SQL Comparison SDK 10 documentation

About SQL Comparison SDK

With the SQL Comparison SDK, you can use the functionality of SQL Compare, SQL Data Compare, and SQL Packager in your own applications.

You can use the SQL Comparison SDK to deploy multiple databases, to upgrade your customers' databases without manual intervention, and for continuous integration.

Breaking changes
Version 10.5 of the SQL Comparison SDK includes changes that might break your projects that use the SQL Compare, SQL Data Compare, or SQL Packager APIs.

For more information, see Breaking changes in SQL Comparison SDK 10.5.

SQL Compare API

Download API reference documentation (CHM)

You may need to unblock the CHM file before you can view it.

Download sample projects

SQL Data Compare API

Download API reference documentation (CHM)

You may need to unblock the CHM file before you can view it.

Download sample projects

SQL Packager API

Download API reference documentation (CHM)

You may need to unblock the CHM file before you can view it.

Download sample projects

Installing

To install the SDK, download the SQL Toolbelt installer, and select SQL Comparison SDK. When you have installed the SQL Comparison SDK, a page called Getting Started can be accessed from the Start menu.

The Getting Started page provides links to API reference documentation, sample projects, and information on using the SQL Comparison SDK. To install or reinstall the databases used in the sample projects, on the Getting Started page, under **Sample Databases** click **Install**.

When you install the SQL Comparison SDK, the APIs comprising the SDK are installed on your computer, as well as sample C# and Visual Basic projects. You do not need to install SQL Compare, SQL Data Compare, and SQL Packager separately.

Code samples
You can get more code samples and example projects from sdk.red-gate.com.

Licensing and distribution

Licensing the SQL Comparison SDK

A SQL Comparison SDK license entitles you to distribute up to ten copies of an application created using the SDK.

When you buy a SQL Comparison SDK license, you will receive an invoice that contains your serial number. Your invoice shows how many instances of a product the serial number can be used to activate. For information on how to activate a SQL Comparison SDK license, see Exampl e: licensing a project.

Finding your serial number
If you can't find your invoice, you can view your serial numbers at http://www.red-gate.com/myserialnumbers. You will need to enter the email address and password you provided when you bought the product.

Licensing requirements

To create licensed applications using the SQL Comparison SDK, you must:

- have a valid SQL Comparison SDK, or SQL Toolbelt serial number
- include a valid *licenses.licx* file in your .NET project

For each project you create using the SQL Comparison SDK, the first time it is compiled, you are prompted for a serial number. Once you have entered a serial number, the project is licensed, and subsequent builds will succeed silently.

SQL Comparison SDK applications that are not licensed with a valid serial number display a trial expiry dialog box when compiled, and will not work after the trial period has expired.

Licenses.licx

When setting up your .NET project in Visual Studio, note that the *licenses.licx* file:

- is case sensitive
- must have a build action of *Embedded resource*
- must contain one or more of the following entries, on separate lines:

```
RedGate.SQLCompare.Engine.Database, RedGate.SQLCompare.Engine
RedGate.SQLDataCompare.Engine.ComparisonSession, RedGate.SQLDataCompare.Engine
RedGate.SQLPackager.Engine.PackagerEngine, RedGate.SQLPackager.Engine
```

  If you are using only the SQL Compare API in your application, include only the first line. To use the SQL Data Compare API, you must include the first two lines. To use the SQL Packager API, include all three lines in the file.

Example: licensing a project

To set up licensing for a project using only the SQL Compare API:

1. In the **Solution Explorer** pane of Visual Studio, right-click the project, click **Add**, and then click **New Item**.
   The Add New Item dialog box is displayed.
2. On the Add New Item dialog box, select **Text File**, and specify the name *licenses.licx*
   If the file already exists, you will be prompted to overwrite it.
   The *licenses.licx* file is displayed.
3. Add the following to *licenses.licx*, and save the file:

```
RedGate.SQLCompare.Engine.Database,RedGate.SQLCompare.Engine
```

   Entries in *licenses.licx* are case sensitive.

4. Right click *licenses.licx*, and click **Properties**. Ensure the **Build action** is set to *Embedded resource*.
   Licensing is now set up for the project. If this is the first time you have licensed the SQL Comparison SDK, you will be prompted for a serial number when you compile the project.

Licensing for ASP.NET applications

If you are using the SQL Comparison SDK in an ASP.NET application, web service, or website, you may encounter errors when licensing the application.

When you first compile an application using the SQL Comparison SDK, a dialog box is displayed prompting you to enter a serial number. The license information you enter is embedded in an assembly. Subsequent builds will succeed silently, and users will not be prompted for licensing information.

ASP.NET 2.0 compiles dynamically, so you cannot follow this standard licensing procedure for web applications.

To use the SQL Comparison SDK in ASP.NET applications, you are recommended to create a separate library (DLL) containing the SQL Comparison SDK functions you want to use. This DLL can be licensed normally when it is compiled, and then referenced in the ASP.NET project.

It may be necessary to license your application using the .NET Framework SDK, for example if the application is compiled by an automated build server. Instructions for this are provided below.

Note that it is not currently possible to license an ASP.NET application if you are using the 14 day free trial version of the SQL Comparison SDK. To license a web application, you must have a valid serial number, or contact support.

Licensing using the .NET Framework SDK

To license an ASP.NET application that uses the SQL Compare API:

1. Create a text file called *License.txt*, listing the licensed components. In this example, type:

   ```
   RedGate.SQLCompare.Engine.Database, RedGate.SQLCompare.Engine
   ```

   and save the file to the folder containing the source code for the project.
2. From the Visual Studio command prompt, use the License Compiler to create the *.licenses* file:

   ```
   lc /target:ApplicationName.Licenses.Resources.dll /complist:licence.txt
    /i:"C:\Program Files\Red Gate\SQL Comparison SDK 8\Assemblies\SQL
   Compare\RedGate.SQLCompare.Engine.dll"
   ```

   > */i:<module>* specifies the location of the components referenced by */complist*. In this example, there is only one: *RedGate.SQL Compare.Engine.dll*
   > To specify additional components, use a separate */i* switch for each one.

3. Enter the serial number when prompted.
4. From the Visual Studio command prompt, use the Assembly Linker to embed the *.licenses* file in a library:

   ```
   al /t:lib /embed:ApplicationName.Licenses.Resources.dll.licenses
    /culture:neutral /out:ApplicationName.Licenses.Resources.dll
   ```

5. Copy the library *ApplicationName.Licenses.Resources.dll* into the Bin folder of the web application.
   Licensing is now set up.

For more information on using the .NET framework SDK, refer to your .NET Framework documentation.

Distributing your SDK applications

When you purchase the SQL Comparison SDK, on its own or as part of the SQL Toolbelt, the license entitles you to distribute up to ten copies of an application created using the SDK.

If you want to distribute an application using the SDK more than ten times, contact our sales department.

For more information, see:

- Licensing the SQL Comparison SDK
- The Red Gate license agreement (PDF)
- Licensing ASP.NET applications

Required files

When you distribute the application to a user, you must also include the assemblies referenced by the APIs you are using. The SQL Comparison SDK installs these files in the following default locations:

- SQL Compare assemblies
  C:\Program Files\Red Gate\SQL Comparison SDK 10\Assemblies\SQL Compare
- SQL Data Compare assemblies
  C:\Program Files\Red Gate\SQL Comparison SDK 10\Assemblies\SQL Data Compare
- SQL Packager assemblies
  C:\Program Files\Red Gate\SQL Comparison SDK 10\Assemblies\SQL Packager

Licensing automated builds with NAnt

Microsoft Visual Studio can licence SQL Comparison SDK assemblies automatically. This can also be done without the need to have Visual Studio installed, for instance on a server that automates software builds using the NAnt automated build tool.

Before you start, ensure that any prerequisites are installed. In addition to the .NET Framework, the .NET Framework Software Development Kit (SDK) must also be installed to provide licensing functionality. This is a free download from Microsoft. Finally, install SQL Comparison SDK, which will provide the .NET assemblies necessary to create a licence available to the build server.

First, a licence needs to be created on the server. Because a form is presented requesting the SQL Comparison SDK serial number, this needs to be done manually, before any builds are configured.

- Open a command prompt and browse to the SDK folder (cd C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin)
- Notepad licence.txt
- Enter these contents into licence.txt:
  `RedGate.SQLCompare.Engine.Database, RedGate.SQLCompare.Engine<return>`

If you were also referencing Data Compare's engine, you would mention it on the next line.

- Save the file and exit.
- Run the licence compiler:

```
lc.exe /target:"MyApp.exe" /complist:"licence.txt" /i:"c:\program files (x86)\Red
Gate\SQL Comparison SDK 10\Assemblies\SQL Compare\RedGate.SQLCompare.Engine.dll"
```

At this point, you can enter your SQL Comparison SDK serial number from your Red Gate invoice, and a SQL_Toolkit_v_x.lic file will be created in the profile for all users of the computer. Once this file is present, you should never need to enter the serial number again, and automated builds of any SQL Toolkit project can be licensed automatically by NAnt.

The next step is to configure your build. Following the instructions in the SQL Comparison SDK documentation, create a licenses.licx file in your SQL Toolkit project and check it into your source control system. When your source control provider checks out the source code again, this file will be present in your build folder so that the NAnt licensing task can process it.

Now add a license task to your NAnt build script, specifying the location to licenses.licx in your build folder and the location of the Red Gate assembly references required:

```
<target name="licence">
  <license input="${build.dir}MyProject\licenses.licx" target="MyProject.exe"
output="${build.dir}MyProject\obj\MyProject.exe.licenses">
   <assemblies>
    <include name="${DataCompare.path}RedGate.SQLCompare.Engine.dll" />
    <include name="${DataCompare.path}RedGate.SQLDataCompare.Engine.dll" />
   </assemblies>
  </license>
 </target>
```

This action will create a file called MyProject.exe.licenses in the obj subfolder of your build folder. The final step is to link the .licenses file into the resources of your output assembly, however, you cannot include the .licenses file in the references section of the <vbc> or <csc> task, because this resource is already compiled and the <resources> section is meant for resources that have not yet been compiled using resgen.exe.

For this reason, an <arg> section is needed inside your <csc> or <vbc> task in order to force the compiler into including the .licences resource in the compiled assembly:

```
<arg line="/res:${build.dir}\MyProject\obj\MyProject.exe.licenses" />
```

Here is an example NAnt build script for a Visual Basic project build with the necessary Toolkit licensing in place.

```
 <?xml version="1.0"?>
<project name="Toolkit Licensing Nunit Test" default="build">
 <property name="build.dir" value="c:\NantBuilds\MyProject\" />
 <property name="DataCompare.path" value="c:\program files\red gate\sql data compare
6\" />
 <target name="licence">
  <license input="${build.dir}MyProject\licenses.licx" target="MyProject.exe"
output="${build.dir}MyProject\obj\MyProject.exe.licenses">
   <assemblies>
    <include name="${DataCompare.path}RedGate.SQLCompare.Engine.dll" />
    <include name="${DataCompare.path}RedGate.SQLDataCompare.Engine.dll" />
   </assemblies>
  </license>
 </target>
 <target name="build">
  <vbc target="winexe"
    output="${build.dir}MyProject\bin\Debug\MyProject.exe"
    debug="true"
    main="MyProject.Form1"
                  optioncompare="text"
                  optionexplicit="true"
                  optionstrict="true"
                  rootnamespace="MyProject"
                  removeintchecks="true"
    verbose="true">
   <sources>
    <include name="${build.dir}MyProject\*.vb" />
   </sources>
   <arg line="/res:${build.dir}\MyProject\obj\MyProject.exe.licenses" />
   <resources dynamicprefix="false" basedir="${build.dir}MyProject\obj"
prefix="MyProject">
    <include name="${build.dir}MyProject\*.resx"/>
   </resources>
   <imports>
    <import namespace="RedGate.SQLCompare.Engine" />
    <import namespace="RedGate.SQL.Shared" />
    <import namespace="RedGate.SQLDataCompare.Engine" />
    <import namespace="System.Threading" />
    <import namespace="System.Windows.Forms" />
    <import namespace="Microsoft.VisualBasic" />
              <import namespace="System.Collections" />
                  <import namespace="System.Diagnostics" />
                  <import namespace="System.Drawing" />
   </imports>
   <references>
    <include name="${DataCompare.path}RedGate.SQLCompare.Engine.dll" />
    <include name="${DataCompare.path}RedGate.SQLDataCompare.Engine.dll" />
    <include name="${DataCompare.path}RedGate.SQL.Shared.dll" />
    <include name="Microsoft.VisualBasic.dll" />
    <include name="System.dll" />
    <include name="System.Data.dll" />
    <include name="System.Deployment.dll" />
    <include name="System.Drawing.dll" />
    <include name="System.Windows.Forms.dll" />
    <include name="System.Xml.dll" />
   </references>
  </vbc>
 </target>
</project>
```

Licensing SDK applications in Visual Studio 2010

When building an SDK application in Visual Studio 2010, the following error may occur:

```
Mixed mode assembly is built against version 'v2.0.50727' of the runtime and cannot be loaded in the 4.0
runtime without additional configuration information.
```

To fix this, add or modify the application configuration file for *lc.exe*, which is the licence compiler for Visual Studio. Typically, this file exists in *%programfiles%\Microsoft SDKs\Windows\v7.0A\bin\NETFX 4.0 Tools*.

Inside this folder, create a new file called lc.exe.config (or modify the existing file) and enter the following information:

```
<?xml version ="1.0"?>
<configuration>
    <runtime>
        <generatePublisherEvidence enabled="false"/>
    </runtime>
      <startup useLegacyV2RuntimeActivationPolicy="true">
            <supportedRuntime version="v4.0"/>
      </startup>
</configuration>
```

Manually licensing Redgate assemblies

SQL Comparison SDK allows you to use the comparison and deployment capabilities of SQL Compare and SQL Data Compare in your own .NET projects. In order to use a SQL Comparison SDK-based application, however, licensing must be performed at build-time. This is done automatically by Microsoft Visual Studio when a *licenses.licx* file is detected, but it is also possible to licence your application manually from the command prompt. This is useful if you do not develop using the Visual Studio IDE or are experiencing difficulty in getting the licensing to work as it should.

Licenses for SQL Comparison SDK are created using *lc.exe*, which is part of Microsoft's .NET Framework Software Development Kit (SDK). *lc.exe* creates a file which needs to be built into your .NET assembly as an embedded resource. These are the steps for performing licensing of a schema comparison project called *MyApp.exe* from a command prompt, outside of Visual Studio:

1. Launch a Visual Studio command prompt from the **Tools** menu of the Visual Studio program group.
2. Change directory to your project folder where your project's source code is located.
3. Create a new *license.txt* document for the licence information.
4. Enter the following into licence.txt:

   ```
   RedGate.SQLCompare.Engine.Database, RedGate.SQLCompare.Engine
   ```

   If you were also referencing Data Compare's engine, you would mention it on the next line:

   ```
   RedGate.SQLDataCompare.Engine.ComparisonSession, RedGate.SQLDataCompare.Engine
   ```

5. Save the file and exit.
6. Run the licence compiler:

   ```
   lc.exe /target:"MyApp.exe" /complist:"licence.txt" /i:"c:\program files\red
   gate\sql compare 7\RedGate.SQLCompare.Engine.dll"
   ```

   If your project also references the Data Compare libraries, they should be included in the references here as well by adding this to the command above:

   ```
   /i:"c:\program files\red gate\sql data compare
   7\RedGate.SQLDataCompare.Engine.dll"
   ```

   This produces a file called MyApp.exe.licenses.
7. Compile the application, including your .licenses file as an embedded resource:

   ```
   csc /res:"MyApp.exe.licenses" /out:"MyApp.exe" /r:"c:\program files\red gate\sql
   compare 7\RedGate.SQLCompare.Engine.dll" /r:"c:\program files\red gate\sql
   compare 7\RedGate.Shared.SQL.dll" /r:"c:\program files\red gate\sql compare
   7\RedGate.Shared.Utils.dll" *.cs
   ```

Breaking changes in SQL Comparison SDK 10.5

Version 10.5 of the SQL Comparison SDK includes changes that might break your projects that use the SQL Compare, SQL Data Compare, or SQL Packager APIs.

To fix your projects, you might need to do one or more of the following:

- Use the new Options class
- Add new references
- Update uses of DifferenceType
- Distribute new assemblies

Use the new `Options` class

In previous versions of the SQL Comparison SDK, comparison options were specified as a bit flags enum. For example:

C#

```
Options.Default | Options.DropAndCreateInsteadOfAlter
```

In version 10.5, options are specified using the methods in the `Options` class. For example:

C#

```
Options.Default.Plus(Options.DropAndCreateInsteadOfAlter)
```

or

C#

```
Options.Default.Except(Options.IgnoreFileGroups, Options.IgnoreComments)
```

Add new references

With version 10.5, projects must reference *RedGate.Shared.ComparisonInterfaces.dll*.

Projects using the SQL Data Compare API must now also reference *RedGate.SQLCompare.Engine.dll*.

Update uses of `DifferenceType`

In version 10.5, the `DifferenceType` enum has moved from `RedGate.SQLCompare.Engine` to `RedGate.Shared.ComparisonInterfaces`.

You need to update any files that use `DifferenceType`:

1. Add `using RedGate.Shared.ComparisonInterfaces` to the list of `using` statements at the top of your files.
2. Replace all uses of `RedGate.SQLCompare.Engine.DifferenceType` with `RedGate.Shared.ComparisonInterfaces.DifferenceType`.

Distribute new assemblies

When you distribute your application, you now need to include the following assemblies:

- RedGate.Shared.ComparisonInterfaces.dll
- System.Threading.dll

- RedGate.Migrations.Core.dll
- Newtonsoft.Json.dll (distributed under the MIT license)
- System.Data.SqlLocalDb.dll (distributed under the Apache license)

For more information, see Distributing your SDK applications.

Getting more from the SQL Comparsion SDK

- Creating an HTML report of schema differences in C#
- Creating an HTML report of schema differences in Visual Basic .NET
- Creating a deployment script without batch markers
- Running SQL code inside SQL Comparison SDK applications
- Using SQL Data Compare mappings in projects using the API
- Excluding a table from a data comparison
- Executing your own SQL queries together with SDK synchronization

Creating an HTML report of schema differences in C#

SQL Compare's API does not directly support the creation of HTML reports as the user interface of the program does. By manually generating an XML document, a report can be generated by transforming the XML using the XSL template supplied with the SQL Compare program. This report code examines the differences between databases WidgetStaging and WidgetProduction, generates suitable XML, and converts it to HTML using the template SQLCompareInteractiveReportTemplate.xsl.

```
//======================================================================
// Save the following as Program.cs
//======================================================================
using System;
using System.Data;
using System.IO;
using RedGate.SQLCompare.Engine; //Reference %programfiles%\Red Gate\SQL Compare
10\RedGate.SQLCompare.Engine.dll
using RedGate.Shared.SQL; // Reference %programfiles%\Red Gate\SQL Compare
10\RedGate.SQL.Shared.dll
using System.Diagnostics; // for ProcessStartInfo
namespace SQLCompareReport
{
/// <summary>
/// Two functions -- CreateHTMLReport and ViewReport -- create an HTML report and
optionally view it.
/// </summary>
class Program
{
static void Main(string[] args)
{
using (Database dbSource = new Database(),
dbTarget = new Database())
{
// Retrieve the schema information for the two databases
Console.WriteLine("Registering databases");
dbSource.Register(new ConnectionProperties(".", "WidgetStaging"), Options.Default);
dbTarget.Register(new ConnectionProperties(".", "WidgetProduction"), Options.Default);
Console.WriteLine("Comparing Databases");
Differences dbSourceVsdbTarget = dbSource.CompareWith(dbTarget, Options.Default);
// Set the filespec for our HTML report
string ReportOutput = @"c:\Program files\Red Gate\SQL Compare 7\htmlreport.html";
// Set the XSL template to use for the report. These ship with the SQL Compare
software
string xsltemplate = @"c:\Program files\Red Gate\SQL Compare
7\SQLCompareInteractiveReportTemplate.xsl";
Console.WriteLine("Creating report...");
HTMLReport.CreateHtmlReport(ReportOutput, dbSource, dbTarget, dbSourceVsdbTarget,
Options.Default,xsltemplate);
Console.WriteLine("Finished creating {0}, viewing", ReportOutput);
HTMLReport.ViewReport(ReportOutput);
}
Console.WriteLine("Press any key to continue");
Console.ReadLine();
}
}
}
//======================================================================
// EOF Program.cs
//======================================================================
//======================================================================
// Save the following as HTMLReport.cs
```

```
//======================================================================
using System;
using System.Collections;
using System.Data;
using System.IO;
using System.Xml;
using System.Xml.Xsl;
using System.Text;
using System.Reflection;
using RedGate.SQLCompare.Engine;
using RedGate.Shared.SQL;
using System.Diagnostics;
using RedGate.Shared.Utils;
using System.Collections.Generic; // for ProcessStartInfo
namespace SQLCompareReport
{
public class HTMLReport
{
// This method will create the XML needed for the report and transform it to an HTML
page
// specified by fileName. It looks in the current folder for the template file.
// Please supply the two database objects, the Differences object that you get after a
comparison,
// and the set of options that you used for the comparison.
public static void CreateHtmlReport(string fileName,
RedGate.SQLCompare.Engine.Database dbSourceDatabase,
RedGate.SQLCompare.Engine.Database dbTargetDatabase,
RedGate.SQLCompare.Engine.Differences obDatabaseDifferences,
RedGate.SQLCompare.Engine.Options enOptions,string xlstemplate)
{
string tempFile = Path.GetTempFileName();
XslCompiledTransform xslt = new XslCompiledTransform();
        //Load the XSL template
        XsltSettings xSettings = new XsltSettings();
        xSettings.EnableScript = true;
        xslt.Load(xlstemplate, xSettings, new XmlUrlResolver());
        try
        {
                XmlTextWriter writer = new XmlTextWriter(tempFile, Encoding.Unicode);
                //Generate the raw data that will go into the report
                GenerateXml(writer, dbSourceDatabase, dbTargetDatabase,
obDatabaseDifferences, enOptions);
                writer.Close();
                xslt.Transform(tempFile, fileName);
        }
        catch (Exception e)
        {
                Console.WriteLine("Unable to generate html report " + e.Message);
        }
        finally
        {
                File.Delete(tempFile);
        }
  }
/// <summary>
/// This is the method that creates the XML data used in the report (SQL Compare v7)
/// </summary>
/// <param name="writer">XmlTextWriter object</param>
/// <param name="dbSourceDatabase">A registered database</param>
/// <param name="dbTargetDatabase">The second registered database</param>
/// <param name="obDatabaseDifferences">The differences between the two
databases</param>
/// <param name="m_Options">Set of options used during the comparison process</param>
private static void GenerateXml(XmlTextWriter writer,
RedGate.SQLCompare.Engine.Database dbSourceDatabase,
RedGate.SQLCompare.Engine.Database dbTargetDatabase,
RedGate.SQLCompare.Engine.Differences obDatabaseDifferences,
RedGate.SQLCompare.Engine.Options options)
{
        writer.WriteStartDocument();
        //Header
        writer.WriteStartElement("comparison");
        writer.WriteAttributeString("direction", "1to2");
        writer.WriteAttributeString("timestamp", DateTime.Now.ToString());
        //Datasources
        writer.WriteStartElement("datasources");
        writer.WriteStartElement("datasource");
        writer.WriteAttributeString("type", "live");
        writer.WriteAttributeString("id", "1");
        writer.WriteStartElement("server");
        writer.WriteString(dbSourceDatabase.ConnectionProperties.ServerName);
        writer.WriteEndElement(); // </server>
        writer.WriteStartElement("database");
        writer.WriteString(dbSourceDatabase.ConnectionProperties.DatabaseName);
        writer.WriteEndElement(); // </database>
        writer.WriteEndElement(); // <datasource[@id=1]>
        //Second database
        writer.WriteStartElement("datasource");
        writer.WriteAttributeString("type", "live");
        writer.WriteAttributeString("id", "2");
        writer.WriteStartElement("server");
        writer.WriteString(dbTargetDatabase.ConnectionProperties.ServerName);
        writer.WriteEndElement(); // </server>
        writer.WriteStartElement("database");
        writer.WriteString(dbTargetDatabase.ConnectionProperties.DatabaseName);
        writer.WriteEndElement(); // </database>
        writer.WriteEndElement(); // </datasource>
        writer.WriteEndElement(); // </datasources>
        //Differences collection
        writer.WriteStartElement("differences");
        foreach (Difference d in obDatabaseDifferences)
        {
        if (d.Type == DifferenceType.Equal)
                continue;
        if (!d.Selected)
                continue;
                writer.WriteStartElement("difference");
                writer.WriteAttributeString("objecttype",
d.DatabaseObjectType.ToString().ToLower());
                writer.WriteAttributeString("status",
d.Type.ToString().Trim().ToLower());
                writer.WriteAttributeString("fqn",
String.Format("{0}-{1}",d.DatabaseObjectType.ToString().ToLower(),d.Name.ToString().To
Lower()));
                switch (d.Type){
                        case DifferenceType.OnlyIn1:
                                writer.WriteStartElement("object");
                                writer.WriteAttributeString("owner",
d.ObjectIn1.Owner);
                                writer.WriteAttributeString("id", "1");
                                writer.WriteString(d.ObjectIn1.FullyQualifiedName);
                                writer.WriteEndElement();
                                writer.WriteStartElement("object");
                                writer.WriteAttributeString("owner","");
                                writer.WriteAttributeString("id", "2");
                                writer.WriteEndElement();
                        break;
                        case DifferenceType.OnlyIn2:
                                writer.WriteStartElement("object");
                                writer.WriteAttributeString("owner", "");
                                writer.WriteAttributeString("id", "1");
                                writer.WriteEndElement();
                                writer.WriteStartElement("object");
                                writer.WriteAttributeString("owner",
d.ObjectIn2.Owner);
                                writer.WriteAttributeString("id", "2");
                                writer.WriteString(d.ObjectIn2.FullyQualifiedName);
                                writer.WriteEndElement();
                        break;
                        default: // object exists in both
                                writer.WriteStartElement("object");
                                writer.WriteAttributeString("owner",
d.ObjectIn1.Owner);
                                writer.WriteAttributeString("id", "1");
                                writer.WriteString(d.ObjectIn1.FullyQualifiedName);
                                writer.WriteEndElement();
                                writer.WriteStartElement("object");
                                writer.WriteAttributeString("owner",
d.ObjectIn2.Owner);
                                writer.WriteAttributeString("id", "2");
                                writer.WriteString(d.ObjectIn2.FullyQualifiedName);
                                writer.WriteEndElement();
                        break;
        }
        // Now we write the actual SQL code for the objects in database 1 and 2
        // Since the reordering of lines is copyright code in SQL Compare
        // we are going to simply dump the SQL in the order it comes
                writer.WriteStartElement("comparisonstrings");
                Work w = new Work();
                Regions regions1 = w.ScriptObject(d.ObjectIn1, options);
                Regions regions2 = w.ScriptObject(d.ObjectIn2, options);
                // Work out which region is "shortest"
                int regionCount=regions1.Count;
                bool oneIsLonger=true;
                if (regions2.Count>regions1.Count)
                {
                        regionCount=regions2.Count;
                        oneIsLonger=false;
                }
                //loop through all SQL regions -- append the longer lines
                int j = 0;
                for (j=0; j < regioncount;="">
                {
                        //Start writing out the lines of SQL code
                        bool oneHasMoreLines = false;
                        string[] linesFrom1;
                        string[] linesFrom2;
                        try
                        {
                                linesFrom1 = regions1[j].SQL.Split('\n');
                        }
                        catch (ArgumentOutOfRangeException) // There are more regions
in region2
                        {
                                linesFrom1 = new
string[regions2[j].SQL.Split('\n').Length];
                                for (int y = 0; y < linesfrom1.length;="">
                                {
                                        linesFrom1[y] = String.Empty;
                                }
                        }
                        try
                        {
                                linesFrom2 = regions2[j].SQL.Split('\n');
                        }
                        catch (ArgumentOutOfRangeException) // There are more regions
in region1
                        {
                                linesFrom2 = new
string[regions1[j].SQL.Split('\n').Length];
                                for(int y=0;y<>
                                {
                                        linesFrom2[y] = String.Empty;
                                }
                        }
                        int sqlLineCount = linesFrom1.Length;
                        int sqlLineCount2 = linesFrom2.Length;
                        if (sqlLineCount > sqlLineCount2)
                        {
                                sqlLineCount = sqlLineCount2;
                                oneHasMoreLines = true;
                        }
                        int l=0;
                        for (; l <>
                        {
                                writer.WriteStartElement("line");
                                writer.WriteAttributeString("type",
String.Compare(linesFrom1[l],linesFrom2[l], true) != 0 ?"different" : "same");
                                // Dump the line of SQL from db1
                                writer.WriteStartElement("left");
                                writer.WriteString(linesFrom1[l].Trim());
                                writer.WriteEndElement(); // </left>
                                // ...and db2
                                writer.WriteStartElement("right");
                                writer.WriteString(linesFrom2[l].Trim());
                                writer.WriteEndElement(); // </right>
                                writer.WriteEndElement(); //</line>
                        }
                        // Write out any "leftover" SQL
                        string[] leftoverSql=linesFrom2;
                        if (oneHasMoreLines) leftoverSql=linesFrom1;
                        for (int m = l; m < leftoversql.length;="">
                        {
                                writer.WriteStartElement("line");
                                writer.WriteAttributeString("type", "different");
                                writer.WriteStartElement("left");
                                if (oneHasMoreLines)
writer.WriteString(leftoverSql[m].Trim());
                                writer.WriteEndElement(); // </left>
                                writer.WriteStartElement("right");
                                if (!oneHasMoreLines)
writer.WriteString(leftoverSql[m].Trim());
                                writer.WriteEndElement(); // </right>
                                writer.WriteEndElement(); //</line>
                        }
                }
                writer.WriteEndElement(); // </comparisonStrings>
                writer.WriteEndElement(); // </difference>
        }
        writer.WriteEndElement(); // </differences>
        writer.WriteEndElement(); // </comparison>
        writer.WriteEndDocument(); //EOF
}
//Feed the .htm file to Windows and let it start the viewer (IE)
public static void ViewReport(string sPath)
{
if (sPath == string.Empty)
return;
// view the doc
try
{
ProcessStartInfo psi = new ProcessStartInfo(sPath);
psi.UseShellExecute = true;
Process.Start(psi);
}
catch { }
}
} //end class
}
```

```
//====================================================================
// EOF HTMLReport.cs
//====================================================================
```

Creating an HTML report of schema differences in Visual Basic .NET

SQL Compare's API does not directly support the creation of HTML reports as the user interface of the program does. By manually generating an XML document, a report can be generated by transforming the XML using the XSL template supplied with the SQL Compare program. This report code examines the differences between databases WidgetStaging and WidgetProduction, generates suitable XML, and converts it to HTML using the template SQLCompareInteractiveReportTemplate.xsl.

```
'=====================================================================
' Save the following as Module1.vb
'=====================================================================
Imports RedGate.SQLCompare.Engine
Imports RedGate.Shared.SQL
Imports System.Diagnostics
Module Module1
        Sub Main()
                Using dbSource As New Database(), dbTarget As New Database()
                        'Retrieve the schema information for the two databases
                        Console.WriteLine("Registering databases")
                        dbSource.Register(New ConnectionProperties(".", 
"WidgetStaging"), Options.Default)
                        dbTarget.Register(New ConnectionProperties(".", 
"WidgetProduction"), Options.Default)
                        Console.WriteLine("Comparing Databases")
                        Dim dbSourceVsdbTarget As Differences = 
dbSource.CompareWith(dbTarget, Options.Default)
                        ' Set the filespec for our HTML report
                        Dim ReportOutput As String = "c:\Program files\Red Gate\SQL 
Compare 7\htmlreport.html"
                        ' Set the XSL template to use for the report. These ship with 
the SQL Compare software
                        Dim xsltemplate As String = "c:\Program files\Red Gate\SQL 
Compare 7\SQLCompareInteractiveReportTemplate.xsl"
                        Console.WriteLine("Creating report...")
                        HTMLReport.CreateHtmlReport(ReportOutput, dbSource, dbTarget, 
dbSourceVsdbTarget, Options.Default, xsltemplate)
                        Console.WriteLine("Finished creating {0}, viewing", 
ReportOutput)
                        HTMLReport.ViewReport(ReportOutput)
                End Using
                Console.WriteLine("Press any key to continue")
                Console.ReadLine()
        End Sub
End Module
'=====================================================================
' EOF Module1.vb
'=====================================================================
'=====================================================================
' Save the following as HTMLReport.vb
'=====================================================================
Imports System
Imports System.Collections
Imports System.Data
Imports System.IO
Imports System.Xml
Imports System.Xml.Xsl
Imports System.Text
Imports System.Reflection
Imports RedGate.SQLCompare.Engine
Imports RedGate.Shared.SQL
Imports System.Diagnostics
Imports RedGate.Shared.Utils
Imports System.Collections.Generic
Public Class HTMLReport
        ' This method will create the XML needed for the report and transform it to an
HTML page
        ' specified by fileName. It looks in the current folder for the template file.
        ' Please supply the two database objects, the Differences object that you get
after a comparison,
        ' and the set of options that you used for the comparison.
        Public Shared Sub CreateHtmlReport(ByVal fileName As String, ByVal
dbSourceDatabase As Database,ByVal dbTargetDatabase As Database, ByVal
obDatabaseDifferences As Differences, ByVal enOptions AsOptions, ByVal xlstemplate As
String)
                Dim tempFile As String = Path.GetTempFileName()
                Dim xslt As XslCompiledTransform = New XslCompiledTransform()
                'Load the XSL template
                Dim xSettings As XsltSettings = New XsltSettings()
                xSettings.EnableScript = True
                xslt.Load(xlstemplate, xSettings, New XmlUrlResolver())
                Try
                        Dim writer As XmlTextWriter = New XmlTextWriter(tempFile,
Encoding.Unicode)
                        ' Generate the raw data that will go into the report
                        GenerateXml(writer, dbSourceDatabase, dbTargetDatabase,
obDatabaseDifferences, enOptions)
                        writer.Close()
                        xslt.Transform(tempFile, fileName)
                Catch e As Exception
                        Console.WriteLine("Unable to generate html report " +
e.Message)
                Finally
                        File.Delete(tempFile)
                End Try
        End Sub
        ''' <summary>
        ''' This is the method that creates the XML data used in the report (SQL
Compare v7)
        ''' </summary>
        ''' <param name="writer">XmlTextWriter object</param>
        ''' <param name="dbSourceDatabase">A registered database</param>
        ''' <param name="dbTargetDatabase">The second registered database</param>
        ''' <param name="obDatabaseDifferences">The differences between the two
databases</param>
        ''' <param name="m_Options">Set of options used during the comparison
process</param>
        Private Shared Sub GenerateXml(ByVal writer As XmlTextWriter, ByVal
dbSourceDatabase As Database,ByVal dbTargetDatabase As Database, ByVal
obDatabaseDifferences As Differences, ByVal options As Options)
                writer.WriteStartDocument()
                ' Header
                writer.WriteStartElement("comparison")
                writer.WriteAttributeString("direction", "1to2")
                writer.WriteAttributeString("timestamp", DateTime.Now.ToString())
                'Datasources
                writer.WriteStartElement("datasources")
                writer.WriteStartElement("datasource")
                writer.WriteAttributeString("type", "live")
                writer.WriteAttributeString("id", "1")
                writer.WriteStartElement("server")
                writer.WriteString(dbSourceDatabase.ConnectionProperties.ServerName)
                writer.WriteEndElement() ' </server>
                writer.WriteStartElement("database")
                writer.WriteString(dbSourceDatabase.ConnectionProperties.DatabaseName)
                writer.WriteEndElement() ' </database>
                writer.WriteEndElement() ' <datasource[@id=1]>
                'Second database
                writer.WriteStartElement("datasource")
                writer.WriteAttributeString("type", "live")
                writer.WriteAttributeString("id", "2")
                writer.WriteStartElement("server")
                writer.WriteString(dbTargetDatabase.ConnectionProperties.ServerName)
                writer.WriteEndElement() ' </server>
                writer.WriteStartElement("database")
                writer.WriteString(dbTargetDatabase.ConnectionProperties.DatabaseName)
                writer.WriteEndElement() ' </database>
                writer.WriteEndElement() ' </datasource>
                writer.WriteEndElement() ' </datasources>
                'Differences collection
                writer.WriteStartElement("differences")
                Dim d As Difference
                For Each d In obDatabaseDifferences
                        If d.Type = DifferenceType.Equal Then Continue For
                        If d.Selected = False Then Continue For ' do not report equal
and nonselected objects
                        writer.WriteStartElement("difference")
                        writer.WriteAttributeString("objecttype",
d.DatabaseObjectType.ToString().ToLower())
                        writer.WriteAttributeString("status",
d.Type.ToString().Trim().ToLower())
                        writer.WriteAttributeString("fqn", String.Format("{0}-{1}",
d.DatabaseObjectType.ToString().ToLower(), d.Name.ToString().ToLower()))
                        Select Case d.Type
                                Case DifferenceType.OnlyIn1
                                        writer.WriteStartElement("object")
                                        writer.WriteAttributeString("owner",
d.ObjectIn1.Owner)
                                        writer.WriteAttributeString("id", "1")

writer.WriteString(d.ObjectIn1.FullyQualifiedName)
                                        writer.WriteEndElement()
                                        writer.WriteStartElement("object")
                                        writer.WriteAttributeString("owner", "")
                                        writer.WriteAttributeString("id", "2")
                                        writer.WriteEndElement()
                                Case DifferenceType.OnlyIn2
                                        writer.WriteStartElement("object")
                                        writer.WriteAttributeString("owner", "")
                                        writer.WriteAttributeString("id", "1")
                                        writer.WriteEndElement()
                                        writer.WriteStartElement("object")
                                        writer.WriteAttributeString("owner",
d.ObjectIn2.Owner)
                                        writer.WriteAttributeString("id", "2")

writer.WriteString(d.ObjectIn2.FullyQualifiedName)
                                        writer.WriteEndElement()
                                Case Else ' object exists in both
```

```
                                        writer.WriteStartElement("object")
                                        writer.WriteAttributeString("owner",
d.ObjectIn1.Owner)
                                        writer.WriteAttributeString("id", "1")

writer.WriteString(d.ObjectIn1.FullyQualifiedName)
                                        writer.WriteEndElement()
                                        writer.WriteStartElement("object")
                                        writer.WriteAttributeString("owner",
d.ObjectIn2.Owner)
                                        writer.WriteAttributeString("id", "2")

writer.WriteString(d.ObjectIn2.FullyQualifiedName)
                                        writer.WriteEndElement()
                        End Select
                        ' Now we write the actual SQL code for the objects in database
1 and 2
                        ' Since the reordering of lines is copyright code in SQL
Compare
                        ' we are going to simply dump the SQL in the order it comes
                        writer.WriteStartElement("comparisonstrings")
                        Dim w As Work = New Work()
                        Dim regions1 As Regions = w.ScriptObject(d.ObjectIn1, options)
                        Dim regions2 As Regions = w.ScriptObject(d.ObjectIn2, options)
                        ' Work out which region is "shortest"
                        Dim regionCount As Integer = regions1.Count
                        Dim oneIsLonger As Boolean = True
                        If regions2.Count > regions1.Count Then
                                regionCount = regions2.Count
                                oneIsLonger = False
                        End If
                        'loop through all SQL regions -- append the longer lines
                        Dim j As Integer = 0
                        For j = 0 To regionCount - 1
                                'Start writing out the lines of SQL code
                                Dim oneHasMoreLines As Boolean = False
                                Dim linesFrom1() As String
                                Dim linesFrom2() As String
                                Try
                                        linesFrom1 = regions1(j).SQL.Split(vbCrLf)
                                Catch a As ArgumentOutOfRangeException ' There are
more regions in region2
                                        ReDim Preserve
linesFrom1(regions2(j).SQL.Split(vbCrLf).Length)
                                        Dim y As Integer = 0
                                        For y = 0 To linesFrom1.Length - 1
                                                linesFrom1(y) = String.Empty
                                        Next y
                                End Try
                                Try
                                        linesFrom2 = regions2(j).SQL.Split(vbCrLf)
                                Catch a As ArgumentOutOfRangeException ' There are
more regions in region1
                                        ReDim Preserve
linesFrom2(regions1(j).SQL.Split(vbCrLf).Length)
                                        Dim y As Integer = 0
                                        For y = 0 To linesFrom2.Length - 1
                                                linesFrom2(y) = String.Empty
                                        Next y
                                End Try
                                Dim sqlLineCount As Integer = linesFrom1.Length
                                Dim sqlLineCount2 As Integer = linesFrom2.Length
                                If sqlLineCount > sqlLineCount2 Then
                                        sqlLineCount = sqlLineCount2
                                        oneHasMoreLines = True
                                End If
                                Dim l As Integer = 0
                                For l = 0 To sqlLineCount - 1
                                        writer.WriteStartElement("line")
                                        writer.WriteAttributeString("type", IIf(Not
String.Compare(linesFrom1(l), linesFrom2(l), True) = 0, "different", "same"))
                                        ' Dump the line of SQL from db1
                                        writer.WriteStartElement("left")
                                        writer.WriteString(linesFrom1(l).Trim())
                                        writer.WriteEndElement() ' </left>
                                        ' ...and db2
                                        writer.WriteStartElement("right")
                                        writer.WriteString(linesFrom2(l).Trim())
                                        writer.WriteEndElement() ' </right>
                                        writer.WriteEndElement() '</line>
                                Next l
                                'Write out any "leftover" SQL
                                Dim leftoverSql() As String = linesFrom2
                                If oneHasMoreLines = True Then leftoverSql =
linesFrom1
                                Dim m As Integer = 0
                                For m = l To leftoverSql.Length - 1
                                        writer.WriteStartElement("line")
                                        writer.WriteAttributeString("type",
"different")
                                        writer.WriteStartElement("left")
                                        If oneHasMoreLines = True Then
writer.WriteString(leftoverSql(m).Trim())
                                        writer.WriteEndElement() ' </left>
                                        writer.WriteStartElement("right")
                                        If oneHasMoreLines = False Then
writer.WriteString(leftoverSql(m).Trim())
                                        writer.WriteEndElement() ' </right>
                                        writer.WriteEndElement() '</line>
                                Next m
                        Next j
                        writer.WriteEndElement() ' </comparisonStrings>
                        writer.WriteEndElement() ' </difference>
                Next
                writer.WriteEndElement() ' </differences>
                writer.WriteEndElement() ' </comparison>
                writer.WriteEndDocument() 'EOF
        End Sub
        ' Feed the .htm file to Windows and let it start the viewer (IE)
        Public Shared Sub ViewReport(ByVal sPath As String)
                If sPath = String.Empty Then Return
                'view the doc
                Try
                        Dim psi As ProcessStartInfo = New ProcessStartInfo(sPath)
                        psi.UseShellExecute = True
                        Process.Start(psi)
                Catch
                End Try
        End Sub
End Class
```

```
'=====================================================================
' EOF HTMLReport.vb
'=====================================================================
```

Creating a deployment script without batch markers

When saving a deployment script produced by SQL Comparison SDK, you may want to save the script for later use by the ADO .NET client (SqlCommand). Since the script produced using SQL Toolkit's BlockSaver class in meant for SQL Server Management Studio, it includes batch separators (GO commands) that aren't understood by ADO .NET and will cause errors when run for this reason.

It is possible to save the SQL Toolkit ExecutionBlock in a way where batch markers are not included in the script by looping through the individual batches and saving only batches that are not marked as a batch marker. Here is an example code snippet which assumes that you have already populated an ExecutionBlock called "block":

```
using (System.IO.StreamWriter sw=new System.IO.StreamWriter(@"c:\temp\test.sql"))
 {
  for (int i = 0; i < block.batchcount;="">
  {
   Batch b = block.GetBatch(i);
   if (b.Marker == false)
   {
    sw.Write(b.Contents + "\r\n");
   }
  }
 }
```

Please note that you may want to use the NoSqlPlumbing option as well when you create the ExecutionBlock to prevent transactional operations from appearing in the script. ADO .NET has its own methods for rolling back and committing transactions if necessary.

```
work.BuildFromDifferences(stagingVsProduction, Options.Default |
Options.NoSQLPlumbing, true);
```

Running SQL code inside SQL Comparison SDK applications

In some cases, you may want to augment the script produced by the SQL Compare or SQL Data Compare API with SQL code that you have written yourself.

SQL Comparison SDK can compare and synchronize schema and data between two instances of SQL Server, however the API is limited to executing the code produced by the API. For instance, an object called BlockExecutor can run the SQL commands contained in an ExecutionBlock object, but it cannot run a SQL statement that you have written yourself.

In order to execute these "custom" SQL Scripts, the libraries provided by ADO .NET would be the choice for running your own SQL code. In the following example, a SqlCommand object is used to run a SQL command that will create a table in the WidgetStaging database.

C# code

```
using System.Data.SqlClient;
...
CreateCommand("CREATE TABLE [test](id int, data varchar(50))", "Data
Source=localhost;Initial Catalog=WidgetStaging;Integrated Security=SSPI");
...
private static void CreateCommand(string queryString,
        string connectionString)
{
        using (SqlConnection connection = new SqlConnection(
                              connectionString))
        {
                SqlCommand command = new SqlCommand(queryString, connection);
                command.Connection.Open();
                command.ExecuteNonQuery();
                connection.Close();
        }
}
```

Using SQL Data Compare mappings in projects using the API

A powerful feature of SQL Data Compare is the ability to use the object mappings set in a saved SQL Data Compare project file programmatically.

Line 65 shows the method `ReplayUserActions`, which allows the mappings to be accessed from the project file.

Visual Basic code

```
Imports RedGate.SQL.Shared
Imports RedGate.SQLCompare.Engine
Imports RedGate.SQLDataCompare.Engine
Imports RedGate.SQLDataCompare.Engine.ResultsStore


Module Test
Sub test()
Dim e As New SyncNow( "D:\TLHORT\Misc Files\SF- Sync.sdc")
e.Synchronize()
End Sub
End Module

Public Class SyncNow
Dim pathToProjectFile As String

Dim project As RedGate.SQLDataCompare.Engine.Project
Dim db1, db2 As Database
Dim mappings As SchemaMappings
Dim session As ComparisonSession
Dim livedb As LiveDatabaseSource

Dim provider As SqlProvider
Dim block As ExecutionBlock

Dim executor As BlockExecutor
Public Sub New( ByVal PhysicalPathToProjectFile As String)
Me.pathToProjectFile = PhysicalPathToProjectFile


End Sub

Public Sub Synchronize()
Try
Compare()
GetScript()
ExecuteScript()
Catch ex As Exception
Throw
Finally
session.Dispose()
block.Dispose()
End Try
End Sub

Private Sub Compare()
project = project.LoadFromDisk(pathToProjectFile)

Dim db1 As New Database
Dim db2 As New Database
```

```
livedb = DirectCast(project.DataSource1, LiveDatabaseSource)
db1.RegisterForDataCompare(livedb.ToConnectionProperties(), Options.Default)


livedb = DirectCast(project.DataSource2, LiveDatabaseSource)
db2.RegisterForDataCompare (livedb.ToConnectionProperties(), Options.Default)


mappings = New SchemaMappings
mappings.Options = project.Options

'The table mappings are NOT imported from the project file
mappings.CreateMappings(db1, db2)
project.ReplayUserActions(mappings)
session = New ComparisonSession
session.Options = project.Options
session.CompareDatabases(db1, db2, mappings)
End Sub

Private Sub GetScript()
provider = New SqlProvider
block = provider.GetMigrationSQL(session, True)
End Sub

Private Sub ExecuteScript()
executor = New BlockExecutor
executor.ExecuteBlock(block, livedb.ServerName, livedb.DatabaseName,
livedb.IntegratedSecurity , livedb.UserName, livedb.Password)
```

```
End Sub

End Class
```

Excluding a table from a data comparison

In the commercial version of SQL Data Compare, it is possible to exclude a whole SQL Server table from a comparison. This is also possible using the SQL Comparison SDK data comparison libraries using the mapping class.

Each Data Compare mapping has a property called "Include", which you can set to true or false, depending on whether the table's data should be included in the comparison. Here is an example where a table called *Widgets*, which had previously been successfully mapped using the CreateMappings method, is being excluded from the data comparison:

C#

```
Database db1=new Database();
 Database db2=new Database();

 db1.RegisterForDataCompare(new ConnectionProperties(".", "WidgetDev"),
Options.Default);
 db2.RegisterForDataCompare(new ConnectionProperties(".", "WidgetLive"),
Options.Default);

 // Create the mappings between the two databases
 TableMappings mappings = new TableMappings();
 mappings.CreateMappings(db1.Tables, db2.Tables);
mappings["[dbo].[Widgets]"].Include=false;

...
```

Executing your own SQL queries together with SDK synchronization

When executing a synchronization script created by the SQL Comparison SDK, it may be desired that some ad-hoc queries be intermixed with the SQL produced by the Red Gate APIs.

Because the BlockExecutor class can only run SQL code by converting ExecutionBlocks to SQL code and submitting them to the SQL Server, custom SQL cannot be introduced into the query stream. It is possible, however, to break an ExecutionBlock into individual query batches and running them using the .NET Framework's ADO .NET methods.

In the following C# example, a SQL query `SET LANGUAGE us_english` needs to be submitted before the synchronization produced by the SQL Data Compare Engine. First, a connection is made to the server using the connection properties of the second database. Then a transaction is created. The custom SQL query is run first, then each batch of SQL from the ExecutionBlock in order. Finally, the transaction is committed. If any errors occur during the execution of the SQL script, then the error is written to the console and the transaction will be rolled back.

```
using System;
using RedGate.SQLCompare.Engine;
using System.IO;
using RedGate.Shared.SQL.ExecutionBlock;
using System.Data.SqlClient;
namespace CompareTwoDatabases
{
        /// <summary>
        /// Demonstration that compares the widgetStaging database to the
widgetProduction database
        /// and synchronizes them using the ADO .NET libraries
        /// </summary>
        class Program
        {
                static void Main(string[] args)
                {
                        using (Database widgetStaging = new Database(),
                        widgetProduction = new Database())
                        {
                                // Retrieve the schema information for the two
databases
                                widgetStaging.Register(new ConnectionProperties(".",
"WidgetStaging"), Options.Default);
                                widgetProduction.Register(new
ConnectionProperties(".", "WidgetProduction"), Options.Default);
                                // Compare widgetStaging to widgetProduction.
Comparing in this order makes WidgetProduction the second database
                                Differences stagingVsProduction =
widgetStaging.CompareWith(widgetProduction, Options.Default);
                                // Select the differences to include in the
synchronization. In this case, we're using all differences.
                                foreach (Difference difference in stagingVsProduction)
                                {
                                        difference.Selected = true;
                                }
                                Work work = new Work();
                                // Create the migration without the transactional bits
--
                                //we will let ADO .NET manage the transactions using
SqlTransaction
                                // The script is to be run on widgetProduction so the
runOnTwo parameter is true
                                work.BuildFromDifferences(stagingVsProduction,
Options.Default | Options.NoSQLPlumbing,true);
                                // We can now access the messages and warnings
                                Console.WriteLine("Messages:");
                                foreach (Message message in work.Messages)
                                {
                                        Console.WriteLine(message.Text);
                                }
                                Console.WriteLine("Warnings:");
                                foreach (Message message in work.Warnings)
                                {
                                        Console.WriteLine(message.Text);
                                }
                                // Disposing the execution block when it's not needed
any more is important to ensure
                                // that all the temporary files are cleaned up
                                using (ExecutionBlock block = work.ExecutionBlock)
                                {
                                        // Display the SQL used to synchronize
                                        Console.WriteLine("SQL to synchronize:");
                                        Console.WriteLine(block.GetString());
                                        // Make a connection string from the second
database connection properties (runOnTwo)
                                        System.Data.SqlClient.SqlConnection conn =
newSystem.Data.SqlClient.SqlConnection("Data Source=" +
widgetProduction.ConnectionProperties.ServerName +";Initial Catalog=" +
widgetProduction.ConnectionProperties.DatabaseName + ";Integrated Security=SSPI");
                                        System.Data.SqlClient.SqlCommand cmd =
conn.CreateCommand();
                                        conn.Open();
                                        SqlTransaction trans =
conn.BeginTransaction("MyTransaction");
                                        cmd.Transaction = trans;
//Run the first command
cmd.CommandText = "SET LANGUAGE us_english";
cmd.ExecuteNonQuery();
                                        //Run batches
                                        for (int i = 0; i < block.batchcount;="">
                                        {
                                                Batch b = block.GetBatch(i);
                                                try
                                                {
                                                        if (!b.Marker) //Do not run
blocks that are simply "GO"
                                                        {
                                                                cmd.CommandText =
b.Contents;
                                                                cmd.ExecuteNonQuery();
                                                        }
                                                }
                                                catch
(System.Data.SqlClient.SqlException se)
                                                {
                                                        Console.WriteLine("Execute SQL
failed: " + se.Message);
                                                        trans.Rollback();
                                                }
                                        }
                                        trans.Commit();
                                }
                                Console.WriteLine("Press [Enter]");
                                Console.ReadLine();
                        }
```

```
                    }
          }
}
```

Troubleshooting

- Troubleshooting the SQL Comparison SDK
- 'Error 1603' occurring during installation
- SQL Compare deployment error 'Full-Text Search is not installed'
- Application licensing invalidated by renaming assembly
- Compatibility of SQL Comparison SDK in 64-bit environments
- Licenses.licx is not a valid Win32 application

Troubleshooting the SQL Comparison SDK

Often there are problems licensing SQL Comparison SDK applications, especially when upgrading versions. The following information outlines the common causes and solutions for problems that occur when applying a Red Gate serial number to a programming project that integrates the Red Gate SQL Comparison APIs.

> Before you begin, make sure you have a valid activated SQL Comparison SDK serial number. If you don't, the applications you distribute will display a trial expiry dialog box and eventually stop working at the deployment site.
>
> It's possible for an SDK application to continue functioning on the computer used for application development if the licence has been activated previously, so it's important to test the licensing has been applied correctly before deploying the application to your users.

Setting up licensing for an application that integrates the SDK's data comparison APIs

Install SQL Data Compare (or Red Gate SQL Bundle containing SQL Data Compare) and note down the installation folder. By default, the location is *C:\Program Files\Red Gate\SQL Data Compare <version>*, where *<version>* is the major version number of the product.

When adding a reference to the Red Gate components in your Visual Studio project, the assemblies you choose are found in this location. Visual Studio should also automatically add this location to your references path, so the necessary Red Gate licensing components will function properly at build time as well. In Windows Vista, make sure you open Visual Studio with an admin account.

1. Make sure you have a valid serial number (if you do not have a valid, activated serial number, the applications you distribute will display a trial expiry dialog box and will stop working when your trial expires.
2. Add a licenses.licx file to your C# or Visual Basic .NET project in Visual Studio. If a *licenses.licx* file does not exist in your Visual Studio .NET project, you must create one by right-clicking the project, selecting **Add > New Item > Text File** and name the file *licenses.licx*. Add the following two lines (this is case sensitive):

```
RedGate.SQLCompare.Engine.Database,RedGate.SQLCompare.Engine
RedGate.SQLDataCompare.Engine.ComparisonSession,RedGate.SQLDataCompare.Engine
```

3. Compile your application (click Build in Visual Studio or compile using the .NET compiler of your choice).
4. If you have a valid licenses.licx file, you will be prompted for your serial number if this is the first time you've built a SQL Toolbelt application.
5. In version 5 of the APIs, each time a new project is created, you are prompted for the serial number. In version 6 and up, the licence is available for all future SDK projects once activated.
6. When your application is built correctly, the end-user can run it without having to licence it again. Otherwise, the end-user will be presented with a dialog informing them that the SQL Comparison SDK is not licensed.
7. Distribute your application to your end-user. You must distribute the following files with your application:

- RedGate.SQL.Shared.dll
- RedGate.SQLCompare.Engine.dll
- RedGate.SQLCompare.Rewriter.dll
- RedGate.SQLCompare.ASTParser.dll
- RedGate.SQLDataCompare.Engine.dll

  If the compare to backup functionality from SQL Data Compare API is used, also include:
- RedGate.BackupReader.dll
- RedGate.SQLCompare.BackupReader.dll

You don't need to distribute RedGate.Licensing.Client.dll as it is only used by Visual Studio during compile time. The licence is embedded in the assembly by the compiler.

Setting up licensing for an application that integrates the SDK's schema comparison APIs

For SQL Compare API, the approach is similar. Again, all references should be pointing to the same directory:

C:\Program Files\Red Gate\SQL Compare <version>

1. If you do not have a valid, activated serial number, the applications you distribute will display a trial expiry dialog box and will stop working after 14 days.
2. Add a licenses.licx file to your C# or Visual Basic .NET project in Visual Studio. If a *licenses.licx* file does not exist in your Visual Studio

.NET project, you must create one by right-clicking the project, selecting **Add > New Item > Text File** and name the file *licenses.licx*. Add the following line (this is case sensitive):

```
RedGate.SQLCompare.Engine.Database,RedGate.SQLCompare.Engine
```

3. If you have a valid serial number and a valid licenses.licx file, your executable file will run without displaying the trial expiry dialog box.
4. Distribute your application. You must distribute the following files:

- RedGate.SQL.Shared.dll
- RedGate.SQLCompare.Engine.dll
- RedGate.SQLCompare.Rewriter.dll
- RedGate.SQLCompare.ASTParser.dll

You do not need to distribute RedGate.Licensing.Client.dll as it is only used by Visual Studio during compile time. The licence is embedded in the assembly by the compiler.

Licenses.licx

A few guidelines for the licenses.licx file:

- Don't copy the licenses.licx file from another folder manually, for instance using a command prompt or Windows Explorer. It should be added as a new item at the Visual Studio project level with a build action of "Embedded Resource". Visual Studio 2003 sometimes ignores changes to embedded resources, so at times it may be advisable to simply recreate the file if changes that you make are not being enforced. If possible, use Explorer and delete licenses.licx, then go back into the project, right-click, add new, file, call it licenses.licx.
- Check the properties of licenses.licx and make sure the build action says 'Embedded Resource'.
- The entries in licenses.licx are case-sensitive, so for SQL Data Compare, so check the entries carefully
- If you go to build the project, and get an error 'unable to resolve type...' , licenses.licx has been picked up by Visual Studio, but the format is wrong.
- If a licence file produced as the result of a successful activation is present on the build computer, the build will silently succeed. To verify that your assembly has been licensed successfully, you may use a tool such as .NET Reflector to open your assembly, expand 'resources', locate a resource called *<YourAssemblyFileName>.licenses*, view the resource, and make sure that the resource contains `RedGate.SQLCompare.Engine.Database` and also `RedGate.SQLDataCompare.Engine.ComparisonSession` if you had also referenced the Data Compare Engine in your project.
- SQL Comparison SDK 6 and later will retain the product license in a global location, so it is only necessary to activate the product once. Subsequent builds will succeed silently without having to re-enter the product serial number. Versions lower than 6 would retain the activated licence in an application-specific location, so the SDK would need to be re-activated every time a new SDK project was created. This is useful because it makes a shortcut possible where the SQL Toolbelt serial number does not need to be specified for every new project. Once an SDK project is activated on the build machine, a licence file (.lic) is saved to your hard disk. Moving this file into the references path for any new project on the same machine should suppress the serial number requester and directly enter the licensing details from the licence file instead.
- You do not need to distribute `RedGate.Licensing.Client.dll` as it is only used by Visual Studio during compile time. The licence is embedded in the assembly by the compiler and there is no further need for this file if the SDK had been licensed successfully. If the application being deployed shows an error message relating to a binding failure to this assembly, this indicates that the licensing for your assembly had not been successful and it needs to be rebuilt and properly licensed.

Removing the serial number that has already been activated

If the registry key below is removed then you can reactivate the product with a different serial number:

HKEY_LOCAL_MACHINE\SOFTWARE\Red Gate\Licensing\<product>\<version>\SerialNumber

(<product> will either be SQL Comparison SDK or SQL Toolbelt)

'Error 1603' occurring during installation

While attempting to install the SQL Comparison SDK, you may see the following error message:

```
Error: 1603
```

SQL Comparison SDK can fail to install if your *My Documents* folder is located on a mapped drive or network share. The installer fails when attempting to update the reference paths in your C# and Visual Basic project files.

To avoid this, you can temporarily change your *My Documents* folder to one on the local disk:

1. In Windows Explorer, right-click your *My Documents* folder and select **Properties**.
2. Right-click the current location and select **Copy**.
3. Change the location to an empty folder on a local disk, for instance C*:\temp-profile*.
4. When asked if you want to copy all of the files to the new folder, select **No**.
5. Run the SQL Comparison SDK installation again.
6. After installation, open the properties of the *My Documents* folder.
7. In the **Location** box, paste the location you copied earlier.
8. Copy the folder *C:\temp-profile\SQL Comparison SDK 8 Samples* to your *My Documents* folder.

SQL Compare deployment error 'Full-Text Search is not installed'

An error may occur when running a deployment produced by SQL Compare API:

```
Full-Text Search is not installed, or full-text component cannot be loaded
```

The deployment scripts have the following:

```
GO
PRINT N'Adding full text indexing'
GO
sp_fulltext_database N'enable'
GO
```

And if the script is run then the message appears:

```
Msg 15601, Level 16, State 1, Procedure sp_help_fulltext_catalogs
Full-Text Search is not enabled for the current database. Use sp_fulltext_database to enable Full-Text
Search. The functionality to disable and enable full-text search for a database is deprecated. Please
change your application.
```

This may occur because SQL Server 2000's full-text indexing features have not been installed. In order to full-text enable a database, SQL Server 2000 setup must be run to install full-text indexing. If the deployment is going to add full-text indexes to the target database, it is also necessary to create any required full-text catalogs manually, as SQL Compare's API will not do this for you. It would be possible to use the ADO .NET client to create a full-text catalog manually:

```
using System.Data.SqlClient;
SqlConnection conn=new SqlConnection("data source=localhost;initial
catalog=master;Integrated Security=SSPI");
SqlCommand cmd=new SqlCommand("exec sp_fulltext_catalog 'MyCatalog','Create'", conn);
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();
```

The alternative to installing and enabling full-text indexing on the SQL Server would be to instruct the API to ignore any full-text catalogs that it finds in the database you are deploying to.

Set this option with any others you require by joining them together with a bitwise `OR` operator `|`, for example:

```
Options options=Options.Default | Options.IgnoreFullTextIndexing;
```

In the Visual Basic language, there is no distinction between bitwise and logical operators, hence the `Or` operator would be used:

```
Dim o As Options=Options.Default Or Options.IgnoreFullTextIndexing
```

This will disable the comparison and deployment of full text objects.

Application licensing invalidated by renaming assembly

When deploying an assembly utilizing SQL Comparison SDK to a remote site where SQL Comparison SDK is not already installed and activated, you may be asked to activate the SDK, even though the assembly had been built and licensed correctly.

The problem occurs after renaming an assembly. SQL Comparison SDK assemblies cannot be renamed after they are built, because the licensing system requires the licence resource embedded into an assembly to bear the same name as the assembly file name.

You can either change the assembly back to its original name, or change the settings in your Visual Studio project so that the output assembly is compiles as a different name and deploy the assembly built with the desired name.

Compatibility of SQL Comparison SDK in 64-bit environments

The SQL Comparison SDK assemblies are compiled as "any CPU" as of version 7.1, however, any reference to the RedGate.SQLCompare.BackupReader assembly will require your application to be compiled as 32-bit, as this assembly has got the 32-bit flag set. You can compile an SDK application on a 64-bit machine as a 32-bit application, but the Visual Studio environment needs to be adjusted to compile your project in 32-bit mode.

For more information, see: Licenses.licx is not a valid Win32 application

Licenses.licx is not a valid Win32 application

Versions: 3,4,5,6,7.1*
Platform: 64-bit editions of Windows
Fix version: SDK 7.0 assemblies are compiled as "Any CPU" and should not encounter this issue
*NB this error will also happen in version 7.1, but only when RedGate.BackupReader.dll is used. This DLL will only run properly in a 32-bit environment, so referencing this DLL in your project will require you to compile it specifically for x86.

When compiling a Toolkit application in Visual Studio 2005 on a 64-bit operating system, a BadImageFormatException is thrown indicating that Licenses.licx is not a valid Win32 application.

The root of the issue is that the licence compiler (lc.exe) that is part of the Framework comes in a 32-bit and a 64-bit version. When compiling a Toolkit application on a 64-bit OS, the 64-bit lc.exe is invoked, leading to the exception message.

You can work around this by setting pre and post-build events to control the compilation of the licence. This, in addition to choosing the target CPU as 32-bit, will allow your project to compile and licence successfully. In VS 2005, right-click the project and select properties. Next, click 'Build Events'.
Set a new pre-build event:
c:\WINDOWS\Microsoft.NET\Framework64\v2.0.50727\Ldr64.exe setwow
Set a new post-build event:
c:\WINDOWS\Microsoft.NET\Framework64\v2.0.50727\Ldr64 set64

The paths may differ based on the installation folder of your .NET Framework.

Release notes and other versions

Version 11.4 (current)	December 21st 2015	Release notes	Documentation
Version 11.3	September 1st, 2015	Release notes	
Version 11.2	June 24th, 2015	Release notes	
Version 11.1	February 2nd, 2015	Release notes	
Version 11.0	November 26th, 2014	Release notes	
Version 10.7	May 15th, 2014	Release notes	Documentation
Version 10.5	February 4th, 2014	Release notes	
Version 10.0	March 2nd, 2012	Release notes	
Version 8.0	March 20th, 2009	Release notes	Documentation (web help)

SQL Comparison SDK was previously known as SQL Toolkit.

SQL Comparison SDK 10.7 release notes

May 15th, 2014

New features

- Minor bug fixes
- Limited SQL Server 2014 support – will connect to SQL Server 2014 databases if no new features are in use

SQL Comparison SDK 10.5 release notes

February 4th, 2014

Features

- Now supports Microsoft Azure SQL databases and SQL Server 2012 databases
- Limited SQL Server 2014 support – will connect to SQL Server 2014 databases if no new features are in use

Version 10.5 of the SQL Comparison SDK includes changes that might break your projects that use the SQL Compare, SQL Data Compare, or SQL Packager APIs.

For more information, see Breaking changes in SQL Comparison SDK 10.5.

SQL Comparison SDK 10.0 release notes

March 2nd, 2012

SQL Comparison SDK 10 contains the following APIs:

- SQL Compare 10.0
- SQL Data Compare 10.0
- SQL Packager 6.6 (updated to handle new collations and limited SQL Server 2012 support)

The improvements to all three engines include Azure database support and script folder support for Data Compare for the first time in a main SDK release.

There are also new examples and updated API documentation available here.

SQL Comparison SDK 8.0 release notes

Version 8.0 – March 30th, 2009

Version 8.0 concentrates on usability enhancements, making it easier to get started with the SDK.

There is now a single installer for all code samples and required assemblies.

The code samples and example projects are easier to install and run, as well as providing better feedback.

A new Getting Started page provides an installer for the example databases, links to API reference documentation, and information on using the SQL Comparison SDK.

Version 8 comprises the following APIs:

- SQL Compare 8.0
- SQL Data Compare 7.1
- SQL Packager 6.0