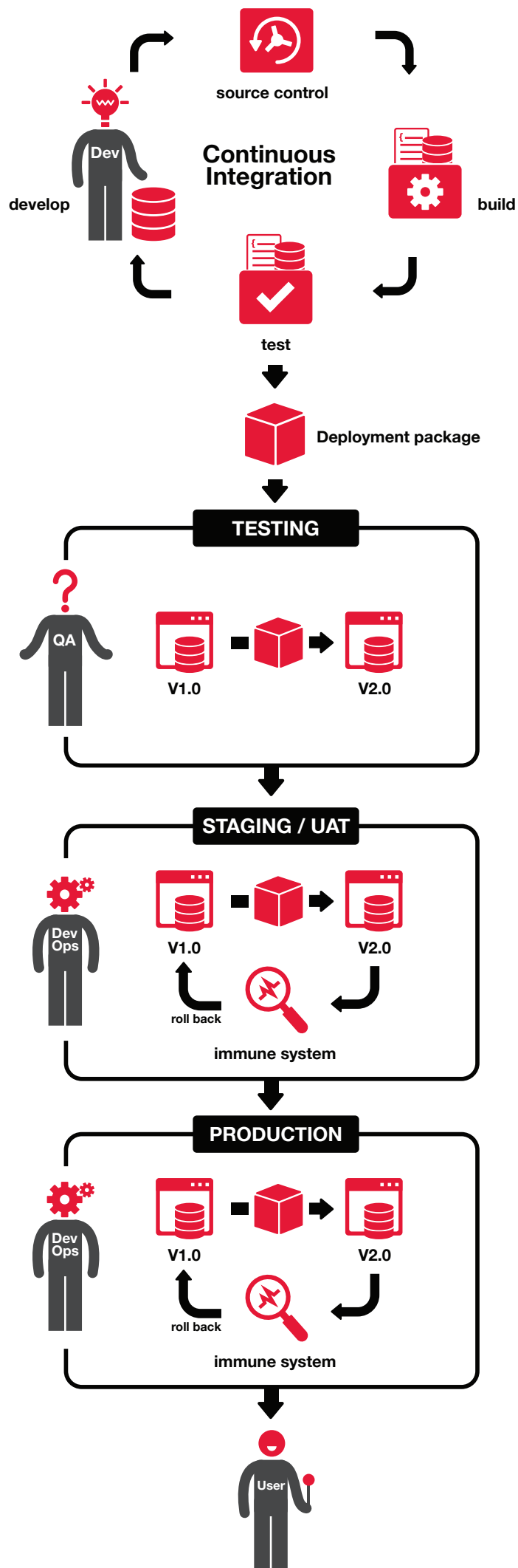


# Continuous integration for databases using Red Gate tools

A technical overview





---

# Continuous integration for databases using Red Gate tools

## Introduction

This whitepaper examines the challenge of integrating SQL Server databases into an existing build automation process, such as continuous integration, and describes how Red Gate tools can be used to automate the process.

Included in this paper are simple examples using SQL Source Control and the command line interfaces of Red Gate tools. SQL Source Control is an add-in to SQL Server Management Studio that source controls your database schema and static data. When changes are checked into source control, the continuous integration process is triggered.

This can do any of the following:

- build a test database from source control
- run and validate an automated deployment process
- run and validate an automated upgrade process
- run automated regression tests

This paper only covers databases, and not the building of your application code or any other configuration or setup required for your product.

This whitepaper is organized into the following sections:

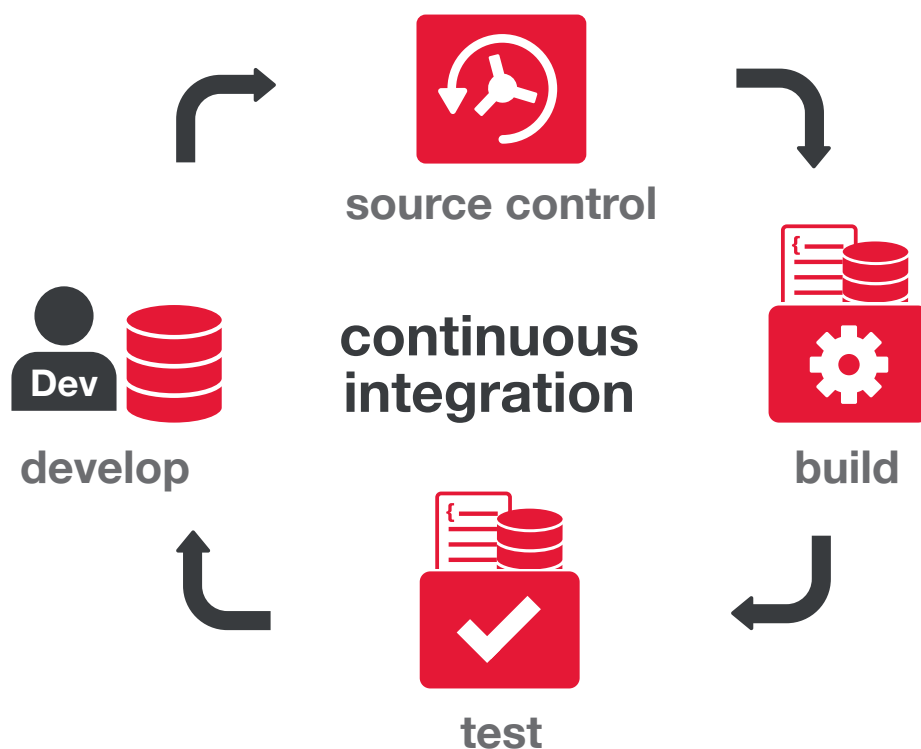
I.	Why continuous integration?	p4
II.	How are databases different?	p6
III.	The database delivery lifecycle	p7
IV.	Database continuous integration	p8
V.	Database deployment	p10
VI.	How Red Gate tools help	p10
VII.	Worked examples	p12
VIII.	Further reading and resources	p17
IX.	Conclusions	p18

## I. Why continuous integration?

Continuous integration (CI) is the process of ensuring that all code and related resources in a development project are integrated regularly and tested by an automated build system. Code changes are checked into source control, triggering an automated build with unit tests and early feedback in the form of errors returned. A stable current build is consistently available, and if a build fails, it can be fixed rapidly and re-tested.

A CI server uses a build script to execute a series of commands that build an application. Generally, these commands clean directories, run a compiler on source code, and execute unit tests. However, for applications that rely on a database back-end, build scripts can be extended to perform additional tasks such as creating, testing, and updating a database.

The following diagram illustrates a typical integration process. The automated continuous integration process begins each time the server detects a change that has been committed to source control by the development team. Continuous integration ensures that if at any stage a process fails, the 'build' is deemed broken and developers are alerted immediately.



CI originated from the Extreme Programming (XP) movement and is now an established development practice.

*“Continuous Integration is a practice designed to ensure that your software is always working, and that you get comprehensive feedback in a few minutes as to whether any given change to your system has broken it.”*

**Jez Humble, ThoughtWorks, co-author of Continuous Delivery**

For many software projects, this will include a database. Author and thought leader, Martin Fowler, recommends that “getting the database schema out of the repository and firing it up in the execution environment” should be part of the automatic build process. However, this is not always simple, which is why this paper seeks to clarify the process of integrating databases into an existing automatic continuous integration process.

## II. How are databases different?

They aren't. Database code is code, and should therefore be treated in the same way as your application code. However, the principal difficulty underlying continuous integration for databases is the lack of a simple way to keep a database in source control and deploy it to a target server.

The database is unlike application code in as much as it contains state that needs to be preserved after an upgrade. Where a production database already exists, DML and DDL queries modify the existing state of a database, and unlike for application code, there is no source code to compile. Migration and deployment therefore rely on creating upgrade scripts specifically for that purpose.

The lack of database source code makes it difficult to maintain a current stable version in source control. Creation and migration scripts can be checked into the source control repository, but despite its importance, the disciplined creation and on-going maintenance of these scripts is often not considered to be a core part of the database development cycle.

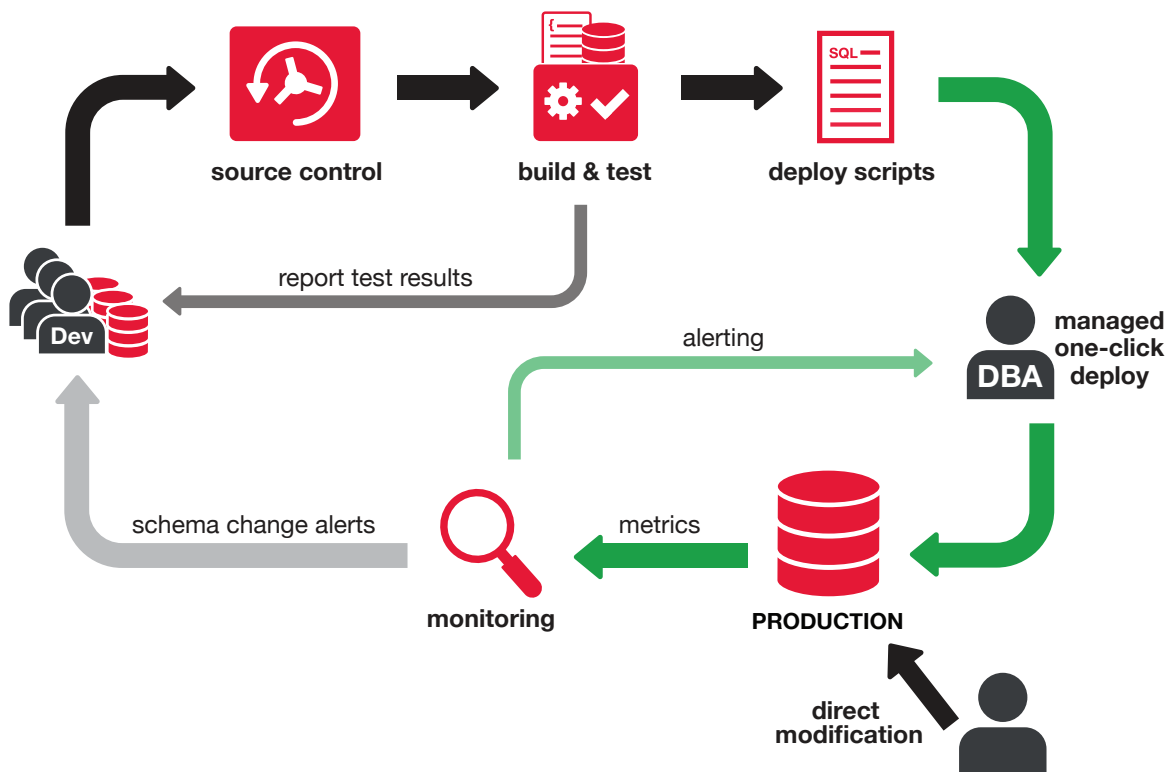
Migration scripts may contain ALTER and UPDATE statements to update the target version of the database with the development version; alternatively, the scripts may create a new database. Where changes are deployed to an existing database, all differences and dependencies must be accounted for. In some production deployments, this involves multiple targets with different schemas and data. In either case, the manual process is time consuming, prone to errors, and one that should not be left unresolved until the end of the project cycle.

Object creation scripts can be generated relatively simply (for example using Microsoft SQL Server Management Studio), but referential integrity is difficult to maintain. Objects and data must be created and populated in the correct order, and as dependency chains can be complex, third party tools are often required. Data migration and test data creation are tedious and time consuming operations when performed manually.

### III. The database delivery lifecycle

Database CI is an important part of a wider database delivery lifecycle. Continuous integration builds and tests a database, validating the upgrade scripts that will eventually be supplied to the DBA for deployment. Once deployed to the production server, monitoring should be put in place to ensure that not only the performance of the database remains acceptable, but also that the end user is benefiting from the changes.

In the event that a change is made directly to the production database bypassing the rigid development and test processes, monitoring ensures that the DBA and development manager are notified. The new change can either be undone by the DBA, or if approved, copied into the development environment.



## IV. Database continuous integration

### 1. Keeping a database up-to-date

Databases may figure in your CI process simply because the application code requires there to be a database present to function correctly. The database schema version corresponds to an analogous application code version. Any changes to the application code or the database structure could in theory break the system and should consequently trigger the CI process.

Once a database is maintained in source control, Red Gate tools are able to build a clean database from its source files to accompany the application build.

If you already have internal test databases that need to match the development databases, you can keep them up-to-date with the latest version using continuous integration.

### 2. Testing database code

Although it is best practice to test application code as part of a CI process, database code and its accompanying business logic is often overlooked. Database code comes in the form of stored procedures, functions, views, triggers, and CLR objects. If it is deemed important to test application code as part of CI, the same must apply to the database code.

Fortunately there are many open source frameworks that can be used for the purposes, some implemented in .NET (e.g. NUnit) and others in SQL (e.g. the popular open source SQL Server unit testing framework, tSQLt). Unit tests can be easily created, run, and managed with Red Gate's SQL Test, a SQL Server Management Studio add-in.

Populating the database with test data for the unit tests is also a very useful feature. Red Gate's SQL Data Generator can be used as part of the build process to fill the test database with sample data for the unit tests to run against.

### 3. Generating database documentation

Development databases undergo frequent changes. To keep track of the state of the database, it can be useful to document the schema. The build process can generate database documentation files as a build artifact. This is created using a SQL Doc project file checked into source control.



#### 4. Validating the database creation script

A database creation script can be created as an artifact of the build process. The script will build the database from scratch. This is useful to build test databases but could also be used to perform new installations of the application; for example for new customers.

#### 5. Validating the database upgrade script

Unlike for application code, upgrading a production database isn't a simple case of replacing it with a fresh copy. Databases have a mission critical state that needs to be preserved.

Safeguarding existing data is the most challenging task to be faced during the upgrade process. This is why it is a highly recommended best practice to repeatedly test the deployment script as part of the CI process and ensure that a working upgrade script can be generated at all times.

In order to test the upgrade, it is necessary to create a database at the version corresponding to the existing production database, and not just create a new database representing the latest version. Using Red Gate tools, the deployment script is generated, applied against the production-level CI database, and subsequently validated against the expected target version. If this validation fails, the failed upgrade process should be regarded as a 'broken build', and measures should be taken to troubleshoot and promptly resolve the issue.

#### 6. Packaging databases

Deploying different versions of a database to different environments and keeping track of changes can be difficult. The CI process can be configured to create database packages and upload them to Deployment Manager. Deployment Manager can then manage all the database and application deployments for your organization.

## V. Database deployment

Once validated in a CI environment, the database (and application) changes need to go through a release process. It is prudent to push these changes through some final test phases using staging and UAT environments.

In many ways continuous integration can be considered a dry run for production deployment. But although the CI environment often mirrors the production environment as closely as possible, it is rarely the case for the database.

Production databases can be huge, and it is therefore impractical to restore a production backup to the CI environment for testing purposes. Test environments rarely benefit from the same storage capacity as for production and pre-production environments, and lengthy restore times make recreating the CI database impractical.

Red Gate's Deployment Manager helps transition code and database changes through the final stages of the release process, by taking the tested output of the CI process encapsulated as 'packages' and deploying these to pre-production test environments. A database package contains the validated deployment scripts along with snapshots of the intended target versions in order to ensure that the actual target is at the expected version and hasn't since 'drifted'. A database package also contains the 'source' database state allowing it to validate the success of the deployment.

## VI. How Red Gate tools help

Red Gate offers the following tools to support the CI and delivery process. Many of these tools feature in the worked examples in the next section.

### SQL Source Control

- Helps maintain database schema and data in a source control system within SQL Server Management Studio
- Allows for the creation of custom migration scripts which are saved to source control

### SQL Connect

- Visual Studio add-in that helps source control database schema by allowing database code to be checked in atomically with application code changes

### SQL Compare and SQL Data Compare command lines

- Creates a database from source files in version control
- Generates schema and data deployment scripts
- Validates that two databases are identical
- Generates pre/post-deployment reports for troubleshooting

### SQL Test

- Allows developers to easily create, run, and manage tSQLt unit tests on databases

### SQL Data Generator command line

- Generates realistic test data based on your existing schema

### SQL Doc command line

- Generates the latest database documentation automatically as part of your CI process

### Deployment Manager

- Makes deployments as easy as possible with automated release management

### SQL Monitor

- Ensures that your production database performance is as expected, and that you are alerted when schema changes are detected

### NAnt and MSBuild scripts

- Allows simple configuration of CI with NAnt and MSBuild using a build script checked into source control

### TeamCity plugin

- Allows configuration of CI through a web interface as a build step in TeamCitysource control

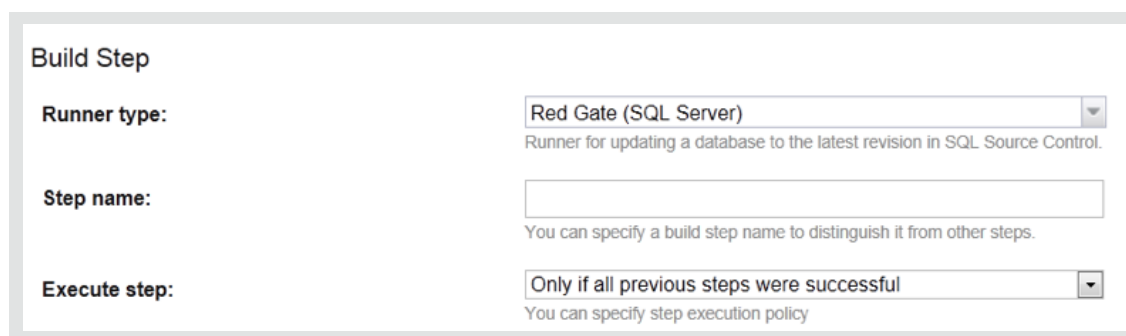
### Licensing

Please contact [sqldev.info@red-gate.com](mailto:sqldev.info@red-gate.com) or visit [www.red-gate.com/CI](http://www.red-gate.com/CI) for information about licensing options for build automation such as Continuous Integration.

## VII. Worked examples

### 1. TeamCity Plugin

The TeamCity Plugin makes configuring CI simple. When it's installed in TeamCity, you can select Red Gate (SQL Server) as the runner type for your build step:



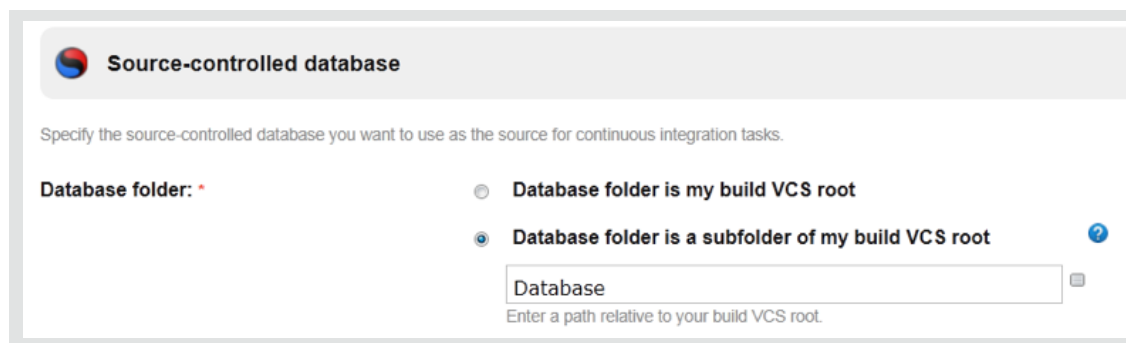
**Build Step**

**Runner type:** Red Gate (SQL Server)  
Runner for updating a database to the latest revision in SQL Source Control.

**Step name:**   
You can specify a build step name to distinguish it from other steps.

**Execute step:** Only if all previous steps were successful  
You can specify step execution policy

Then set the folder for your source-controlled database:



**Source-controlled database**

Specify the source-controlled database you want to use as the source for continuous integration tasks.

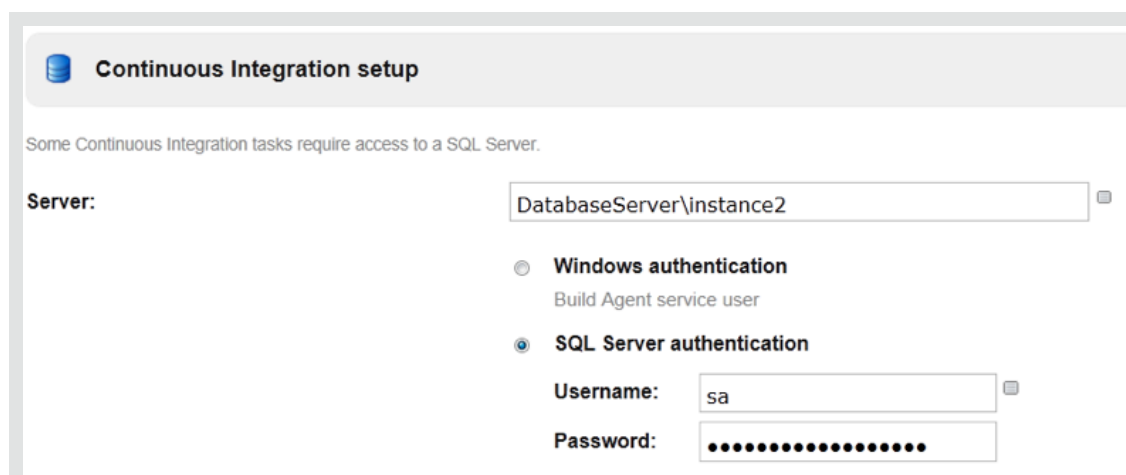
**Database folder: \***

☐ Database folder is my build VCS root

☒ Database folder is a subfolder of my build VCS root

**Database**   
Enter a path relative to your build VCS root.

Then specify connection details for a SQL Server:



**Continuous Integration setup**

Some Continuous Integration tasks require access to a SQL Server.

**Server:** DatabaseServer\Instance2

☐ Windows authentication  
Build Agent service user

☒ SQL Server authentication

**Username:** sa

**Password:**

Now you can choose which continuous integration tasks you want to use. You can:

- update a target database to the latest version in source control on every build:

**Database**

☒ **Update a database on every build** ?

Specify a database you'd like to keep up to date with the latest changes in source control.  
This will only update one database. To use Deployment Manager to do database packaging and deployment, use the options below.

- enable tSQLt regression tests on a copy of the database. You can also check a SQL Data Generator project file in to source control. The test database will be populated with test data before the tSQLt tests are run:

**tSQLt**

☒ **Run tSQLt unit tests on the database** ?

☒ **Populate the database with test data** ?

Enter the path of a SQL Data Generator project (.sqlgen) file.  
The path must be relative to the VCS root.

- have the database documented on every build, based on a SQL Doc project file in source control:

**Documentation**

☒ **Create documentation of the database** ?

Enter the path of a SQL Doc project (.sqldoc) file.  
The path must be relative to the VCS root.

- generate a creation script and have it tested by creating and dropping a temporary copy of the database:

**Creation script**

☒ **Generate database creation script** ?



Generates a script as a build artifact that will create the source database.

☒ **Validate creation script** ?


Creates a temporary database with the script and reports errors in the build output.

- generate or validate upgrade scripts from previous versions of the database in Deployment Manager:

**Upgrade scripts**

- ☒ **Generate database upgrade scripts**   
Generates scripts that will upgrade the database from the previous version in Deployment Manager.  
Requires Deployment Manager details to be specified below.
- ☒ **Validate upgrade scripts**   
Upgrades a temporary database of the previous version from Deployment Manager and reports errors in the build output.

- package each build of the database and publish it to Deployment Manager:

 **Deployment Manager**

You can use Deployment Manager to automate deployments of database packages. Find out more at [red-gate.com/dm](https://red-gate.com/dm).


**Database Package** ☒ **Create a Database Package**  
Deployment Manager uses database packages to deploy databases.

**Package ID:** \*   
The identifier for the nuget package to create.

**Server URL:** \*   
The URL of your Deployment Manager web interface, for example *http://my-server-name:80*

**DM API Key:** \*   
You can find the API key on your Profile page in the Deployment Manager web interface.

- customize any of the behavior by specifying additional SQL Compare command line parameters to run:

 **Additional Commandline switches**

Specify any additional SQL Compare switches you want the command line to run.

**Additional parameters:**  [Syntax](#) | [Examples](#)

## 2. Build Scripts

The build scripts allow simple configuration of CI for build systems that use NAnt or MSBuild. When the build scripts are checked in to source control, you provide your build system with the sqlCI.proj file for TFS and MSBuild and the sqlCI.build file for NAnt.

Then you need to specify at least the following XML elements:

- **<scriptsfolder>**  
the location of your database scripts folder
- **<databaseserver>**  
the address of the database server
- **<databaseIntegratedAuthentication>**  
whether you want to use Windows authentication or SQL authentication

Then you can specify any of the following XML options to configure continuous integration. You can:

- **<databasename>**  
update a target database to the latest version in source control on every build
- **<enableTsqli>**  
run tSQLt tests in source control on a test copy of the database
- **<sqlDataGeneratorProject>**  
populate the database with test data, based on a SQL Data Generator project file in source control
- **<sqlDocProject>**  
generate database documentation, based on a SQL Doc project file in source control
- **<generateCreationScript>**  
generate a creation script as a build artifact
- **<validateCreationScript>**  
have the creation script tested by creating and dropping a temporary copy of the database

- **<generateUpgradeScriptForCurrentlyDeployedVersions>**  
generate upgrade scripts to the current version of the database from previous versions of the database in Deployment Manager
- **<validateUpgradeScript>** validate the upgrade scripts
- **<deploymentManagerUrl>**
- **<deploymentManagerApiKey>**
- **<packageVersion>** package each build of the database and publish it to Deployment Manager
- **<additionalCompareArgs>** customize the behavior by specifying SQL Compare command line parameters

### 3. Command line tools

The command line tools for SQL Compare, SQL Packager, SQL Doc, and SQL Data Generator allow you to write more customized and advanced scripts for CI and automation.

For more information on the command line tools, see their individual product pages:

**SQL Automation Pack:** [www.red-gate.com/automationpack](http://www.red-gate.com/automationpack)

**SQL Compare:** [www.red-gate.com/sqlcompare/commandline](http://www.red-gate.com/sqlcompare/commandline)

**SQL Packager:** [www.red-gate.com/sqlpackager/commandline](http://www.red-gate.com/sqlpackager/commandline)

**SQL Doc:** [www.red-gate.com/sqldoc/commandline](http://www.red-gate.com/sqldoc/commandline)

**SQL Data Generator:** [www.red-gate.com/sqldatagen/commandline](http://www.red-gate.com/sqldatagen/commandline)

For more information on using **sqlCI.exe**, see [www.red-gate.com/CI/switches](http://www.red-gate.com/CI/switches)



## VIII. Further reading and resources

Please visit [www.red-gate.com/CI](http://www.red-gate.com/CI) for the latest resources and further reading.  
Information on Deployment Manager can be found by visiting [www.red-gate.com/DM](http://www.red-gate.com/DM)



## IX. Conclusions

This article has outlined some best practices and worked examples for implementing databases as part of your CI development process. As with application code, database code is managed in version control using SQL Source Control, and automatically deployed to a CI environment. Red Gate command line tools handle the scripting and deployment process, removing the hurdle that has previously obstructed CI and source control for databases. A database package can be created as part of the CI process and, with minimal effort, deployed through your dev, staging, and production environments using Red Gate's Deployment Manager.





redgate  
ingeniously simple