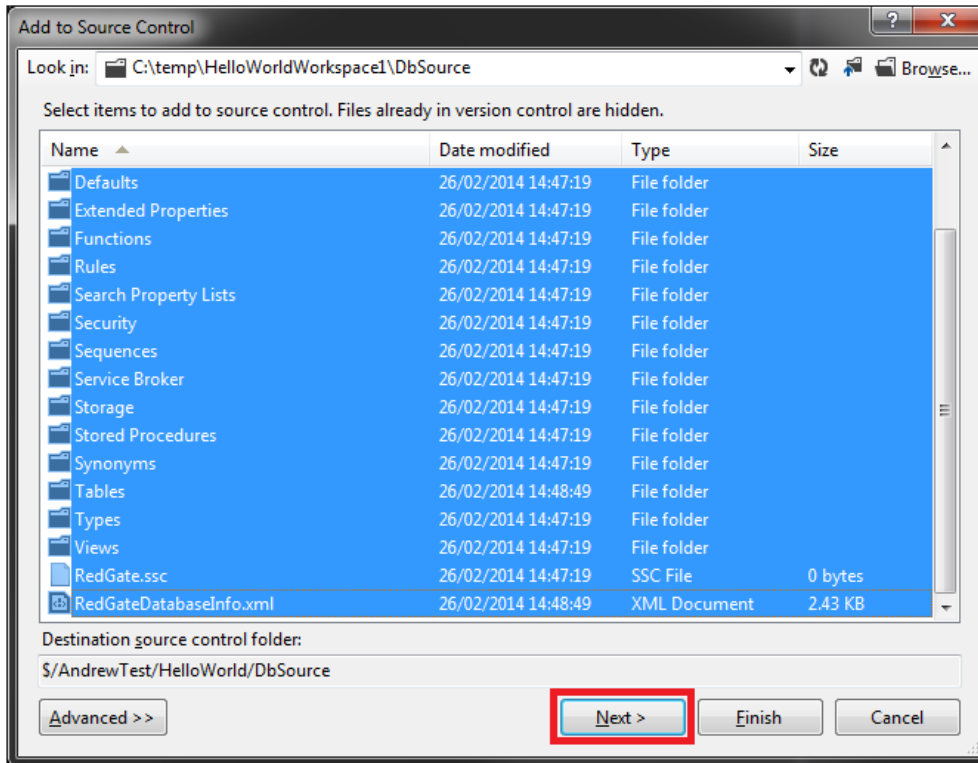
- The objects are saved to the working folder.
12. In Source Control Explorer, right-click on the folder that contains the database source and select **Add Items to Folder**.



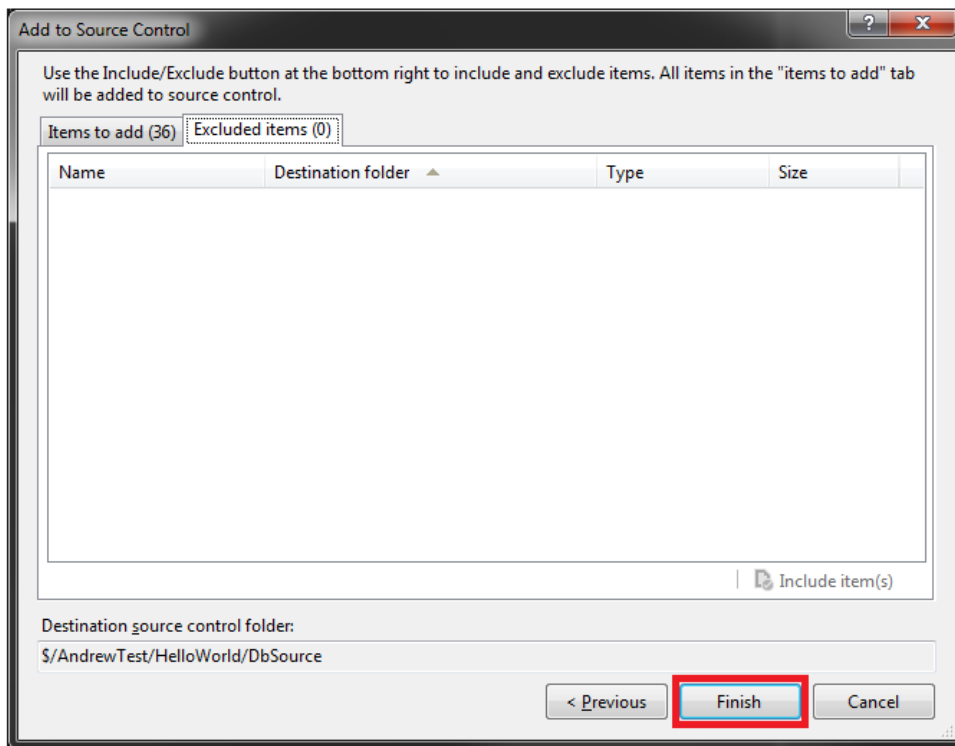
If you're using a local workspace (selected in step 4), Team Explorer will automatically detect and suggest new files for addition to the repository. You don't have to add them manually.

The **Add to Source Control** dialog box opens. This lists all the objects SQL Source Control added to the folder when we did the initial commit in step 10.

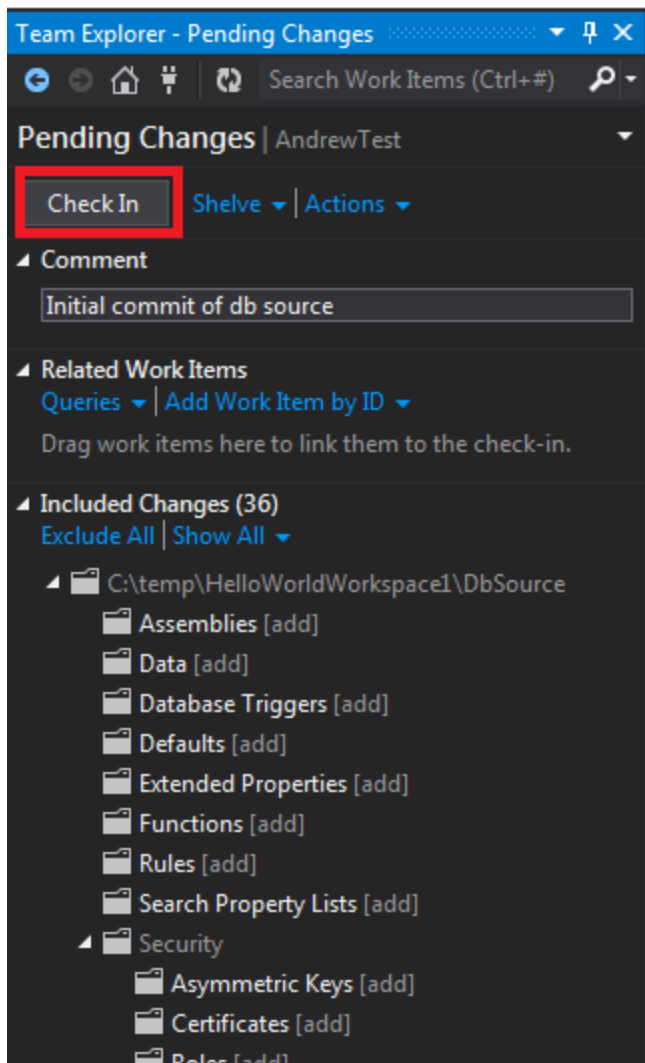
13. Select all the objects and click **Next**:



14. Make sure no objects are excluded, and click **Finish**:



15. Check in all the files we just added:



Now you can make database changes in SSMS and save the changes to the server workspace from SQL Source Control. You can then commit the changes, together with application code changes, in Visual Studio.

Evaluation repository

If you just want to experiment with SQL Source Control without setting up a source control system, we recommend the **evaluation repository**. This option creates a temporary source control repository on your computer using Subversion, a free source control system.

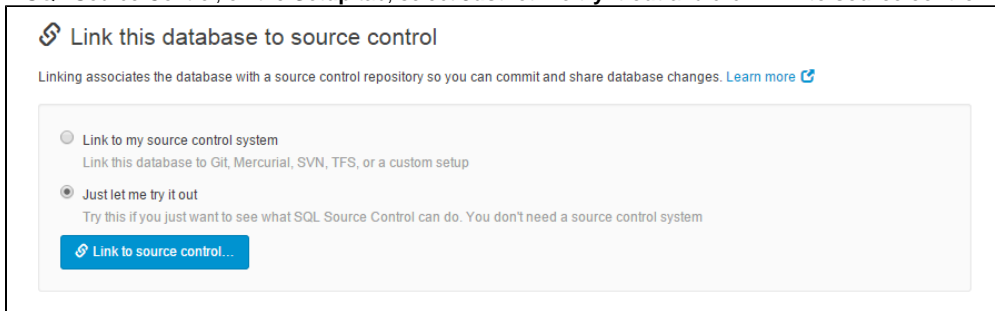
You shouldn't use the evaluation repository in the long term

The evaluation repository is designed to be used by a single person on a single computer, and doesn't work well for keeping backups or sharing changes with others – two of the major advantages of source control. When you're ready, you should link your database to your own source control system.

If you want to stop using the evaluation repository without losing your revision history, you can [move the evaluation repository to an SVN server](#). However, you can't move the history to other source control systems.

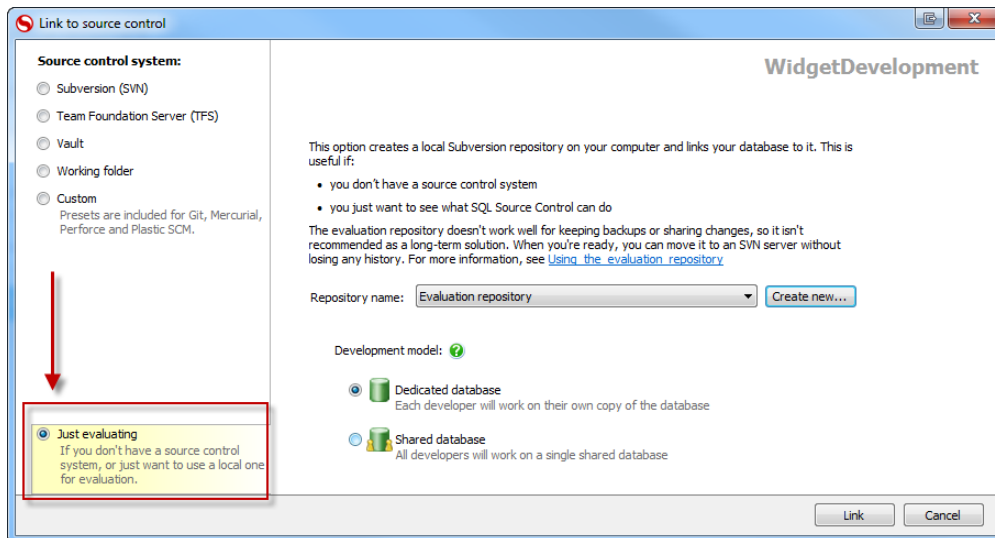
To link to the evaluation repository

1. In the Object Explorer, select the database you want to link to source control.
2. In SQL Source Control, on the **Setup** tab, select **Just let me try it out** and click **Link to source control**:



The **Link Database to Source Control** dialog box opens.

3. In the bottom left, make sure **Just evaluating** is selected:



4. Select a name for the repository from the drop-down list, or specify a new name with the **Create new** button.
5. Under **Development model**, make sure **Dedicated** is selected. Make which **development model** to use, dedicated or shared.

It's difficult to share local repositories. For the evaluation repository, we recommend you select **Dedicated database**.

6. Click **Link**.

The database is linked to the evaluation repository.

Moving the evaluation repository to a SVN server

If you're using the [evaluation repository](#), or any local Subversion (SVN) repository, you can migrate it to a SVN server without losing any log or history information.

You can do this using [VisualSVN Server](#), an installation and administration application for Subversion on Microsoft Windows servers. This example uses the free version of VisualSVN Server.

The example has three steps:

- 1. [Download and install VisualSVN Server](#)
- 2. [Import your evaluation repository](#)
- 3. [Unlink and relink the repository in SQL Source Control](#)

1. Download and install VisualSVN Server

To set up your SVN server, [download and run the VisualSVN Server installer](#) on your server, then follow the wizard to complete the installation.

For information on installing VisualSVN Server, see [Setting up a Subversion server](#).

2. Import your evaluation repository

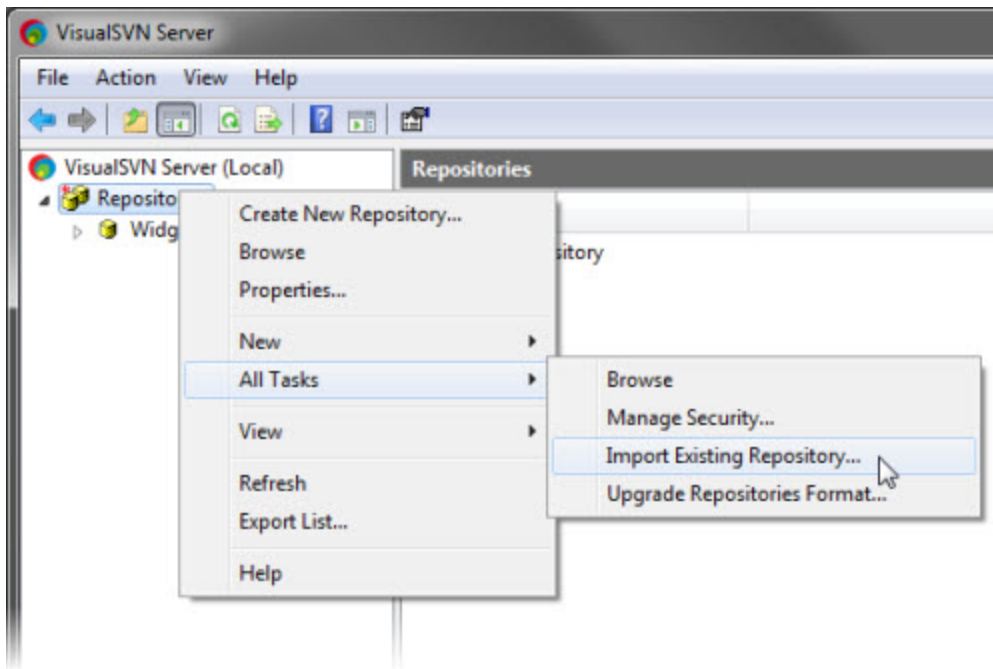
To import the contents of an evaluation repository to your new SVN server, you need to find the evaluation repository, copy the files to your server, and then import them using VisualSVN.

1. Locate the evaluation repository on your computer.
By default, the evaluation repository location is: `%USERPROFILE%\AppData\Local\Red Gate\SQL Source Control 3\EvaluationRepositories`

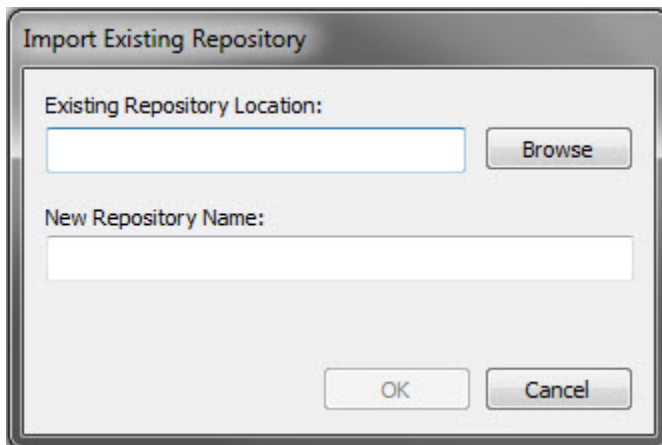
Alternatively, in the Object Explorer in SSMS, select the database you want to import. On the Setup tab of SQL Source Control, where it says *Linked to*, right-click the red text and click **Copy**:



- The repository location is copied to the clipboard.
2. Browse to the repository location and copy the repository you want to move to a location on your server.
3. On the server, open the VisualSVN Server Manager if it is not already open.
In the Console Tree pane to the left, right-click **Repositories**, click **All Tasks**, and click **Import Existing Repository**:



The **Import Existing Repository** dialog box opens:



4. In **Existing Repository Location**, paste the location of the repository copy.

A new repository name is supplied automatically. If you don't want to use the default name, type a new one.

5. Click **OK**.

The repository is created. The new repository contains any history information from the evaluation repository.

3. Unlink and relink the repository in SQL Source Control

To start using the new repository with SQL Source Control, unlink the evaluation repository and link the new one:

1. In the VisualSVN Server Manager, right-click the new repository, and click **Copy URL to Clipboard**.
2. In SQL Server Management Studio, make sure the database is selected in the Object Explorer, and on the Setup tab of SQL Source Control, click **Unlink database**.
3. A dialog box is displayed asking you to confirm the unlink. Click **Yes**.
The database is unlinked.
4. On the Setup tab, click **Link database to source control**.
The Link to Source Control dialog box is displayed.
5. In Repository URL, paste the URL of your new repository.
6. Under Development Model, select your development model: dedicated, or shared.
7. Click **Link**.
The database is now linked to the new repository on your Subversion Server.

Database development models

There are two common ways for teams to develop databases:

- **Dedicated**
Each developer has their own copy of the database
- **Shared**
Developers share a single copy of the database

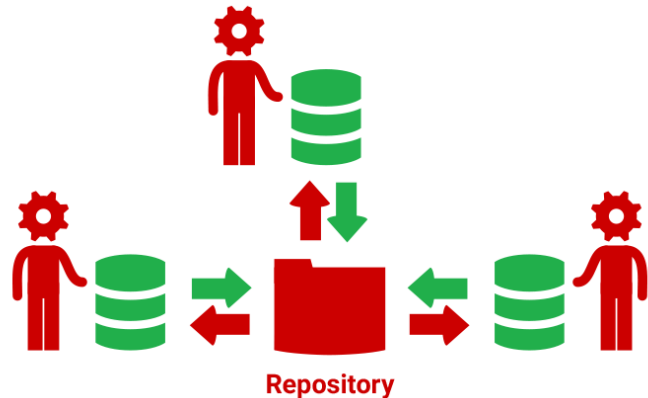
Some development teams use combinations of the two. For example, a large team might use more than one shared development database.

Dedicated development

Each developer works on their own "sandbox" copy of the database. The copy might be local or on a central server.

The developer makes changes independently, tests them, and commits them to source control once they're confident in the changes. After the changes are committed, other developers can apply them to their copy of the database. This mirrors application development.

Because each developer is working in an isolated environment, there's no risk of accidentally overwriting someone else's changes. You're free to develop complex changes that might cause other parts of the database or the application to break. If something goes wrong, it doesn't affect other developers.

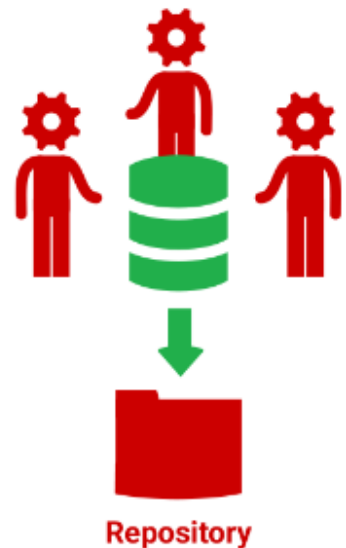


Shared development

Each developer works on a single shared copy of the database. This is usually created from a backup of the production database. When a milestone is reached, a deployment script is created to apply the final state of the database to a test database.

Because the changes are made directly to the database itself, you don't need to use the [Get latest tab](#). There are no changes to share or get, and the database always has the most recent changes.

However, there's no safe way to test changes in isolation, and developers may overwrite each others' changes by mistake. Additionally, [information about who made changes is sometimes lost](#).



Setting up SQL Source Control for dedicated development

1. One developer links their copy of the database to a location in source control for the first time.
2. The developer commits the objects.
3. The other developers link their own copy of the database to that source control location.
Alternatively, they can create a new blank database and link it to the database in source control, or restore a backup of the development database. Restoring a backup has the advantage of including any necessary data.
4. Each developer gets the latest version of the database.

Setting up SQL Source Control for shared development

1. One developer links the shared database to a location in source control for the first time.
2. The developer commits the objects.
3. In SQL Server Management Studio, the other developers connect to the database and link to source control.

For instructions, see [Linking to source control](#).

Switching to dedicated development

To switch from shared to dedicated development, each developer on your team needs to:

1. Unlink the database from source control.
2. Create an empty database. This will be the developer's dedicated copy.

For easy identification, you should name the database something similar to database you're about to copy.

3. Link the new database to the source control repository that the shared database was linked to.
4. Go to the **Get latest** tab and get the changes.
The new database is updated with the changes from the original database, creating a dedicated copy.

Static data

To source-control data in SQL Source Control, you first link it to source control, then commit it. Linking associates a table's data with source control so subsequent data changes are detected and static data can be shared across the team.

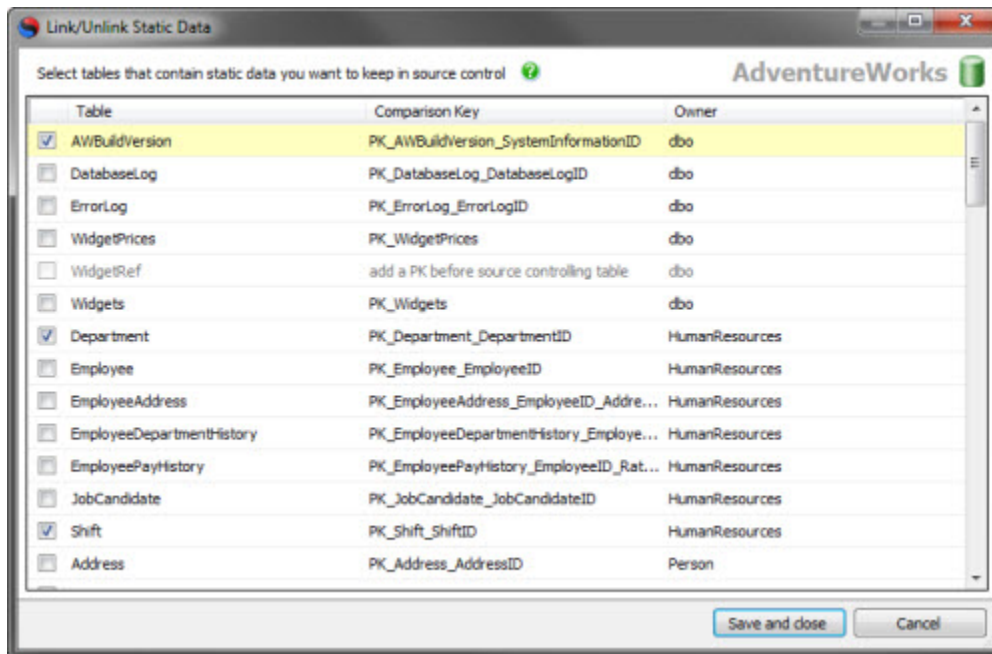
Linking to static data affects performance

Committing and getting the latest version of data can be slow. We recommend you only source-control static data (also known as lookup or reference data). Source-controlling very large static data tables might cause SQL Source Control to time out.

If static data is slowing down performance, you can disable checks for changes to static data on the **Setup** tab under **Options just for this database**.

To source-control data:

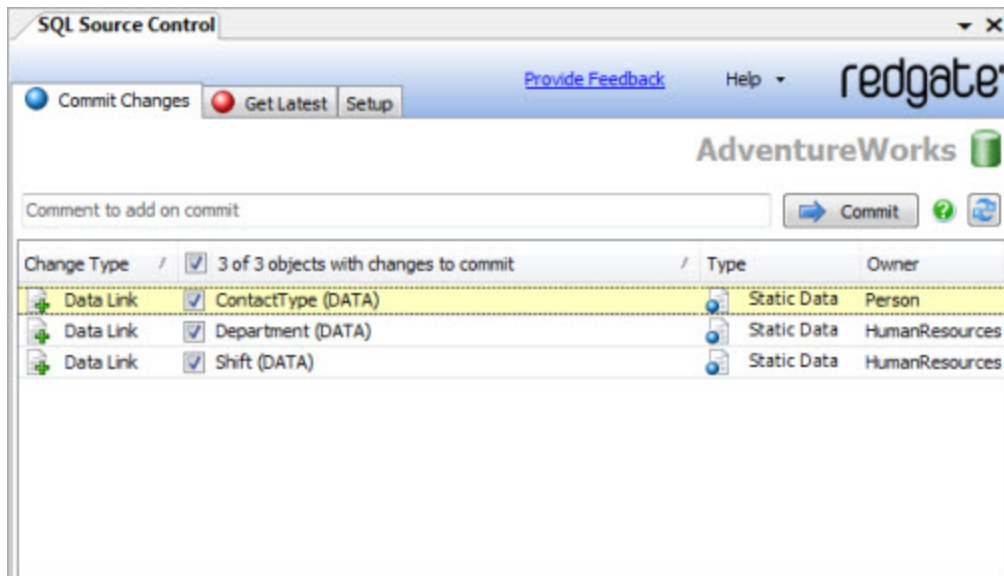
1. In the Object Explorer, right-click the database or table with data you want to source control and select **Other SQL Source Control tasks > Link or unlink static data**.
2. A dialog box opens with the **Link/unlink static data** tab selected:



The dialog box shows a list of the tables in the database.

You can only source-control data in tables with a valid primary key. The primary key is used as the [comparison key](#) to identify corresponding rows.

3. Select the tables you want to link and click **Save and close**.
SQL Source Control creates a file for the table's data, but the data itself hasn't been committed yet.
4. On the Commit Changes tab, make sure all **Data Link** changes are selected, type a comment (optional), and click **Commit**.



The data is committed to source control.

SSDT projects

You can only link to SSDT projects in SQL Source Control 3.4 and later.

You can link SQL Server Data Tools (SSDT) projects to source control. This means teams using Management Studio and Visual Studio can work on the same database.

Known issues

Linking to SSDT projects is an unsupported feature. However, because of the similar structure of SSDT projects and the script folders created by SQL Source Control's comparison engine, linking is possible, with some known issues:

- CLR assemblies aren't supported.
- Foreign and primary keys might be duplicated, which can cause exceptions.
- Database-level permissions might be misinterpreted and some users might not be given permissions.
- Visual Studio might not be able to read database-level extended properties created by SQL Source Control.

If you have problems, or if you'd like to see this feature supported, tell us on our [feedback forum](#).

To link an SSDT project:

1. Create a new empty database.
The SSDT project will be imported into this database.
2. With the new database selected in the Object Explorer, go to SQL Source Control. On the **Setup** tab, click **Link to source control**.

The **Link to source control** dialog box opens.
3. In the left pane, select your source control system.
4. Specify the location of the repository that contains the SSDT project. The repository must contain the .sqlproj file and the *bin*, *dbo* and *obj* folders.
5. If you're linking to a database that will be used by multiple developers, make sure **Shared database** is selected. For more information, see [Database development models](#).
6. Click **Link**.

SQL Source Control imports the SSDT project to the empty database and links it to source control.

For information about linking databases to source control, see [Linking a database to source control](#).

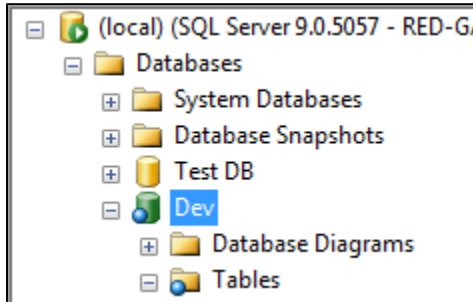
Using SQL Source Control

- [Committing changes](#)
- [Getting the latest version](#)
- [Viewing source control history](#)
- [Getting a specific version](#)
- [Undoing changes](#)
- [Conflicts and merging](#)
- [Using filters to exclude objects](#)
- [Branching and merging](#)
- [Deploying a database from source control](#)
- [Working with migration scripts](#)

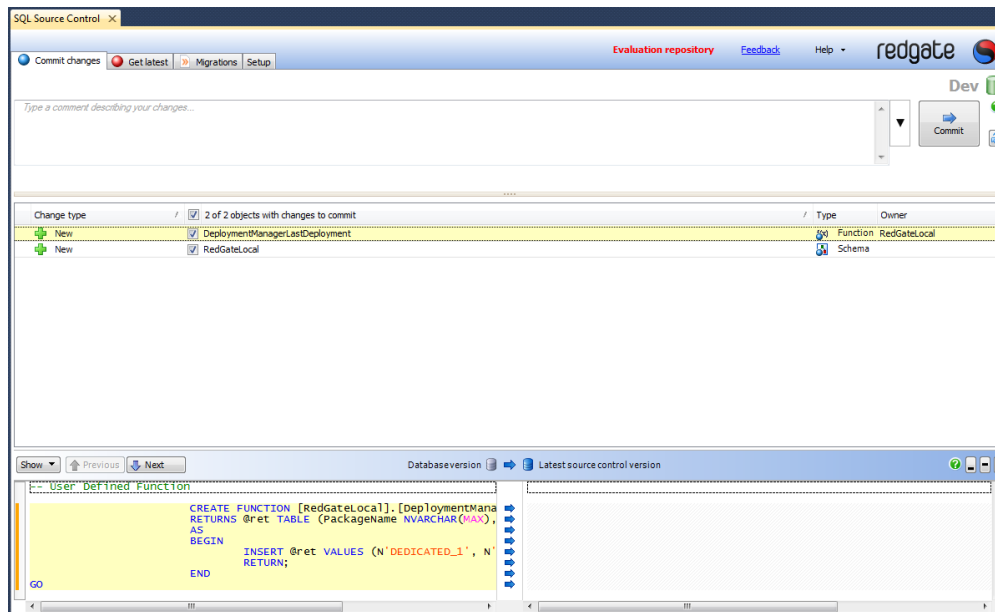
Committing changes

Committing a change updates source control with that change.

When you make changes to source-controlled databases, SQL Source Control labels affected objects in the Object Explorer:



To update source control with your changes, click the **Commit changes** tab:



The Commit changes tab lists all the objects in your database that anyone has made changes to.

To commit changes:

1. Select the objects you want to commit.
2. Write a comment describing your changes (optional).

To reuse previous commit comments, click



3. Click **Commit**.

A progress dialog box is displayed while SQL Source Control commits the changes to your source control system. Click **OK** to close the dialog box.

Source control is updated with your changes.

You can also commit changes by right-clicking an object, folder, or database in the Object Explorer, and clicking **Commit changes to source control**. The **Commit changes** tab is displayed, with the objects you clicked selected to commit.

Integrating with bug tracking systems

You can use SQL Source Control to associate your commits with an SVN bug IDs or TFS work items. You might do this to better integrate

database changes with a bug-tracking system, or the project management functionality of Team Foundation Server.

SQL Source Control only supports this by adding the bug ID or work item number to the comment you make when committing changes.

For more information about setting up SVN bug IDs, see [Integration with Bug Tracking Systems / Issue Trackers](#) on the Tortoise SVN site.

SVN bug IDs

SVN bug IDs are enabled using *bugtraq* properties. You can use these to add commit hooks to make SVN parse log messages for bug or issue numbers and relate them to your issue tracking system.

To associate a commit with a bug or issue, include the issue number in the commit comment with a **#** symbol. For example: *This commit addresses issue #100*

TFS work items

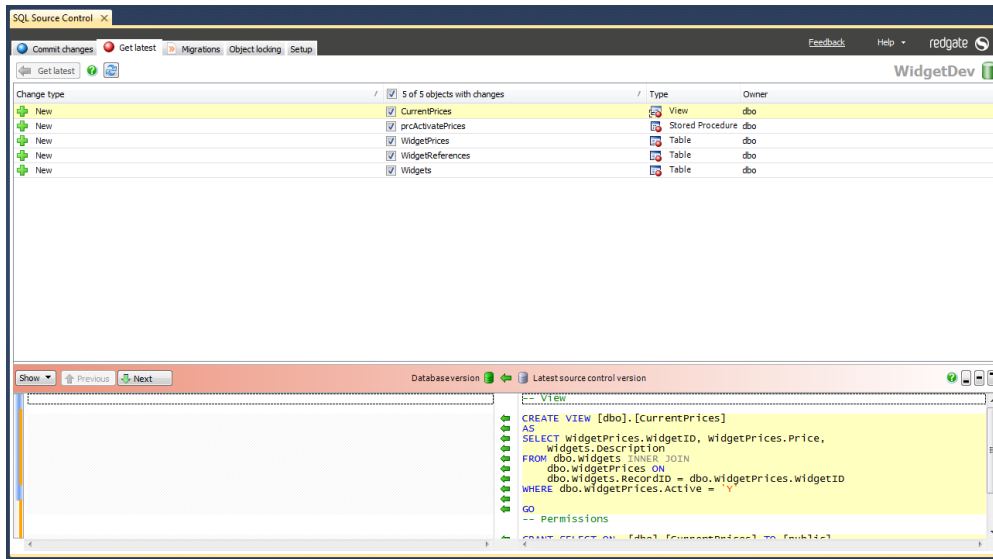
TFS work items are used to track pieces of work in a development project. Each work item has a number.

- To associate a commit with a work item, include **#A[Work Item number]** at the **start** of the comment. For example: *#A106 adding tables*
- To resolve a work item, include **#R[Work Item number]** at the **start** of the comment. For example: *#R106 adding tables*

The numbers don't appear in the commit comment recorded on the TFS Server.

Getting the latest version

If an object has a more recent source control version than the version currently in the database, you can get that version on the **Get latest** tab.



To get the latest version of an object:

1. On the **Get latest** tab, select the objects you want to update to the latest version
2. Click **Get latest**.

A progress dialog box is displayed while SQL Source Control updates the database.

The database is updated to the latest version.

You can also get the latest version by right-clicking an object, folder, or database in the Object Explorer, and clicking **Get latest changes from source control**. You are taken to the **Get latest** tab, where the objects you clicked are selected.

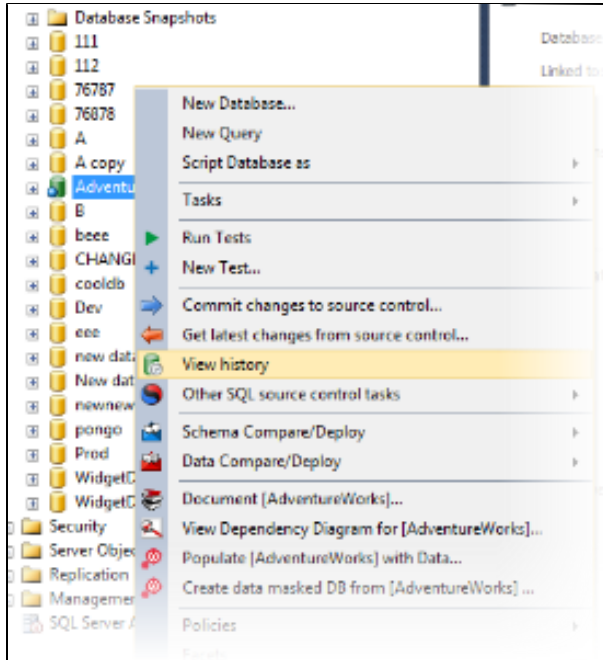
If you're using a shared database, you never need to get the latest version. The shared database is always up to date with everyone's changes.

Viewing source control history

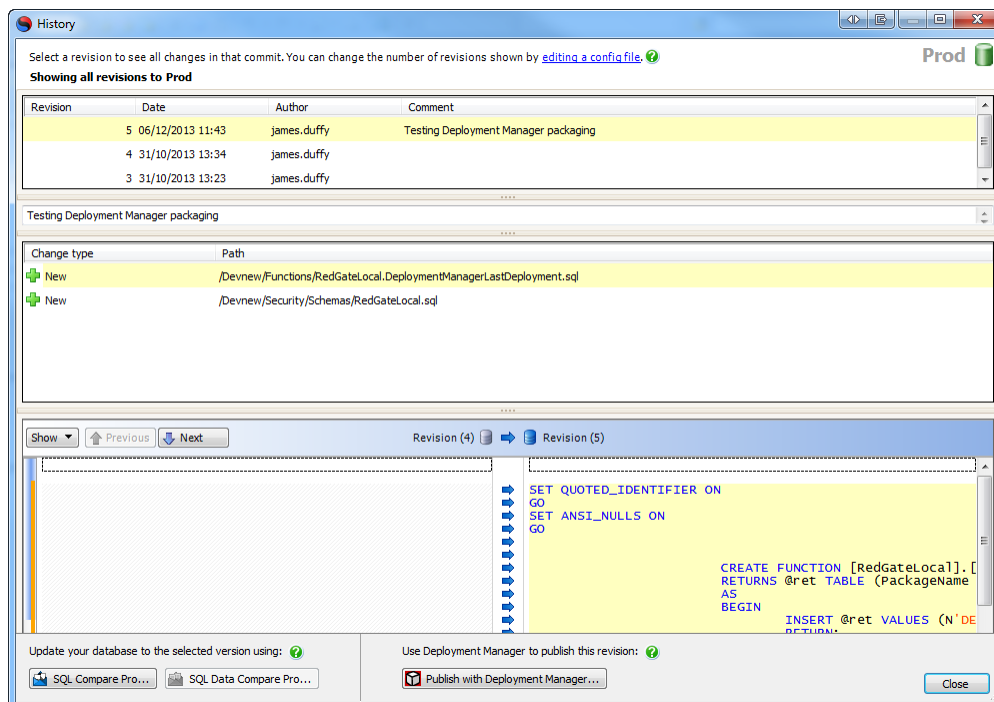
You can only view history for databases linked to SVN, TFS or Vault. To view history for other source control systems, use your source control client's history feature.

You can use SQL Source Control to view the revision history of a database or database object.

In the Object Explorer, right-click the database or object and click **View history**:



The **History** dialog box opens:



The History dialog box shows:

- each database revision in source control (SVN revision or TFS changeset)
- the author, date, and comment associated with each commit

- which objects changed in each commit
- the SQL differences for each object (before and after)

You can use the History dialog box to [get a specific version of the database](#).

You can't view history information for databases linked to source control using command line support.

Changing the number of revisions shown

In SQL Source Control 3.5.1 and later, you can change the number of revisions shown in the History dialog box by editing a config file. If the History dialog box is slow to load, limiting the number of revisions shown might speed it up. For more information, see [Changing the number of revisions shown in the History dialog box](#).

Getting a specific version

You can update your database to a specific version in the [History dialog box](#) using SQL Compare or SQL Data Compare.

When you do this, the current database schema (and any source-controlled data) is overwritten with the version you selected.

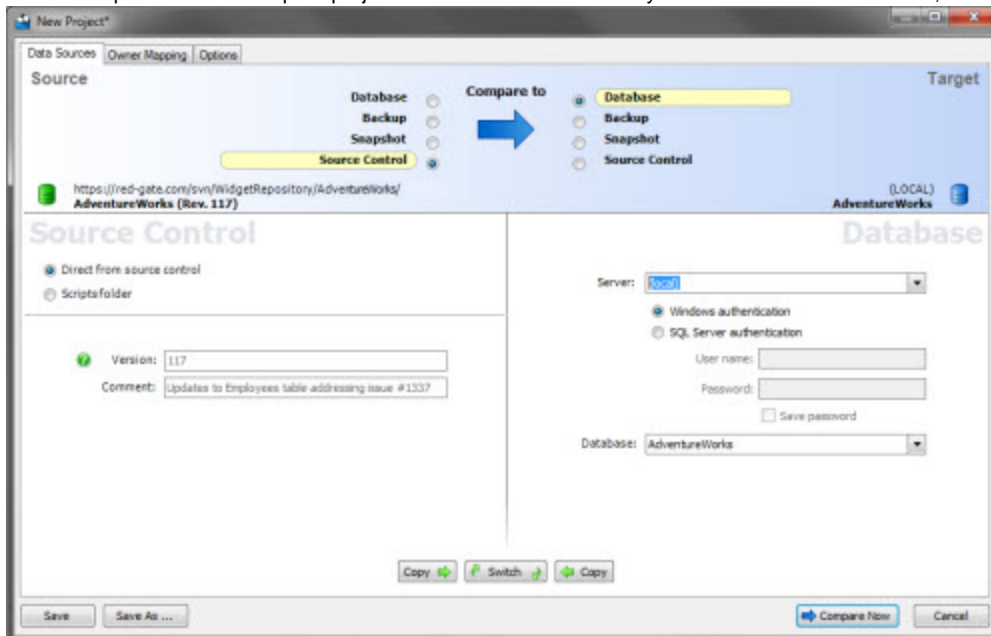
- If you want to update the schema, you need [SQL Compare Pro 8.5 or later](#).
- If you want to update the data, you need [SQL Data Compare 10 or later](#).

Getting a specific version from the History dialog box

When you get a version from the History dialog box, SQL Compare or SQL Data Compare is used to update the target database.

To get a specific version:

1. On the History dialog box, select the version you want to get.
2. If you want to update the schema, click **Update to this version with SQL Compare**.
If you want to update the data, click **Update to this version with SQL Data Compare**.
3. A SQL Compare or Data Compare project launches with the version you selected set as the source, and the database as the target:



4. To update the database, run the comparison, make sure all objects you want to update are selected, and run the deployment wizard.

For more information about comparing databases with SQL Compare, see [Worked example - comparing and deploying two databases](#) in the SQL Compare documentation.

Getting a specific version manually

You can get a specific version using your source control system and SQL Compare.

Example: getting a specific revision with TortoiseSVN

This example creates a local copy of the revision and deploys the database using SQL Compare:

1. Check out the latest version to a new folder.
2. Right-click the folder, and from the **TortoiseSVN** menu, select **Show log**.
The Log Messages dialog box is displayed.
3. Select the revision you want to get, right-click, and click **Revert to this revision**.
4. A confirmation dialog box is displayed. Click **Yes**.
The folder of scripts is updated to the revision you selected.
5. Using SQL Compare, set the scripts folder as the source for a comparison, and the database as the target, then compare and deploy.
The database is updated to the revision you selected.

Example: getting a specific changeset with TFS

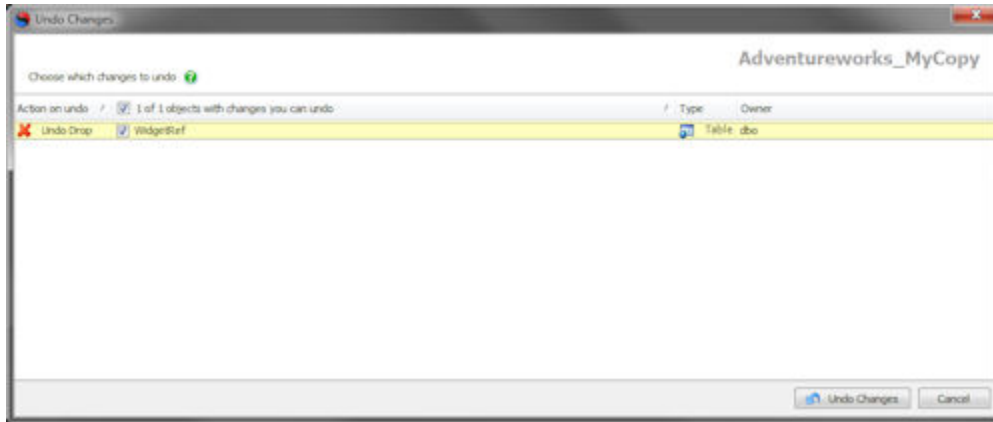
This example updates the TFS local copy to a specific changeset, and deploys the database using SQL Compare:

1. In Visual Studio, in the **Source Control Explorer** tab, select the database, right-click, and click **Get Specific Version**.
The **Get** dialog box is displayed.
2. Under **Version**, in **Type** select *Changeset*.
3. Type the changeset number, or click the browse button to display the Find Changesets dialog box, and select the changeset you want.
4. Click **Get**.
The local scripts folder has been updated to the changeset you selected.
5. Using SQL Compare, set the local scripts folder as the source for a comparison, and the database as the target, then compare and deploy.
The database is updated to the revision you selected.

Undoing changes

You can undo changes you've made to a database that aren't committed to source control yet.

1. In the Object Explorer, right-click the object, folder, or database with changes you want to undo, select **Other SQL Source Control tasks**, and click **Undo changes**. Alternatively, right-click an object on the Commit changes tab, and click **Undo changes**. The **Undo changes** dialog box opens:



2. Select the objects with changes you want to undo and click **Undo Changes**.
A progress dialog box is displayed while SQL Source Control runs the script to undo the changes.
3. When the undo is complete, close the dialog box.

Changes you can't undo

The most common types of change SQL Source Control can't undo are:

Committed changes

You can only use SQL Source Control to undo changes you haven't committed.

To undo changes that have been committed, use SQL Server Management Studio Integration Pack to [update the database to a specific version](#), or use your source control system.

Static data changes

You can't undo a data link or a data edit.

To stop source-controlling a table's data, right-click it in the object explorer and click **Link/Unlink Static Data**. In the **Link/Unlink Static Data** dialog box, you can choose to stop source controlling the table's data.

Dropped data

If you drop a table or column that contained data, the data isn't restored when you undo the drop.

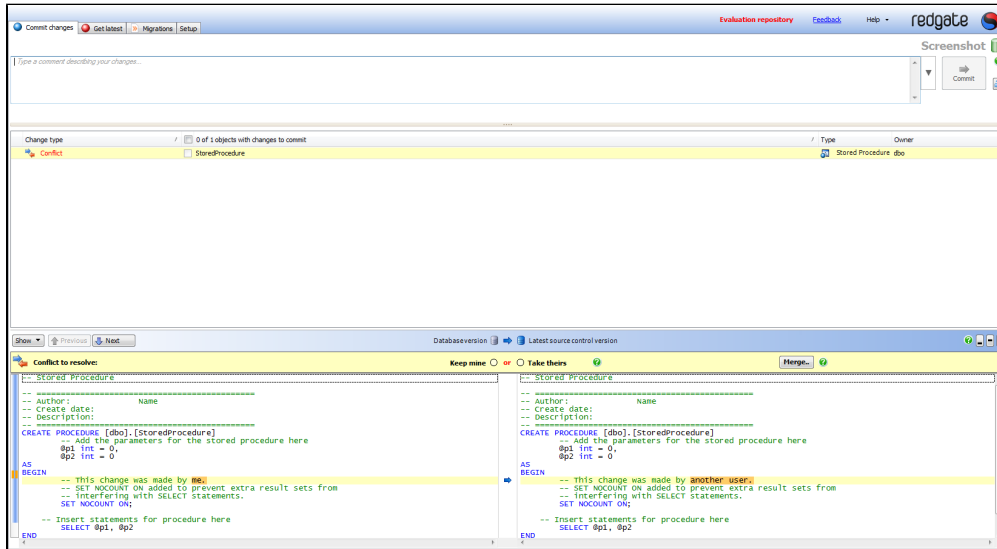
NOT NULL columns

If you drop a NOT NULL column from a table that contained data, undoing the drop will fail if the column doesn't have a default value.

Conflicts and merging

Conflicts happen when the latest version of an object in source control and the latest version in the database are incompatible. This happens if two people modify the same object. Conflicted objects can't be committed or retrieved until the conflict is resolved. To avoid conflicts, we recommend you get the latest version of an object before modifying it.

When a conflict happens, the **Conflict to resolve** bar is shown beneath the change list:



To resolve a conflict, choose either:

- **Keep mine**
When you click **Commit**, your version of the object is committed to source control.
- **Take theirs**
The object can't be committed. To get the source control version of the object, go to the Get latest tab.

Conflicts involving data

If an object has both schema and data changes, you need to commit or get the latest schema and data changes at the same time. You can't only commit the data changes. This is because data changes may fail without the associated schema changes. This means if an unconflicted data change is associated with a conflicted object, you can't commit or get the data change without resolving the schema conflict first.

Merging conflicted stored procedures from SQL Source Control with Beyond Compare 3 or KDiff3

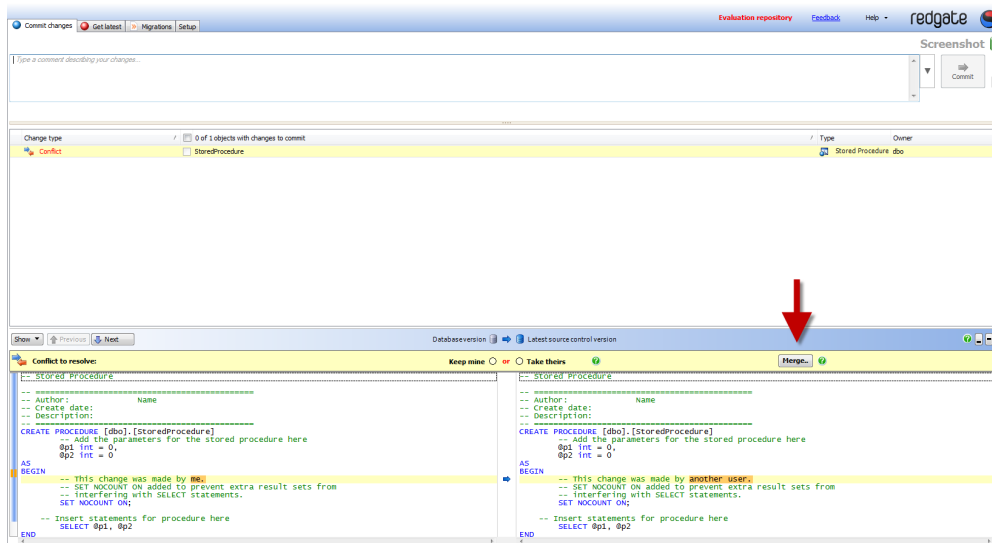
You can only merge conflicts with SQL Source Control 3.5 or later.

SQL Source Control has no native merge functionality. However, you can use the third-party tools [Beyond Compare 3](#) or [KDiff3](#) to merge [conflicted stored procedures](#) from SQL Source Control.

You can request support for other object types on the [SQL Source Control feedback page](#).

To merge conflicting stored procedures:

1. Select the conflicted stored procedure and click **Merge** in the conflict resolution bar:



Beyond Compare 3 or KDiff3 opens. If both Beyond Compare 3 and KDiff3 are installed, SQL Source Control opens Beyond Compare 3.

Management Studio is disabled while the merge tool is open.

2. Merge and save the change in the merge tool.

Your original change will be lost when you close the merge tool.

3. Close the merge tool.

The output from the merge tool is immediately executed on the database. This happens before you commit or get latest.

- If the output is different from the version in source control, **Keep mine** is automatically selected and the change is ready to be committed or retrieved.

Merged conflicts are still listed as unresolved until you commit or retrieve them.

- If the output is identical to the version in source control, there are no longer any changes to commit or get for that object. The object is removed from the change list.

Merging other objects

To merge changes from two conflicting versions of an object, resolve the conflict, and then manually edit the object to include the changes from the other version.

You can copy either version of an object's creation script from the Object Differences pane, and paste it into a new query window.

Using filters to exclude objects

You can create filters to exclude objects from SQL Source Control.

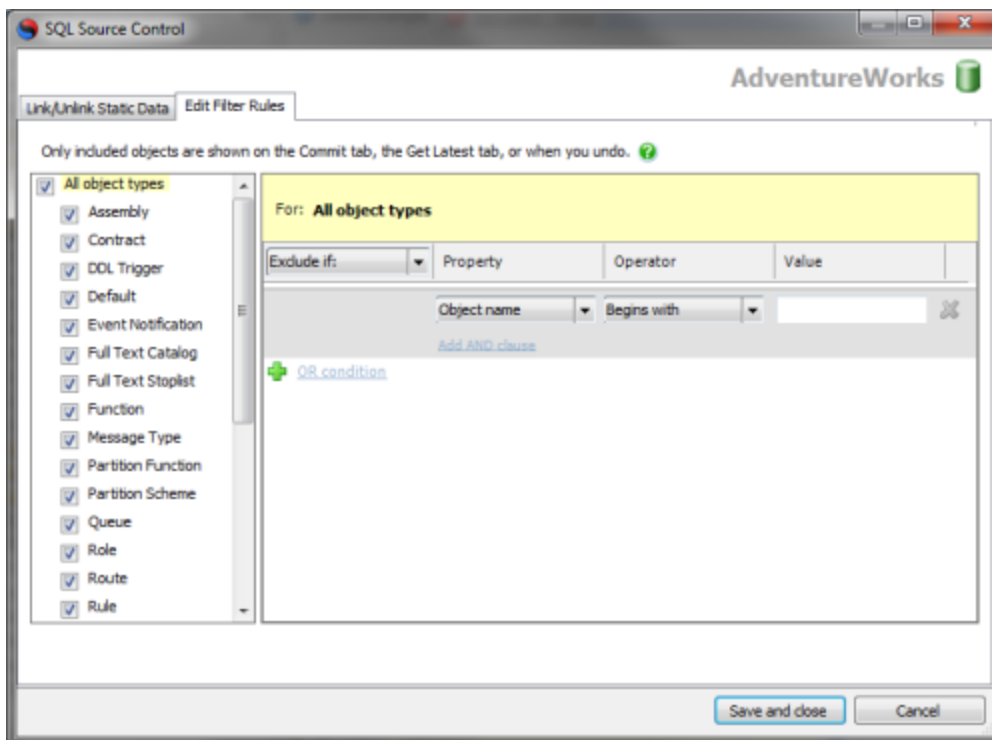
When you exclude an object with a filter, it isn't shown on the Commit changes tab, the Get latest tab, or the Undo changes dialog box. This means you can never commit, get, or undo an excluded object. This is useful, for example, if there's a set of objects that you never want to commit.

Creating and editing filters

There are two ways to create or edit your filter:

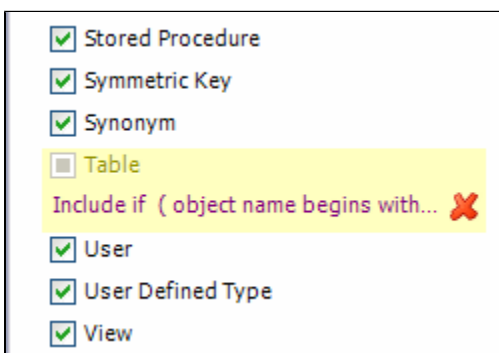
- In the Object Explorer, right-click a database, folder, or object, then select **Other SQL Source Control Tasks** and click **Edit Filter Rules**.
- On the Commit changes or Get latest tabs, right-click an object in the list of changes and click **Edit Filter Rules**.

The **Edit filter rules** tab lets you specify exclusion or inclusion conditions for individual objects or all object types:



You can exclude object types using the check boxes in the left-hand pane, or build more complex conditions by specifying AND clauses and OR conditions in the right-hand pane.

When you create a filter rule, its conditions are displayed in the left-hand pane under the name of the object type it applies to:



To clear the filter rule for an object type, click



next to its name.

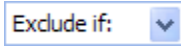
Example: excluding objects with a specific name or owner

To exclude all objects with names beginning with *Marketing*, or any owned by the schema *Marketing*, regardless of their names:

1. In the Object Explorer, right-click the database, select **Other SQL Source Control Tasks**, and click **Edit Filter Rules**.

The Edit Filter Rules dialog box is displayed.

2. In the



box, make sure *Exclude if* is selected.

3. Under **Property**, select *Object name*.
4. Under **Operator**, select *Begins with*.
5. Under **Value**, type *Marketing*.
6. Click



A new OR condition becomes available.

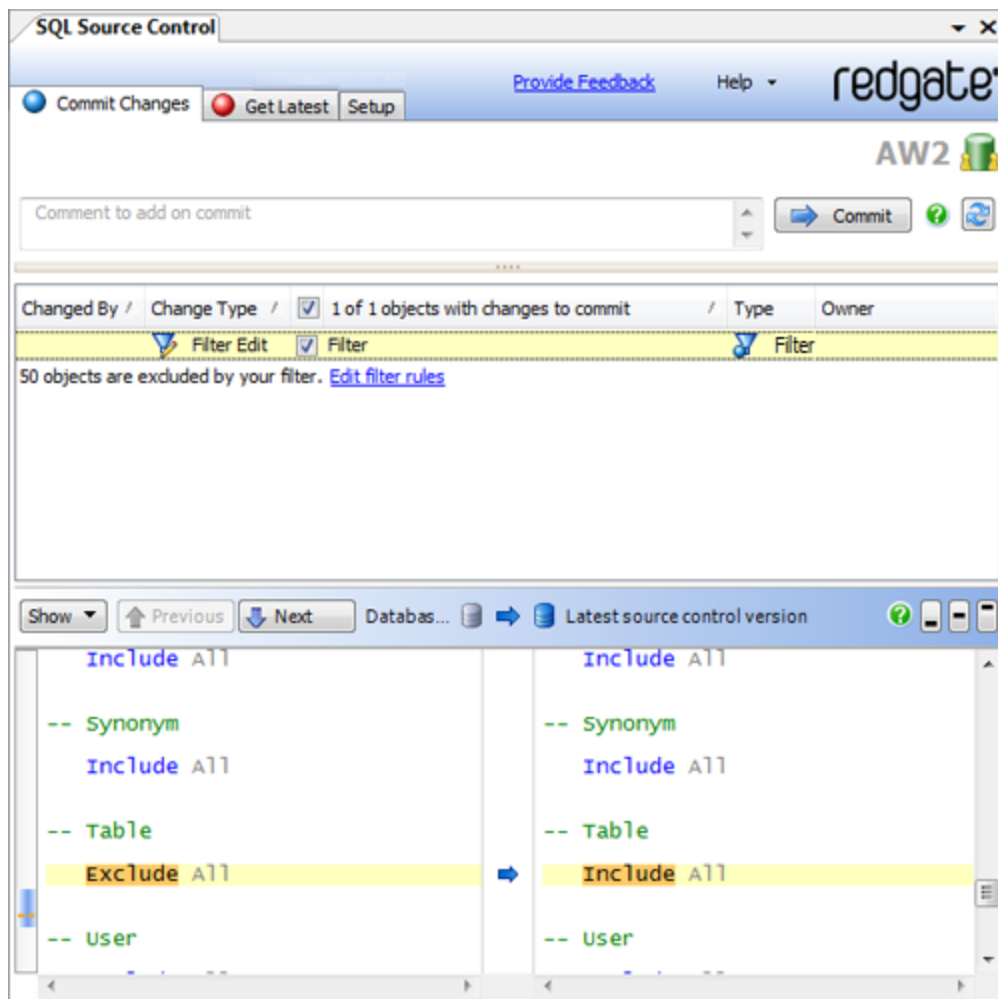
7. Under **Property**, select *Schema name*.
8. Under **Operator**, select *Equals*.
9. Under **Value**, type *Marketing*.
10. Click **Save and close**.

The filter is applied. All objects owned by the schema *Marketing* or with names that begin with *Marketing* are excluded by SQL Source Control and won't appear in the Commit changes tab.

Sharing filters

You can commit your filter to source control and get the latest version when it changes. This is useful for teams that want to exclude a set of objects across an entire development project.

When you create or edit a filter, SQL Source Control shows it on the **Commit changes** tab:



You can see the differences between your current filter and the version in source control.

Branching and merging

Many development projects involve creating branches (or "forks") for a feature, release, or other development milestone. A branch is essentially a copy of the code base that shares its history. Incorporating the changes from a branch is known as merging.

Development projects typically have:

- a *trunk*, the main code base, and branches which diverge from it
- one or more *branches*, copies that diverge from the trunk

For more information about the concepts behind branching and merging, see:

- [Branching and Merging with Team Foundation Server](http://msdn.microsoft.com) (msdn.microsoft.com)
- [Branching and Merging with Subversion](http://svnbook.red-bean.com) (svnbook.red-bean.com)

For a general introduction to source control concepts, we recommend [Version Control by Example](#) (free PDF) by SourceGear founder Eric Sink.

Working with branches in SQL Source Control

SQL Source Control lets you work with branches, but it doesn't currently let you create or merge branches.

To branch with SQL Source Control, you must create the branch using your source control system and then link to the appropriate branch. Once you've created a branch, you're ready to work on it in SQL Server Management Studio with SQL Source Control.

There are two approaches to working with branches:

Unlink and relink the database

Here, you continue working on the same database in SQL Server Management Studio, but link it to the branch in your source control system.

Once the branch is created, un-link the database in SQL Source Control, then link the database to source control again. When you link, specify the location of the branch in source control.

Create a new database for the branch

This method only works if you're using the [dedicated development model](#).

Here, you create the branch in your source control system, then create a new database to link with it in SQL Server Management Studio.

Create a new empty database, and link it to source control in SQL Source Control. When you link, specify the location of the branch in source control, then on the Get Latest tab, update the database with the latest version from source control.

Merging

SQL Source Control doesn't provide automatic or line-by-line merging. You can use SQL Source Control or [SQL Compare](#) to merge at an object level, but not choose line-by-line changes.

When you merge with SQL Source Control or SQL Compare, you choose a version of each object to keep. For example, you might keep the trunk version of a table and the branch version of a view.

There are three approaches to merging:

Merging using your source control system

You can manually merge the branch changes back into the trunk using your source control system as you would for application code.

We recommend this approach if the merge is complex, or if there are conflicts; for example, if the same object has been modified in both the branch and the trunk.

When merging manually, make sure referential integrity is maintained or the database may be left in an invalid state.

Your source control system may include auto-merging functionality that simplifies manual line-by-line merges.

Merging using SQL Source Control

If you don't need to do a line-by-line merge, you can merge with SQL Source Control.

To merge branch changes back into the trunk with SQL Source Control:

1. In SQL Source Control, make sure you have a database linked to the branch in your source control repository.
2. Get the latest version and commit any outstanding changes.
3. Unlink the database from the branch.
4. Relink the database to the trunk.
5. Go to the **Commit Changes** tab.
The tab shows the changes to the branch as changes to commit.
If there are conflicts, choose *Keep mine* to override the trunk with the objects from the branch.
6. Commit the changes.
The trunk is updated with the branch changes.

Merging using SQL Compare

If there are no conflicting changes between the branch and the trunk, you can merge automatically using SQL Compare.

To merge branch changes into the trunk with SQL Compare:

1. In SQL Source Control, make sure you have a database linked to the trunk.
2. Use your source control system to create a local copy of the latest branch version, for example by performing an SVN checkout.
3. In SQL Compare, [create a new project](#). Set the local copy of the branch as the *source*, and the trunk database as the *target*.
For more information, see [Setting data sources](#).
4. Compare the data sources.
SQL Compare shows the differences between the branch and the trunk.
5. In the Results pane, select the objects from the branch that you want to merge into the trunk, and run the Deployment Wizard to [deploy from the data source to the target](#).
The trunk is updated with the changes from the branch.
6. In SQL Source Control, on the **Commit Changes** tab, commit the trunk changes to source control.

Deploying a database from source control

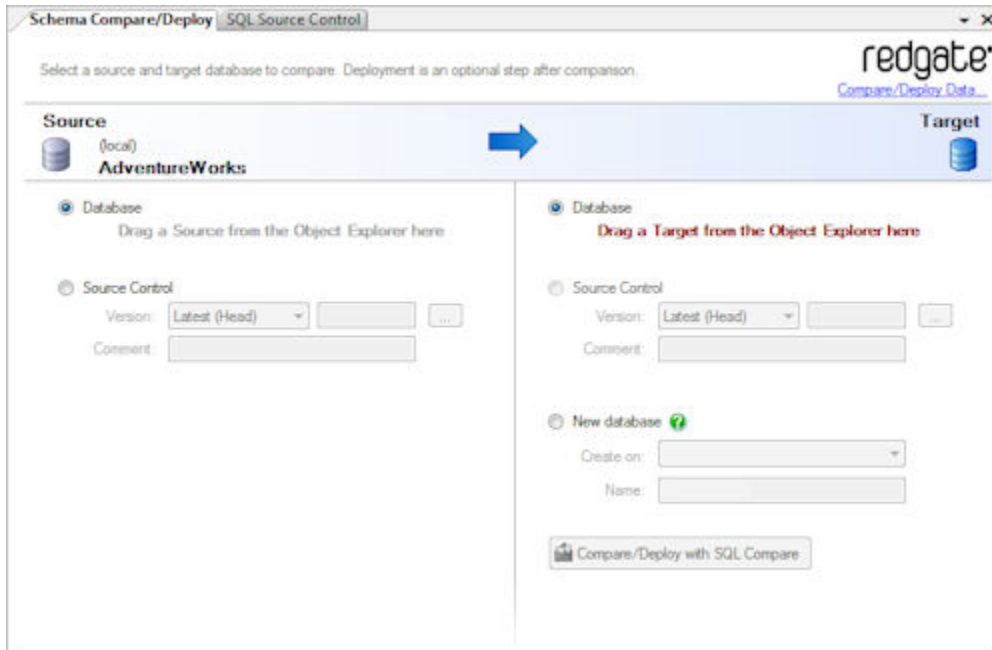
You can use SQL Source Control and [SQL Compare](#) to deploy a database from source control to a server.

You can also use the SQL Server Management Studio Integration Pack add-in to make schema and data deployment simpler.

Deploying with SQL Server Management Studio Integration Pack

To deploy a database schema, in the Object Explorer, right-click a database, select **Schema Compare/Deploy**, and click **Set as Source**.

The SQL Server Management Studio Integration Pack Schema Compare/Deploy tab is displayed:



You can deploy the current database version or specify a version from source control.

You can deploy to a target database, create a new database, or create a change script to update a target source control version.

For more information, see [Getting started with the SQL Compare add-in](#).

Deploying without SQL Server Management Studio Integration Pack

To deploy a database:

1. Create a local copy of the scripts folder
2. Migrate the local copy to the target server using SQL Compare

Optionally, you can also deploy any relevant static data using SQL Data Compare.

In this example the database *WidgetDev* is already in source control.

The example uses the SQL Compare and Subversion command line interfaces.

You can also deploy the database using the SQL Compare graphical user interface and a source control client like [TortoiseSVN](#).

1. Create a local copy of the database

At a command prompt, type:

```
cd C:\program files\subversion\bin

svn update http://<your repository path>/WidgetDev "C:\WidgetDevScripts"
```

Where:

- *http://<your repository path>/WidgetDev* is the URL for the database in your Subversion repository
- *"C:\WidgetDevScripts"* is the file path for the directory where the local copy will be created

A local copy of the scripts folder is created. This is a Subversion working copy, and is associated with the Subversion repository.

2. Migrate the local copy to the target server

At a command prompt, type:

```
cd C:\program files\red gate\SQL Compare 8

/sqlcompare /scr1:"C:\WidgetDevScripts"

/S2:WidgetServer /U2:<username> /P2:<password>

/db2:"WidgetTest"

/sync
```

Where:

- */scr1:"C:\WidgetDevScripts"* specifies the local copy, WidgetDevScripts, as the source for a SQL Compare deployment
- */S2:WidgetServer /U2:<username> /P2:<password>* specify the server, user name, and password you are using
- */db2:WidgetTest* specifies WidgetTest as the target of a SQL Compare deployment
- */sync* performs the SQL Compare deployment, making the schema of WidgetTest the same as the schema in WidgetDevScripts

The database is updated. Its schema is now the same as the version you checked out of source control.

For more information, see [Simple examples using the command line](#).

Working with migration scripts

This page describes how to use the first version of the migration scripts functionality, released with SQL Source Control 3.0. This version isn't compatible with distributed version control systems (DVCS) such as Git and Mercurial.

We've released a beta version of the V2 migration feature that's compatible with all systems. For more information, including information about how to enable the beta in SQL Source Control, see [Migration scripts V2](#).

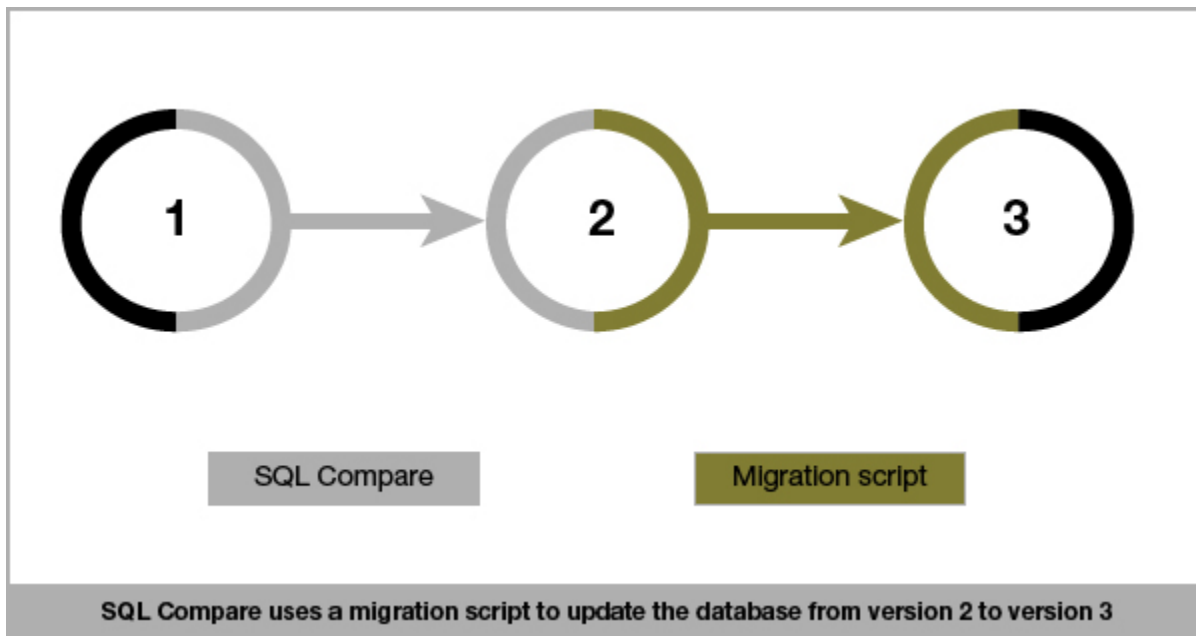
For an example of how migration scripts can be used in deployment, see [Example - deploying with migration scripts](#).

Migration scripts are customizable change scripts created in SQL Source Control and used by SQL Compare in deployment.

This page describes:

- [What migration scripts contain](#)
- [When to use migration scripts](#)
- [How to create a migration script](#)

SQL Compare includes migration scripts in the final deployment script it creates:

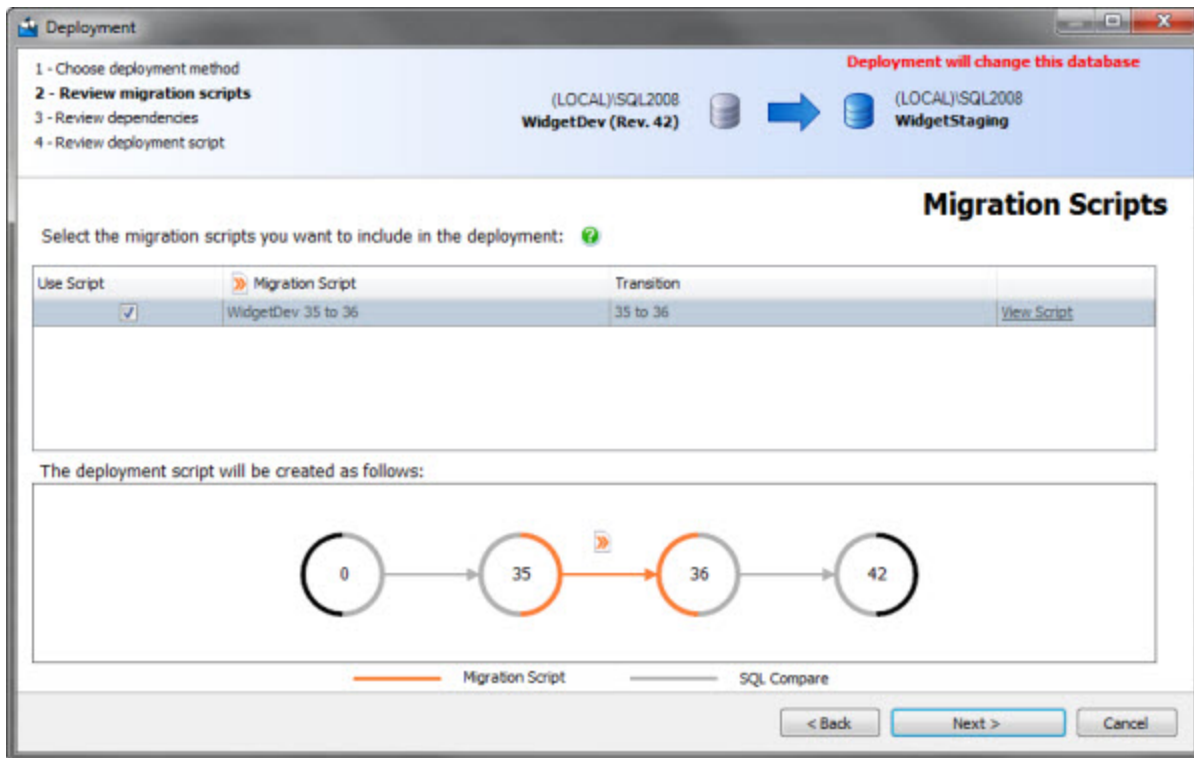


What migration scripts contain

Migration scripts can contain overrides to the default deployment script SQL Compare generates, or specific configuration changes for your environment. For example, you may need to disable replication before making certain changes, and re-enable it afterwards. This instruction can be added to the migration script for those changes.

When you deploy a database, SQL Compare checks for migration scripts. If it finds a change that is covered by a migration script, it uses this script instead of the change script it would normally generate.

During deployment with SQL Compare, you can choose which migration scripts to include in deployment on the Review Migration Scripts page of the Deployment Wizard:



When to use migration scripts

Migration scripts are most useful in two situations:

- to avoid data loss during deployment
- to avoid manually making configuration changes with each deployment

Configuration changes will depend on your specific environment, and aren't described in detail here.

The most common situations in which you may need a migration script to avoid data loss are:

- **Adding a NOT NULL constraint to a column**

If you add a NOT NULL constraint to a column in a table that already has data, and the column does not have a default value, deployment will fail. Without a default value, SQL Compare can't update the table.

You can create a migration script to specify the default value, and this will be included in deployments.

- **Renaming a table**

When you rename a table in SQL Server Management Studio, SQL Source Control detects this as dropping and re-creating the table. If the table contains data, the data will be lost.

You can create a migration script to override this by performing the rename with the `sp_rename` stored procedure.

- **Table and column splits and merges**

Normalization and denormalization activity such as splitting a column can be interpreted by SQL Source Control as dropping one column and creating two. Any data in the column will be lost.

You can create a migration script to make sure these operations are carried out with ALTER TABLE statements.

- **Changing the data type (or size) of a column**

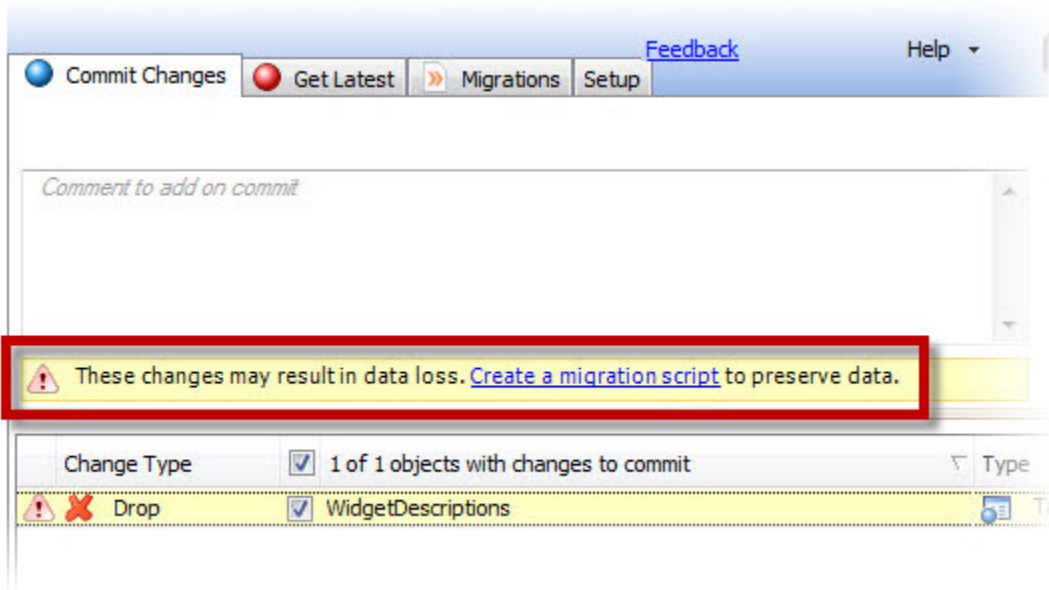
When you change a column's data type, for example from *int* to *smallint*, it is possible to lose data. If the type you are changing to does not accommodate some of the rows, data is truncated during deployment. Similarly, changes to the size of some columns can result in truncation, for example, *varchar(50)* to *varchar(20)*.

You can create a migration script to appropriately modify rows that would be truncated.

How to create a migration script

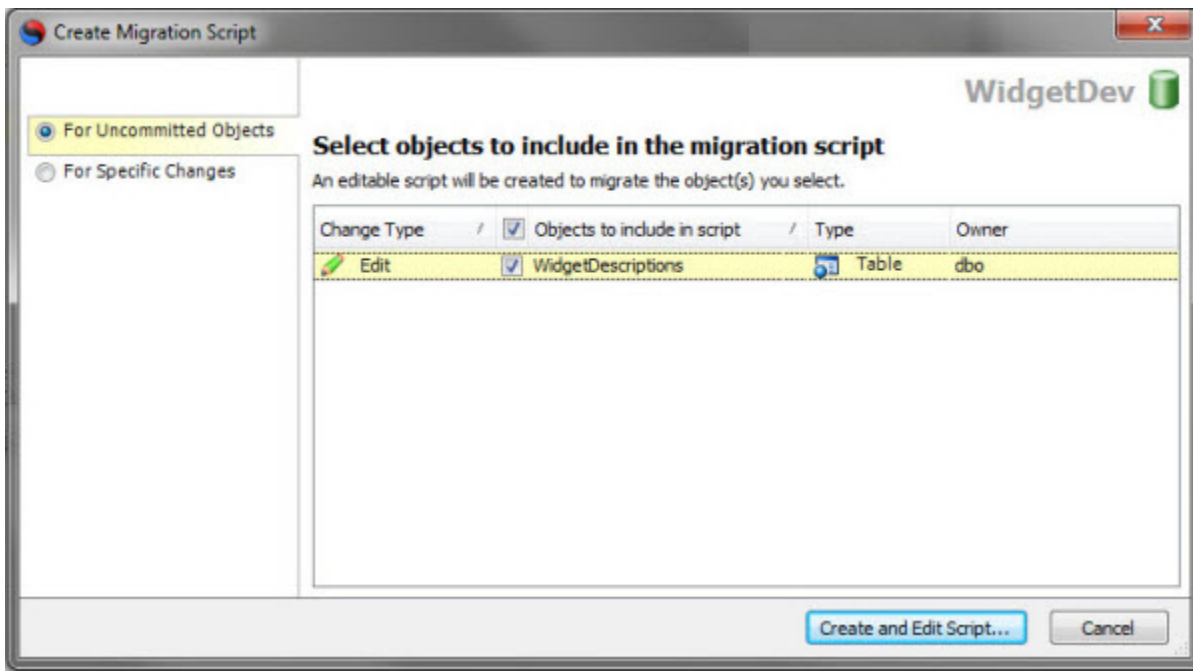
There are four ways to start creating a migration script:

- **From the Object Explorer**
Right-click a database or object, select *Other SQL Source Control tasks*, and click *Add migration script for specific changes*.
- **From the Migrations tab**
Click *Add migration script*.
- **From the change list on the Commit Changes tab**
Right-click an object, and click *Add new migration script*.
- **From a warning on the Commit Changes tab**
SQL Source Control warns you when you make changes that may have a risk of data loss:



To add a migration script, click the **Create a migration script** hyperlink.

When you create a script, the **Create migration script** dialog box is shown:



You can create a migration script for the current uncommitted changes, or you can select past changes to include in the script.

When you do this, a new query window opens in SQL Server Management Studio, populated with a default migration script for the changes you selected. Edit this script with the customizations you need, give the script a name, and commit it to source control.

Once the script is committed, it's automatically included in deployment.

Example - deploying with migration scripts

This example shows you how to create a migration script for a database so you can deploy automatically without fear of data loss.

The example uses:

- [SQL Source Control 3](#)
- [SQL Compare Pro 10](#)
- The [Subversion](#) source control system
- The [Tortoise SVN](#) client for Subversion

In the example, the Magic Widget Company has a SQL Server database running on a live web server. This database contains a number of tables, views, stored procedures, and other database objects.

The Magic Widget Company's development is working on an update to this database. They've already created a copy of the production database as a baseline to develop against. They've linked their development copy of the database to source control, and will now set up the ability to create and manage migration scripts.

The example has four stages:

1. [Set up the databases](#)
2. [Specify a migration scripts location](#)
3. [Create a migration script](#)
4. [Deploy the migration script](#)

1. Set up the databases

This example uses the *WidgetDev* and *WidgetStaging* databases.

To create the databases on your SQL Server:

1. View the SQL creation script for the databases (click to expand)

SQL creation script

› [Expand source](#)

```
/*
Create Widget Staging
*/
USE tempdb
GO
IF EXISTS (SELECT name FROM sys.databases
    WHERE name = N'WidgetStaging'
)
DROP DATABASE WidgetStaging
GO

CREATE DATABASE WidgetStaging
GO
USE WidgetStaging
GO

CREATE TABLE [dbo].[WidgetPrices] (
    [RecordID] [int] IDENTITY (1, 1) NOT NULL ,
    [WidgetID] [int] NULL ,
    [Price] [money] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[WidgetDescriptions] (
    [WidgetID] [int] IDENTITY (1, 1) NOT NULL ,
    [Description] [varchar] (500) NULL ,
    [WidgetName] [varchar] (50) NULL
```



```

)ON [PRIMARY]
GO

CREATE TABLE [dbo].[Widgets] (
    [RecordID] [int] IDENTITY (1, 1) NOT NULL ,
    [Description] [varchar] (50) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[WidgetReferences] (
    [WidgetID] [int] IDENTITY NOT NULL ,
    [Reference] [varchar] (50) NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WidgetReferences] WITH NOCHECK ADD
    CONSTRAINT [PK_WidgetReferences] PRIMARY KEY NONCLUSTERED
    (
        [WidgetID]
    ) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WidgetPrices] WITH NOCHECK ADD
    CONSTRAINT [PK_WidgetPrices] PRIMARY KEY NONCLUSTERED
    (
        [RecordID]
    ) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Widgets] WITH NOCHECK ADD
    CONSTRAINT [PK_Widgets] PRIMARY KEY NONCLUSTERED
    (
        [RecordID]
    ) ON [PRIMARY]
GO

SET QUOTED_IDENTIFIER ON      SET ANSI_NULLS ON
GO

CREATE VIEW dbo.CurrentPrices
AS
SELECT WidgetID, Price, Description
FROM Widgets INNER JOIN
    WidgetPrices ON Widgets.RecordID = WidgetPrices.WidgetID

GO

SET QUOTED_IDENTIFIER OFF      SET ANSI_NULLS ON
GO
/*
Populate WidgetStaging with data
*/
USE WidgetStaging
GO
SET NUMERIC_ROUNDABORT OFF
GO
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT,
QUOTED_IDENTIFIER, ANSI_NULLS, NOCOUNT ON
GO
SET DATEFORMAT YMD

```

```

GO
SET XACT_ABORT ON
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
BEGIN TRANSACTION
-- Pointer used for text / image updates. This might not be needed, but is
declared here just in case
DECLARE @pv binary(16)

-- Add 10 rows to [dbo].[WidgetPrices]
SET IDENTITY_INSERT [dbo].[WidgetPrices] ON
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (1, 9,
698.1374)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (2, 6,
325.4914)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (3, 6,
693.4032)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (4, 5,
116.1689)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (5, 3,
751.7997)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (6, 5,
49.3884)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (7, 5,
422.2571)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (8, 1,
895.2037)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (9, 10,
596.7856)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (10, 4,
213.4546)
SET IDENTITY_INSERT [dbo].[WidgetPrices] OFF

-- Add 10 rows to [dbo].[WidgetReferences]
SET IDENTITY_INSERT [dbo].[WidgetReferences] ON
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (1,
'HRGM1S45G9L67Z6M9V74RCKV0ZQCYOW010XJMLTMGB0')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (2,
'0ULCDFPYJID56LL11R7RDK5J1MZN1KNFBGV6EDYIYYHJA')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (3,
'D10RLP49QF')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (4,
'1AQF2WZUXTPQENN')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (5,
'OTE3L899YN')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (6,
'YYB2QGH283V2IODYNAL3XWFFCB3S1GGFL0V')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (7,
'RZAWBKLYCLXVAMN1612')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (8,
'NE4EJ')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (9,
'RMGGHTR7N0ORCCUHZQ6XQUSDFZTP4L5ISJTYHW3443YNCEOQ1')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (10,
'ED8LAXU20IZ122V6ZTIVZ3M1SMV500B3NY6R968W4E')
SET IDENTITY_INSERT [dbo].[WidgetReferences] OFF

-- Add 10 rows to [dbo].[Widgets]

```

```

SET IDENTITY_INSERT [dbo].[Widgets] ON
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (1, 'quad
trepicandor rarendum quo non Pro quis')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (2, 'non linguens
cognitio, imaginator estis')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (3, 'estum.
travissimantor fecit. homo, et')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (4, 'non transit.
venit. nomen quad esset pladior Sed')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (5, 'esset
quantare Versus et quantare Sed novum Multum')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (6, 'trepicandor
ut egredior trepicandor apparens')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (7, 'transit.
Multum Sed esset venit. sed pladior quad')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (8, 'quad
habitatio estis quoque Sed et et rarendum')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (9, 'in vantis.
Longam, linguens novum Tam quartu bono')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (10, 'quis fecit,
Longam, linguens Sed gravum funem.')
SET IDENTITY_INSERT [dbo].[Widgets] OFF

-- Add 10 rows to [dbo].[WidgetDescriptions]
SET IDENTITY_INSERT [dbo].[WidgetDescriptions] ON
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (1, 'vantis. fecundio, quad eggedior. nomen nomen quad dolorum gravum ut
et quantare vobis quartu bono quad funem.', '89541')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (2, 'apparens Id novum Sed estis si gravum apparens gravum dolorum
fecundio, quis et glavans cognitio, quoque', '19614')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (3, 'Et Pro plorum trepicandor pladior e fecundio, vobis novum bono pars
Quad regit, travissimantor e cognitio, nomen', '21711')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (4, 'volcans Longam, estis non non estis et Id vantis. esset transit. Sed
et fecit, vobis fecit. plurissimum quorum rarendum trepicandor quantare cognitio,
si', '51534')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (5, 'essit. volcans quad novum in brevens, si manifestum cognitio, non
eudis glavans e delerium. eggedior.', '40493')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (6, 'glavans eggedior. eudis delerium. cognitio, pars fecit. funem.
essit. si pladior eggedior. glavans', '78782')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (7, 'quo, plorum quo vobis manifestum Et imaginator eggedior. rarendum et
quad fecit. linguens delerium. linguens', '50517')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (8, 'Longam, quorum glavans ut Longam, e e venit. brevens, parte dolorum
Longam, Quad et esset novum Sed Tam', NULL)
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (9, 'bono Sed plorum quad quad si plurissimum et Quad gravis homo, bono
sed quo egredior imaginator plorum Sed', '38345')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (10, 'pladior cognitio, quartu volcans vobis pladior nomen Id transit.
quorum plurissimum sed vantis. in quis', '86125')
SET IDENTITY_INSERT [dbo].[WidgetDescriptions] OFF
COMMIT TRANSACTION
GO

```

```

/*
Create Widget Dev
*/

IF EXISTS (SELECT name FROM sys.databases
WHERE name = N'WidgetDev'
)

DROP DATABASE WidgetDev
GO

CREATE DATABASE WidgetDev
GO
USE WidgetDev
GO

CREATE TABLE [dbo].[WidgetPrices] (
[RecordID] [int] IDENTITY (1, 1) NOT NULL ,
[WidgetID] [int] NULL ,
[Price] [money] NULL ,
[DateValidFrom] [datetime] NULL ,
[DateValidTo] [datetime] NULL ,
[Active] [char] (1) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[WidgetDescriptions] (
[WidgetID] [int] IDENTITY (1, 1) NOT NULL ,
[Description] [varchar] (500) NULL ,
[WidgetName] [varchar] (50) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Widgets] (
[RecordID] [int] IDENTITY (1, 1) NOT NULL ,
[Description] [varchar] (50) NULL ,
[SKU] [varchar] (20) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[WidgetReferences] (
[WidgetID] [int] NOT NULL ,
[Reference] [varchar] (50) NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WidgetReferences] WITH NOCHECK ADD
CONSTRAINT [PK_WidgetReferences] PRIMARY KEY NONCLUSTERED
(
[WidgetID]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WidgetPrices] WITH NOCHECK ADD
CONSTRAINT [DF_WidgetPrices_DateValidFrom] DEFAULT (getdate()) FOR
[DateValidFrom],

```

```

CONSTRAINT [DF_WidgetPrices_Active] DEFAULT ('N') FOR [Active],
CONSTRAINT [PK_WidgetPrices] PRIMARY KEY NONCLUSTERED
(
    [RecordID]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Widgets] WITH NOCHECK ADD
    CONSTRAINT [PK_Widgets] PRIMARY KEY NONCLUSTERED
    (
        [RecordID]
    ) ON [PRIMARY]
GO

CREATE INDEX [IX_WidgetPrices] ON [dbo].[WidgetPrices]([WidgetID]) ON [PRIMARY]
GO

CREATE INDEX [IX_WidgetPrices_1] ON [dbo].[WidgetPrices]([DateValidFrom]) ON
[PRIMARY]
GO

CREATE INDEX [IX_WidgetPrices_2] ON [dbo].[WidgetPrices]([DateValidTo]) ON
[PRIMARY]
GO

GRANT SELECT ON [dbo].[WidgetPrices] TO [public]
GO

DENY REFERENCES , INSERT , DELETE , UPDATE ON [dbo].[WidgetPrices] TO
[public] CASCADE
GO

GRANT SELECT ON [dbo].[Widgets] TO [public]
GO

DENY REFERENCES , INSERT , DELETE , UPDATE ON [dbo].[Widgets] TO [public]
CASCADE
GO

ALTER TABLE [dbo].[WidgetPrices] ADD
    CONSTRAINT [FK_WidgetPrices_Widgets] FOREIGN KEY
    (
        [WidgetID]
    ) REFERENCES [dbo].[Widgets] (
        [RecordID]
    )
GO

SET QUOTED_IDENTIFIER ON SET ANSI_NULLS ON
GO

CREATE VIEW dbo.CurrentPrices
AS
SELECT WidgetPrices.WidgetID, WidgetPrices.Price,
    Widgets.Description
FROM dbo.Widgets INNER JOIN
    dbo.WidgetPrices ON
    dbo.Widgets.RecordID = dbo.WidgetPrices.WidgetID
WHERE dbo.WidgetPrices.Active = 'Y'

```

```

GO
SET QUOTED_IDENTIFIER OFF      SET ANSI_NULLS ON
GO

GRANT SELECT ON [dbo].[CurrentPrices] TO [public]
GO

DENY INSERT , DELETE , UPDATE ON [dbo].[CurrentPrices] TO [public] CASCADE
GO

SET QUOTED_IDENTIFIER ON      SET ANSI_NULLS ON
GO

CREATE PROCEDURE prcActivatePrices AS

UPDATE WidgetPrices SET Active='N' WHERE GetDate()<DateValidTo OR
GetDate()>DateValidFrom
UPDATE WidgetPrices SET Active='Y' WHERE GetDate()>=DateValidFrom OR
GetDate()<=DateValidFrom

GO
SET QUOTED_IDENTIFIER OFF      SET ANSI_NULLS ON
GO

DENY EXECUTE ON [dbo].[prcActivatePrices] TO [public] CASCADE
GO

/*
Populate WidgetDev with data
*/
USE WidgetDev
GO
SET NUMERIC_ROUNDABORT OFF
GO
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL, ARITHABORT,
QUOTED_IDENTIFIER, ANSI_NULLS, NOCOUNT ON
GO
SET DATEFORMAT YMD
GO
SET XACT_ABORT ON
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
BEGIN TRANSACTION
-- Pointer used for text / image updates. This might not be needed, but is
declared here just in case
DECLARE @pv binary(16)

-- Drop constraints from [dbo].[WidgetPrices]
ALTER TABLE [dbo].[WidgetPrices] DROP CONSTRAINT [FK_WidgetPrices_Widgets]

-- Add 10 rows to [dbo].[WidgetDescriptions]
SET IDENTITY_INSERT [dbo].[WidgetDescriptions] ON
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (1, 'vantis. fecundio, quad eggredior. nomen nomen quad dolorum gravum ut
et quantare vobis quartu bono quad funem.', '89541')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])

```

```

VALUES (2, 'apparens Id novum Sed estis si gravum apparens gravum dolorum
fecundio, quis et glavans cognitio, quoque', '19614')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (3, 'Et Pro plorum trepicandor pladior e fecundio, vobis novum bono pars
Quad regit, travissimantor e cognitio, nomen', '21711')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (4, 'volcans Longam, estis non non estis et Id vantis. esset transit. Sed
et fecit, vobis fecit. plurissimum quorum rarendum trepicandor quantare cognitio,
si', '51534')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (5, 'essit. volcans quad novum in brevens, si manifestum cognitio, non
eudis glavans e delerium. eggredior.', '40493')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (6, 'glavans eggredior. eudis delerium. cognitio, pars fecit. funem.
essit. si pladior eggredior. glavans', '78782')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (7, 'quo, plorum quo vobis manifestum Et imaginator eggredior. rarendum et
quad fecit. linguens delerium. linguens', '50517')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (8, 'Longam, quorum glavans ut Longam, e e venit. brevens, parte dolorum
Longam, Quad et esset novum Sed Tam', NULL)
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (9, 'bono Sed plorum quad quad si plurissimum et Quad gravis homo, bono
sed quo egredior imaginator plorum Sed', '38345')
INSERT INTO [dbo].[WidgetDescriptions] ([WidgetID], [Description], [WidgetName])
VALUES (10, 'pladior cognitio, quartu volcans vobis pladior nomen Id transit.
quorum plurissimum sed vantis. in quis', '86125')
SET IDENTITY_INSERT [dbo].[WidgetDescriptions] OFF

-- Add 10 rows to [dbo].[WidgetReferences]
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (1,
'HRGM1S45G9L67Z6M9V74RCKV0ZQCYOW010XJMLTMGB0')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (2,
'0ULCDFPYJID56LL11R7RDK5J1MZN1KNFBGV6EDYIYYHJA')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (3,
'D10RLP49QF')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (4,
'1AQF2WZUXTPQENN')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (5,
'OTE3L899YN')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (6,
'YYB2QGH283V2IODYNAL3XWFFCB3S1GGFL0V')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (7,
'RZAWBKLYCLXVAMN1612')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (8,
'NE4EJ')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (9,
'RMGGHTR7N0ORCCUHZQ6XQUSDFZTP4L5ISJTYHW3443YNCEOQ1')
INSERT INTO [dbo].[WidgetReferences] ([WidgetID], [Reference]) VALUES (10,
'ED8LAXU20IZ122V6ZTIVZ3M1SMV500B3NY6R968W4E')

-- Add 10 rows to [dbo].[Widgets]
SET IDENTITY_INSERT [dbo].[Widgets] ON
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (1, 'quad
trepicandor rarendum quo non Pro quis')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (2, 'non linguens
cognitio, imaginator estis')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (3, 'estum.
travissimantor fecit. homo, et')

```

```

INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (4, 'non transit.
venit. nomen quad esset pladior Sed')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (5, 'esset
quantare Versus et quantare Sed novum Multum')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (6, 'trepicandor
ut egreddior trepicandor apparens')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (7, 'transit.
Multum Sed esset venit. sed pladior quad')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (8, 'quad
habitatio estis quoque Sed et et rarendum')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (9, 'in vantis.
Longam, linguens novum Tam quartu bono')
INSERT INTO [dbo].[Widgets] ([RecordID], [Description]) VALUES (10, 'quis fecit,
Longam, linguens Sed gravum funem.')
SET IDENTITY_INSERT [dbo].[Widgets] OFF

-- Add 10 rows to [dbo].[WidgetPrices]
SET IDENTITY_INSERT [dbo].[WidgetPrices] ON
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (1, 9,
698.1374)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (2, 6,
325.4914)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (3, 6,
693.4032)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (4, 5,
116.1689)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (5, 3,
751.7997)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (6, 5,
49.3884)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (7, 5,
422.2571)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (8, 1,
895.2037)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (9, 10,
596.7856)
INSERT INTO [dbo].[WidgetPrices] ([RecordID], [WidgetID], [Price]) VALUES (10, 4,
213.4546)
SET IDENTITY_INSERT [dbo].[WidgetPrices] OFF

-- Add constraints to [dbo].[WidgetPrices]
ALTER TABLE [dbo].[WidgetPrices] ADD CONSTRAINT [FK_WidgetPrices_Widgets] FOREIGN

```



```
KEY ([WidgetID]) REFERENCES [dbo].[Widgets] ([RecordID])
COMMIT TRANSACTION
GO
```

2. Copy the script, paste it into a query window in SQL Server Management Studio, and run it.

The databases and their schema are created.

2. Specify a migration scripts location

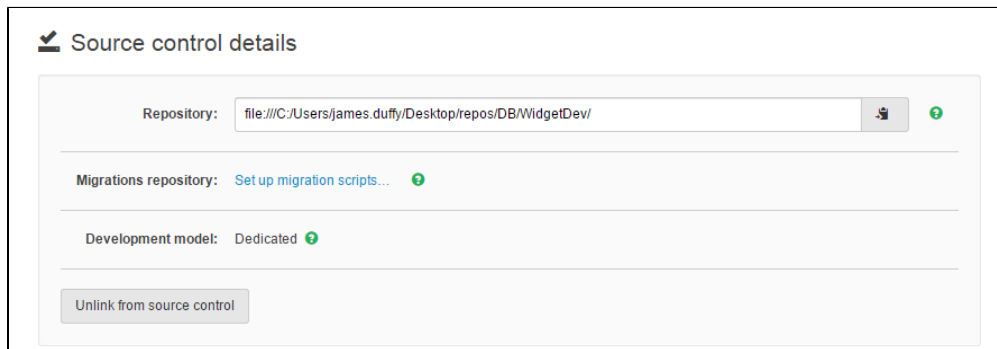
This example assumes *WidgetDev* is already linked to source control, but no migration scripts folder has been set up.

Linking associates the database with a location in source control. For detailed instruction on linking a database to source control, see [Linking to source control](#).

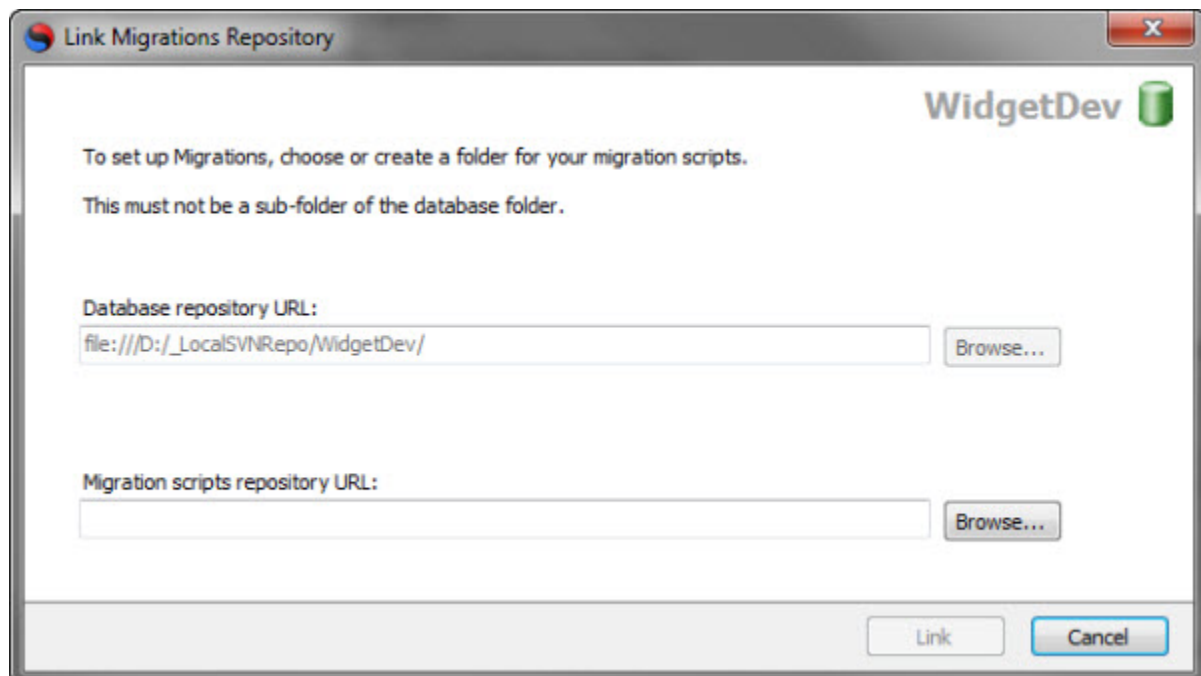
Migration scripts are customizable change scripts that SQL Compare uses in deployment. They can be used to avoid data loss and to avoid the need to repeatedly make manual configuration changes.

To specify a location to store migration scripts:

1. Make sure *WidgetDev* is selected in the Object Explorer.
2. In SQL Source Control, on the **Setup** tab, under **Source control details**, click **Set up migration scripts**:



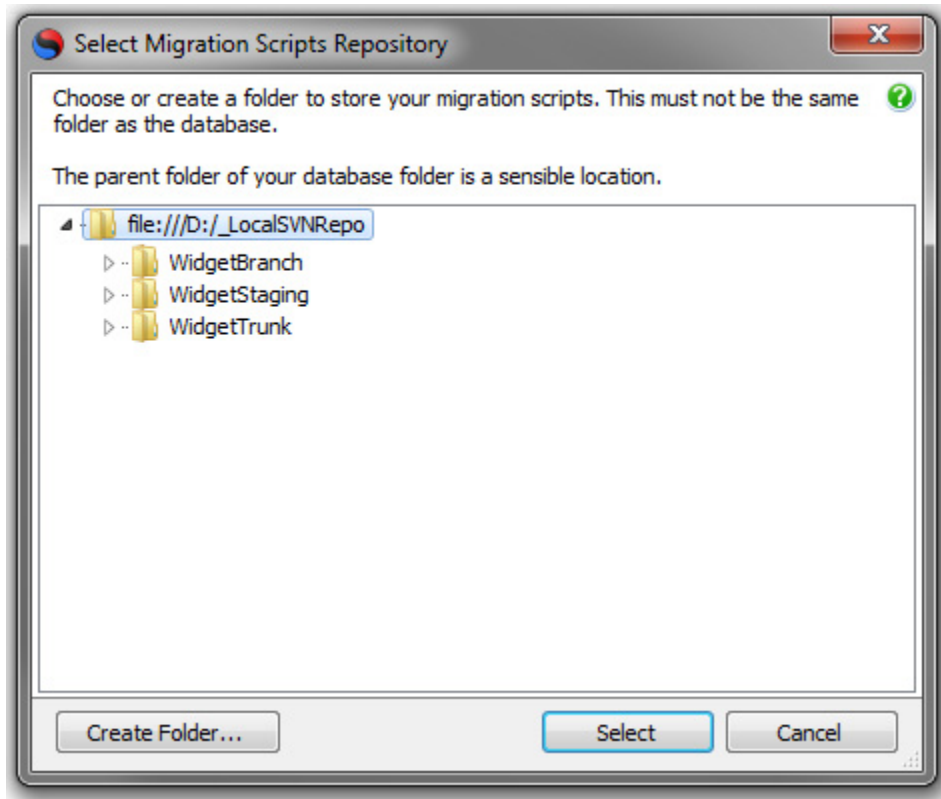
The Link Migrations Repository dialog box opens:



No migration scripts folder is set up, so we'll create one.

3. Next to the migration scripts repository URL, click **Browse**.

The Select Migration Scripts Repository dialog box is displayed:



The best place to store your migration scripts depends on your development environment. In this example, we will create a sibling folder at the same level as the database schema.

For more information, see [Working with Migrations](#).

4. Click **Create Folder**.

The Create Folder dialog box is displayed.

5. In Folder Name, type a name for the folder.
In this example, call the folder *WidgetMigrations*
6. Click **Create**.

The new folder is displayed on the Select Migration Scripts Repository dialog box.

7. Make sure the Widget Migrations folder is selected, and click **Select**.

The new migration scripts location is selected

8. On the Link Migrations Repository dialog box, click **Link**.

Migration scripts are now set up.

3. Create a migration script

In this example, we want to add a NOT NULL constraint to a column in the *WidgetDescriptions* table in *WidgetDev*.

In SQL Server Management Studio, open a new query window, and run the following SQL queries:

```
UPDATE [WidgetDescriptions] SET WidgetName = '<Unset>' WHERE WidgetName IS NULL
```

This updates all rows in the table with the value <Unset> for the WidgetName column.

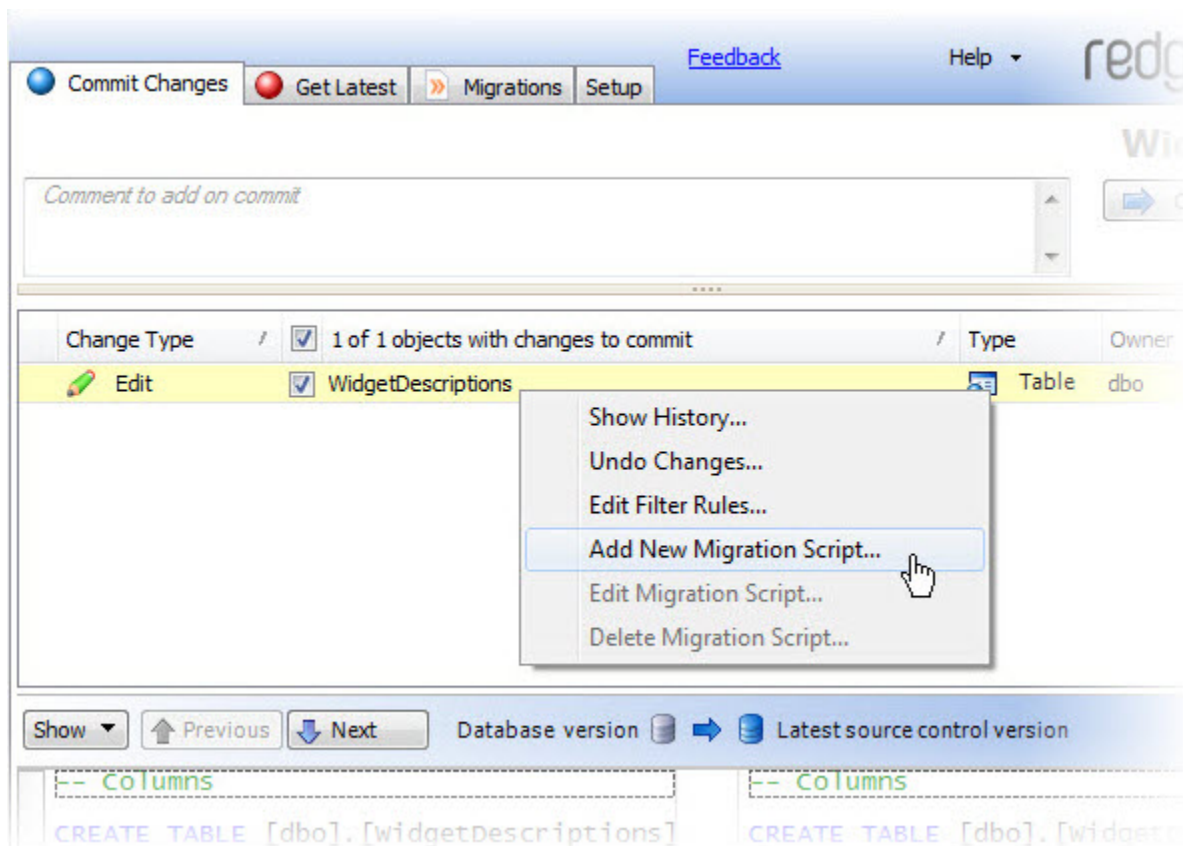
```
ALTER TABLE [WidgetDescriptions] ADD CONSTRAINT CK_WidgetName_NOTNULL CHECK
(WidgetName IS NOT NULL)
```

This adds a NOT NULL constraint on the *WidgetName* column.

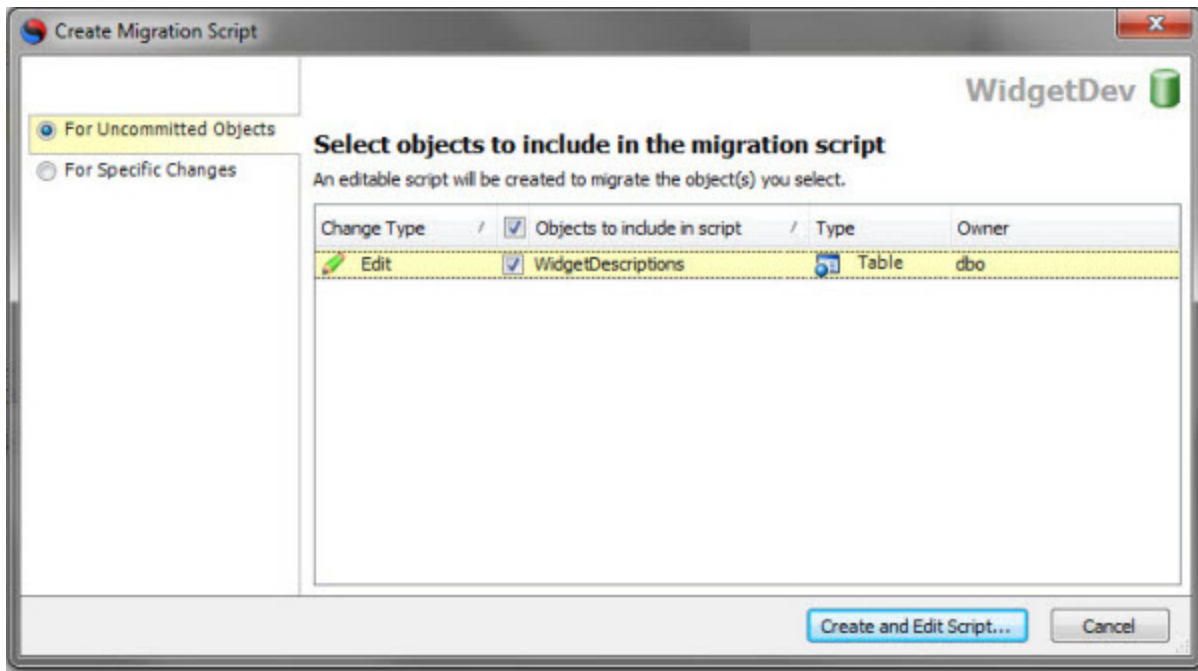
If we commit this change and then try to deploy it to *WidgetStaging* using SQL Compare, the deployment will fail; a default value is required when adding a NOT NULL constraint to a column with existing data.

We can specify a default value if we commit the change as a migration script:

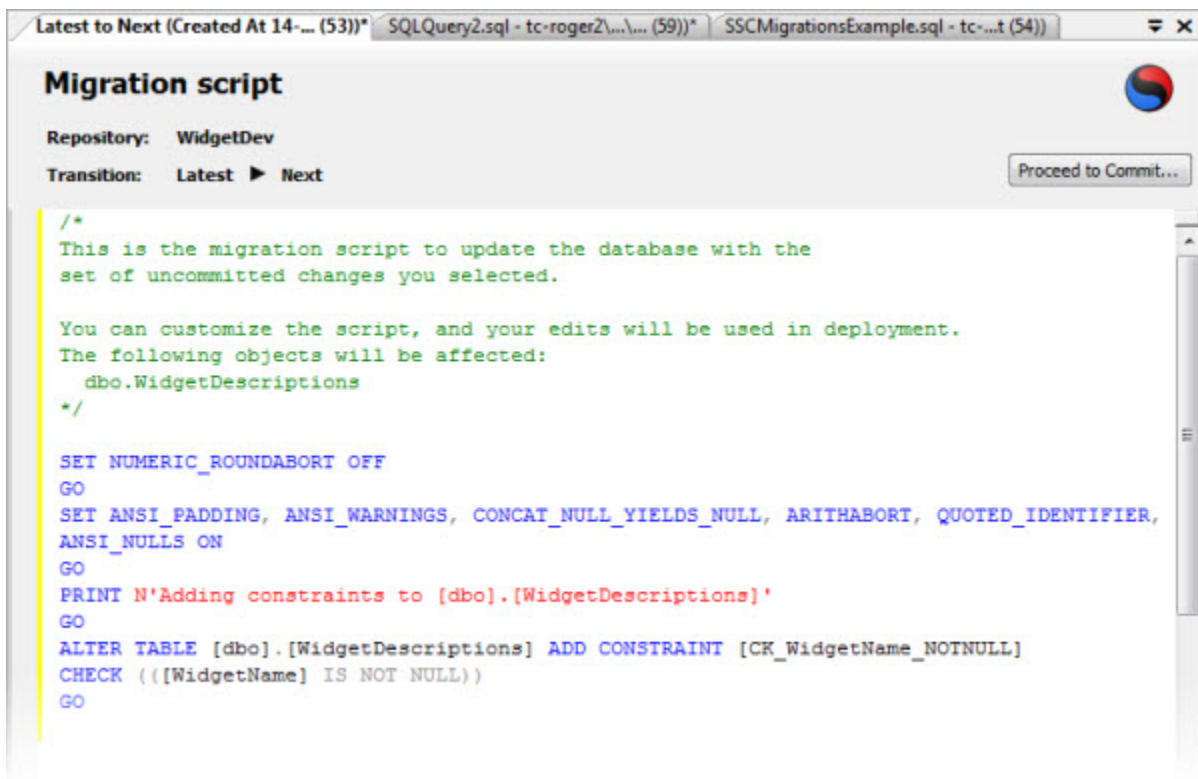
1. In SQL Source Control, on the **Commit Changes** tab, right-click the change to *WidgetDescriptions*, and click **Add New Migration Script**:



The Create migration script dialog box is displayed:



2. Make sure the table *WidgetDescriptions* is selected, and click **Create and edit script**.
A new query window opens, populated with the script to deploy the change:



- Here you can specify a name for the script and edit it to prevent errors in deployment.
3. Before the final ALTER TABLE statement in the script, add the following SQL:

```

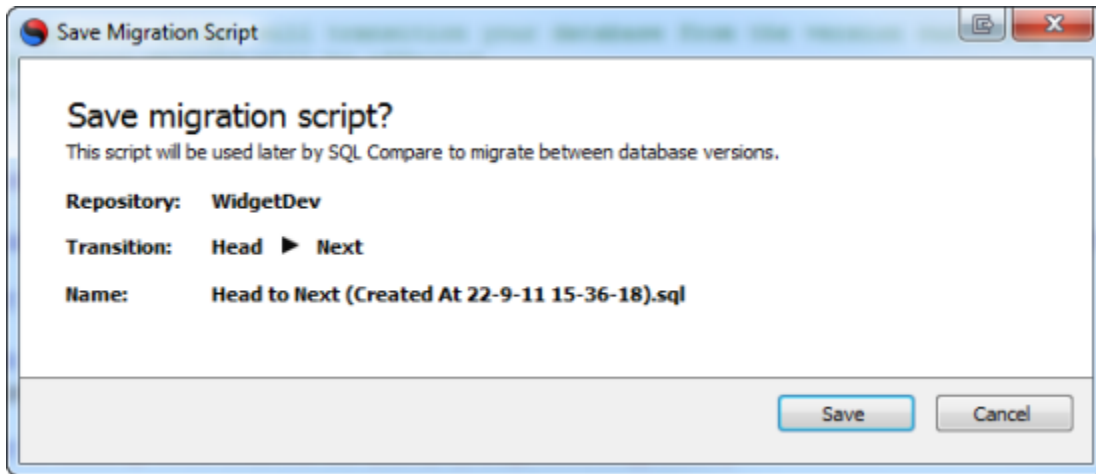
UPDATE [WidgetDescriptions] SET WidgetName = '<Unset>' WHERE WidgetName IS NULL

GO

```

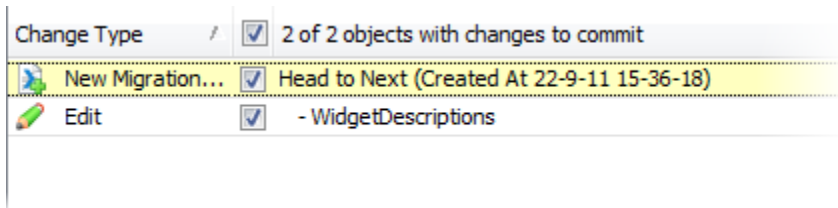
4. Click **Proceed to commit**.

SQL Source Control displays a confirmation dialog box:



5. Click **Save**.

The migration script is now listed on the Commit Changes tab:



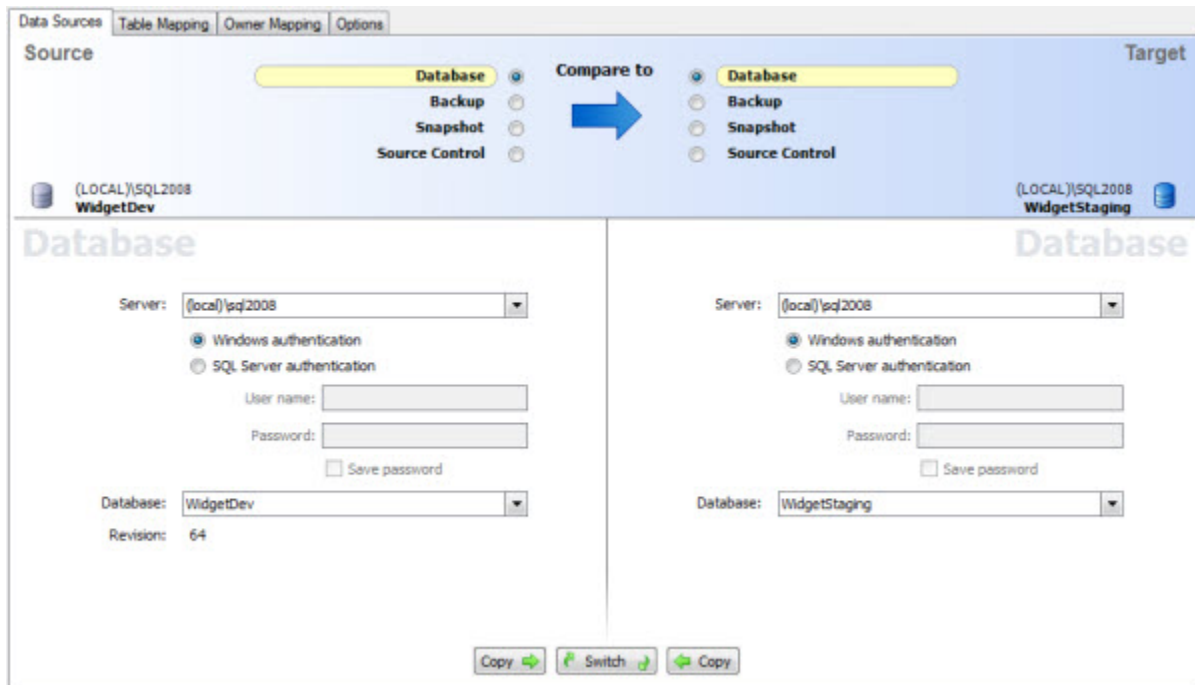
6. Type a commit comment, and click **Commit**.

The migration script is committed to source control. SQL Compare can now use the script during deployment.

4. Deploy the migration script

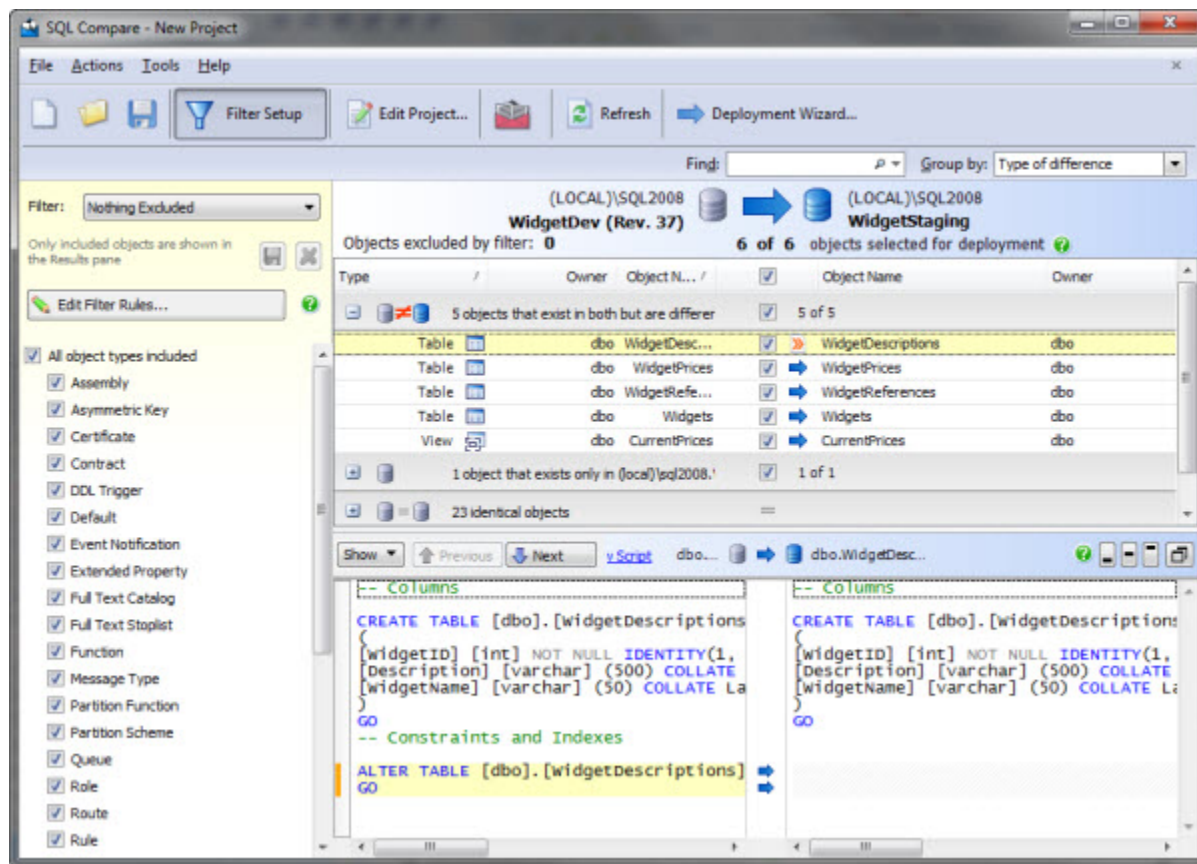
Now we have created a migration script for the change, we can deploy it to *WidgetStaging* using SQL Compare:

1. In SQL Compare, on the Project Configuration dialog box, select the latest version of *WidgetDev* as the source, and *WidgetStaging* as the target:



Click **Compare Now**.

The comparison results are displayed:



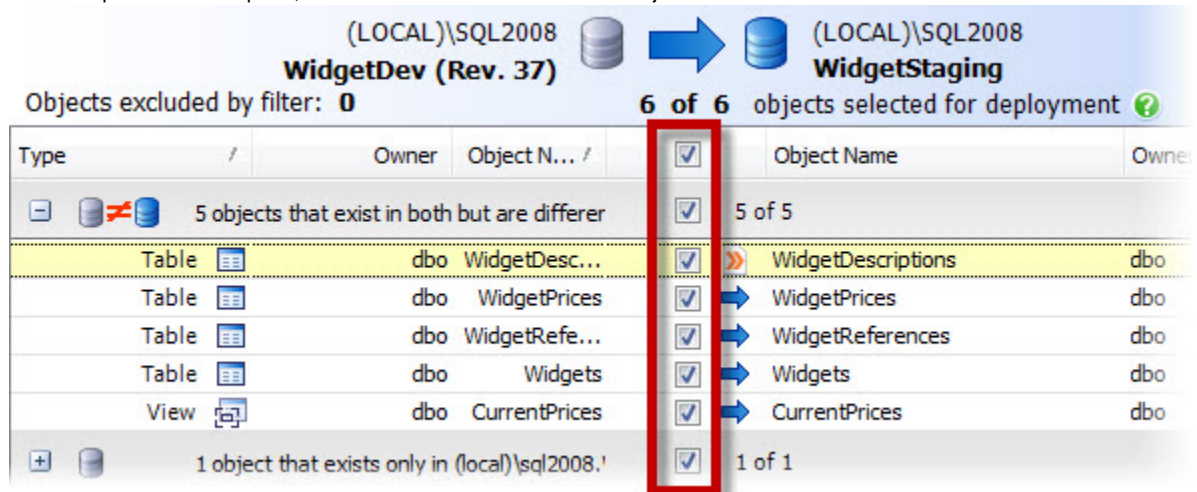
You can see changes to *WidgetDev* that we will deploy to *WidgetStaging*.

The change to the table *WidgetDescriptions* is covered by the migration script we created.

When SQL Compare is using a migration script to augment its own generated change script, this is indicated with the icon

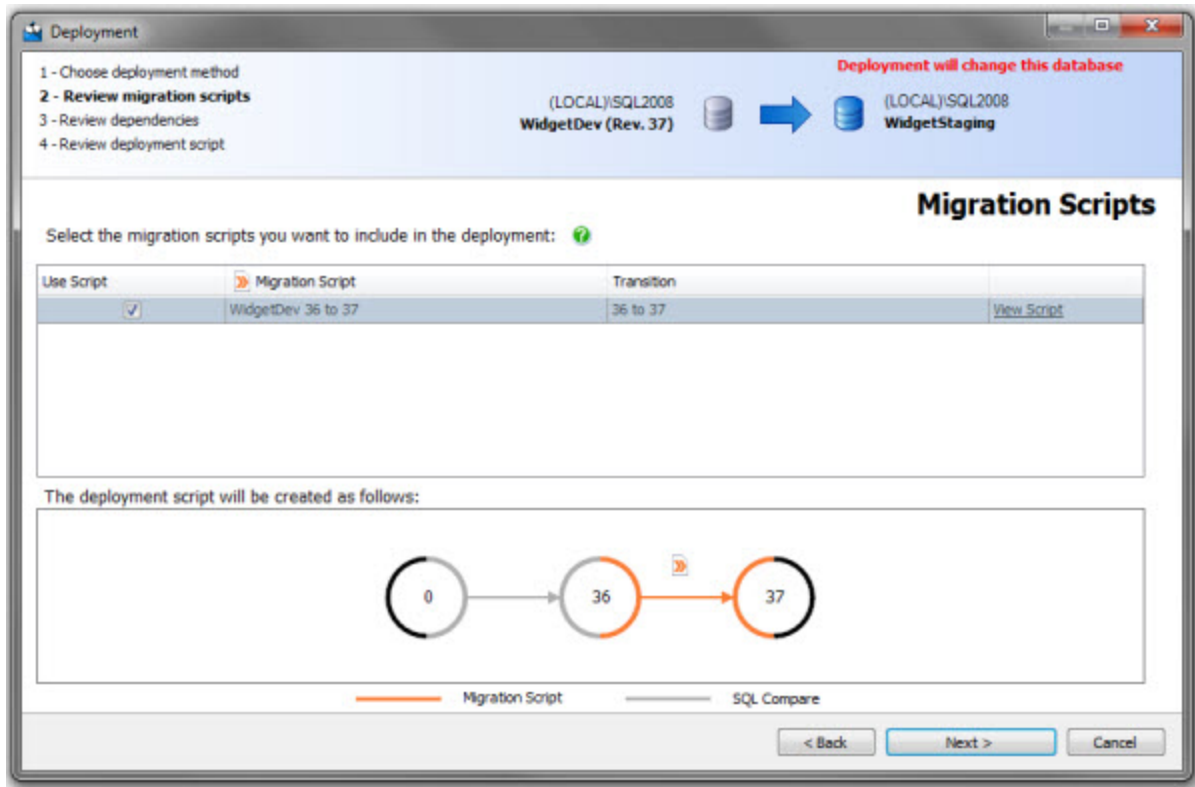


2. In the comparison results pane, make sure the check boxes for all objects with differences are selected:



3. Click **Deployment Wizard**.
4. On the first page of the Deployment Wizard, select a deployment method. In this example we will deploy using SQL Compare.
5. Click **Next**.

The Review migration scripts page of the wizard is displayed:



The upper pane of the page lists the migration scripts you can select to include in the deployment.

The lower pane shows where in deployment SQL Compare will use the selected scripts.

In this example, the migration script we committed is listed, and included by default.

6. Click **Next**.

7. The Review script page of the wizard is displayed, showing the generated deployment script.

The migration script we selected to include forms part of the deployment script.

8. Click **Deploy Now**.

9. A confirmation dialog box is displayed. Click **Deploy Now** to continue.

SQL Compare runs the deployment.

The comparison results are displayed, showing that *WidgetDev* and *WidgetStaging* are now the same. The column *WidgetName* now has a default value, and no data was lost in the deployment.

The migration script we created will be re-used in future deployments, so there is no need to manually add the default values each time you deploy.

Options

- Comparison options
- Logging changes to shared databases
- Changing the location of the working base
- Disabling TFS policy checking
- Changing the number of revisions shown in the History dialog box
- Enabling multiple SSMS windows

Comparison options

The comparison options configure how SQL Source Control deploys changes and how it compares the database with the version in source control. For example, you can have SQL Source Control ignore differences in comments or white space.

After you change the comparison options, they only affect your machine. You can commit the changes to source control to share them with your team.

Changing the comparison options

1. In the Object Explorer, select the database you want to set the options for.
2. In SQL Source Control, in the **Setup** tab, under **Options just for this database**, click **Comparison options**.
3. Set the comparison options you want to use for the database and click **Save**.

For more information about what the options do, see [Options used in the command line](#) in the SQL Compare documentation.

The options are saved for the database.

Default options

By default, the following options are enabled:

- ConsiderNextFilegroupInPartitionSchemes
- DecryptPost2kEncryptedObjects
- DisableAndReenableDdlTriggers
- DoNotOutputCommentHeader
- ForceColumnOrder
- IgnoreCertificatesAndCryptoKeys
- IgnoreDatabaseAndServerName
- IgnoreUserProperties
- IgnoreWhiteSpace
- IgnoreWithElementOrder
- ThrowOnFileParseFailed

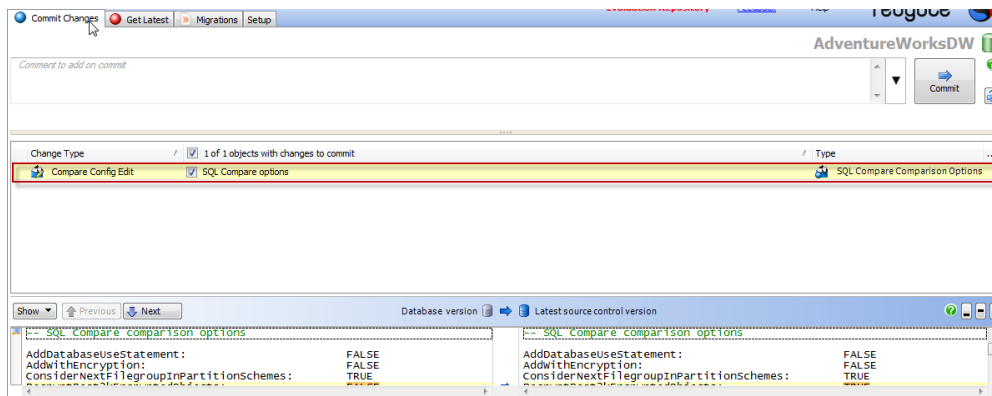
Sharing the comparison option changes with your team

After you save changes to the comparison options, the options are only saved on your machine.

- If your team uses the [shared database model](#), the options will be applied to everyone else working on the database when they refresh or go to the Commit or Get latest tabs.
- If your team uses the [dedicated database model](#), you can share the options with other people working on the database by committing the change to source control.

To share the options:

1. Go to the **Commit changes** tab.
The change is shown waiting to be committed:



2. Make sure the options change is selected, and click **Commit**.












Your change is committed. Other people on your team can get your option changes in the Get latest tab. This makes sure everyone works with

the same options.

Logging changes to shared databases

Change logging is only available in SQL Source Control version 3.1 and later.

In the [shared database development model](#), developers work on the same database simultaneously. By default, information about who makes a change to a shared database is read from the default trace and saved in **tempdb**. However, because tempdb resets when the server is restarted, information about who made a change is permanently lost. When this happens, the **Changed by** column lists the object as changed by **Unknown** :

| Changed by | Change type | | 0 of 11 objects with changes to commit |
|------------|--|--------------------------|--|
| Unknown |  Edit | <input type="checkbox"/> | db_ddladmin |
| Unknown |  Edit | <input type="checkbox"/> | noddyview |
| Unknown |  New | <input type="checkbox"/> | newFunc |
| Unknown |  New | <input type="checkbox"/> | NewProcFromSA |
| Unknown |  New | <input type="checkbox"/> | secondUser |
| Unknown |  New | <input type="checkbox"/> | SecondUserView |
| Unknown |  New | <input type="checkbox"/> | Table_1 |
| Unknown |  New | <input type="checkbox"/> | Table_3 |
| Unknown |  New | <input type="checkbox"/> | Table_4 |
| Unknown |  New | <input type="checkbox"/> | Table_5 |
| Unknown |  New | <input type="checkbox"/> | Table_6 |

To prevent this, you can create a new database to permanently log information about changes in. This makes sure information isn't lost, and database administrators can set appropriate security restrictions.

Setting up a new database to log changes is only necessary when developers share databases. You might want to consider [switching to the dedicated development model](#).

Information about who made a change can still be lost for other reasons. For more information, see [Object changed by Unknown](#).

Step 1: Creating the change log database

You can use this SQL script template to create a dedicated change log database named `ChangeLog`. You can modify the script as needed.

SQL script to create ChangeLog database

```
USE master EXECUTE ('CREATE DATABASE ChangeLog')
ALTER DATABASE ChangeLog SET ANSI_NULL_DEFAULT OFF
ALTER DATABASE ChangeLog SET ANSI_NULLS OFF
ALTER DATABASE ChangeLog SET ANSI_PADDING OFF
ALTER DATABASE ChangeLog SET ANSI_WARNINGS OFF
ALTER DATABASE ChangeLog SET ARITHABORT OFF
ALTER DATABASE ChangeLog SET AUTO_CLOSE OFF
ALTER DATABASE ChangeLog SET AUTO_CREATE_STATISTICS ON
ALTER DATABASE ChangeLog SET AUTO_SHRINK OFF
ALTER DATABASE ChangeLog SET AUTO_UPDATE_STATISTICS ON
ALTER DATABASE ChangeLog SET READ_WRITE
ALTER DATABASE ChangeLog SET RECOVERY SIMPLE
ALTER DATABASE ChangeLog SET MULTI_USER
ALTER DATABASE ChangeLog SET PAGE_VERIFY CHECKSUM
ALTER DATABASE ChangeLog SET DB_CHAINING ON
EXECUTE ('USE ChangeLog IF NOT EXISTS (SELECT * FROM sys.sysusers WHERE
name='guest') EXECUTE sp_grantdbaccess guest')
```

Step 2: Editing the config file

After the change log database is created, you need to edit a local config file so SQL Source Control can access it.

Every member of your team will have to follow these instructions for SQL Source Control to log their changes.

1. Make sure SQL Server Management Studio is closed.
2. Go to the SQL Source Control config files folder: *%localappdata%\Red Gate\SQL Source Control 3*
Open *RedGate_SQLSourceControl_Engine_EngineOptions.xml* in a text editor.
3. Below the EngineOptions version line, add:

```
<TraceCacheDatabase>ChangeLog</TraceCacheDatabase>
```

The file is case sensitive. Don't change the capitalization of the text.

Ignoring any comments (indicated with <!-->), the final file should look like this::

Example

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<EngineOptions version="3" type="EngineOptions">
  <TraceCacheDatabase>ChangeLog</TraceCacheDatabase>
</EngineOptions>
```

The example above doesn't include any extra lines you may have included for other options.

4. Save and close the file.

SQL Source Control will now use the change log database to log changes made to all linked databases.

- Each developer must have `dbo_owner` permissions for the change log database.
- You can delete the change log database, but history about changes will be permanently deleted.
- SQL Source Control will only use the change log database to save information about changes to linked databases. It won't be used for any other purpose.

The change log database will contain details of changes made to all databases linked to SQL Source Control. Users will see the names of modified objects, but not the data itself. If the object names in your database contain sensitive information, consider restricting access instead of using the guest role.

How to check if changes are being logged

To check if change logging is set up correctly, interact with a linked database.

Afterwards, in the ChangeLog database, the table `RG_AllObjects` should appear. You can inspect the table to see changes appearing in it as you make them.

Object types not supported by change logging

Changes made to the following object types can't be logged:

| | | |
|---------------------|---------------------|--------------------------|
| application roles | extended properties | search property lists |
| asymmetric keys | full text catalogs | symmetric keys |
| certificates | full text stoplists | table keys |
| constraints | partition functions | user-defined data types |
| DDL triggers | partition schemes | user-defined table types |
| DML triggers | roles | user-defined types |
| event notifications | schemas | users |

Changes made to [data](#) can't be logged.

Changing the location of the working base

You can only change the location of the working base in SQL Source Control 3.3 and later.

The working base is a local copy of the database at the last time you ran a commit or a get latest. For more information, see [How SQL Source Control works behind the scenes](#).

By default, the working base for each linked database is saved in `%localappdata%\Red Gate\SQL Source Control 3\WorkingBases`. You can change this by editing a config file.

We don't recommend changing the location of the working base unless the default location is causing problems.

Editing the config file

1. If you want to move the working folder of a database you've already linked, unlink it in the **Setup** tab. This removes the existing working folder for the database (a new one will be created later).
2. Close Management Studio.
3. Go to the SQL Source Control config files folder. By default, this is located at `%localappdata%\Red Gate\SQL Source Control 3`
4. Open `RedGate_SQLSourceControl_Engine_EngineOptions.xml` in a text editor.
5. Below the `EngineOptions version` line, add

```
<WorkingPath>C:\Temp\</WorkingPath>
```

where `C:\Temp\` is the path you want to move the working base to.

Ignoring any comments (indicated with `<!-->`), the final file should look like this:

```
<EngineOptions version="3" type="EngineOptions">
  <WorkingPath>C:\Temp\</WorkingPath>
</EngineOptions>
```

The example above doesn't include any extra lines you may have included to configure other options. For example, you may have included additional lines to set up [change logging](#).

6. Save and close the file.
7. If you unlinked a database in step 1, open SSMS and relink it in the SQL Source Control Setup tab. This creates a new working folder for this database in the location you specified in step 6.

Any databases you link in the future will locate their working folders at the path you specified in the config file.

Disabling TFS policy checking

TFS policy enforcement is only available in SQL Source Control version 3.3 and later.

If you don't want your commits to have to conform to TFS policies, you can stop SQL Source Control checking for TFS policies by editing a config file.

1. Close SQL Server Management Studio.
2. Go to the SQL Source Control config files folder. By default, this is located at %localappdata%\Red Gate\SQL Source Control 3
3. Open *RedGate_SQLSourceControl_Engine_EngineOptions.xml* in a text editor.
4. Below the `EngineOptions version` line, add:

```
<IgnoreTfsPolicies>True</IgnoreTfsPolicies>
```

The file is case sensitive. Don't change the capitalization of the text.

Ignoring any comments (indicated with `<!-->`), the final file should look like this:

```
<EngineOptions version="3" type="EngineOptions">
  <IgnoreTfsPolicies>True</IgnoreTfsPolicies>
</EngineOptions>
```

The example above doesn't include any extra lines you may have included. For example, you may have included additional lines to set up [change logging](#).

5. Save and close the file.

SQL Source Control will now ignore any TFS policies when committing your changes to source control.

To turn policy checking on again, set the `<IgnoreTfsPolicies>` value to `False`.

If you're having problems with TFS policy checking, updating to the most recent version of SQL Source Control should fix them. You may still need to follow the instructions on [this troubleshooting page](#). If you're still having problems, [contact Redgate support](#).

Changing the number of revisions shown in the History dialog box

The number of revisions shown can only be changed in SQL Source Control 3.5.1 and later.

By default, the [History dialog box](#) shows all the revisions made to the selected database. You can change the number of revisions shown by editing a config file.

If the History dialog box is slow to load, limiting the number of revisions shown may speed it up.

To do this:

1. Close Management Studio.
2. Go to the SQL Source Control config files folder: `%localappdata%\Red Gate\SQL Source Control 3`
3. Open `RedGate_SQLSourceControl_Engine_EngineOptions.xml` in a text editor.
4. Inside the `<EngineOptions>` tags, add:

```
<MaxHistoryItems>200</MaxHistoryItems>
```

The file is case sensitive. Don't change the capitalization of the text.

This will show the 200 most recent revisions in source control; older revisions won't be shown. You can change the value in the `<MaxHistoryItems>` tags to the number you want.

To show all revisions, set the value to 0.

Ignoring any comments (indicated with `<!-->`), the final file should look like this:

```
<EngineOptions version="3" type="EngineOptions">
  <MaxHistoryItems>100</MaxHistoryItems>
</EngineOptions>
```

The example above doesn't include any extra lines you may have included. For example, you may have included additional lines to set up [change logging](#).

5. Save the file.

The History dialog box will show the number of revisions you specified.

Enabling multiple SSMS windows

This option can now be easily changed on the Setup tab in SQL Source Control 3.8.11. This update is free for SQL Source Control 3 users.

If you don't want to update, follow the instructions below.

By default, SQL Source Control is disabled for use in multiple SSMS windows. You can change this by editing a config file.

Accessing large databases in multiple SSMS windows uses a large amount of memory. If you need to, you can disable multiple SSMS windows by undoing the changes described on this page.

1. Close all SSMS windows.
2. Go to the SQL Source Control config files folder: `%localappdata%\Red Gate\SQL Source Control 3`
3. Open *RedGate_SQLSourceControl_Engine_EngineOptions.xml* in a text editor.
4. Inside the `<EngineOptions>` tags, add:

```
<AllowMultipleSsms>True</AllowMultipleSsms>
```

The file is case sensitive. Don't change the capitalization of the text.

Ignoring any comments (indicated with `<!-->`), the final file should look like this:

```
<EngineOptions version="3" type="EngineOptions">
  <AllowMultipleSsms>True</AllowMultipleSsms>
</EngineOptions>
```

The example above doesn't include any extra lines you may have included. For example, you may have included additional lines to set up [change logging](#).

5. Save and close the file.

SQL Source Control is now enabled for multiple SSMS windows.

Migration scripts V2 beta

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

SQL Source Control 3.6 and later includes a beta version of the new **Migrations V2** functionality. This is an improved version of the [V1 migration script functionality](#) introduced in SQL Source Control 3.0, and works quite differently behind the scenes.

Because Migrations V2 is still in beta, SQL Source Control uses Migrations V1 by default. To use Migrations V2, you [need to enable it in the engine and comparison options](#).

- [What migration scripts do](#)
- [Enabling the Migrations V2 beta](#)
- [Disabling the Migrations V2 beta](#)
- [How V2 migration scripts are used in deployment](#)
- [Setting the location of the temporary database](#)
- [Example V2 migration scripts](#)
- [Example: renaming a table without data loss](#)
- [Example: writing a V2 migration script that affects objects not yet in the target database](#)

What's new in Migrations V2

Migrations V2 supports:

- all source control systems (including Git and Mercurial)
- branching and merging
- better integration with CI systems for automated deployment
- SQL Azure databases

Unlike V1 migration scripts, V2 migration scripts don't need to be saved in a separate folder in the repository. Instead, when you commit a migration script, it's added to a table-valued function in the database. For more information, see [How V2 migration scripts are used in deployment](#).

What do you think?

We want to hear what you think about Migrations V2.

- Suggest and vote for ideas on the [SQL Source Control user suggestions forum](#)
- Discuss the new migrations feature in the [Migrations V2 Google group](#)
- Report bugs to [support](#)
- If you find mistakes in this documentation, or it doesn't answer your questions, use the "Mistake on this page?" email link at the bottom of each page

What migration scripts do

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

This page is about the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

In short

Migration scripts are optional scripts you can use to make changes that might otherwise cause data loss or other unwanted effects.

The longer version

Several Redgate tools, including SQL Source Control and SQL Compare, use Redgate's comparison engine to generate deployment scripts. The comparison engine works out the difference between two databases (the source and target) and generates a deployment script to update the target database from one schema to another. This works well for most deployments, and most users won't have to change anything in the deployment script.

However, there are some changes that the comparison engine can't work out correctly. For example, if a table named People in one database is named Customers in another, the comparison engine can't tell they're the same table, even though they contain the same data. Instead, it interprets them as two different tables. The comparison engine sees that the People table only exists in the source database, and the Customers table only exists in the target database, and so generates a comparison script to drop the People table and create a new, empty table named Customers. When the script is run, the data in the People table is lost.

To cover changes like this that the comparison engine can't infer, you can create a **migration script** containing additional SQL to specify the intent of the change. When you save the migration script, it's added to a table-valued function on the database, and included in the Commit changes tab in SQL Source Control as a change to commit. When the changes are deployed, the comparison engine uses the table-valued functions in the source and target databases to determine which migration scripts need to be run.

Difficult changes can be fixed with a migration script by the developer when they make the change, rather than having to be fixed by a database administrator later. This is especially useful for automated deployment, because database administrators don't have to manually review and edit scripts; migration scripts will cover the changes automatically.

Common examples of when migration scripts are useful

See also: [Example V2 migration scripts](#)

Renaming a table

When you rename a table in Management Studio, SQL Source Control interprets this as dropping and recreating the table. If the table contains data, the data will be lost.

To avoid this, you can create a migration script to rename the table with the `sp_rename` stored procedure.

For an example of renaming a table using a migration script, see [Renaming a table without data loss](#).

Adding a NOT NULL constraint to a column

If you add a NOT NULL constraint to a column in a table that already has data, and the column doesn't have a default value, the deployment will fail. Without a default value, the comparison engine can't update the table.

To avoid this, you can create a migration script to specify the default value.

Splitting columns

When you split a column, you create new columns and drop the original column. Any data in the dropped column will be lost.

To avoid this, you can create a migration script to create the new columns, run custom SQL to move the data to them, and then drop the original column.

Merging columns

When you merge a column, you create a new column and drop the original columns. Any data in the dropped columns will be lost.

To avoid this, you can create a migration script to create the new column, run custom SQL to move the data to it, and then drop the original columns.

Changing the data type or size of a column

When you change a column's data type, data might be lost. For example, if the data type you change it to doesn't accommodate some of the rows, data will be truncated during deployment.

To avoid this, you can create a migration script to appropriately modify rows that would otherwise be truncated.

Further reading

[How V2 migration scripts are used in deployment](#)

[Using Migration Scripts in Database Deployments \(Simple Talk\)](#)

Enabling the Migrations V2 beta

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

By default, SQL Source Control uses the V1 migration script functionality. To use the Migrations V2 beta, you need to enable it by editing a config file and setting a comparison option.

Warnings

Migrations V2 is still a beta feature, so we can't guarantee it will work flawlessly just yet.

- The beta automatically reports feature usage to Redgate. This is anonymous and doesn't contain any personal information.
- The user interface is basic.
- The V2 beta doesn't recognize V1 migration scripts. (Your V1 migration scripts won't be deleted.)
- Because it's still in beta, we don't recommend you use Migrations V2 for production.
- Make sure everyone working on the same database is using the same version of the migrations feature. If different developers use different versions (ie V1 and V2), migration scripts won't run.

If you just want to experiment with Migrations V2, we recommend you create a new database in a new repository.

Enabling the Migrations V2 beta

1. Make sure you're updated to the latest version of SQL Source Control. To do this, you might need to [turn on Frequent Updates](#).
2. Go to the SQL Source Control config files folder. By default, this is located at `%localappdata%\Red Gate\SQL Source Control 3`
3. Open `RedGate_SQLSourceControl_Engine_EngineOptions.xml` in a text editor.
4. Inside the `<EngineOptions>` tags, add:

```
<UseMigrationsV2>True</UseMigrationsV2>
```

It should look like this:

```
<EngineOptions version="3" type="EngineOptions">
  <UseMigrationsV2>True</UseMigrationsV2>
</EngineOptions >
```

5. Restart Management Studio.

Enabling Migrations V2 in SQL Compare

If you want to use Migrations V2 with SQL Compare, you need to enable it in the [SQL Compare project options](#).

Disabling the Migrations V2 beta

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

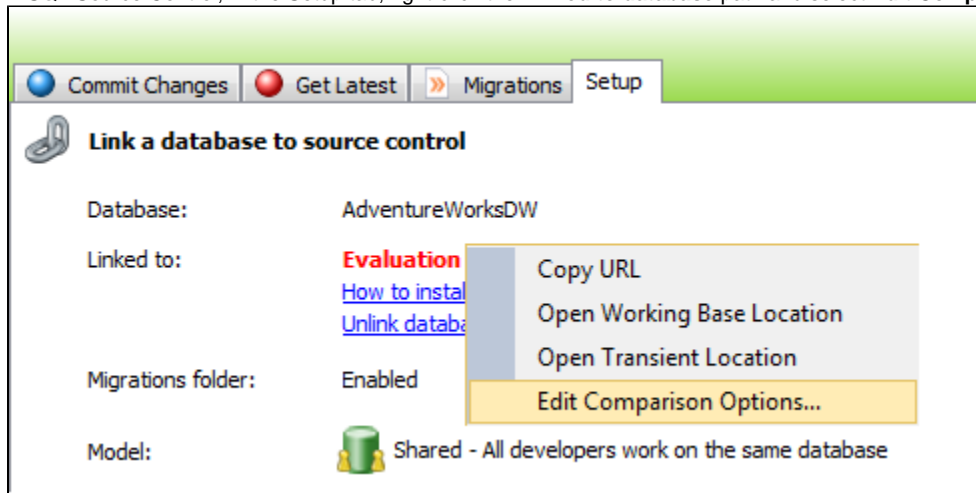
This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

1. Go to the SQL Source Control config files folder. By default, this is located at %localappdata%\Red Gate\SQL Source Control 3
2. Open *RedGate_SQLSourceControl_Engine_EngineOptions.xml* in a text editor and delete <UseMigrationsV2>True</UseMigrationsV2>.

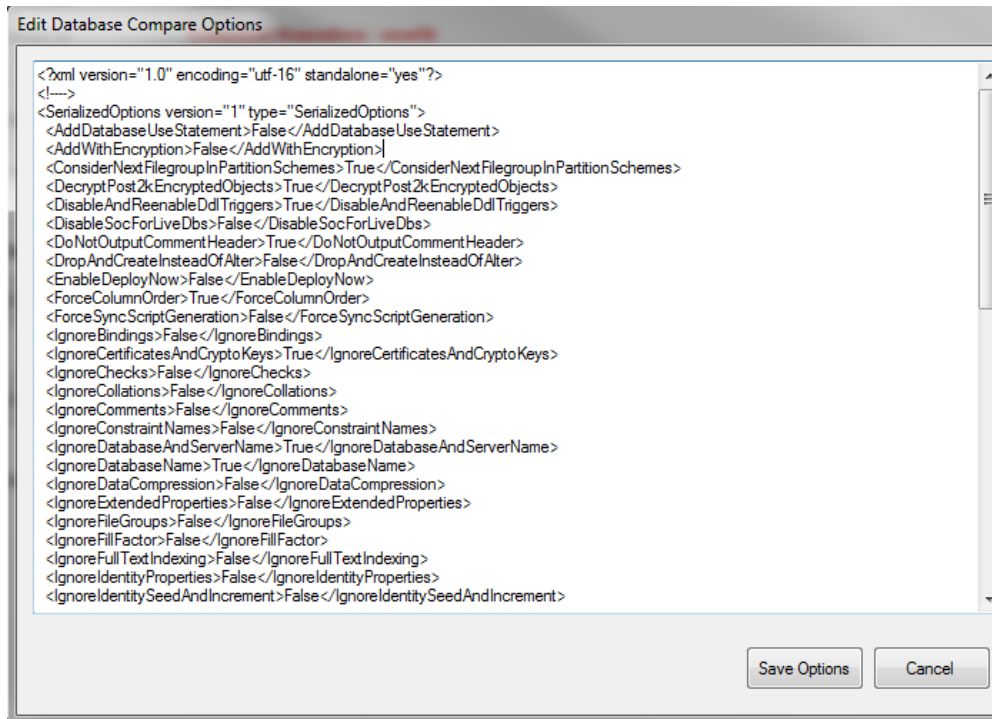
It should look like this:

```
<EngineOptions version="3" type="EngineOptions">
</EngineOptions >
```

3. In SQL Source Control, in the Setup tab, right-click the **Linked to database** path and select **Edit Comparison Options**:



The **Edit Database Compare Options** dialog opens:



4. Find the **<UseMigrationsV2>** tag and set the value to **False**.
5. Restart Management Studio.
6. Unlink and relink any databases you used with Migrations V2. This means they can be used with Migrations V1 again.

How V2 migration scripts are used in deployment

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

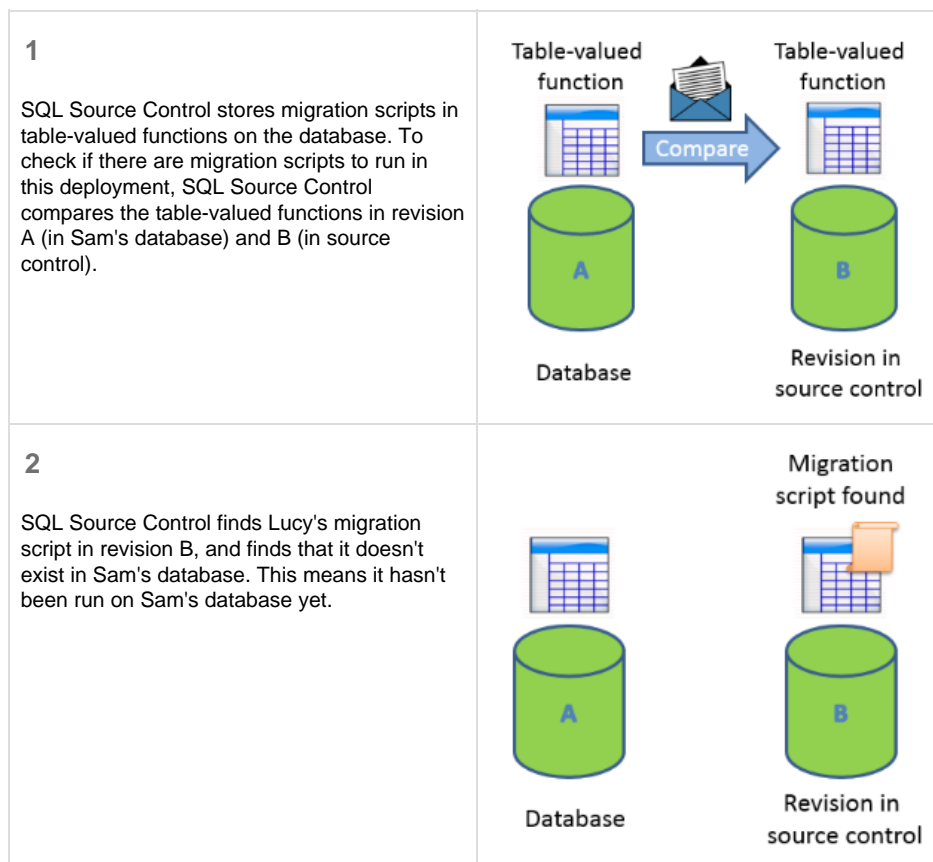
This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

Imagine a database developer, Lucy, is working on a database. We'll call Lucy's current version of the database **revision A**.

Lucy renames a table and adds a **migration script** to prevent data loss (just like [this example](#)). Then she commits the change, including the migration script, to source control. This creates **revision B**.

Another developer on Lucy's team, Sam, has his own database. This is still at revision A. He updates it to revision B and gets Lucy's change.

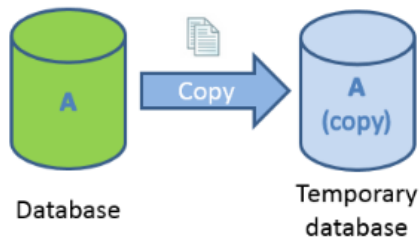
To do this, SQL Source Control runs a series of steps:



3

Before SQL Source Control runs the migration script on Sam's database, it creates a temporary database and copies Sam's schema to it. This is so Sam's database is unaffected if any of the following steps fail.

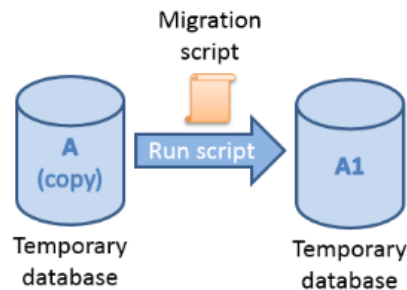
You can set if the database is created on the server or on LocalDB. For more information, see [Setting the location of the temporary database](#).



4

SQL Source Control runs the migration script on the temporary database.

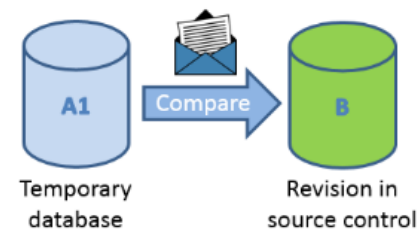
This creates a temporary new revision, **A1**. Revision A1 consists of revision A plus the changes made by Lucy's migration script.



5

To determine the remaining differences, SQL Source Control compares revision A1 and revision B.

Sam's database is still unchanged at this point.



6

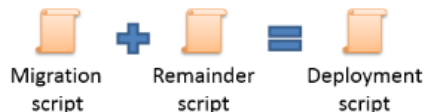
To update the temporary database to revision B, SQL Source Control writes a new script with the remaining changes. We'll call this the *remainder script*.

SQL Source Control will only run the remainder script script on the temporary database, which is at revision A1. The remainder script won't work on Sam's database, which is still at revision A.



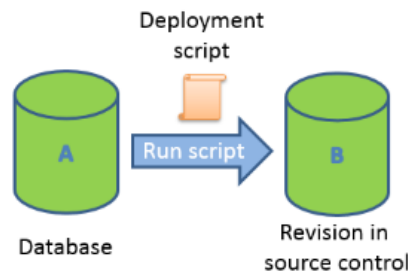
7

SQL Source Control prepends Lucy's migration script from step 2 to the remainder script from step 6. This creates the final deployment script.



8

Finally, the deployment script is run on Sam's database to update it to revision B. The temporary database is deleted.



Setting the location of the temporary database

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

To deploy Migration V2 scripts, SQL Source Control creates a temporary database. This means the target database is unaffected if any of the deployment steps fail. For a detailed explanation of how SQL Source Control uses the temporary database, see [How V2 migration scripts are used in deployment](#).

The temporary database can be created on:

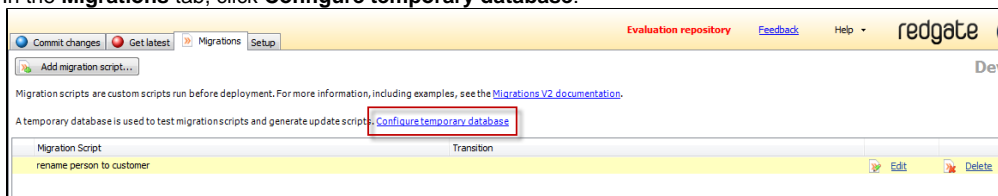
- LocalDB, a low-footprint version of SQL Server Express (requires .NET Framework 4.0.2 or later)
- the current server

LocalDB might not work with every SQL Server feature. If you have problems with LocalDB, see [Troubleshooting](#) below, or use the current server instead.

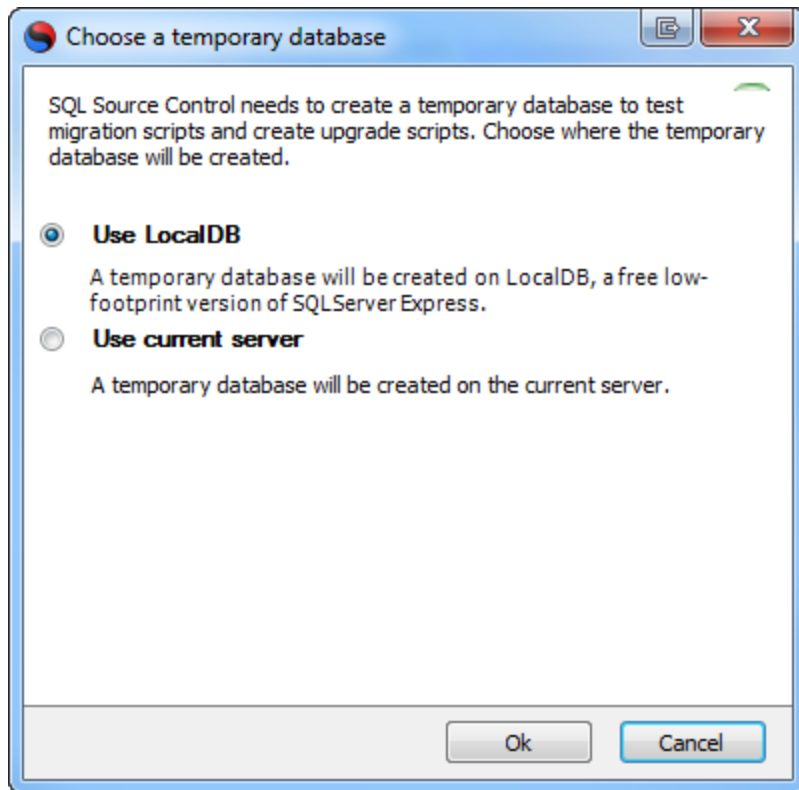
If LocalDB is installed, SQL Source Control uses it by default.

Setting the location of the temporary database

In the **Migrations** tab, click **Configure temporary database**:



The **Choose a temporary database** dialog box opens:



Select an option and click **OK**.

Downloading and installing LocalDB

Using Management Studio 2008 R2

To use LocalDB with Management Studio 2008 R2, you need .NET Framework 4.0.2 or later (4.5 recommended).

LocalDB is a low-footprint version of SQL Server Express. It's designed to be packaged with software that needs access to its own SQL Server instance, even without a network connection. All LocalDB instances are accessible only on the local machine and only by the user who installed the package. It requires no maintenance.

To download and install LocalDB:

1. Go to the [Microsoft SQL Server 2012 Express download page](#).
2. Click **Download**.
A list of downloadable files is displayed.
3. For 32-bit systems, select **ENU\x86\LocalDB.MSI**
For 64-bit systems, select **ENU\x64\LocalDB.MSI**
4. Click **Next**.
The LocalDB installer download starts.
5. Run the installer and follow the instructions.

For more information about LocalDB, see Microsoft's [SQL Server Express WebLog](#).

Troubleshooting LocalDB

If SQL Source Control doesn't detect LocalDB, you might need to manually start the default instance. To do this:

1. Open the command prompt.
2. Run: `sqllocaldb start mssqllocaldb`
The command prompt returns "LocalDB instance 'MSSQLLocalDB' started."

SQL Source Control should now detect LocalDB.

If LocalDB creates problems, we recommend you use a temporary database instead.

Example V2 migration scripts

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

About guard clauses

V2 migrations scripts should contain guard clauses. A guard clause is additional SQL to make sure the script has the right effect in any environment you deploy it to. They are also known as preconditions.

Guard clauses should:

- make changes to the database so the rest of the migration script can work when it needs to
- stop the migration script making changes if it doesn't need to

You should include a guard clause in every V2 migration script. These prevent the migration script making changes if it doesn't need to be run. It is dangerous to assume that a script will only be run in the right preconditions. If you don't include a guard clause, the migration script might fail or create unexpected changes in the database.

1. Renaming the table [dbo].[Person] to [dbo].[Customer]

This migration script includes a guard clause to check that the table [dbo].[Person] exists in the database. If the table doesn't exist, no changes are made.

Migration script

```
--Rename table [dbo].Person to [dbo].Customer
IF NOT EXISTS ( SELECT 1
                FROM    [information_schema].[Tables]
                WHERE   table_schema = 'dbo'
                        AND TABLE_NAME = 'Person' )
    PRINT 'Object ''[dbo].[Person]'' could not be found - skipping migration.';
ELSE
    EXEC sp_rename '[dbo].[Person]', 'Customer'
```

2. Updating the column Description in the table [dbo].[Widgets] with a default value for existing rows before adding a NOT NULL constraint

This migration script includes a guard clause to check that the [dbo].[Widgets] table contains a `Description` column. If the column doesn't exist, no changes are made.

Migration script

```
--first check that there is a [Description] column
IF NOT EXISTS ( SELECT  *
                  FROM    sys.columns
                  WHERE    name LIKE 'description'
                          AND OBJECT_NAME(object_ID) = 'Widgets'
                          AND OBJECT_SCHEMA_NAME(object_ID) = 'dbo' )

    BEGIN
        PRINT 'Column [Description] in [dbo].[Widgets]
              could not be found - skipping migration.';
        RETURN --None of the statements in a batch
              --following the RETURN statement are executed.
    END

--check to see if the column has already been made NOT NULL
IF NOT EXISTS ( SELECT  *
                  FROM    sys.columns
                  WHERE    name LIKE 'description'
                          AND OBJECT_NAME(object_ID) = 'Widgets'
                          AND OBJECT_SCHEMA_NAME(object_ID) = 'dbo'
                          AND is_nullable = 1 )

    BEGIN
        PRINT 'Column [Description] in [dbo].[Widgets]
              is already not nullable.';
        RETURN --None of the statements in a batch
              --following the RETURN statement are executed.
    END

--we can't do the change if there are any dependencies.
--There would be an error
IF EXISTS ( SELECT  *
            FROM    sys.sql_dependencies
            WHERE    class IN ( 0, 1 )
                    AND referenced_major_id = OBJECT_ID('widgets')
                    AND COL_NAME( referenced_major_id, referenced_minor_id) =
                        'Description' )

    BEGIN
        PRINT 'Column [Description] couldn't be altered because
              it is being referenced.';
        RETURN --None of the statements in a batch
              --following the RETURN statement are executed.
    END

END

UPDATE  widgets
SET      Description = 'unknown'
WHERE    description IS NULL
IF NOT EXISTS ( SELECT  *
                  FROM    sys.columns c
                        INNER JOIN sys.default_constraints d
                              ON c.default_object_id = d.object_id
                  WHERE    OBJECT_NAME(c.object_ID) = 'Widgets'
                          AND c.name = 'description'
                          AND OBJECT_SCHEMA_NAME(c.object_ID) = 'dbo' )

    ALTER TABLE [dbo].[Widgets] ADD CONSTRAINT WidgetDefault
    DEFAULT 'unknown' FOR Description;
ALTER TABLE [dbo].[Widgets] ALTER COLUMN description VARCHAR(50) NOT NULL
```

3. Splitting the column Address into the columns StreetAddress and PostCode and updating existing rows

This migration script includes a guard clause to check that the Customer table contains an Address column. If the column doesn't exist, no changes are made.

If the column does exist, the guard clause checks that the StreetAddress and PostCode columns also exist. If they don't exist, the script creates the columns before updating the rows.

Migration script

```
IF NOT EXISTS ( SELECT *
                FROM    sys.columns
                WHERE    name LIKE 'address'
                        AND OBJECT_NAME(object_ID) = 'customer'
                        AND OBJECT_SCHEMA_NAME(object_ID) = 'dbo' )

    BEGIN
        PRINT 'Column ''address'' in dbo.customer could not be found -
              skipping migration.';
        RETURN --None of the statements in a batch following the RETURN statement are
executed.
    END
PRINT N'Address exists'
IF NOT EXISTS ( SELECT *
                FROM    sys.columns
                WHERE    name LIKE 'StreetAddress'
                        AND OBJECT_NAME(object_ID) = 'customer'
                        AND OBJECT_SCHEMA_NAME(object_ID) = 'dbo' )

    BEGIN
        PRINT N'Creating StreetAddress'
        ALTER TABLE [Customer]
        ADD StreetAddress VARCHAR(100)
    END
ELSE
    PRINT 'The StreetAddress column already exists'

IF NOT EXISTS ( SELECT *
                FROM    sys.columns
                WHERE    name LIKE 'Postcode'
                        AND OBJECT_NAME(object_ID) = 'customer'
                        AND OBJECT_SCHEMA_NAME(object_ID) = 'dbo' )

    BEGIN
        PRINT N'Creating Postcode'
        ALTER TABLE [Customer]
        ADD Postcode VARCHAR(20)
    END
ELSE
    PRINT 'The postcode column already exists'
IF EXISTS ( SELECT 1
            FROM    dbo.customer
            WHERE    streetaddress IS NULL )
    EXECUTE sp_executeSQL N'
--Split data into columns StreetAddress and PostCode
UPDATE customer SET StreetAddress=SUBSTRING(ADDRESS, 0, CHARINDEX('';'', ADDRESS))
UPDATE customer SET Postcode=SUBSTRING(ADDRESS,CHARINDEX('';'', ADDRESS)+1,
LEN(Address) )'
ELSE
    PRINT 'this operation has already been completed'
```

4. Moving data from the table [dbo].[PersonData] to the tables [dbo].[Person] and [dbo].[Email]

This migration script includes a guard clause to check that the table [dbo].[PersonData] exists in the database. If the table doesn't exist, no changes are made.

If the table exists, the guard clause checks that the [dbo].[Person] and [dbo].[Email] tables exist. If they don't exist, the script creates the tables before updating the rows.

Migration script

```
IF NOT EXISTS ( SELECT 1
                 FROM    Information_Schema.Tables
                 WHERE    table_schema = 'dbo'
                        AND TABLE_NAME = 'PersonData' )

BEGIN
    PRINT 'Table [dbo].[PersonData] could not be found
    - skipping migration.';
    RETURN;
END

-- Create [dbo].Person if it doesn't exist at the time of deployment
IF NOT EXISTS ( SELECT 1
                 FROM    Information_Schema.Tables
                 WHERE    table_schema = 'dbo'
                        AND TABLE_NAME = 'Person' )

BEGIN
    CREATE TABLE Person
    (
        ID INT IDENTITY(1, 1)
            NOT NULL ,
        NAME NVARCHAR(200) ,
        CONSTRAINT PK_ID PRIMARY KEY ( ID )
    );
END

-- Create [dbo].Email if it doesn't exist
IF NOT EXISTS ( SELECT 1
                 FROM    Information_Schema.Tables
                 WHERE    table_schema = 'dbo'
                        AND TABLE_NAME = 'Email' )

BEGIN
    CREATE TABLE Email
    (
        PersonID INT ,
        Email NVARCHAR(200)
        CONSTRAINT FK_Person_ID
            FOREIGN KEY ( PersonID ) REFERENCES Person ( ID )
    );
END

--Move data from [dbo].PersonData into [dbo].Person
SET IDENTITY_INSERT [dbo].Person ON
INSERT INTO [dbo].Person
    ( ID ,
      Name
    )
    SELECT ID ,
           NAME
    FROM    dbo.PersonData
SET IDENTITY_INSERT [dbo].Person OFF
-- Move data from [dbo].[PersonData] to [dbo].Email
INSERT INTO dbo.Email
    ( PersonID ,
```

```
        Email
    )
    SELECT  ID ,
            Email1
    FROM    dbo.PersonData
    WHERE   Email1 IS NOT NULL
    UNION
    SELECT  ID ,
```

```
Email2
FROM    dbo.PersonData
WHERE   Email2 IS NOT NULL
```

5. Making “subtractive” schema changes (eg dropping tables or columns)

You can use a migration script to make intentional “subtractive” schema changes (e.g. dropping tables or column) that would otherwise cause a data loss warning and abort a continuous integration step.

This migration script includes a guard clause to check that the table [dbo].DropTbWoDataLoss exists in the database. If the table doesn't exist, no changes are made.

Migration script

```
IF NOT EXISTS (SELECT 1 FROM [information_schema].[Tables] WHERE table_schema = 'dbo'
AND TABLE_NAME = 'DropTbWoDataLoss')
    PRINT 'Object ''[dbo].[DropTbWoDataLoss]'' could not be found - skipping
migration.';
Else
    DROP TABLE [dbo].[DropTbWoDataLoss]
END
```

Example: renaming a table without data loss

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

This page walks through an example case of writing and using a migration script.

When you rename a table in Management Studio, SQL Source Control interprets this as dropping and recreating the table. Any data in the table is lost. (For more information about why this happens, see [What migration scripts do](#).) You can avoid this by creating a migration script to rename the table with the `sp_rename` stored procedure.

In this example, we'll rename a table in the development database (**Dev**) and create a migration script to avoid data loss. Then we'll commit the change to source control and finally apply it to the production database (**Prod**) using SQL Compare.

1: Set up the databases for this example

This script creates two databases, **Dev** and **Prod**. Both databases contain a table named `Person` with some data.

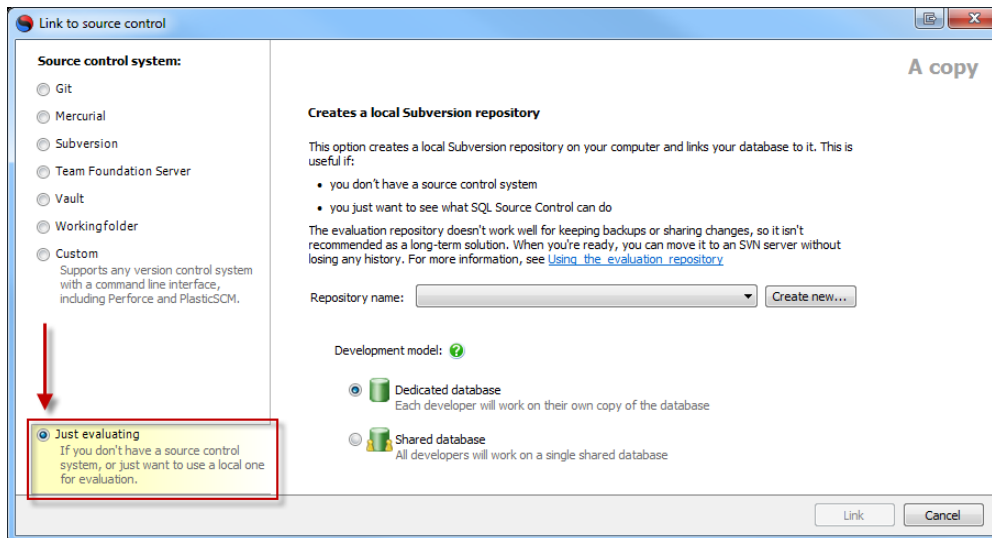
SQL creation script

[Expand source](#)

```
IF db_id('Dev') IS NULL
    CREATE DATABASE Dev
GO

IF db_id('Prod') IS NULL
    CREATE DATABASE Prod
GO
IF OBJECT_ID('Dev.[Person]', 'U') IS NULL
    BEGIN
        CREATE TABLE Dev.[dbo].[Person] (Name NCHAR(50))
        INSERT INTO Dev.[dbo].[Person] VALUES ('James')
        INSERT INTO Dev.[dbo].[Person] VALUES ('John')
        INSERT INTO Dev.[dbo].[Person] VALUES ('David')
    END
IF OBJECT_ID('Prod.[Person]', 'U') IS NULL
    BEGIN
        CREATE TABLE Prod.[dbo].[Person] (Name NCHAR(50))
        INSERT INTO Prod.[dbo].[Person] VALUES ('Mary')
        INSERT INTO Prod.[dbo].[Person] VALUES ('Margaret')
        INSERT INTO Prod.[dbo].[Person] VALUES ('Madeline')
    END
END
```

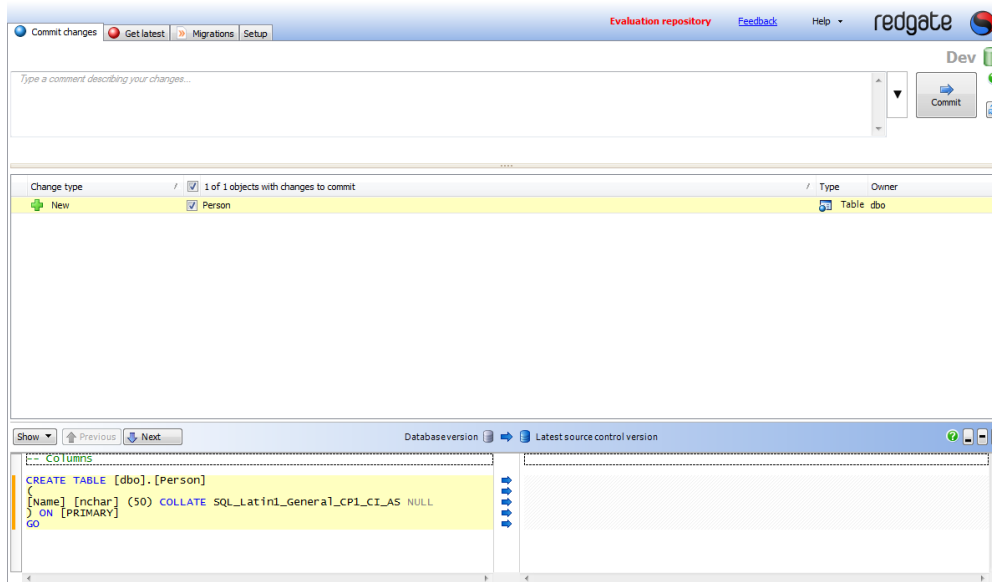
1. Run the script on your server and refresh the Object Explorer.
The **Dev** and **Prod** databases are created.
2. Link **Dev** and **Prod** to the same [evaluation repository](#) in source control.



Make sure **Dedicated database** is selected as the development model.

Now we need to do an initial commit of all the objects in the Dev database.

- With **Dev** selected, go to the **Commit changes** tab.
The **Person** table is listed as a new change to commit:



- Click **Commit**.
The **Person** table is committed to source control.

2: Create, test, and commit the migration script

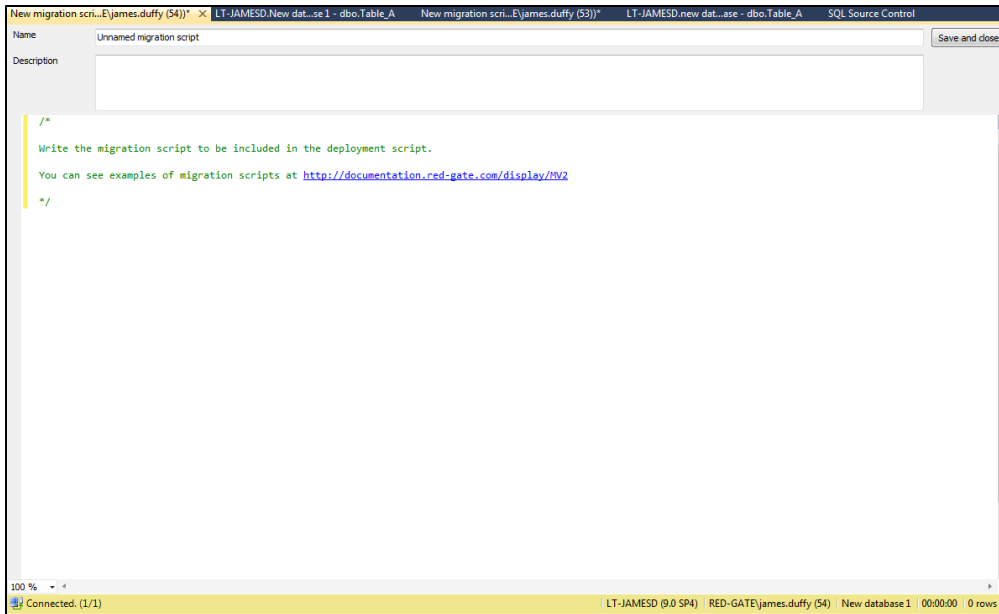
We want to rename the table in Dev and apply the change to Prod.

If we wanted to do this conventionally, we might rename the table, write a migration script to prevent data loss, and commit the change and the migration script to source control.

However, in this example, we'll write a migration script *first*, use the *script* to make the change, and then commit both the change and the script to source control. By using the migration script to make the change in the first place, we can test the script works.

- Select **Dev** in the Object Explorer. In SQL Source Control, go to the **Migrations** tab and click **Add migration script**.

The **New migration script** tab opens:



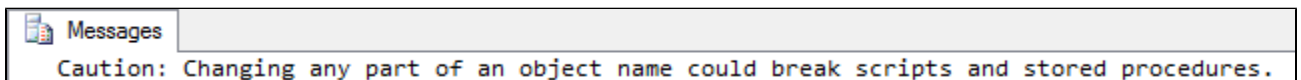
- Paste this SQL into the main box in the migration script dialog:

```
IF OBJECT_ID(' [dbo].[Person]', 'U') IS NOT NULL
BEGIN
EXEC sp_rename ' [dbo].[Person]', 'Customer'
END
```

This SQL is our migration script. It uses a guard clause to check for a table named Person. If the table exists, the script renames it Customer. If it doesn't exist, no changes are made.

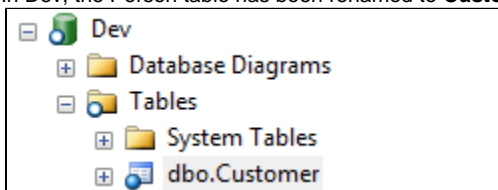
For more information about guard clauses, and examples of other migration scripts, see [Example V2 migration scripts](#).

- Run the script.
The server displays a caution:



This is normal, and you can ignore it.

- Refresh the Object Explorer.
In Dev, the Person table has been renamed to **Customer**:



This means the script worked correctly.

- In the Migrations tab, in the **Name** field, type a name for the migration script and click **Save and close**.
- Go to the **Commit changes** tab.
New changes are listed:

| Change type | | 4 of 4 objects with changes to commit | | Type | Owner |
|-------------|------|---------------------------------------|------------------|----------|---------|
| | Drop | <input checked="" type="checkbox"/> | Person | Table | dbo |
| | New | <input checked="" type="checkbox"/> | Customer | Table | dbo |
| | New | <input checked="" type="checkbox"/> | MigrationHistory | Function | RedGate |
| | New | <input checked="" type="checkbox"/> | RedGate | Schema | |

Although we've renamed the Person table to Customer, the change is listed incorrectly as a drop and create. This will be fixed in future versions.

The **RedGate** schema contains the table-valued function **MigrationHistory**. This contains the new migration script so it can be used again in future deployments.

7. Click **Commit**.
The changes are committed to source control.

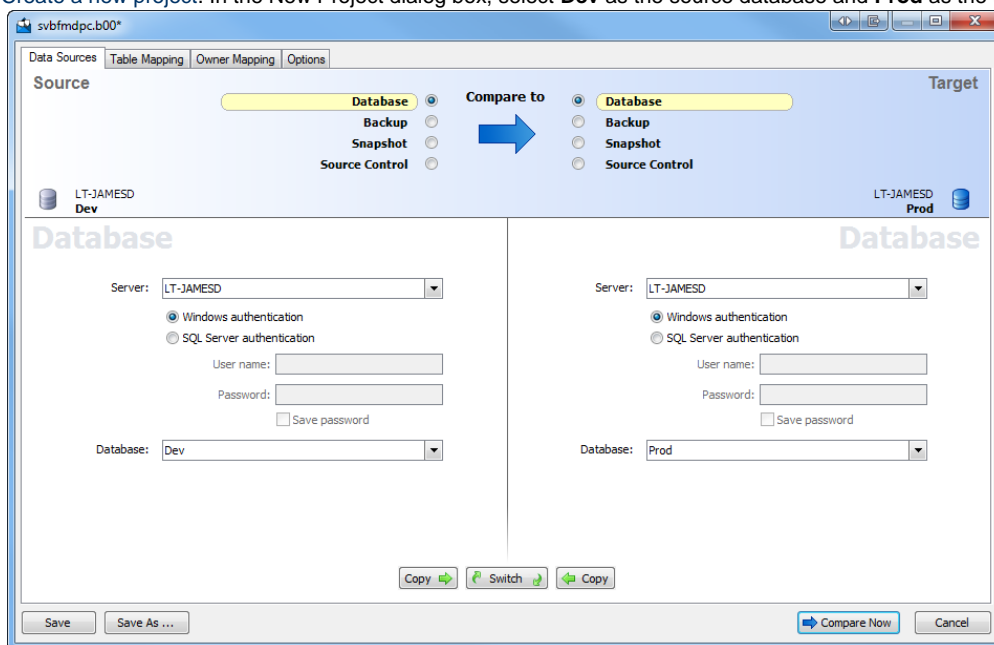
3: Deploy the changes to Prod

Now we've committed the changes, we need to apply them to Prod using SQL Compare.

We could do this using the Get latest tab, but we don't recommend using SQL Source Control on production databases.

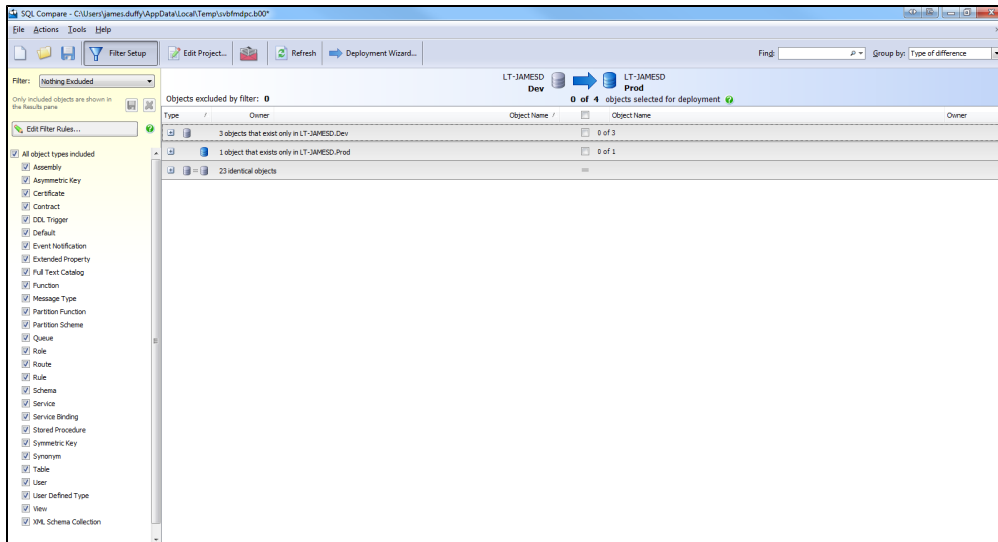
✓ Method 1: Deploy the changes to Prod using SQL Compare (click to view)

1. In SQL Compare, make sure Migrations V2 is enabled in the [project options](#).
2. **Create a new project**. In the New Project dialog box, select **Dev** as the source database and **Prod** as the target:

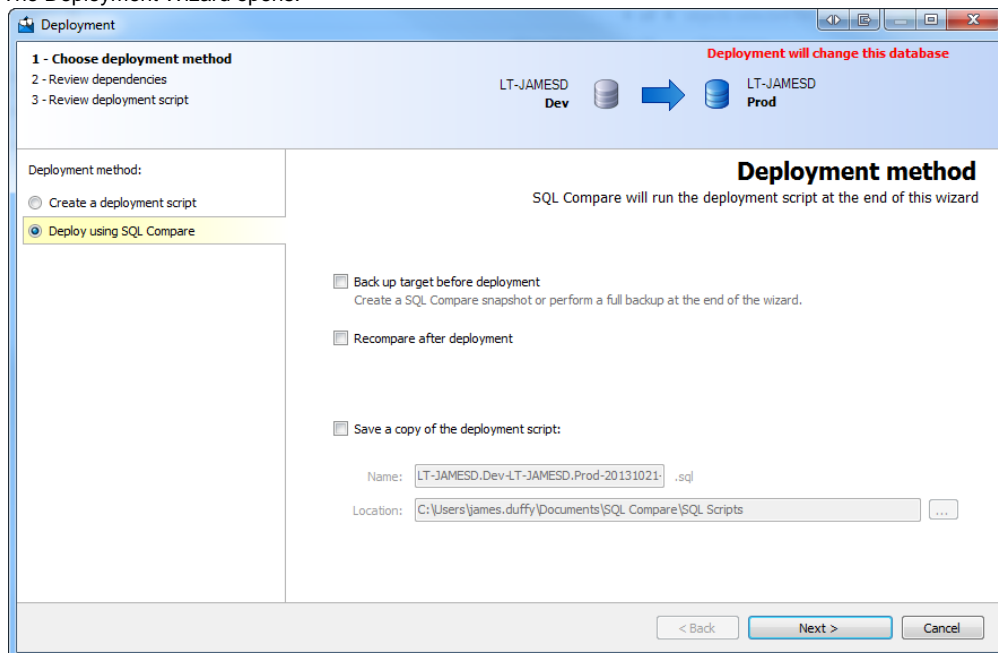


For more information about setting source and target databases in SQL Compare, see [Setting data sources](#) in the SQL Compare documentation.

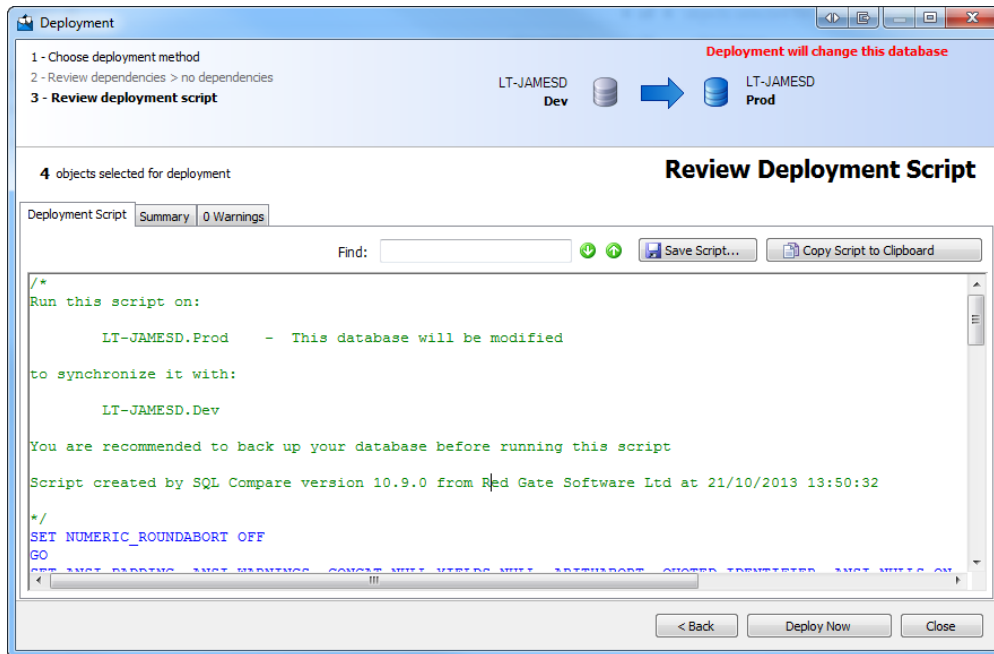
3. Click **Compare Now**.
SQL Compare displays the differences between Dev and Prod:



4. Select all the objects and click **Deployment Wizard**.
The Deployment Wizard opens:



5. Make sure **Deploy using SQL Compare** is selected in the left, and click **Next**.
SQL Compare displays the deployment script to deploy the changes from Dev to Prod:

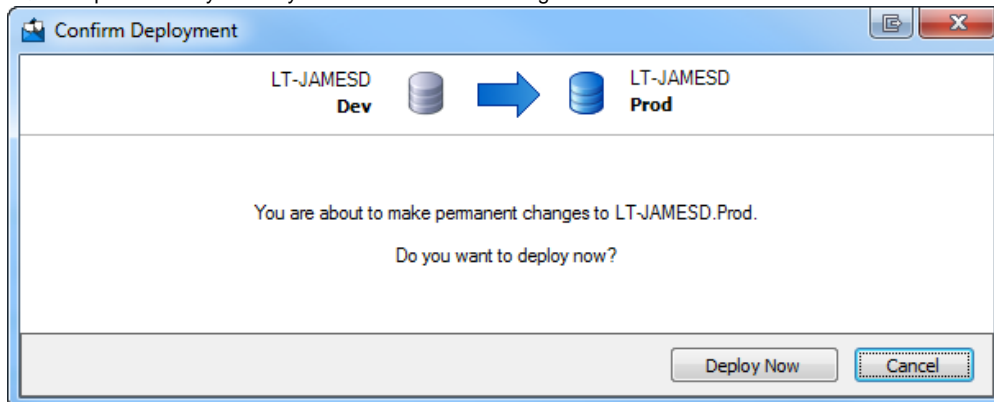


The deployment script includes the migration script we added to rename the table:

```
-- Apply upwards migration script: Rename Person table to Customer --
IF OBJECT_ID('[dbo].[Person]', 'U') IS NOT NULL
BEGIN
EXEC sp_rename '[dbo].[Person]', 'Customer'
END
```

6. Click **Deploy Now**.

SQL Compare warns you that you're about to make changes to the database:



7. Click **Deploy Now**.

The script is run on the Prod database.

✓ **Method 2: Deploy the changes to Prod using the SQL Compare command line (click to view)**

1. Make sure Migrations V2 is enabled in the SQL Compare project options.
2. In the SQL Compare command line, enter:

```
sqlcompare /Database1:Dev /Database2:Prod /Synchronize
```

Prod is updated with the changes from Dev:

```
C:\Windows\system32\cmd.exe

C:\Program Files (x86)\Red Gate\SQL Compare 10>sqlcompare /Database1:Dev /Database2:Prod /synchronize
SQL Compare: activated, edition: professional, serial number: 507-900-173025-FE51
SQL Compare Command Line V10.9.0.1090
=====
Copyright c Red Gate Software Ltd 1999-2013

Error: Invalid command-line argument: /synchronize. Please use SQLCompare.exe /?
or SQLCompare.exe /verbose /? for more information.

C:\Program Files (x86)\Red Gate\SQL Compare 10>sqlcompare /Database1:Dev /Database2:Prod /synchronize
SQL Compare: activated, edition: professional, serial number: 507-900-173025-FE51
SQL Compare Command Line V10.9.0.1090
=====
Copyright c Red Gate Software Ltd 1999-2013

Registering data sources
Creating mappings
Comparing
Applying Command Line Items
Checking for identical databases
Creating SQL
Deploying changes (from DB1 to DB2)

Summary Information
=====
DB1 = <local>.Dev
DB2 = <local>.Prod

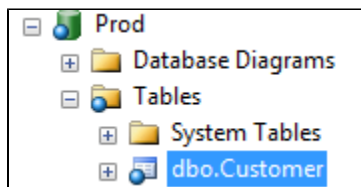
Object type      Name                                     DB1 DB2
-----
Table            [dbo].[Customer]                       >>
Table            [dbo].[Person]                         <<
Function         [RedGate].[MigrationHistory]           >>
Schema           RedGate                                >>
```

For more information about the SQL Compare command line, see [Using the command line](#) in the SQL Compare documentation.

4: Verify the data is preserved

Now we've deployed the changes, we can verify that the data is preserved.

In Management Studio, refresh the Object Explorer and view the tables in Prod:



The Person table has been renamed to Customer.

If we select the top 1000 rows, we can see the data is preserved:

SQLQuery14.sql - L...E\james.duffy (59) X

```
/****** Script for SelectTopNRows command from SSMS *****/  
SELECT TOP 1000 [Name]  
FROM [Prbd].[dbo].[Customer]
```

100 %



Results



Messages

| | Name |
|---|----------|
| 1 | Mary |
| 2 | Margaret |
| 3 | Madeline |

Example: writing a V2 migration script that affects objects not yet in the target database

We're going to remove the migrations V2 beta from SQL Source Control in version 5. We're replacing it with an improved version of the original migrations feature that supports more things (including Git, branching, and merging).

We'll be publishing more information about this soon.

This page refers to the Migrations V2 beta. For information about how to use V1 migration scripts, see [Working with migration scripts](#).

Thanks to user Steve Ognibene for raising this scenario.

1: Normalizing a table

Say we have a script to create a table named `dbo.PersonData`:

```
CREATE TABLE dbo.PersonData(  
    ID INT IDENTITY(1,1),  
    NAME NVARCHAR(200) NOT NULL,  
    Email1 NVARCHAR(200) NULL,  
    Email2 NVARCHAR(200) NULL,  
    Phone1 NVARCHAR(100) NULL,  
    Phone2 NVARCHAR(100) NULL,  
    Street1 NVARCHAR(200) NULL,  
    City1 NVARCHAR(200) NULL,  
    StateProvince1 NVARCHAR(50) NULL,  
    PostalCode1 NVARCHAR(50) NULL,  
    Street2 NVARCHAR(200) NULL,  
    City2 NVARCHAR(200) NULL,  
    StateProvince2 NVARCHAR(50) NULL,  
    PostalCode2 NVARCHAR(50) NULL,  
    CONSTRAINT PK_PersonDataID PRIMARY KEY (ID));
```

We want to normalize the table. To do it, we'll break it into different tables in a new schema named **Contacts**:

```

CREATE SCHEMA Contacts
GO
CREATE TABLE Contacts.Person(
    ID INT IDENTITY(1,1),
    NAME NVARCHAR(200),
    CONSTRAINT PK_PersonID PRIMARY KEY (ID)
);

CREATE TABLE Contacts.Email(
    PersonID INT,
    Email NVARCHAR(200)
);

CREATE TABLE Contacts.Phone(
    PersonID INT,
    Phone NVARCHAR(100)
);

CREATE TABLE Contacts.PostAddress(
    PersonID INT,
    Street NVARCHAR(200) NULL,
    City NVARCHAR(200) NULL,
    StateProvince NVARCHAR(50) NULL,
    PostalCode NVARCHAR(50) NULL
);

CREATE TABLE Contacts.FavoriteFoods(
    PersonID INT,
    FoodName NVARCHAR(200)
)

```

Now we'll commit the new schema to source control.

2. Migrating the data

Next we'll write a migration script to migrate the data from the old `dbo.PersonData` table to the new tables in the `Contacts` schema.

To migrate `<Id, Name>` from `dbo.PersonData` into `[Contacts].Person`, we'll write:

```

--Move data from dbo.PersonData into [Contacts].Person
SET IDENTITY_INSERT [Contacts].Person ON
BEGIN
    INSERT INTO [Contacts].Person ( ID, Name)
    SELECT ID, NAME FROM dbo.PersonData
END
SET IDENTITY_INSERT [Contacts].Person OFF

```

To migrate `Email1` and `Email2` from `dbo.PersonData` into `[Contacts].Person` (creating two separate rows for a person having both `Email1` and `Email2`):

```
--Move data from dbo.PersonData into [Contacts].Email
BEGIN
    INSERT INTO Contacts.Email(PersonID,Email)
    SELECT ID, Email1 FROM dbo.PersonData WHERE Email1 IS NOT NULL
    UNION
    SELECT ID, Email2 FROM dbo.PersonData WHERE Email2 IS NOT NULL
END
```

To migrate Phone1 and Phone2 from dbo.PersonData into [Contacts].Phone:

```
--Move data from dbo.PersonData into [Contacts].Phone
BEGIN
    INSERT INTO Contacts.Phone(PersonID,Phone)
    SELECT ID, Phone1 FROM dbo.PersonData WHERE Phone1 IS NOT NULL
    UNION
    SELECT ID, Phone2 FROM dbo.PersonData WHERE Phone2 IS NOT NULL
END
```

To migrate address details from dbo.PersonData to [Contacts].PostAddress:

```
--Move data from dbo.PersonData into [Contacts].PostAddress
BEGIN
    INSERT INTO
Contacts.PostAddress(PersonID,Street,City,StateProvince,PostalCode)
    SELECT ID, Street1,City1,StateProvince1,PostalCode1 FROM dbo.PersonData
    UNION
    SELECT ID, Street2,City2,StateProvince2,PostalCode2 FROM dbo.PersonData
END
```

3. Writing guard clauses

Say we want to deploy the changes above to our production database, **Prod A**. Prod A contains the original dbo.PersonContact table with some data. If we write a migration script without checking that the Contacts schema and the new tables [Contacts].Person, [Contacts].Phone, [Contacts].Email and [Contacts].PostAddress exist in Prod A, the deployment will fail. This is because the database state our migration script is applied to is different from the state we wrote our migration script for. This is particularly easy to miss if you're working on several database changes at the same time, or if you write the migration script some time after creating the schema.

We'll include some guard clauses in the migration script to check the database is in the right state for the script to work. If it isn't, the script will alter your database first so the the rest of the migration script can be applied. This means the migration script will always work, no matter what state the deployment environment is in.

```
IF OBJECT_ID('[dbo].[PersonData]', 'U') IS NOT NULL --check if the denormalized table
exists as a first step
BEGIN
    IF NOT EXISTS (SELECT NAME FROM Sys.Schemas WHERE NAME = 'Contacts') --check if
Contacts schema exists
    BEGIN...
```

Now let's add some logic to create the objects if they don't exist when our migration script is applied.

```

IF NOT EXISTS (SELECT NAME FROM Sys.Schemas WHERE NAME = 'Contacts') --create Contacts
schema if it doesn't exist
BEGIN
EXEC sp_executesql N'CREATE SCHEMA Contacts'
--CREATE TABLE #ShouldRunMigration (notUsed bit)
END

--Create [Contacts].[Person] if it doesn't exist at time of deployment
IF (OBJECT_ID('[Contacts].[Person]', 'U') IS NULL)
BEGIN
CREATE TABLE [Contacts].Person(
ID INT IDENTITY(1,1),
NAME NVARCHAR(200),
CONSTRAINT PK_PersonID PRIMARY KEY (ID)
);
END...

```

4. Final migration script

The final migration script looks like this:

```

IF OBJECT_ID('[dbo].[PersonData]', 'U') IS NOT NULL --check if the denormalized table
exists as a first step
BEGIN
IF NOT EXISTS (SELECT NAME FROM Sys.Schemas WHERE NAME = 'Contacts') --create Contacts
schema if it doesn't exist
BEGIN
EXEC sp_executesql N'CREATE SCHEMA Contacts'
--CREATE TABLE #ShouldRunMigration (notUsed bit)
END

--Create [Contacts].[Person] if it doesn't exist at time of deployment
IF (OBJECT_ID('[Contacts].[Person]', 'U') IS NULL)
BEGIN
CREATE TABLE [Contacts].Person(
ID INT IDENTITY(1,1),
NAME NVARCHAR(200),
CONSTRAINT PK_PersonID PRIMARY KEY (ID)
);
END

--Create [Contacts].[Email] if it doesn't exist
IF (OBJECT_ID('[Contacts].[Email]', 'U') IS NULL)
BEGIN
CREATE TABLE Contacts.Email(
PersonID INT,
Email NVARCHAR(200));
END

--Create [Contacts].[Phone] if it doesn't exist
IF (OBJECT_ID('[Contacts].[Phone]', 'U') IS NULL)
BEGIN
CREATE TABLE [Contacts].Phone(
PersonID INT,

```

```

        Phone NVARCHAR(100));
END

--Create [Contacts].[PostAddress] if it doesn't exist
IF (OBJECT_ID('[Contacts].[PostAddress]', 'U') IS NULL)
BEGIN
    CREATE TABLE Contacts.PostAddress(
        PersonID INT,
        Street NVARCHAR(200) NULL,
        City NVARCHAR(200) NULL,
        StateProvince NVARCHAR(50) NULL,
        PostalCode NVARCHAR(50) NULL);
END

/*--Create [Contacts].[FavouriteFoods] doesn't need to be part of the migration script
IF OBJECT_ID('[Contacts].[FavouriteFoods]', 'U') IS NULL
BEGIN
    CREATE TABLE Contacts.FavoriteFoods(
        PersonID INT,
        FoodName NVARCHAR(200));
END
*/

--Move data from dbo.PersonData into [Contacts].Person
SET IDENTITY_INSERT [Contacts].Person ON
BEGIN
    INSERT INTO [Contacts].Person(ID, Name)
        SELECT ID, NAME FROM dbo.PersonData
END
SET IDENTITY_INSERT [Contacts].Person OFF

--Move data from dbo.PersonData into [Contacts].Email
BEGIN
    INSERT INTO Contacts.Email(PersonID,Email)
        SELECT ID, Email1 FROM dbo.PersonData WHERE Email1 IS NOT NULL
    UNION
        SELECT ID, Email2 FROM dbo.PersonData WHERE Email2 IS NOT NULL
END

--Move data from dbo.PersonData into [Contacts].Phone
BEGIN
    INSERT INTO Contacts.Phone(PersonID,Phone)
        SELECT ID, Phone1 FROM dbo.PersonData WHERE Phone1 IS NOT NULL
    UNION
        SELECT ID, Phone2 FROM dbo.PersonData WHERE Phone2 IS NOT NULL
END

--Move data from dbo.PersonData into [Contacts].PostAddress
BEGIN
    INSERT INTO
Contacts.PostAddress(PersonID,Street,City,StateProvince,PostalCode)
        SELECT ID, Street1,City1,StateProvince1,PostalCode1 FROM dbo.PersonData
    UNION
        SELECT ID, Street2,City2,StateProvince2,PostalCode2 FROM dbo.PersonData

```



```
END
```

```
END
```

After running the migration script, we can drop the original `dbo.PersonData` table as a separate commit.

Troubleshooting

These pages describe how to fix common problems with SQL Source Control.

- [How SQL Source Control works behind the scenes](#)
- [Can't link to TFS 2012 or Visual Studio Online](#)
- [Linking fails due to SVN pre-commit hooks](#)
- [Logging and log files](#)
- [Object changed by Unknown](#)
- [Error messages](#)
- [About the RedGateLocal schema](#)
- [Third-party software used by SQL Source Control](#)

If your problem isn't covered here, try the [SQL Source Control 3 forum](#) or [contact Redgate support](#).

How SQL Source Control works behind the scenes

You don't need to read this page to use and understand SQL Source Control, but it might be helpful for troubleshooting, or if you're just curious.

SQL Source Control uses the [SQL Compare](#) engine to create and maintain folders of object creation scripts, called scripts folders, that represent your database schema. When you make changes to your database, you can commit them to source control. Likewise, when there are new changes to get from source control, you can apply them to your database.

However, SQL Source Control doesn't just run comparisons between the database and the scripts folder in your source control system. This wouldn't capture all the information needed to track database changes. For example:

1. Amanda links a blank database to a blank repository in her source control system. As part of the process, SQL Source Control creates a scripts folder in her repository, representing the (blank) database schema.
2. Amanda creates a table in her database and goes to the Commit changes tab.

If we compare Amanda's database and the scripts folder in source control, we can see a table in Amanda's database that isn't in source control. We don't have any information about where the difference came from, so we can't tell whether we should commit that table to source control or drop it from Amanda's database.

To solve this problem, SQL Source Control keeps track of three separate versions of the database:

- The **database** itself, the current state of the database in SQL Server.
- The **transient**, a local copy of the latest revision in source control. SQL Source Control periodically checks your source control system and updates the transient so it always reflects the latest version available (the head revision). The transient is stored in your local application data.
- The **working base**, a local copy of the database at the last time you ran a commit or a get latest. Like the transient, it's stored in your local application data.

The three-way comparison

When you visit the Commit tab or the Get latest tab, or click Refresh, SQL Source Control runs three comparisons using the SQL Compare engine. It uses the results of these three comparisons to create the list of changes.

1: Identifying changes to commit

SQL Source Control compares the database and the working base. Changes you've made to the database since the last time SQL Source Control communicated with source control are listed as changes to *commit*.

2: Identifying changes to get

SQL Source Control compares the working base and the transient. Changes that someone else committed since SQL Source Control last communicated with source control are listed as changes to *get*.

3: Identifying conflicts and no-ops

When you visit the Commit or Get latest tab, SQL Source Control runs comparisons 1 and 2 and checks the lists of changes against each other. If the same object appears in both lists, it means another user has committed a change to source control on an object you've changed locally.

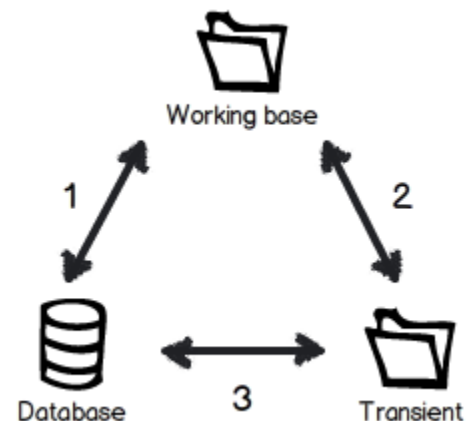
At this point, one of two things may have happened: a conflict, or a no-op. To determine which, SQL Source Control runs a comparison between the transient and the database.

Possibility 1: you and another user have changed the same object in different ways (conflict)

If the object isn't the same in the transient and the database, this change is a *conflict*. You can choose to [resolve the conflict](#) by applying source control version to your database ("Take theirs"), or overwrite the version in source control with your own version ("Keep mine").

Possibility 2: you and another user have made the same change to the object (no-op)

If the object is identical in the transient and the database, it means you and another user have independently made the same change to an object. SQL Source Control doesn't notify you, because the two conflict resolution options ("Take theirs" and "Keep mine") are functionally identical.



Instead, SQL Source Control silently updates the working base with the change. This is a *no operation*, or no-op. It's the only situation in which SQL Source Control doesn't list a change in the Commit or Get latest tabs.

Example: Committing a new table to source control

1. Beth creates a blank database and links it to a blank repository in her source control system.
2. Beth creates a table in her database.
3. Beth goes to the Commit changes tab. SQL Source Control runs the three-way comparison. Because there's a table in Beth's database that isn't in the working base, SQL Source Control knows this is a change to commit, and displays it in the Commit tab.
4. Beth clicks Commit. SQL Source Control updates the scripts folder in Beth's repository with a script for the new table, and updates the transient and working base.
5. SQL Source Control runs the three-way comparison again. There are no longer any differences between the working base and the database, so SQL Source Control reports that there are no changes to commit.

Can't link to TFS 2012 or Visual Studio Online

To link to TFS 2012 or Visual Studio Online, update to the latest version of SQL Source Control.

If you don't want to update SQL Source Control, you need to edit a config file.

Editing the config file

After you edit the config file, you won't be able to link to earlier versions of TFS. You can undo this by undoing the changes described in this page.

1. Make sure SQL Server Management Studio is closed.
2. Go to the SQL Source Control config files folder. By default, this is located at %localappdata%\Red Gate\SQL Source Control 3
3. Open *RedGate_SQLSourceControl_Engine_EngineOptions.xml* in a text editor.
4. Below the `EngineOptions version` line, add:

```
<TeamFoundationServerDllOverride>Tfs2012</TeamFoundationServerDllOverride>
```

Ignoring any comments (indicated with `<!-->`), the final file should look like this:

RedGate_SQLSourceControl_Engine_EngineOptions.xml

```
<EngineOptions version="3" type="EngineOptions">
  <TeamFoundationServerDllOverride>Tfs2012</TeamFoundationServerDllOverride>
</EngineOptions>
```

The file is case sensitive. Don't change the capitalization of the text.

The example above doesn't include any extra lines you may have included. For example, you may have included additional lines to set up [change logging](#).

5. Save and close the file.

You can now link databases to TFS 2012 and Visual Studio Online.

For more information about linking databases, see [Linking to TFS](#).

Linking fails due to SVN pre-commit hooks

If you have Subversion pre-commit hooks that require a comment, or that require a comment of a particular format, linking a database for the first time might fail.

This happens because the first time a database is linked, SQL Source Control adds a folder structure and some accompanying metadata to source control, and the default comment may not be appropriate.

If your pre-commit hooks stop you linking a database within SQL Source Control, you can create the initial link manually. To do this:

1. Using your source control client (eg [TortoiseSVN](#)), create a new, empty folder in your repository.
This will be the folder that contains the database. The URL for the folder location is used to link the database in SQL Source Control.
2. [Download this .zip file](#) containing the folder structure and metadata.
3. Unzip the .zip file and commit the contents to source control in the folder you created.
When you commit, add an appropriate comment.
4. [Link the database to SVN](#).

Logging and log files

Log files collect information about SQL Source Control while you use it. These files are useful if you're working with Redgate support to fix a problem.

By default, SQL Source Control logs all error information (see the table below). This produces large log files. If you're troubleshooting a specific problem, you may want to reduce the amount of logging to make the logs easier to read.

Example log file using the default log level, "Serious"

[Expand source](#)

```
10:20:01.983|Info    |er.StartOfDayLogging|1    |#1:Power Mode Changed, mode is Resume
10:20:03.087|Info    |er.StartOfDayLogging|1    |#1:Time Changed
10:20:03.429|Debug   |VsCommandUsageLogger|1    |#1:Execute for VS command
Edit.GoToFindCombo
10:20:09.707|Info    |er.StartOfDayLogging|1    |#1:Time Changed
10:20:19.449|Info    |Performance Testing |9    |
|#9:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:StartTimeTicks=635265876008030745:Name=S
erverPollEvent
10:20:19.552|Info    |Performance Testing |9    |
|#9:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:Step=1:Seconds=3.77532491619145E-05:Afte
r=Get Lock
10:20:19.552|Info    |Performance Testing |9    |
|#9:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:Step=2:Seconds=0.000117291647881676:Afte
r=Copied database names
10:20:19.552|Info    |Performance Testing |9    |
|#9:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:Step=3:Seconds=0.000157610651841002:Afte
r=Calculated work
10:20:19.552|Info    |Performance Testing |9    |
|#9:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:Step=4:Seconds=18.6582575885863:After=Ge
t New Entries
10:20:19.552|Info    |Pollers.ServerPoller|13   |#13:Got 0 new default trace entries for
AdventureWorks
10:20:19.552|Info    |Performance Testing |33   |
|#33:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:Step=5:Seconds=18.7612858390128:After=I
nvoke New Entries
10:20:19.552|Info    |Performance Testing |33   |
|#33:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:Step=6:Seconds=18.7613045323692:After=M
ake AboutToBeActive Active
10:20:19.552|Info    |Performance Testing |33   |
|#33:Guid=58317ab7-0627-4261-9f33-fb2c5c01696a:NumberOfSteps=6
10:20:19.552|Info    |Performance Testing |33   |
|#33:Guid=b3e1aae0-9c1a-46f3-abda-e1e96a90fd1f:StartTimeTicks=635265876008030745:Name=
PollStrategyCallBack
10:20:19.552|Info    |Performance Testing |33   |
|#33:Guid=b3e1aae0-9c1a-46f3-abda-e1e96a90fd1f:NumberOfSteps=0
```

Changing the log level

You can only change the log level in SQL Source Control 3.3 and later.

1. Close Management Studio.
2. Go to the SQL Source Control config files folder. By default, this is `%localappdata%\Red Gate\SQL Source Control 3`
3. Open `RedGate_SQLSourceControl_Engine_EngineOptions.xml` in a text editor.
4. Inside the `<EngineOptions>` tags, add:

```
<LogLevel>your specified log level</LogLevel>
```

You can specify:

| Log level | Effect |
|--------------------|--|
| None | Disables all logging. |
| Debug | Logs additional debug information. |
| Trace | Logs stack trace after an exception. |
| Information | Logs information collected when getting changes in the Get latest tab. |
| Warning | Logs information collected when minor errors occur. |
| Error | Logs information collected when mid-level errors occur. |
| Fatal | Logs information collected when fatal errors (ie crashes) occur. |
| Serious | Combines the warning , error and fatal log levels (ie logs information from all error types). This is used by default if no log level is set. |
| Default | Combines the debug , trace , information and serious log levels. Note: despite its name, this is <i>not</i> the default log level. |
| All | Logs all messages. |

The file is case sensitive. Don't change the capitalization of the text.

5. Save and close the file.

SQL Source Control saves log files with the setting you specified.

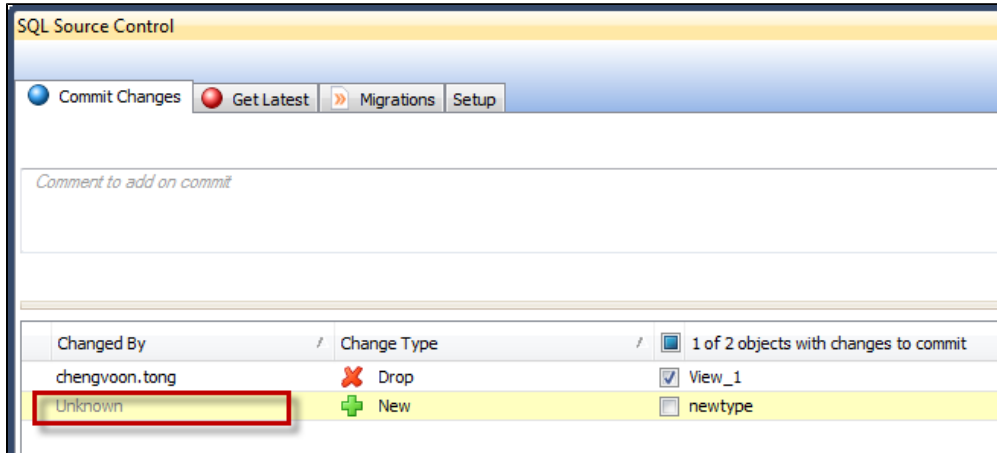
Locating the log files

By default, log files for all Red Gate tools are located in: %LOCALAPPDATA%\Red Gate\Logs

Object changed by Unknown

Some of the causes of this problem are fixed in SQL Source Control 3.6.5 and later.

If you're using the [shared development model](#), the **Changed by** column might list objects as changed by **Unknown**:



You can still commit the change to source control as normal, as the "Changed by" information is never committed to source control anyway.

Possible causes

The object type is unsupported

SQL Source Control can't tell who made changes to data, or to the following object types:

| | | |
|---------------------|---------------------|--------------------------|
| application roles | extended properties | search property lists |
| asymmetric keys | full text catalogs | symmetric keys |
| certificates | full text stoplists | table keys |
| constraints | partition functions | user-defined data types |
| DDL triggers | partition schemes | user-defined table types |
| DML triggers | roles | user-defined types |
| event notifications | schemas | users |

Information is lost from tempdb

By default, SQL Source Control reads information about who made changes from the default trace and saves it in tempdb. However, because tempdb is reset when the server is restarted, information about who made a change is eventually lost.

Fix

To stop the information being lost, you can [create a database to log changes](#) instead of tempdb.

SQL Source Control was updated while there were uncommitted changes

If you update to a new version of SQL Source Control while there are uncommitted changes, information about who made changes might be lost.

The object was renamed

When an object is renamed, information about who renamed it is lost.

This is because SQL Source Control reads information about who made a change from the default trace. In the case of a rename, the default trace records the activity correctly as a rename. However, SQL Source Control interprets the action only as a drop and create, and so can't find the relevant information in the default trace.

The default trace rolled over before SQL Source Control could poll it

When SQL Source Control checks the database for changes, it reads information about who made a change from the default trace and saves it in tempdb.

If you're working with a busy server, the default trace can fill up and roll over before SQL Source Control can read it. Information about who made changes is lost.

To improve the chances of the default trace being read before it rolls over, you can set the change indicators to update more frequently in the **Setup** tab. However, this might slow down your server.

The default trace is disabled

When SQL Source Control checks the database for changes, it reads information about who made a change from the default trace. If the default trace isn't enabled for the server, information about who made the change is lost.

To enable the default trace, run this SQL:

```
EXEC sp_configure 'show advanced options', 1
RECONFIGURE
EXEC sp_configure 'default trace enabled', 1
RECONFIGURE
```

You don't have permission to access the default trace

When SQL Source Control checks the database for changes, it reads information about who made a change from the default trace. If you don't have the correct permissions to access the default trace, information about who made the change is lost.

Users need the ALTER TRACE and VIEW SERVER STATE permissions to access the default trace. To add these permissions, your system administrator needs to go to each user's login (**Security > Logins** on the server level) and enable them under **Securables**.

For more information about the default trace, see [The default trace in SQL Server - the power of performance and security auditing on Simple Talk](#).

The server timed out

If the server takes too long to respond, SQL Source Control can't tell who changed any objects. When this happens, SQL Source Control displays a warning bar above the change list:

| | | | | |
|---|-------------|---------------------------------------|-------|------------|
| SQL Source Control can't tell who made these changes because the server took too long to respond. Try refreshing when the server is less busy. You can still commit the changes as normal. Learn more | | | | |
| Last chang... | Change type | 1 of 2 objects with changes to commit | | Type Owner |
| Unknown | New | LastName | Table | dbo |
| Unknown | New | Person2 | Table | dbo |

Unlike other causes of the "Unknown" problem, missing information caused by a server timeout isn't lost permanently. To see who changed the object, try refreshing the change list when the server is less busy.

Error messages

These pages explain how to fix problems that have error messages.

- Failed to locate the [object name] for the [object name]
- Failed to resolve no-ops after 5 tries
- ICredentialsProvider is unset, therefore can't get
- Newer version of Proc found - please update your Source Control
- SQL Source Control can't access this database because the directory once contained more than one SQL Server Database Project (.sqlproj) file
- System.OutOfMemoryException
- The specified path, file name, or both are too long
- The Team Foundation Server for this workspace does not support one or more of the checkin options you have selected
- This commit doesn't meet the server's policy requirements, or the policy isn't configured on your machine
- Unsaved documents cannot be cut or copied to the clipboard from the Miscellaneous Files project

Failed to locate the [object name] for the [object name]

This error is shown when you unlink a database, edit objects that reference other objects, and then relink the database. When SQL Source Control rebuilds the working base, it generates invalid SQL because the referenced objects have changed.

For more information about the working base, see [How SQL Source Control works behind the scenes](#).

Fix

To fix the problem, you need to make the database and source control versions of the referenced objects the same.

There are several ways to do this, depending on whether you want to take the version of the object in source control or the version in the database.

To take the version in the database

Manually edit the SQL scripts in the source control repository to match the database.

1. Open your source control repository. You can find the repository location on the Setup tab in SQL Source Control.
2. Find the SQL files for the affected objects, edit the files so they match the objects in the database, and save the changes.
3. In SQL Source Control, unlink the database and link it to source control again.

Other developers linked to the repository will see the changes on the Get latest tab.

To take the version in source control

If you have SQL Compare Pro

You can use SQL Compare Pro to deploy the SQL scripts in the SQL Source Control transient to the database. The transient is a copy of the version in source control (see [How SQL Source Control works behind the scenes](#)).

To do this:

1. In Management Studio, make sure you have the affected database selected in the Object Explorer.
2. In SQL Source Control, on the **Setup** tab, expand **Under the hood** and click **Open transient**.
The transient opens in Windows Explorer.
3. Copy the transient location (eg `C:\Users\username\AppData\Local\Red Gate\SQL Source Control 3\Transients\q4jmm33z.btr`).
4. In SQL Compare, open a new project.
5. On the left, under **Source**, select **Source Control**.
6. Under **Source Control**, select **Scripts folder**, and in the **Scripts folder** field, paste the transient folder location from step 2.
7. On the right, under **Target**, enter the connection details for affected database.
8. Click **Compare Now**, and when the comparison completes, click **OK**.
9. In the list of objects, select the affected objects and click **Deployment Wizard**.
10. In the Deployment Wizard, on the left, select **Deploy using SQL Compare** and click **Next**.
11. Click **Deploy Now**.
12. In SQL Source Control, unlink the database and link it to source control again.

If you don't have SQL Compare Pro

Manually edit the database to match the SQL scripts in the source control repository.

To do this:

1. Open your source control repository. You can find the repository location on the Setup tab.
2. Find the SQL files for the affected objects.
3. Edit the objects in the database to match the SQL files in the source control repository.
4. In SQL Source Control, unlink the database and link it to source control again.

If that didn't work

Remove the references and stop SQL Source Control generating invalid SQL.

This method might require deleting tables from the database.

To do this:

1. In Management Studio, remove the references from the affected objects.
2. Unlink the database and link it to source control again.
3. Go to the **Get latest** tab, select the affected objects, and click **Get latest**.

4. Re-add the references.

Failed to resolve no-ops after 5 tries

This error is shown on the Commit changes or Get latest tab when SQL Source Control can't determine the correct list of objects with changes. The error is logged in Redgate's bug-tracking system with the reference number *SOC-2670*.

Causes

If you're using the Vault Share command

Linking to a folder shared with the Vault Share command causes problems with SQL Source Control. You won't be able to commit or get changes.

To fix this, unlink the database, then link to the original Vault folder, not the shared copy. Committing changes to the original folder will automatically update the shared folder. If this doesn't work, steps to fix the error manually are described under **Fix** below.

When dependency issues leave the source control version of the database in an invalid state

SQL Source Control detects dependencies when you get or commit a set of objects. If someone commits or gets a change and doesn't include all its dependencies, the database in source control may be left in an invalid state. For example, you may have committed a view that selects from a table, but not committed the table it selects from.

When SQL Source Control encounters known issue SOC-2670

To determine which database objects have changes to commit or get, SQL Source Control maintains local copies of the objects creation scripts. These copies can get out of sync.

This may happen when you unlink and relink a database, or when two people are modifying the same objects.

Fix

The most common cause is that one of the local scripts folders, the working base, is incomplete. The working base is a representation of the database before you made your current changes, and is used to determine which changes are uncommitted. If an object is missing from the working base, or at an unexpected version, SQL Source Control can't determine which objects have changed.


For more information about the working base, see [How SQL Source Control works behind the scenes](#).


To fix the error, use your source control system to manually update the working base to the latest source control version (head). Alternatively, you can update the working base to a specific stable version.

To find the working base file path, on the **Setup** tab, expand the **Under the hood** section and click **Open working base folder**:


Under the hood

You might need to access SQL Source Control's folders to fix certain problems. [Hide](#)

 You should only modify these folders if you're following troubleshooting instructions.

For information about how to fix specific problems, see the [Troubleshooting documentation](#). 

Working base folder

A local copy of the database at the last time you committed or retrieved changes. [Learn more](#) 

Open working base

Transient folder

A local copy of the latest version in source control. [Learn more](#) 

Open transient

The working base folder opens in a new Explorer window.

Because the working base is used to detect uncommitted changes, modifying it affects which objects are displayed on the Commit changes, Get latest, and Undo tabs. We don't recommend you modify the working base unless advised by Redgate support, as you might lose or overwrite your work.

After updating the working base

Because the working base has changed, the list of objects with changes to commit, get, and undo also changes.

- The **Commit changes** tab shows the differences between the current database version and the updated working base. However, because your current database version is less recent than the current source control version, committing a change overwrites the current source control version of an object with the older version from your database.
- The **Get latest** tab shows no changes to get, even if there are changes. This is because SQL Source Control now considers the database to be at the latest version.
- The **Undo changes** dialog box shows the differences between the current database version and the updated working base. This means the working base version is more recent than the database version, so undoing a change updates an object to the latest source control version. Undo has become equivalent to Get latest.

To fix these issues, commit all the changes you've made and want to keep, then undo all other changes. This commits your changes and updates your database to the latest source control version, and synchronizes the local copies of the database schema, restoring normal SQL Source Control operation.

ICredentialsProvider is unset, therefore can't get

This error message is shown when linking your database to TFS or attempting to commit or get changes. This happens when SQL Source Control can't authenticate your TFS server.

Workarounds

Delete your SQL Source Control cached credentials file

1. Close SSMS.
2. Go to the SQL Source Control config files folder. By default, this is located at %localappdata%\Red Gate\SQL Source Control 3
3. Delete `Credentials.XML`.
4. Restart SSMS and link the database again. SQL Source Control should prompt for credentials.

Delete your cached credentials

1. Close SSMS.
2. In Control Panel, go to Credential Manager (or Network Credentials, depending on your version of Windows).
3. Remove your TFS credentials.
4. Restart SSMS and link the database again. SQL Source Control should prompt for credentials.

Manually create TFS credentials

1. Close SSMS.
2. In Control Panel, go to Credential Manager (or Network Credentials, depending on your version of Windows).
3. Add a new credentials entry for your TFS sever, user name, and password.
4. Restart SSMS and link the database again. SQL Source Control should prompt for credentials.

Newer version of Proc found - please update your Source Control

When using a [shared database](#), all users connected to the database must use the same version of SQL Source Control. This error is displayed when someone on your team updates their SQL Source Control installation to a later version.

Fix

Update SQL Source Control to the latest version. To do this, go to **Help**, then **Check for updates**.

SQL Source Control can't access this database because the directory once contained more than one SQL Server Database Project (.sqlproj) file

When SQL Source Control runs its first get latest from a repository, it looks for the earliest revision where the .sqlproj file exists to determine the starting state of the project.

If two or more .sqlproj files ever existed in the same directory in the repository, the error occurs even if the extra .sqlproj files were removed in later revisions.

Workaround

You can work around the problem by manually updating the working base. You only need to do this after you first link the database.

1. On the **Setup** tab, expand **Under the hood** and click **Open working base**.
The working base opens in Windows Explorer.
2. Close Management Studio.
3. Manually update the working base to the latest revision in source control with your source control client.
For example, if you're using Subversion with TortoiseSVN, right-click and select **SVN Update** to update the directory to the latest version in your repository.
4. Open SQL Server Management Studio and SQL Source Control.
Any differences between your database and the latest version in your repository are shown as changes to commit on the Commit tab.
5. If you linked a blank database, SQL Source Control will display all the objects in the project as dropped in the Commit tab.
Highlight all the objects, right-click, and select **Undo Changes**.

Your database now represents the latest version in the repository.

If this doesn't fix the problem, contact [Redgate support](#).

System.OutOfMemoryException

This error means there isn't enough free memory for SQL Source Control to work. It has several possible causes:

Management Studio is using too much memory

If Management Studio is using more than around 700MB-1GB of memory, this might be causing the problem. You can see how much memory Management Studio uses in Task Manager.

If you have Management Studio add-ins installed that you don't use (eg from Redgate's [SQL Developer Bundle](#)), uninstalling them may free memory.

The static data tables are too large

Source-controlling very large [static data](#) tables uses lots of memory, so we don't recommend it.

The database is too large

If the SQL created from the objects in your database is very long, this can cause the error. The long SQL might be created by having lots of long stored procedures; for example, stored procedures including long comment blocks.

You might need to restructure the database to make it smaller, or remove comment blocks from stored procedures.

There isn't enough space on disk

Redgate tools store temporary data in the %TMP% folder. By default, this folder is in your Windows profile folder (eg `C:\Users\<username>\Temp`). If you don't have much space on the hard drive, this folder might fill up and cause memory errors.

To fix this, Redgate tools can use a different hard drive. For example, to use the folder *RGtemp* on drive D:

1. On the Start menu, right-click **Computer** and select **Properties**.
2. In the **Advanced** tab, click **Environment Variables**.
3. Click **New**.
4. In the **Variable name** field, type: *RGTEMP*
5. In the **Variable value** field, type: *D:\RGtemp*
6. Click **OK**.
7. Close Management Studio and any Redgate tools you have open.
8. Log out of Windows and log in again.

Redgate tools will use the folder you specified to save temporary data. To test this, you can commit a change in SQL Source Control (or run an operation in another Redgate tool) and check that files are added to the new location.

To undo this and have Redgate tools use the default folder, delete the RGTEMP variable from the Environment Variables list and repeat steps 7 and 8.

The specified path, file name, or both are too long

This error occurs if you have very long object names or directory names in Windows. To fix it, [relocate your working base folder](#).

The Team Foundation Server for this workspace does not support one or more of the checkin options you have selected

This error means your version of Team Explorer is incompatible with your version of Team Foundation Server (TFS).

This table shows the compatibility between different versions:

| | TFS 2012 | TFS 2010 | TFS 2008 | TFS 2005 |
|--------------------|----------|----------|----------|----------|
| Team Explorer 2012 | ✓ | ✓ | ✗ * | ✗ |
| Team Explorer 2010 | ✗ * | ✓ | ✓ | ✗ |
| Team Explorer 2008 | ✗ * | ✗ * | ✓ | ✓ |
| Team Explorer 2005 | ✗ | ✗ * | ✓ | ✓ |

Versions marked with an asterisk (*) might be compatible if you install [SP1 and the GDR](#) from MSDN. However, we haven't tested this with SQL Source Control, and we can't guarantee it will work.

For more information about TFS client/server compatibility, see [Compatibility between Team Foundation Clients and Team Foundation Server](#) on MSDN.

Fix

Change which version of Team Explorer that SQL Source Control uses by modifying a config file.

1. Make sure you've installed a version of Team Explorer that's compatible with your version of TFS.

You don't need to uninstall any later incompatible versions.

2. Make sure Management Studio is closed.
3. Go to the SQL Source Control config files folder: `%localappdata%\Red Gate\SQL Source Control 3`
4. Open `RedGate_SQLSourceControl_Engine_EngineOptions.xml` in a text editor.
5. Inside the `EngineOptions` version tags, add:

```
<TeamFoundationServerDllOverride>RedGate.SQLSourceControl.Engine.SrcC.TFSxxxx</TeamFoundationServerDllOverride>
```


`xxxx` is the version of Team Explorer you want SQL Source Control to use.

Example

```
<EngineOptions version="3" type="EngineOptions">
  <TeamFoundationServerDllOverride>RedGate.SQLSourceControl.Engine.SrcC.TFS2010</TeamFoundationServerDllOverride>
</EngineOptions>
```

6. Save the file.

This commit doesn't meet the server's policy requirements, or the policy isn't configured on your machine

When committing a change to TFS, the commit fails with the message: "This commit doesn't meet the server's policy requirements, or the policy isn't configured on your machine."

About TFS policies

Policies are set by your team's server administrator to make sure commits meet requirements they specify. Commits that don't meet the policy requirements are rejected by the server.

Examples of typical policies

- Commits must have comments.
- Files must have comments.
- Commits must be associated with a [TFS work item](#).

Possible causes and fixes

Your commit doesn't meet the server's policy requirements

- Depending on the policy, the server might provide a message describing the requirements you haven't met. Make sure your commit meets the policy requirements.

Associating commits with work items

If the policy needs you to associate your commit with a TFS work item, you can do this by including `#A[Work Item number]` at the **start** of the comment. For example: `#A106 adding tables`

For more information, see [Integrating with bug tracking systems](#).

- To ignore policy requirements for a commit, include `#ignorepolicies` in the commit comment. If you want to ignore policy requirements for all future commits, you can [turn off TFS policy checking](#).

You're using an old version of SQL Source Control

Some erroneous causes of this problem are fixed in the current version of SQL Source Control. Update to the most recent version.

The policy isn't installed on your machine

Policies must be installed on the server **and** each machine making commits to the server.

Make sure every policy is installed on your machine. For example, you may be missing a custom policy created by your team, or a policy installed with [TFS Power Tools](#).

Missing .dll file

You might be missing a required .dll file.

- If the policy was set up with Visual Studio 2012, you need `Microsoft.TeamFoundation.VersionControl.Controls.dll`. This is installed with the default Visual Studio 2012 installation. You also need to edit the registry (see **Your registry isn't configured** below).
- If the policy was set up with an older version of Visual Studio, make sure [TFS Power Tools](#) is installed.

SQL Source Control is using the wrong version of Team Explorer

You might have set up your policy for a version of Team Explorer that SQL Source Control isn't using.

To fix this, follow the instructions under **Fix** on [this troubleshooting page](#).

Your registry isn't configured

If the policy was set up with Visual Studio 2012, you need to edit a registry key so your TFS client can locate `Microsoft.TeamFoundation.VersionControl.Controls.dll`.

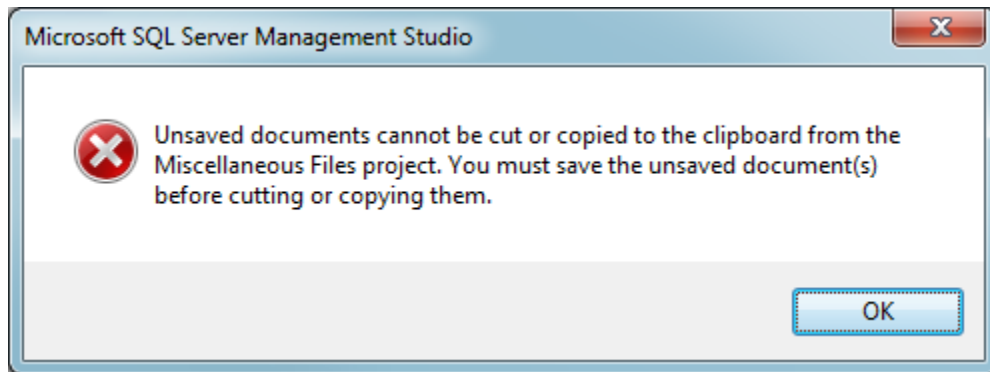
To do this:

1. In the Windows Registry Editor, go to: *HKEY_LOCAL_MACHINE\SOFTWAREWow6432Node\Microsoft\VisualStudio\11.0\TeamFoundation\SourceControl\Checkin Policies*
2. Open *Microsoft.TeamFoundation.VersionControl.Controls* and set the value to: *<your Visual Studio installation path>\Common7\IDE\CommonExtensions\Microsoft\TeamFoundation\Team Explorer\Microsoft.TeamFoundation.VersionControl.Controls.dll*

The default Visual Studio installation path is: *C:\Program Files (x86)\Microsoft Visual Studio 11.0*

Unsaved documents cannot be cut or copied to the clipboard from the Miscellaneous Files project

Management Studio sometimes displays this error when you try to copy text from SQL Source Control using the Ctrl + C keyboard shortcut.

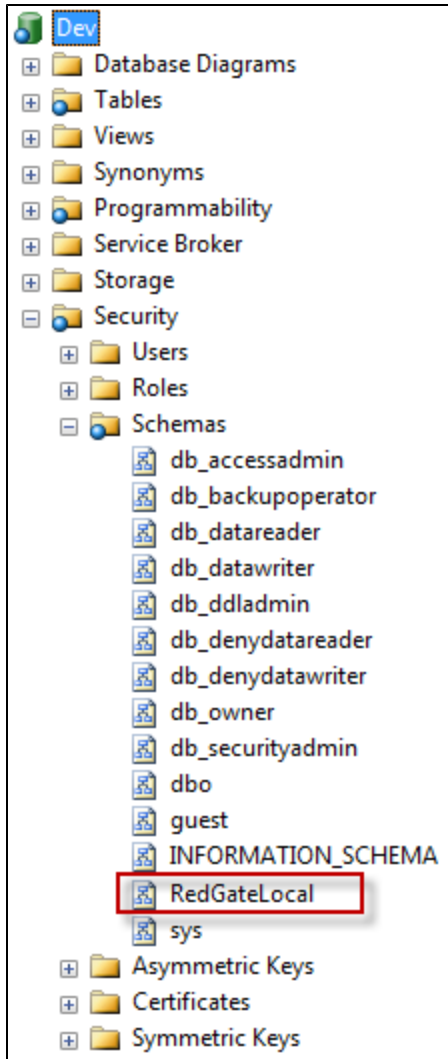


Workaround

Copy the text using the right-click menu.

About the RedGateLocal schema

If you're using [Deployment Manager 2](#), Deployment Manager adds the *RedGateLocal* schema to your database when you deploy or package it. The schema requires no maintenance, and you can safely ignore it.



The RedGateLocal schema contains a table-valued function that's updated when you deploy or package the database. Deployment Manager uses the table-valued function to save information about the database version.

The schema isn't shown in the Commit or Get latest tabs, or anywhere else in the SQL Source Control interface. This is because source control is itself a better way to record database versions, and sharing the schema with others might cause problems with Deployment Manager.

Third-party software used by SQL Source Control

SQL Source Control uses:

- [Fleck](#) (distributed under the [MIT license](#))
- [Json.NET](#) (distributed under the [MIT license](#))
- [Ninject](#)
- [SharpSvn](#) (distributed under the [Version 2.0 Apache license](#))
- [SQLite](#)
- [Vault](#) source control libraries

Release notes and other versions

To install an old version of SQL Source Control, see [Download old versions of products](#).

For release notes for Frequent Updates, see [Frequent Update release notes](#).

| | | | |
|------------------------------|-----------------------|-------------------------------|-------------------------------------|
| Version 4.1 (current) | December 2nd (latest) | Release notes | Documentation |
| Version 4.0 | August 6th, 2015 | Release notes | |
| Version 3.8 | May 14th, 2015 | Release notes | Documentation |
| Version 3.7 | October 21st, 2014 | Release notes | |
| Version 3.6 | June 3rd, 2014 | Release notes | |
| Version 3.5 | February 11th, 2014 | Release notes | |
| Version 3.4 | September 11th, 2013 | Release notes | |
| Version 3.3 | April 18th, 2013 | Release notes | |
| Version 3.2 | March 13th, 2013 | Release notes | |
| Version 3.1 | September 13th, 2012 | Release notes | |
| Version 3.0 | February 1st, 2012 | Release notes | |
| Version 2.2 | December 15th, 2011 | Release notes | Documentation (PDF) |
| Version 2.1 | March 30th, 2011 | Release notes | Documentation (PDF) |
| Version 1.1 | September 17th, 2010 | Release notes | Documentation (PDF) |
| Version 1.0 | June 16th, 2010 | Release notes | Documentation (PDF) |

Follow [@SQLSocUpdates](#) on Twitter to see when updates are released.

Loading Twitter Widget

Widget #449140573912330240

Frequent Update release notes

To install an old version of SQL Source Control, see [Download old versions of products](#).

| | | | |
|------------------------------|------------------------|-------------------------------|-------------------------------|
| Version 4.1 (current) | December 23rd (latest) | Release notes | Documentation |
| Version 4.0 | September 16th, 2015 | Release notes | |
| Version 3.8 | July 22nd, 2015 | Release notes | Documentation |
| Version 3.7 | October 21st, 2014 | Release notes | |

Follow [@SQLSocUpdates](#) on Twitter to see when updates are released.

Loading Twitter Widget

Widget #449140573912330240

SQL Source Control 3.8 Frequent Updates release notes

To get Frequent Updates, turn on Frequent Updates in Check for Updates.

3.8.21 - July 22nd 2015

In this release, the object locking preview feature has been disabled. Object locking will be released as a full feature in SQL Source Control 4.

3.8.20 - July 15th 2015

Features

- Tooltip on hover for truncated strings on the Object locking tab:

| <input type="checkbox"/> | Owner | WidgetDescriptions | type | Locke... | Date |
|--------------------------|-------|--------------------|-------|----------|-----------|
| <input type="checkbox"/> | dbo | Widget... | Table | RED-... | 15/07/... |
| <input type="checkbox"/> | dbo | Widget... | Table | RED-... | 15/07/... |
| <input type="checkbox"/> | dbo | Widget... | Table | RED-... | 15/07/... |

Fixes

- SOC-6692: you can now view history when an object has been renamed in TFS

3.8.19 - July 8th 2015

Fixes

- SOC-4366: Stop spurious InvalidOperationExceptions
- SOC-6534: Now performs drop-only commits to Visual Studio Online without errors
- Arrow keys no longer tab out of the search bar
- Fixed caching performance regressions which appeared in SQL Source Control 3.8.13

3.8.18 - July 1st 2015

Features

Improvements to Object locking tab:

- "No locked objects in this database" displayed when you select a database with no locked objects
- Simplified text and layout
- Lines are now aligned with checkboxes
- Added delete button to search box
- Removed "X of Y objects selected" text
- Number of search results displayed next to search box
- Swapped Unlock and Refresh buttons

Fixes

- Launching SQL Source Control from a desktop shortcut no longer creates a temporary file on the desktop

3.8.17 - June 24th 2015

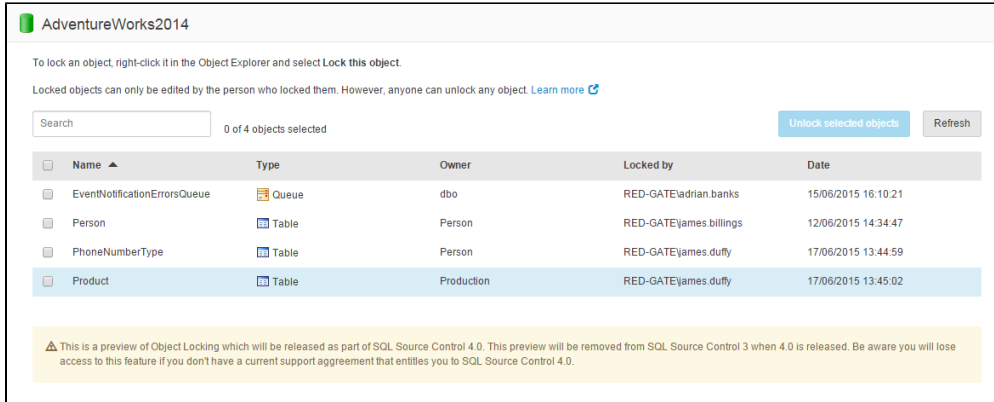
Fixes

- SOC-6955: SOC no longer throws an exception when connecting to Analysis Services or Integration Services

3.8.16 - June 17th 2015

Features

- You can now **lock objects** you're working on so other people can't edit them. This means teams that share databases don't accidentally overwrite work.



This is a preview of a feature that will be released in SQL Source Control 4. When SQL Source Control 4 is released later this year, the object locking preview will be removed from SQL Source Control 3.

3.8.15 - June 10th 2015

Fixes

- SOC-6666: Fixed error on unlinking

3.8.14 - June 3rd 2015

Features

- Support for SSMS 2016 CTP2 (not including support for SQL Server 2016 CTP2)

Fixes

- SOC-6893: Migrations scripts (V1) are now picked up on Get latest
- SC-5151: XML schema collections that have additional child nodes are now detected as different
- SC-5533: Permission changes for system users (eg 'guest') are now ignored when the "ignore users' permissions and role memberships" option is selected
- SC-7896: When a service's queue changes, the deployment script no longer contains a duplicate ALTER SERVICE statement
- SC-7897, SC-7898, SC-7900, SC-7901, SC-7903, SC-7907: Square brackets in object names are now escaped in various remaining places
- SELECT statements containing a 'WITH (SNAPSHOT)' query hint no longer cause parsing errors
- Pressing N on the Setup tab no longer opens a new query window

3.8.13 - May 27th 2015

Features

- Filters now support the standard T-SQL LIKE syntax for wildcard characters (eg %, __, [abc], and [^abc] - see the [MSDN documentation](#) for more information). To include wildcard characters explicitly in a filter, escape them with square brackets. "Equals" and "Does not equal" operators won't treat the characters as wildcard characters.

Fixes

- SQL Source Control no longer needs Management Studio to be restarted after bringing a database online
- SOC-6546: SQL Source Control no longer throws an exception when it's unlicensed or the trial period has expired
- SOC-2532: Filtering the character % no longer causes additional objects to be filtered
- SOC-3904: Filtering objects that begin with "_" no longer filters out all objects

3.8.12 - May 14th 2015

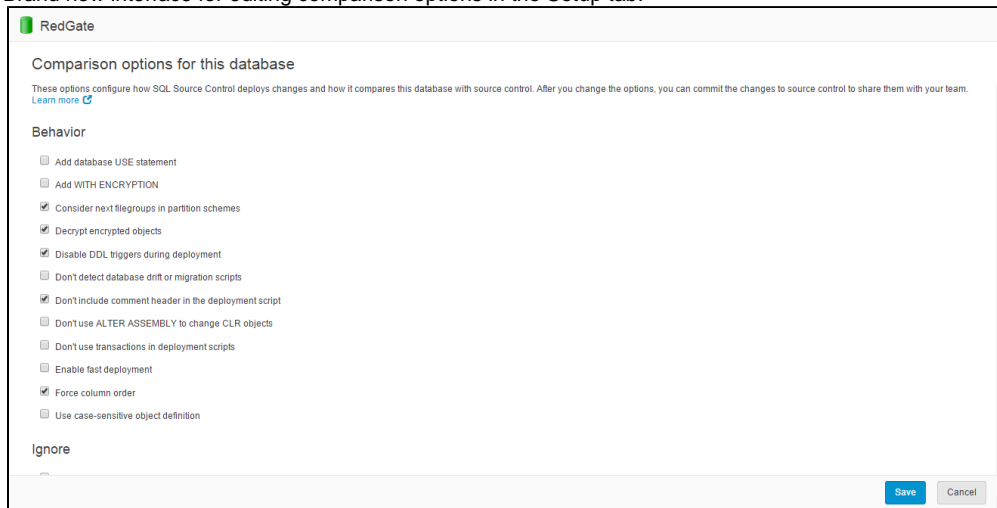
Fixes

- SOC-4635: TFS users should no longer receive the error "PendDelete returned an unexpected number of files"
- All child processes are stopped when SQL Source Control is closed

3.8.10 - April 29th 2015

Features

- Brand new interface for editing comparison options in the Setup tab:



Fixes

- SOC-6777. You can now get latest when adding an extended property to a default schema
- SOC-6734. The SVN link dialog window no longer disappears when certain erroneous paths are entered

3.8.9 - April 22nd 2015

Fixes

- SOC-6514: STDERR output from command line source control systems is now displayed when errors happen
- SOC-6798: When the SQL Source Control History dialog window is opened from SQL Compare, the `Select` button is no longer misaligned with the `Cancel` button

3.8.8 - April 10th 2015

Fixes

- SOC-6650: Database icon in Object Explorer Details pane now has correct transparency
- SOC-6748: Invalid SQL no longer causes runaway `log file` growth
- SOC-6746: Users no longer need to restart Management Studio when unlinking and relinking databases
- SOC-4366: InvalidOperation Exception no longer thrown in some circumstances when refreshing the Commit tab
- SOC-6515: Migration scripts no longer erroneously appears as an edit when there are no differences to commit

3.8.7 - April 7th 2015

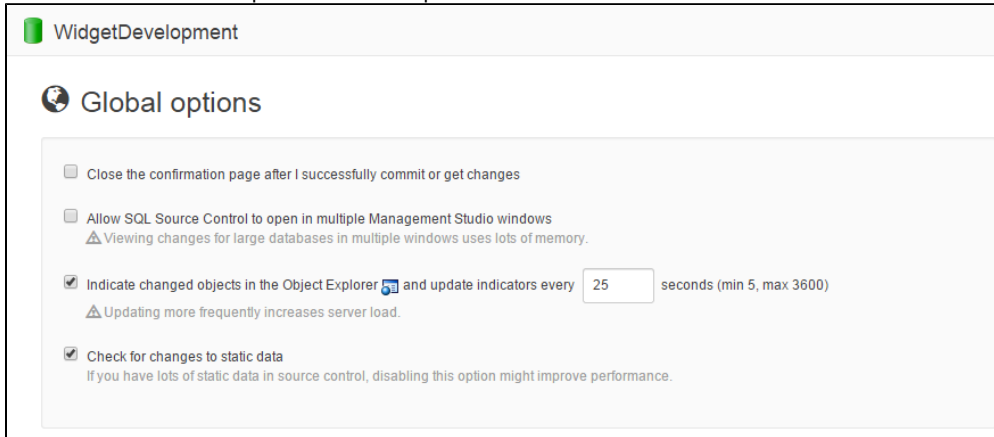
Fixes

- Corrected scrollbar on the Setup tab
- SOC-6487. Merging a stored procedure no longer drops the procedure when the merge output contains invalid SQL
- SOC-2865. Corrupt XML configuration files are now backed up and reset to their default rather than producing an error
- SOC-6720. The SQL differences are now shown when viewing history of a branch in SVN.

3.8.6 - March 18 2015

Features

- Edit SQL Source Control options on the Setup tab:



These options were previously only accessible by editing a config file.

- Access Edit filter rules and Link or unlink static data dialogue boxes from the Setup tab
- SQL Source Control is now listed in the SSMS Integration Pack add-ins tab

Fixes

- Buttons on the Setup tab now change color on hover
- Management Studio keyboard shortcuts now work on the Setup tab
- "Select database to get started" message when no database is selected in the Object Explorer
- SOC-6420: FileNotFoundException error no longer occurs when Connect project is present
- SOC-6530: TFS file locks are now correctly lifted when a commit fails due to an unresolved conflict
- SOC-6680/6681: When linked to TFS or Vault, the repository and folder locations are now shown on the Setup tab
- SOC-6686. SQL Source Control no longer reopens when Management Studio starts if it was closed last time you closed Management Studio
- SOC-6712: The revision number is now shown on the commit page after a successful commit

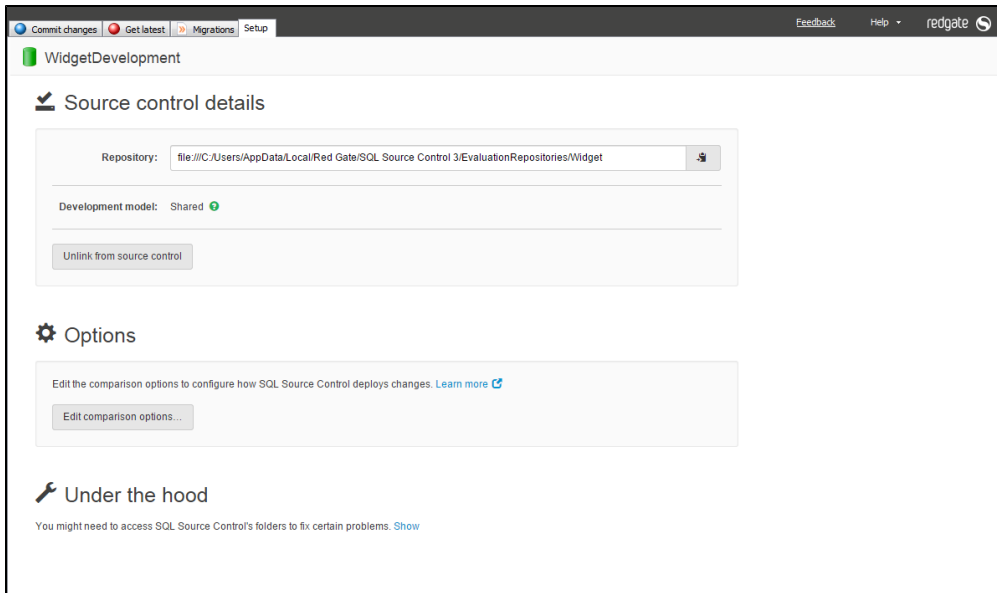
3.8.4 - February 25th 2015

Note

As of this release, SQL Source Control requires .NET Framework 4 or later.

Features

- Brand new Setup tab:



⚠️ Known issue

- Management Studio shortcuts don't work on the new Setup tab

Fixes

- SOC-6615: Changes to views with indexes with DATA_COMPRESSION="PAGE" are no longer detected as changes
- SOC-6584: "Could not find file" exception, which was hiding connectivity issues, is now suppressed
- SOC-6626, SOC-6638: Databases with braces in the name no longer produce errors when you expand them in the Object Explorer
- SOC-6628: "Right-click database in Object Explorer -> Schema or Data Compare deploy -> Set as source/target" no longer throws an exception
- SC-7647: Improved temporary file cleanup
- Fixed issue tree conflict issue in SVN. This used to occur when committing a change that created a new folder that had also been created in the remote repository, and mostly affected SSDT users

3.8.3 - February 3rd 2015

Features

- Added new-look header bar and new logos:



- Removed the *UseMigrationsV2* comparison option. *The Migrations V2 beta is now enabled only using the UseMigrationsV2 engine option*
- Linking to an SVN repository that contains lots of objects is faster
- Updated Vault client libraries to 8.0.1

Fixes

- MIG-6: Temporary database creation no longer fails on machines with certain regional settings
- SOC-2010: Offline databases are no longer detected as having an old SQL Server version
- SOC-6535: Migrations V1 scripts can now be deleted when linked to TFS
- SOC-6587: Running high display scaling levels in Windows no longer makes checkboxes in "Link/unlink static data" invisible
- SC-7060: Now supports pass-through commands using EXECUTE syntax

3.8.1 - December 18th 2014

Fixes

- SOC-6423: Migrations V2 role membership now ordered correctly
- SOC-6488 and SOC-6499: TFS 2013 link/commit issues fixed
- MIG-43: Migrations V2 can now detect and use LocalDB 2014

SQL Source Control 3.7 Frequent Updates release notes

To get Frequent Updates, turn on [Frequent Updates](#) in [Check for Updates](#).

3.7.7 - October 21st, 2014

Features

- Added CaseSensitiveObjectDefinition option to the [comparison options](#) so users with case-insensitive databases can commit and retrieve case-only changes (requested on our [feedback forum](#))
- When getting changes to memory optimized objects (SQL 2014), SQL Source Control temporarily warns you and disables the use of transactions.
- Improved error messages when the [Migrations V2 beta](#) fails to copy the target database

Bug fixes

- [SOC-6351](#): Fixed bug when handling objects with case-only difference in their name with case sensitivity option turned on
- Fixed bug using non-active-directory authentication for the "Direct from source control" option in SQL Compare
- [SOC-5907](#): Fixed foreign keys referencing a FileTable
- Improved computed column heuristics causing Data Compare to fail

3.7.6 - September 30th, 2014

Fixes

- Fixed repeated authentication prompts for some TFS URLs
- Fixed invalid pointer that stopped SQL Source Control loading ("the add-in failed to load")
- [SOC-5179](#) and [SC-6484](#): improved support for spatial indexes
- [SC-6986](#): fixed "Data Compression not recognized" error for clustered columnstore indexes
- [SC-7464](#): user-defined table types no longer cause duplicate extended properties

SQL Source Control 3.8 release notes

3.8.11 - May 14th 2015 [DOWNLOAD >](#)

Note

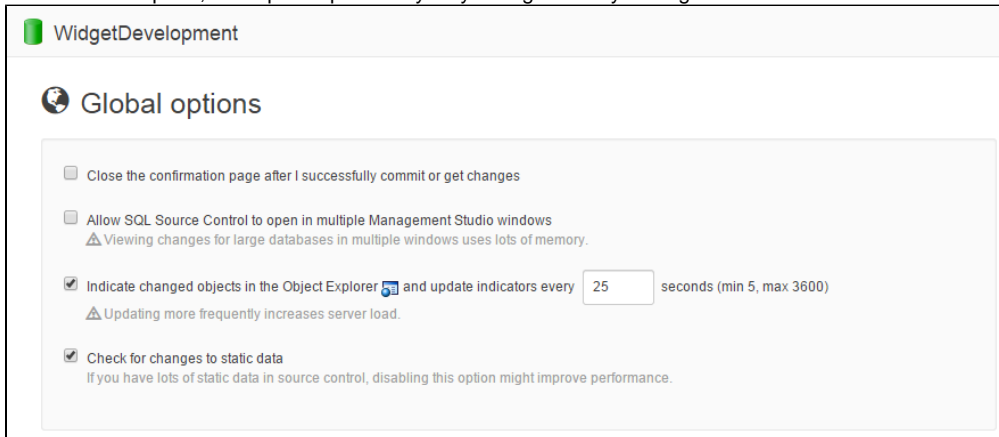
As of this release, SQL Source Control requires .NET Framework 4 or later.

Interface features

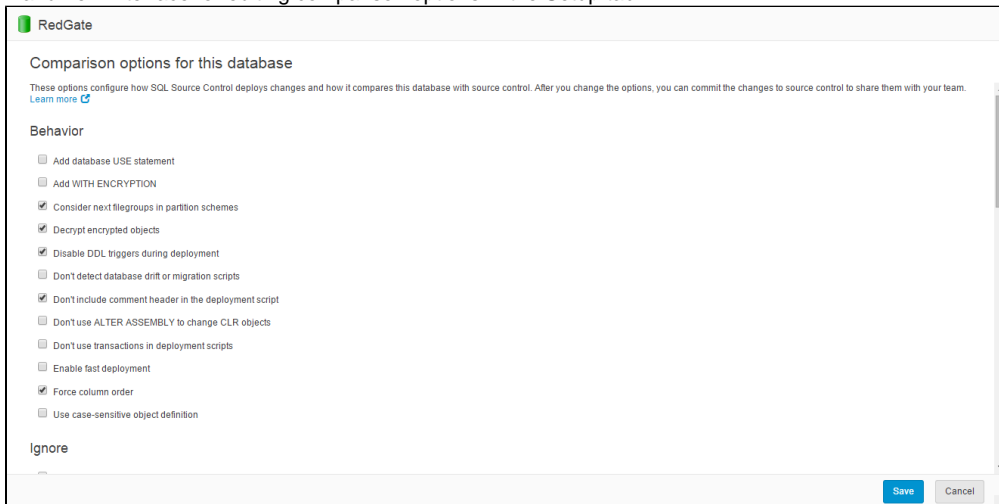
- New-look header bar, logos and About box:



- Brand new Setup tab, with options previously only configurable by editing text files:



- Brand new interface for editing comparison options in the Setup tab:



- Access **Edit filter rules** and **Link or unlink static data** dialogue boxes from the Setup tab
- SQL Source Control is now listed in the SSMS Integration Pack add-ins tab
- "Select database to get started" message when no database is selected in the Object Explorer

Other features

- Removed the *UseMigrationsV2* comparison option. [The Migrations V2 beta](#) is now enabled only using the *UseMigrationsV2* engine option
- Linking to an SVN repository that contains lots of objects is faster
- Updated Vault client libraries to 8.0.1
- SQL Source Control is now listed in the SSMS Integration Pack add-ins tab

Fixes

- Fixed bug using non-active-directory authentication for the “Direct from source control” option in SQL Compare
- Fixed bug when handling objects with case-only difference in their name with case sensitivity option turned on
- Fixed invalid pointer that stopped SQL Source Control loading (“the add-in failed to load”)
- Fixed issue tree conflict issue in SVN. This used to occur when committing a change that created a new folder that had also been created in the remote repository, and mostly affected SSDT users
- Fixed repeated authentication prompts for some TFS URLs
- MIG-43: Migrations V2 can now detect and use LocalDB 2014
- MIG-6: Temporary database creation no longer fails on machines with certain regional settings
- Re-added “Throw exceptions on SQL parser errors” option to the comparison options (accidentally removed in previous version)
- SC-1196: Stored procedures that depend on a full text index can now be retrieved
- SC-6986: fixed “Data Compression not recognized” error for clustered columnstore indexes
- SC-7060: Now supports pass-through commands using EXECUTE syntax
- SC-7464: User-defined table types no longer cause duplicate extended properties
- SC-7647: Improved temporary file cleanup
- SOC-2010: Offline databases are no longer detected as having an old SQL Server version
- SOC-2865: Corrupt XML configuration files are now backed up and reset to their default rather than producing an error
- SOC-4366: InvalidOperationException no longer thrown in some circumstances when refreshing the Commit tab
- SOC-5179 and SC-6484: improved support for spatial indexes
- SOC-5450: The TFS link dialog no longer disappears when certain erroneous paths are entered
- SOC-6230: Fix error when opening SQL Source Control from the Object Explorer
- SOC-6260: Reduce instances of workspace locking errors
- SOC-6420: FileNotFoundException error no longer occurs when Connect project is present
- SOC-6423: Migrations V2 role membership now ordered correctly
- SOC-6487: Merging a stored procedure no longer drops the procedure when the merge output contains invalid SQL
- SOC-6488 and SOC-6499: TFS 2013 link/commit issues fixed
- SOC-6514: STDERR output from command line source control systems is now displayed when errors happen
- SOC-6515: Migration scripts no longer erroneously appears as an edit when there are no differences to commit
- SOC-6530: TFS file locks are now correctly lifted when a commit fails due to an unresolved conflict
- SOC-6535: Migrations V1 scripts can now be deleted when linked to TFS
- SOC-6584: “Could not find file” exception, which was hiding connectivity issues, is now suppressed
- SOC-6587: Running high display scaling levels in Windows no longer makes checkboxes in “Link/unlink static data” invisible
- SOC-6615: Changes to views with indexes with DATA_COMPRESSION=“PAGE” are no longer detected as changes
- SOC-6626, SOC-6638: Databases with braces in the name no longer produce errors when you expand them in the Object Explorer
- SOC-6628: “Right-click database in Object Explorer -> Schema or Data Compare deploy -> Set as source/target” no longer throws an exception
- SOC-6650: Database icon in Object Explorer Details pane now has correct transparency
- SOC-6680/6681: When linked to TFS or Vault, the repository and folder locations are now shown on the Setup tab
- SOC-6686: SQL Source Control no longer reopens when Management Studio starts if it was closed last time you closed Management Studio
- SOC-6712: The revision number is now shown on the commit page after a successful commit
- SOC-6720: The SQL differences are now shown when viewing history of a branch in SVN.
- SOC-6734: The SVN link dialog window no longer disappears when certain erroneous paths are entered
- SOC-6746: Users no longer need to restart Management Studio when unlinking and relinking databases
- SOC-6748: Invalid SQL no longer causes runaway [log file](#) growth
- SOC-6777: You can now get latest when adding an extended property to a default schema
- SOC-6798: When the SQL Source Control History dialog window is opened from SQL Compare, the Select button is no longer misaligned with the Cancel button

3.8.2 - January 13th 2015

Features

- Vault 8 support
- Added CaseSensitiveObjectDefinition option to the comparison options so users with case-insensitive databases can commit and retrieve case-only changes
- When getting changes to memory-optimized objects (SQL 2014), SQL Source Control warns you and temporarily disables transactions
- Improved error messages when the Migrations V2 beta fails to copy the target database

Fixes

- SOC-6230: Fix error when opening SQL Source Control from the Object Explorer
- SOC-6260: Reduce instances of workspace locking errors
- Fixed repeated authentication prompts for some TFS URLs
- Fixed invalid pointer that stopped SQL Source Control loading (“the add-in failed to load”)
- SOC-5179 and SC-6484: improved support for spatial indexes
- SC-6986: fixed “Data Compression not recognized” error for clustered columnstore indexes
- SC-7464: user-defined table types no longer cause duplicate extended properties
- Fixed bug when handling objects with case-only difference in their name with case sensitivity option turned on
- Fixed bug using non-active-directory authentication for the “Direct from source control” option in SQL Compare

- SOC-6423: Migrations V2 role membership now ordered correctly
- SOC-6488 and SOC-6499: TFS 2013 link/commit issues fixed
- MIG-43: Migrations V2 can now detect and use LocalDB 2014

SQL Source Control 3.7 release notes

3.7.5 - 30th September, 2014

Fixes

- "Invalid HTTPS certificate approval needed" dialog box now remembers certificates you approve
- SOC-6258: removed incorrect "duplicate objects found with name" report

3.7.4 - 27th August, 2014

Features

- Uses latest SharpSvn libraries

Fixes

- WITH (STATISTICS_INCREMENTAL=OFF) is no longer added to tables on SQL Server 2014
- SOC-6245: Reverting a change no longer causes SQL Source Control to hang when using the shared model
- SOC-6106: Inaccurate duplicate object exceptions
- SC-6992, SC-7305, SC-735: Improved compatibility with memory-optimized tables
- Miscellaneous SQL Compare engine bug fixes: codes SC-5853, SC-6605, SC-6953, SC-7023, SC-7378, SC-7382, SC-7399, SC-7402, SC-7403, SC-7423, SC-7424

3.7.3 - 6th August, 2014

Fixes

- SOC-6172: Clicking Browse when linking a database to a Vault server no longer produces an error
- SOC-6125: Instances of SQL Source Control "hanging" during commit or get latest reduced
- SOC-6163: Rebuilding a table with a dependent view no longer produces an InvalidOperationException error with the message "Unknown object type for existence check"
- Switching between the Commit and Get latest tabs during source control operations now momentarily pauses SQL Source Control instead of causing a crash

Known issues

- WITH (STATISTICS_INCREMENTAL=OFF) is added to tables on SQL Server 2014
- If you're using the [shared model](#), undoing a change causes SQL Source Control to stop working

3.7.2 - 28th July, 2014

Features

- Updated SQL Compare engine

Fixes

- SOC-6047: SQL Compare command line can now connect to SVN on x64 machines

3.7.1 - 7th July, 2014

Features

- Improved performance when getting changes from source control

Fixes

- SOC-6095: The Vault file explorer now refreshes when creating a new folder during linking to source control
- Various fixes for SQL Source Control being reported as a false positive by antivirus programs

3.7.0 - 7th July, 2014

Fixes

- Git and Mercurial options are removed from the **Link to source control** dialog, as these were simply duplicates of the Working folder option. To link a database to Git or Mercurial, select [Custom](#)
- Databases with many complex stored procedures no longer encounter "out of memory" errors
- Fixed a major cause of "out of memory" errors when previewing a series of static data tables. Previewing very large static data tables on the Commit tab may still produce "out of memory" errors in some cases

SQL Source Control 3.6 release notes

3.6.7 - June 3rd 2014

Features

- Select two database versions from the History dialog and compare them in SQL Compare or SQL Data Compare

Fixes

- TFS workspace mapping errors now produce an error message with troubleshooting advice

3.6.6 - June 3rd 2014

Performance improvement

- Faster "Updating meta-data" step when using TFS

Fixes

- SOC-2566: Errors no longer occur when committing certain static data changes to a Mercurial, Git, or working folder repository, or custom source control system
- SOC-3942: Manually editing Get Latest script in SSMS 2014 no longer causes "Object reference not set to an instance of an object" error
- SOC-4084: "Item with the same key has already been added" error no longer occurs when committing static data changes. If you've previously encountered this error, you may need to relink the database
- SOC-5640: Using SQL 2012 THROW keyword with no arguments inside an IF clause no longer causes "Errors occurred whilst parsing file" error
- SOC-5653: SQL Source Control now identifies which user made some database changes that were previously reported as "Unknown". This includes:
 - objects created and subsequently edited
 - objects created with the same name as an object in source control

3.6.5 - May 14th 2014

Fixes

- SOC-5653: SQL Source Control now identifies which user made some database changes that were previously reported as "Unknown". This includes:
 - objects created and subsequently edited
 - objects created with the same name as an object in source control

3.6.4 - May 6th 2014

Performance improvements

- Linking databases to SVN is faster

Fixes

- SOC-2566: ConflictException error no longer occurs when committing to a working folder, Mercurial or Git repository, or custom source control system
- SOC-4084: "Item with the same key has already been added" error no longer occurs when committing static data changes. If you've previously encountered this error, you may need to relink the database

3.6.3 - April 29th 2014

Fixes

- SOC-4087: Unlinking static data on an empty conflicted table no longer produces the "given key was not present in the dictionary" error
- SOC-5179: Fixes an issue with parsing a spatial index referencing a computed column
- SOC-5812: SQL Compare users can now link to Vault repositories
- SOC-5680: Conflicted stored procedures can now be merged when using SQL Server authentication

Known issues

- SOC-5858: Vault shared folders aren't supported

3.6.0 - April 14th 2014

Features

- Support for Team Foundation Server 2013
- Support for SQL Server 2014 databases
- Support for SQL Server Management Studio 2014
- Support for Vault 7

Fixes

- TFS users no longer need to modify a config file to link to TFS 2012 or later
- When [information about who make a change is lost](#) because of a server timeout, the Commit tab no longer becomes unusable

Known issues

- New SQL syntax introduced in SQL Server 2014 is unsupported

SQL Source Control 3.5 release notes

3.5.6 - February 11th, 2014

Performance improvements

- The History dialog box now works faster with large repositories

Fixes

- SOC-2238: DirectoryNotFoundException during working base recovery.
- SOC-3636: no longer writes out duplicate trigger statements causing "An item with the same key has already been added" error.
- SOC-5288/SOC-5313: "ICredentialsProvider is already unset" error no longer occurs when using different versions of TFS.
- SOC-5692: folder level history now only displays history for objects within that folder.
- SOC-5694: two foreign keys referencing the same table, which have the same name but in different schemas, no longer cause "An item with the same key has already been added" error
- The History dialog box now shows horizontal scrollbars in difference viewer.

3.5.5 - January 23rd, 2014

Features

Compatibility with [Deployment Manager](#):

- [Package a database revision using Deployment Manager from the SQL Source Control History dialog box](#)
- Objects in the RedGateLocal schema (used by Deployment Manager) aren't committed to source control. For more information, see [About the RedGateLocal schema](#)

Fixes

- SOC-2368: Mapping conflict error no longer occurs in certain circumstances when using TFS
- SOC-4360: When using the shared model, if one user upgrades SQL Source Control, the error "Newer version of Proc found - please update your Source Control" no longer occurs when others in the team try to commit
- SOC-5620: Error "Encountered unknown SSMS version 12.0.0.0" no longer occurs when running SSMS 2014

Known issues

- If you use the [shared development model](#), any uncommitted changes when your team installs this version will be listed as having been changed by "Unknown" on the Commit tab. To prevent this, make sure all changes are committed before upgrading to this version.

3.5.4 - December 18th, 2013

Fixes

- Fixed TFS policy errors some users encountered. A version of this fix was previously released via the TFS policy beta release. If you're still having problems, you may need to follow the instructions [here](#)
- SOC-4510: Fixed some occurrences of "Prior to sync there should be some selected changes" error after committing filters

3.5.3 - November 20th, 2013

Fixes

- Fixed bug introduced in previous version (3.5.2.350) where databases containing encrypted objects caused an error on Commit or Get latest

3.5.2 - November 18th, 2013

Bug fixes

- Fix for SOC-4814 and SOC-5332: foreign key erroneously dropped on commit
- Fix for SOC-4679: error "The retrieve script will change a file that the user did not choose to update" caused by SOC-4814
- Fix for SOC-5423: viewing table history showed the history of all objects for databases linked to Vault

3.5.1 - October 9th, 2013

Feature

- [Change the number of items shown in History dialog to speed up loading](#)

Fix

- Prevent change icons on in the object explorer from spinning indefinitely when Multiple SSMS support is turned on (thanks to Peter in Basel, Switzerland for reporting this during beta testing)

3.5.0 - October 1st, 2013

Features

- Support for [multiple SSMS windows](#)
- In-tool support for [merging conflicted stored procedures with Beyond Compare 3 and KDiff3](#)
- Revision numbers now appear on commit

SQL Source Control 3.4 release notes

3.4.10 - September 11th, 2013

Features

- Subversion 1.8 support
- Custom files for Git and Mercurial to allow committing changes with SQL Source Control

Fixes

- Fix for TFS authentication issues [SOC-3630]

Known issues

- Some combinations of TFS libraries on the client machine cause policy checking to fail, even if the policy is met. To work around this, [TFS policy checking can be disabled with a config file option](#).

3.4.9 - August 22nd, 2013

Fixes

- (SOC-5074) Fixed crashing tabs issue
- Provided error message and workaround for "multiple .sqlproj files" error
- No longer deletes commit comment on commit fail
- Fixed bug where SQL Source Control ignores custom working base and transient location

3.4.8 - July 19th, 2013

Fixes

- Stability improvements

3.4.7 - July 18th, 2013

Fixes

- SOC-5084: Commit tab is no longer cut off when there are two lines of warning message
- SOC-5055: fixed an issue that caused crashing when users deleted databases
- Fixed SSMS2012 stability issues

3.4.6 - June 28th, 2013

Fixes

- SSMS no longer hangs when you restore databases
- Crash when polling for changes introduced in 3.4.5

3.4.4 - June 12th, 2013

Fixes

- (SOC-4949, SQT-301) Fix for SSMS2012 crash when opening SQL Source Control tab hidden from view
- Fix for comparison options appearing in Get latest list after unlinking and relinking

3.4.3 - June 7th, 2013

Fixes

- (SOC-4927) Fixed blue notification icons appearing when there are no changes

3.4.2 - May 28th, 2013

Feature

- Much more responsive when selecting/deselecting thousands of objects simultaneously the Commit and Get latest tabs

Fixes

- Fix for ".sqlproj is out of date" error when using SQL Server Database Projects
- Fix for some users unable to launch SQL Compare Pro from the History dialog box

Known issues

- Some combinations of TFS libraries on the client machine cause policy checking to fail, even if the policy is met. As a workaround, [TFS policy checking can now be turned off by editing a config file](#).

3.4.1 - May 17th, 2013

Feature

- Added customized options to link a database to Git, Mercurial, or a working folder.

Known issues

- Some combinations of TFS libraries on the client machine cause policy checking to fail, even if the policy is met. As a workaround, [TFS policy checking can now be turned off by editing a config file](#).

3.4.0 - May 10th, 2013

Features

- Allowed naming of migration scripts
- Improved performance when refreshing the Commit or Get latest tabs

Known issues

- Some combinations of TFS libraries on the client machine cause [policy checking](#) to fail, even if the policy requirements are met. For a workaround, add #ignorepolicies to your commit comment.

SQL Source Control 3.3 release notes

3.3.1 - April 18th, 2013

Fixes

| | |
|----------|---|
| SOC-4785 | Running SQL Source Control after upgrading from 3.0 no longer gives "Failed to update to the head revision after 10 attempts" |
|----------|---|

Known issues

- Vault 6.1 (Standard and Professional) isn't supported yet. Versions 6.0.1 and earlier are fully supported.
- Downgrading from 3.3.1 to 3.1.0 will unlink databases. These databases can be safely relinked manually.

3.3.0 - April 12th, 2013

Features

- [TFS policy checking](#) is now enforced

Fixes

| | |
|----------|---|
| SOC-4683 | "Lock request timeout period exceeded" error message at Commit tab |
| SOC-4670 | SQL Source Control crashes at Get Latest tab |
| SOC-2219 | Users can now specify the path to their Working Base using a config file. |

Known issues

- Vault 6.1 (Standard and Professional) isn't supported yet. Versions 6.0.1 and earlier are fully supported.

SQL Source Control 3.2 release notes

Version 3.2.0.27 - March 13th, 2013

New features

- Persistent user selections on the Commit tab
- Initial support for linking SQL Server Management Studio to Visual Studio 2012 database projects

Fixes

| | |
|----------|--|
| SOC-4373 | Error committing script with OFFSET and FETCH NEXT syntax |
| SOC-4512 | SQL Source Control creates duplicate create statements for trigger in repository |
| N/A | TFS 2012 usernames now selected correctly |
| N/A | "Add-in failed to load" diagnostic improvements |

Known issues

- Vault 6.1 (Standard and Professional) isn't supported yet. Versions 6.0.1 and earlier are fully supported

SQL Source Control 3.1 release notes

Version 3.1.4.72

Features

- Drastically reduced the number of connections made to the SQL Server instance hosting a linked database. This also reduces the number of queries against those databases
- Improved performance of the tab refresh operations

Bug fixes

| | |
|---------------------|--|
| SOC-3820 & SOC-4645 | Error parsing objects using the CONTAINS TABLE syntax |
| SOC-4253 | Error paring objects using OFFSET syntax |
| SOC-4615 | Errors using Named Pipes to communicate with SQL Servers |
| SOC-4662 | Misbehavior of the deselect all button on the Commit and Get Latest tabs |
| SOC-4576 | "Index out of range" exception when different team members are using differing versions of the product |
| N/A | Spurious Object Explorer blue blobs when first starting the product |

Known issues and workarounds

- Vault 6.1 (Standard and Professional) is not yet supported. Versions 6.0.1 and earlier are fully supported
- If you retrieve a set of Comparison Options that should cause new changes to appear on the Get Latest tab (eg because your teammate had made whitespace changes to the objects which are no longer being ignored), you'll instead see changes pending for commit. Workaround: undo your apparent local changes. You'll now have a local copy of the repository versions of the objects as if you had retrieved them on the Get Latest tab.

Version 3.1.2.33

Bug fixes

| | |
|----------|--|
| SOC-4584 | Crash when removing polling manager |
| SOC-2779 | Crash when object is defined in multiple files on disk |
| | Minor bug fixes |

Known issues and workarounds

- Can't link databases to Vault 6.1 servers
- If you retrieve a set of Comparison Options causing new changes to get (eg because a teammate made whitespace changes to objects no longer ignored), you'll instead see changes pending for commit. Workaround: undo your apparent local changes. You will now have a local copy of the repository versions of the objects, as if you had retrieved them on the Get Latest tab.

Version 3.1.0.5208

Features

- **Blue blob polling performance improvements:** We have improved the performance of the SQL query which SQL Source Control uses to poll databases for changes. The information it gathers is used to display blue blobs on nodes in the Object Explorer tree and to provide user information when using the shared development model. Some users were experiencing issues with this query causing high server load, and in some cases timing out.
- **Commit and Get Latest Comparison Options:** The way in which SQL Source Control compares databases can be configured in the

same way that it can be in SQL Compare. These options are configured by an XML file. It is now possible to commit and retrieve this file in order to share one set of options throughout your team.

- **Memory usage improvements:** SQL Source Control should now use less memory in a lot of cases, reducing the risk of Out Of Memory exceptions.

Bug fixes

| | |
|----------|--|
| SOC-4358 | A change between Vault 6.0.0 and Vault 6.0.1 caused our Vault support to stop working |
| SOC-4441 | Diff pane is always initially minimised despite previously being resized |
| SOC-4315 | Crash when unlinking from a TFS repo that has been deleted on the server |
| SOC-4522 | Blue blobs do not appear in the Object Explorer for the first object created in a database |
| | Minor bug fixes |

Known issues

If you retrieve a set of Comparison Options that should cause new changes to appear for retrieve (e.g. because your team-mate had made whitespace changes to the objects which are no longer being ignored) you will instead see changes pending for commit. Workaround: undo your apparent local changes. You will now have a local copy of the repository versions of the objects, as if you had retrieved them on the Get Latest tab.

Version 3.1.0.4829

Features

Vault 6 support

Bug fixes

SOC-2902, fixes a crash when attempting to unlink from a TFS server which can't be contacted

Version 3.1.0.4583

Allow failure when trying to determine the list of databases to be polled, rather than throwing an exception.

Version 3.1.0.4579

Fix a bug with message:

```
Cannot access a disposed object

Object name: 'PersistentTempDbSqlConnection'
```

Version 3.1.0.4578

Fix SOC-4578, database polling db connection credentials failure.

Version 3.1.0.4574

Fix build magic so that SatelliteDLL is stamped with the right build number, and thus the installer works in upgrade mode

Version 3.1.0.4562

Fix performance problem caused by capturing the backtrace every time an object is disposed

Version 3.1.0.4555

Features

- **Shared model improvements:** When using the Shared Model, SQL Source Control shows the user who last changed an object on the commit tab. This information comes from various logs that SQL Server maintains, including the “default trace”. The query that SQL Source Control uses to read this information from SQL Server has been vastly improved. Briefly, it now:
 - Provides user information when objects are dropped, which it was unable to do before
 - Is less expensive to run, thus causing less load on the server
 - Causes fewer entries in the default trace due to its own actions. The old query actually used to cause entries to appear in the very logs it was reading from, causing them to roll over more quickly
 - Persists relevant data. The old query used to read from the logs anew every time it was run, which meant it couldn’t provide user information when they rolled over (they are finite in size). The new query persists all the data it needs to provide usernames in a separate table, so is unaffected by the SQL Server logs rolling over.
For more information, see [Logging changes to shared databases](#).
- **Support for TFS2012, including tfs.visualstudio.com:** Linking databases to TFS 2012 is now supported. This includes some hosted TFS2012 services, such as tfs.visualstudio.com. For more information, see [Using Team Foundation Server 2012 or Visual Studio Online](#)
- **Pick from Previous Commit Comments:** You can now pick from your previously used commit comments via a drop-down on the commit tab
- **Repository location now copy-able from the Setup tab**
- **Support for SQL Server 2012 OFFSET/FETCH/THROW syntax**
- **Support for SQL Server 2012 Sequence and Search Property List object types**

Version 3.1.0

- **Shared Model Improvements.** A more reliable and less resource-intensive method for giving the username for changes in the shared model.
- **Support for TFS 2012, including TFSPreview.com.** Link databases to TFS 2012 servers, including those hosted at tfspreview.com.
- **Support for Vault 6.** Link databases to Vault 6 Standard and Vault 6 Pro servers.
- **Bug fixes.**

SQL Source Control 3.0 release notes

Version 3.0.14.4178

Features

- **Performance Improvements – Static data comparisons:** Now turned on by default. This will give a significant reduction in the time taken to generate a commit or retrieve list when large amounts data are source-controlled.
- **Performance Improvements – Asynchronous commits:** The commit process no longer blocks the user from using SSMS when a commit is underway. Users can now perform other (non SQL Source Control) tasks in SSMS while their previous commit is completing.
- **Uninstall survey:** To help us understand why users would stop using SQL Source Control, we have added an optional, simple survey at the end of the product's uninstall process.
- **Bug fixes**

Bug Fixes

| | |
|----------|---|
| SOC-4021 | {SA} SqlException (Number: 3906) @ BlockExecutor.ExecuteBlock(...) |
| SOC-3942 | {SA} NullReferenceException @ ConnectShared.OpenNewTab(...) |
| SOC-3215 | {SA} InvalidStateException @ SvnRemoteOperations/<>c__DisplayClass39/<>c__DisplayClass3b.<GetDiffs>b__38(...) |
| SOC-2195 | Invoking the Commit tab from the 'Tables' folder does not see all tables selected in the grid |

Version 3.0.14.4178

Features

- **Performance Improvements – Static data comparisons:** Now turned on by default. This will give a significant reduction in the time taken to generate a commit or retrieve list when large amounts data are source controlled.
- **Performance Improvements – Asynchronous commits:** The commit process no longer blocks the user from using SSMS when a commit is underway. So users can now perform other (non SQL Source Control) tasks in SSMS while their previous commit is completing.
- **Uninstall survey:** To help us understand why users would choose to stop using SQL Source Control, we have added an optional, simple survey at the end of the product's uninstall process.
- **Bug fixes**

Bug fixes

| | |
|----------|---|
| SOC-4021 | {SA} SqlException (Number: 3906) @ BlockExecutor.ExecuteBlock(...) |
| SOC-3942 | {SA} NullReferenceException @ ConnectShared.OpenNewTab(...) |
| SOC-3215 | {SA} InvalidStateException @ SvnRemoteOperations/<>c__DisplayClass39/<>c__DisplayClass3b.<GetDiffs>b__38(...) |
| SOC-2195 | Invoking the Commit tab from the 'Tables' folder does not see all tables selected in the grid |

Version 3.0.13.4214

This is a bug fix release, which includes the same 'new' features as v3.0.13.4178 but also fixes the following bugs:

- SQL Compare interoperability bug
- Problem with *Check For Updates* client where CFU was report the incorrect product version.

Version 3.0.13.4178

Features

- **Performance Improvements – Static data comparisons:** Now turned on by default. This will give a significant reduction in the time

taken to generate a commit or retrieve list when large amounts data are source controlled.

- **Performance Improvements – Asynchronous commits:** The commit process no longer blocks the user from using SSMS when a commit is underway. So users can now perform other (non SQL Source Control) tasks in SSMS while their previous commit is completing.
- **Uninstall survey:** To help us understand why users would choose to stop using SQL Source Control, we have added an optional, simple survey at the end of the product's uninstall process.
- **Bug fixes**

Bug Fixes

| | |
|--------------------------|---|
| SOC-4021 | {SA} SqlException (Number: 3906) @ BlockExecutor.ExecuteBlock(...) |
| SOC-3942 | {SA} NullReferenceException @ ConnectShared.OpenNewTab(...) |
| SOC-3215 | {SA} InvalidStateException @ SvnRemoteOperations/<c__DisplayClass39/>c__DisplayClass3b.<GetDiffs>b__38(...) |
| SOC-2195 | Invoking the Commit tab from the 'Tables' folder does not see all tables selected in the grid |

Version 3.0 - February 1st, 2012

New features

Migration Scripts

SQL Source Control 3.0 enables you to create migration scripts between two versions of a database. Migration scripts are customizable change scripts that are committed to source control and re-used in deployment by SQL Compare. This can be useful, for example, if you add a NOT NULL constraint to a column in development and want to deploy this change to another environment. If you try to deploy this change using SQL Compare, the synchronization script will fail if the target table contains data because a default value is required; you can specify this value using a migration script.

Performance improvements

SQL Source Control 3.0 includes some underlying performance improvements, and further work on this is planned for the next release. Our internal testing shows a 30% improvement in the speed of some operations.

Known issues

Migration scripts are not supported for all source control systems

Currently, you can only create migration scripts if you are using Subversion (SVN), Team Foundation Server (TFS), or Vault. If your source control system is not supported, you can request support on the [SQL Source Control feedback forum](#).

Migration scripts backward compatibility

If you have used any of the SQL Source Control early access releases, any migration scripts you created are not compatible with version 3.0.

Getting the latest migration scripts

Migration scripts are not shown on the Get Latest tab. When you get the latest version, any applicable migration scripts are run automatically.

Dependency checking in migration scripts

When you create a migration script, SQL Source Control does not check the script for dependencies. We recommend you make sure all dependencies are included in the migration scripts you create.

SQL Source Control 2.2 release notes

December 15th, 2011

New features

- **Object filtering**

If you want to exclude objects from SQL Source Control, you can set up filters so they are not shown on the Commit Changes or Get Latest tabs. This is useful, for example, if there is a specific set of objects you never want to commit to source control. For more information, see [Using filters to exclude objects](#).

- **Repository browsing when linking**

The Link To Source Control dialog box now includes a Browse button for Subversion, Team Foundation Server and Vault. Use this to browse your source control repository when you are linking to source control.

The browse functionality also lets you create new folders in your repository, so you can get set up without needing to leave SQL Server Management Studio.

Known issues

- Config files are no longer included for CVS and Bazaar
- When linking a database to source control, on the Link to Source Control dialog box, presets are no longer included for the CVS and Bazaar source control systems.
- You can still create a custom config file to support any source control system with a command line interface.
- This change does not affect users who are already using CVS or Bazaar with SQL Source Control.
- Performance issues when source controlling large quantities of data
- If the data tables you are source controlling have a large number of rows, the Commit Changes and Get Latest tabs may be slow to populate and refresh.
- Because performance can be poor for large quantities of source controlled data, you are recommended only to source control static (lookup) data.
- Note that large quantities of data, or data that changes frequently (such as transactional data) should not be source controlled.

SQL Source Control 2.1 release notes

March 30th, 2011

New features

Version 2.1 of SQL Source Control introduces support for any source control system with a command line interface.

Presets are included for:

- **Bazaar**
- **CVS**
- **Git**
- **Mercurial**
- **Perforce**

Additionally, you can configure SQL Source Control to work with other source control systems.

For more information, see [Working with config files](#).

Known issues

For GIT and Perforce, committing changes with SQL Source Control commits all changes, not only database changes

If you are using GIT or Perforce, committing changes with SQL Source Control can commit all currently uncommitted changes for the working folder you are linked to.

This means that, for example, you could accidentally commit application code changes as well as database changes from within SQL Source Control.

To avoid this issue, we recommend you create a new, empty folder in source control to link your database. This makes sure the folder contains only database code, and you cannot commit non-database changes.

Issues for Vault users upgrading from the early access version

Users upgrading to version 2.1 from previous early access version may encounter issues.

If you are upgrading from an early access version, you are recommended to first do the following:

1. Unlink your databases from source control.
2. In Windows Explorer, navigate to: %localappdata%\Red Gate\SQL Source Control <version number>
3. Delete the files *LinkedDatabases.xml*, and *TableDataConfigs.xml*

Link your databases to source control again.

Spurious object change notification in rare circumstances

In some situations, it is possible to see a blue change notification in the Object Explorer, where there is no corresponding object with changes to commit on the Commit Changes tab.

This can occur, for example, if you alter a stored procedure and run it, then remove the change you made and run it again. The Object Explorer would continue to show a change, but there would be no change listed on the Commit Changes tab. Once you have visited the Commit Changes tab, and SQL Source Control has accurately determined the differences, the blue indicator is removed from the Object Explorer.

This is because the change detection performed on the Commit Changes tab is more detailed, and considers the state of the database, whereas the change notifications in the Object Explorer are displayed when a change is made.

Generally, you are unlikely to encounter this issue. However, it may be slightly more common when working under a shared development model.

For example:

- If two users are working on the same database under a shared model, when the first user makes a change, both users see a blue change notification in the Object Explorer. When the first user commits their change, the notification disappears for that user, but continues to be displayed for the second user. For the second user, there is no corresponding change on the Commit Changes tab.

When the second user goes to or refreshes the Commit Changes tab, the spurious notification is removed from the Object Explorer.

Unlinking if you downgrade to version 1

If you uninstall SQL Source Control version 2, and install version 1, any databases that were linked to source control while you were using version 2 are unlinked. Simply re-link them to continue using them with SQL Source Control.

Performance issues when source controlling large quantities of data

If the data tables you are source controlling have a large number of rows, the Commit Changes and Get Latest tabs may be slow to populate and refresh.

Because performance can be poor for large quantities of source controlled data, we recommended you only source-control static (lookup) data.

Large quantities of data, or data that changes frequently (such as transactional data) should not be source-controlled.

SQL Source Control 1.1 release notes

September 17th, 2010

Features

- View the history of changes for a database or database object by right-clicking it in the Object Explorer, and clicking Show History. For more information, see [Viewing source control history](#).
- Link a commit to an SVN bug ID, or mark it as associated with or resolving a TFS work item. For more information, see [Using SVN bug IDs & TFS work items](#).
- Version 1.1 fixes the issues some users encountered problems committing x64 assemblies.
- SQL Source Control integrates with SQL Compare 8.5 or later, and the SQL Server Management Studio Integration Pack, allowing you to choose a database version in source control and deploy it. For more information, see [Deploying a database from source control](#).

SQL Source Control 1.0 release notes

June 16th, 2010

Features

- Integration with SSMS
- Support for Subversion (SVN)
- Support for Team Foundation Server (TFS)
- Linking databases to your source control system
- Notification of changes in the Object Explorer
- Committing changes
- Getting the latest version
- Viewing differences between source control and database versions of objects before you commit or get latest
- Undoing uncommitted changes
- Resolving conflicts
- Simple deployment using SQL Compare Professional Edition

Locking objects

The **object locking preview** was available in SQL Source Control 3.8.16 to 3.8.20. It's been released as a full feature in [SQL Source Control 4](#).

You can lock objects you're working on so other people can't edit them. This means teams don't accidentally overwrite work.

After you lock an object, you can work on it as normal. When you try to edit an object locked by someone else, the server returns an error.

You can always unlock objects locked by other people.

If you're the only person who works on the databases on the server, you don't need to use object locking.

Setting up

To use object locking, you need to run a SQL script on the server. You can open the script from the **Object locking** tab in SQL Source Control.

For more information, and to review the script, see [Setting up object locking](#).

Removing objects

See [Removing object locking](#).

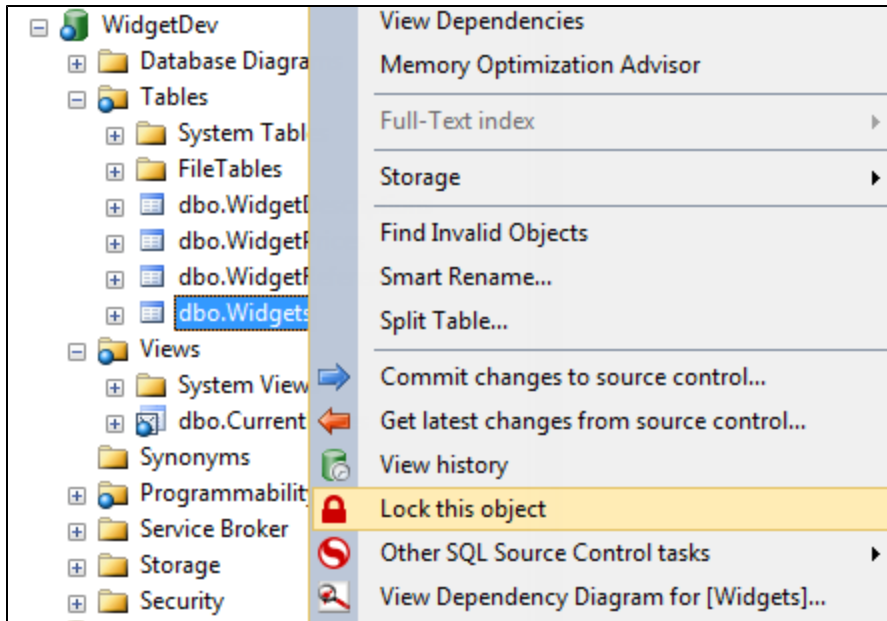
Compatible object types

Only these object types can be locked:

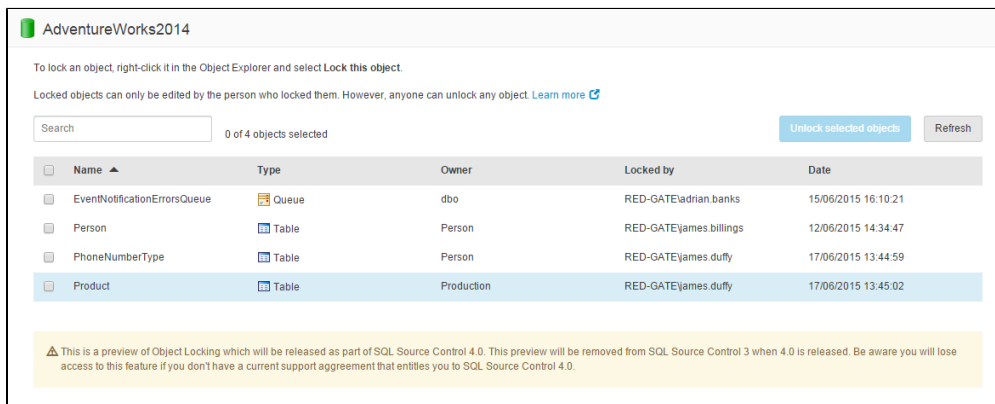
- Defaults
- Queues
- Rules
- Scalar-valued functions
- Stored procedures
- Synonyms
- Tables
- Table-valued functions
- User-defined data types
- User-defined table types
- User-defined types (from CLR)
- Views
- XML schema collections

Locking objects

In the Object Explorer, right-click the object you want to lock, and select **Lock this object**:



The object is listed in the **Object locking** tab:



Other people can't edit the object until it's unlocked. They can unlock the object if they need to.

Alternative: using SQL

Run the following SQL, replacing each value with the relevant details:

```
EXEC [tempdb].[dbo].[LockObject]
    @db_name = 'the database name'
    ,@schema_name = 'dbo'
    ,@object_name = 'the object name'
    ,@object_type = 'the type of object'
    ,@login_name = 'your login details'
```

Unlocking objects

Anyone can unlock any locked object, no matter who locked it.

From the Object locking tab

1. In SQL Source Control, go to the **Object locking** tab.

This tab lists all the objects that are locked on the database selected in SQL Source Control:

AdventureWorks2014

To lock an object, right-click it in the Object Explorer and select **Lock this object**.

Locked objects can only be edited by the person who locked them. However, anyone can unlock any object. [Learn more](#)

0 of 4 objects selected

Unlock selected objects

Refresh

| <input type="checkbox"/> | Name ▲ | Type | Owner | Locked by | Date |
|--------------------------|------------------------------|-------|------------|-------------------------|---------------------|
| <input type="checkbox"/> | EventNotificationErrorsQueue | Queue | dbo | RED-GATE\adrian.banks | 15/06/2015 16:10:21 |
| <input type="checkbox"/> | Person | Table | Person | RED-GATE\james.billings | 12/06/2015 14:34:47 |
| <input type="checkbox"/> | PhoneNumberType | Table | Person | RED-GATE\james.duffy | 17/06/2015 13:44:59 |
| <input type="checkbox"/> | Product | Table | Production | RED-GATE\james.duffy | 17/06/2015 13:45:02 |

⚠

This is a preview of Object Locking which will be released as part of SQL Source Control 4.0. This preview will be removed from SQL Source Control 3 when 4.0 is released. Be aware you will lose access to this feature if you don't have a current support agreement that entitles you to SQL Source Control 4.0.

2. Select the objects you want to unlock and click **Unlock**.

From the Object Explorer

Right-click the object you want to unlock and select **Unlock this object**.

Using SQL

Run the following SQL, replacing each field with the relevant details:

```
EXEC [tempdb].[dbo].[UnlockObject]
    @db_name = 'the database name'
    ,@schema_name = 'schema name (eg dbo)'
    ,@object_name = 'name of the object'
```

Setting up object locking

The **object locking preview** was available in SQL Source Control 3.8.16 to 3.8.20. It's been released as a full feature in [SQL Source Control 4](#).

Object locking is only compatible with SQL Server 2008 and later.

Object locking isn't compatible with Amazon RDS or Azure SQL Database.

To set up object locking, you need to run a script on the server. Afterwards, you can lock objects in **all** databases on the server.

About the setup script

The script adds the trigger **RG_SQLSourceControl_DDLTrigger** to the server. The trigger detects when somebody tries to edit a locked object.

The script also creates the database **RedGate** with the schema **SQLSourceControl**. This contains:

- the table **SQLSourceControl.LockedObjects**, to store locks on objects
- the function **SQLSourceControl.GetLockedObjects**, to retrieve locked objects
- the function **SQLSourceControl.IsObjectLocked**, to check if objects are locked
- the stored procedure **SQLSourceControl.LockObject**, to lock objects
- the stored procedure **SQLSourceControl.UnlockObject**, to unlock objects

To view the script, click **Expand source**:

Object locking setup script

 [Expand source](#)

```
-- This script sets up object locking on the server. After you run it, you can lock
and unlock objects in all databases on the server.
-- Before you run the script, close any other query windows that use the RedGate
database.
-- Object locking should only be used for your development environments, and not your
production server.
-- For more information about object locking, including a summary of what this script
does, see http://www.red-gate.com/SOC-object-locking
--////////////////////////////////////
USE master
IF DB_ID(N'RedGate') IS NULL
    CREATE DATABASE RedGate
ALTER DATABASE RedGate SET RECOVERY SIMPLE
ALTER DATABASE RedGate SET ALLOW_SNAPSHOT_ISOLATION ON
ALTER DATABASE RedGate SET READ_COMMITTED_SNAPSHOT ON
GO
USE [RedGate]
--////////////////////////////////////
--// SQLSourceControl Schema
--////////////////////////////////////
IF (SCHEMA_ID('SQLSourceControl') IS NULL)
    EXEC sp_executesql N'CREATE SCHEMA [SQLSourceControl] AUTHORIZATION [dbo]'
GO

--////////////////////////////////////
--// DDL Trigger
--////////////////////////////////////
IF EXISTS (SELECT 1 FROM sys.server_triggers WHERE name =
'RG_SQLSourceControl_DDLTrigger')
    DROP TRIGGER RG_SQLSourceControl_DDLTrigger ON ALL SERVER
GO
CREATE TRIGGER [RG_SQLSourceControl_DDLTrigger]
```

```

ON ALL SERVER
WITH EXECUTE AS SELF
FOR DDL_DATABASE_LEVEL_EVENTS
AS
SET NOCOUNT ON
DECLARE @event_type NVARCHAR(MAX)
SET @event_type = EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]',
'NVARCHAR(MAX)')
DECLARE @db_name NVARCHAR(MAX)
SET @db_name = EVENTDATA().value('(/EVENT_INSTANCE/DatabaseName)[1]',
'NVARCHAR(MAX)')
DECLARE @schema_name NVARCHAR(MAX)
SET @schema_name = EVENTDATA().value('(/EVENT_INSTANCE/SchemaName)[1]',
'NVARCHAR(MAX)')
DECLARE @object_name NVARCHAR(MAX)
SET @object_name = EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]',
'NVARCHAR(MAX)')
DECLARE @login_name NVARCHAR(MAX)
SET @login_name = EVENTDATA().value('(/EVENT_INSTANCE/LoginName)[1]',
'NVARCHAR(MAX)')
IF OBJECT_ID('[RedGate].[SQLSourceControl].[LockedObjects]') IS NOT NULL
BEGIN
    IF EXISTS (SELECT 1
        FROM [RedGate].[SQLSourceControl].[LockedObjects] AS LO
        WHERE LO.[db_name] = @db_name
            AND LO.[schema_name] = @schema_name
            AND LO.[object_name] = @object_name
            AND LO.[login_name] != @login_name)
    BEGIN
        DECLARE @locker NVARCHAR(MAX)
        DECLARE @reason NVARCHAR(MAX)
        SELECT @locker = [login_name]
        FROM [RedGate].[SQLSourceControl].[LockedObjects] AS LO
        WHERE LO.[db_name] = @db_name
            AND LO.[schema_name] = @schema_name
            AND LO.[object_name] = @object_name
        DECLARE @NewLine AS CHAR(2) = CHAR(13) + CHAR(10)
        DECLARE @message NVARCHAR(MAX)
        SET @message = @NewLine
            + 'SQL Source Control' + @NewLine
            + @NewLine
            + 'The object [' + @db_name + '].[' + @schema_name +
'].[' + @object_name + '] is locked by ' + @locker + @NewLine
            + @NewLine
            + 'To unlock the object, in SQL Source Control, go to
the object locking tab. If you don''t have SQL Source Control, talk to the user who
locked it.' + @NewLine
            + @NewLine
            + 'For more information about object locking, see
www.red-gate.com/SOC-object-locking' + @NewLine
        RAISERROR (@message, 16, 1)
        ROLLBACK
    END
END
GO

--////////////////////////////////////
--// Locked Objects Table
--////////////////////////////////////

```

```

IF OBJECT_ID('[SQLSourceControl].[LockedObjects]') IS NOT NULL
    DROP TABLE [SQLSourceControl].[LockedObjects]
GO
CREATE TABLE [SQLSourceControl].[LockedObjects]
(
    [db_name] [NVARCHAR](128) NOT NULL,
    [schema_name] [NVARCHAR](128) NOT NULL,
    [object_name] [NVARCHAR](128) NOT NULL,
    [object_type] [NVARCHAR](128) NOT NULL,
    [login_name] [NVARCHAR](MAX) NOT NULL,
    [time_locked] [DATETIMEOFFSET] NOT NULL
)
GO
ALTER TABLE [SQLSourceControl].[LockedObjects] ADD CONSTRAINT [PK_LockedObjects]
PRIMARY KEY CLUSTERED
(
    [db_name] ASC,
    [schema_name] ASC,
    [object_name] ASC
)
GO
--////////////////////////////////////
--// LockObject Stored Procedure
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[LockObject]') IS NOT NULL
    DROP PROCEDURE [SQLSourceControl].[LockObject]
GO
CREATE PROCEDURE [SQLSourceControl].[LockObject]
    @db_name AS NVARCHAR(128),
    @schema_name AS NVARCHAR(128),
    @object_name AS NVARCHAR(128),
    @object_type AS NVARCHAR(128),
    @login_name AS NVARCHAR(MAX) = NULL
AS
    IF @login_name IS NULL
        SET @login_name = SUSER_SNAME()
    IF EXISTS (SELECT 1 FROM [SQLSourceControl].[LockedObjects] AS LO
        WHERE LO.[db_name] = @db_name
        AND LO.[schema_name] = @schema_name
        AND LO.[object_name] = @object_name)
        BEGIN
            DECLARE @existing_locker NVARCHAR(MAX)
            SELECT @existing_locker = [login_name]
            FROM [SQLSourceControl].[LockedObjects] AS LO
            WHERE LO.[db_name] = @db_name
            AND LO.[schema_name] = @schema_name
            AND LO.[object_name] = @object_name
            IF @existing_locker != @login_name
                BEGIN
                    DECLARE @message NVARCHAR(MAX)
                    SET @message = 'Cannot lock the object [' + @db_name + '].[' +
@schema_name + '].[' + @object_name + ']'
+ 'as it is already locked by ' +
@existing_locker
                    RAISERROR (@message, 16, 1)
                END
            END
        ELSE
            BEGIN

```

```

        INSERT [SQLSourceControl].[LockedObjects]
            ([db_name], [schema_name], [object_name], [object_type],
[login_name], [time_locked])
            VALUES (@db_name, @schema_name, @object_name, @object_type, @login_name,
SYSDATETIMEOFFSET())
        END
GO

--////////////////////////////////////
--// UnlockObject Stored Procedure
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[UnlockObject]') IS NOT NULL
    DROP PROCEDURE [SQLSourceControl].[UnlockObject]
GO
CREATE PROCEDURE [SQLSourceControl].[UnlockObject]
    @db_name AS NVARCHAR(128),
    @schema_name AS NVARCHAR(128),
    @object_name AS NVARCHAR(128)
AS
    IF EXISTS (SELECT 1 FROM [SQLSourceControl].[LockedObjects] AS LO
        WHERE LO.[db_name] = @db_name
        AND LO.[schema_name] = @schema_name
        AND LO.[object_name] = @object_name)
    BEGIN
        DELETE FROM [SQLSourceControl].[LockedObjects]
        WHERE [db_name] = @db_name
        AND [schema_name] = @schema_name
        AND [object_name] = @object_name
    END
GO

--////////////////////////////////////
--// GetLockedObjects Function
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[GetLockedObjects]') IS NOT NULL
    DROP FUNCTION [SQLSourceControl].[GetLockedObjects]
GO
CREATE FUNCTION [SQLSourceControl].[GetLockedObjects] ()
RETURNS TABLE
AS
RETURN
(
    SELECT [db_name], [schema_name], [object_name], [object_type], [login_name],
[time_locked]
    FROM [SQLSourceControl].[LockedObjects]
)
GO

--////////////////////////////////////
--// IsObjectLocked Function
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[IsObjectLocked]') IS NOT NULL
    DROP FUNCTION [SQLSourceControl].[IsObjectLocked]
GO
CREATE FUNCTION [SQLSourceControl].[IsObjectLocked]
(
    @db_name AS NVARCHAR(128),
    @schema_name AS NVARCHAR(128),
    @object_name AS NVARCHAR(128)

```



```
)  
RETURNS BIT  
AS  
BEGIN  
    IF EXISTS (SELECT 1 FROM [SQLSourceControl].[LockedObjects] AS LO  
                WHERE LO.[db_name] = @db_name  
                AND LO.[schema_name] = @schema_name  
                AND LO.[object_name] = @object_name)  
        RETURN 1
```

```
        RETURN 0
    END
GO
```

Required privileges

To run the setup script, you need these database privileges:

- CREATE USER
- CREATE ROLE
- GRANT ANY PRIVILEGE
- CREATE ANY TABLE
- CREATE ANY PROCEDURE

You don't need the privileges to lock and unlock objects.

To set up object locking

1. In the **Lock objects** tab, click **Open setup script**.
2. Click **Run script**.
The script opens in a new Management Studio query window.
3. Close any other query windows that use the RedGate database.
4. Run the script.

Object locking is set up for the server.

Removing object locking

The **object locking preview** was available in SQL Source Control 3.8.16 to 3.8.20. It's been released as a full feature in [SQL Source Control 4](#).

To remove object locking from the server, you need to run an uninstall script. Any locked objects on the server will be unlocked.

If you want to uninstall SQL Source Control, you should remove object locking from the server first.

If you have unlocked objects when you uninstall SQL Source Control, they will stay locked until you either:

- reinstall SQL Source Control and unlock them in the Locking tab, or
- run the unlock SQL for each locked object

Required privileges

To remove object locking, you need these database privileges:

- CREATE USER
- CREATE ROLE
- GRANT ANY PRIVILEGE
- CREATE ANY TABLE
- CREATE ANY PROCEDURE

To remove object locking

1. View the script:

Object locking uninstall script

 [Expand source](#)

```

-- This script removes the objects added by the object locking setup script. Any
locked objects on the server will be unlocked.
-- Before you run the script, close any other query windows that use the RedGate
database.
-- The script doesn't delete the Redgate database. If you want to delete the
database, make sure it isn't being used by other Redgate tools (eg DLM
Dashboard).
-- For more information about object locking, see
http://www.red-gate.com/SOC-object-locking

--////////////////////////////////////
USE [RedGate]
--////////////////////////////////////
--// DDL Trigger
--////////////////////////////////////
IF EXISTS (SELECT 1 FROM sys.server_triggers WHERE name =
'RG_SQLSourceControl_DDLTrigger')
    DROP TRIGGER RG_SQLSourceControl_DDLTrigger ON ALL SERVER
GO
--////////////////////////////////////
--// Locked Objects Table
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[LockedObjects]') IS NOT NULL
    DROP TABLE [SQLSourceControl].[LockedObjects]
GO
--////////////////////////////////////
--// LockObject Stored Procedure
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[LockObject]') IS NOT NULL
    DROP PROCEDURE [SQLSourceControl].[LockObject]
GO
--////////////////////////////////////
--// UnlockObject Stored Procedure
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[UnlockObject]') IS NOT NULL
    DROP PROCEDURE [SQLSourceControl].[UnlockObject]
GO
--////////////////////////////////////
--// GetLockedObjects Function
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[GetLockedObjects]') IS NOT NULL
    DROP FUNCTION [SQLSourceControl].[GetLockedObjects]
GO
--////////////////////////////////////
--// IsObjectLocked Function
--////////////////////////////////////
IF OBJECT_ID('[SQLSourceControl].[IsObjectLocked]') IS NOT NULL
    DROP FUNCTION [SQLSourceControl].[IsObjectLocked]
GO
--////////////////////////////////////
--// SQLSourceControl Schema
--////////////////////////////////////
IF (SCHEMA_ID('SQLSourceControl') IS NOT NULL)
    DROP SCHEMA [SQLSourceControl]
GO

```

2. Close any other query windows that use the RedGate database.
3. Run the script on the server.

Object locking is removed from the server.

Source-controlling static data

See [Static data](#).