

# **SQL Packager 5.5**

July 2008

Note: these pages apply to a version of this product that is not the current released version.

For the latest support documentation, please see <http://documentation.red-gate.com>

# Contents

---

Getting started.....	2
Worked example: Packaging a database as an .EXE .....	3
Worked example: Packaging an upgrade as a C# project .....	11
Working with projects.....	19
Specifying the package contents.....	21
Previewing the SQL scripts.....	27
Creating an .EXE.....	29
Creating a C# project.....	31
Setting packaging options .....	33
Running the package .....	34
Understanding the results.....	38
Upgrading databases on different SQL Server versions .....	41
Upgrading the database structure and data.....	43
Troubleshooting .....	44
Common error messages .....	47
Using the command line interface.....	49
Basic command line features.....	50
Using XML to specify command line arguments.....	52
Integrating the command line with applications.....	54
Frequently asked questions for the command line .....	55
Acknowledgements .....	57
Entering extra package information .....	58
Specifying a package type.....	60
Setting data packaging options .....	61
Setting SQL Packager options .....	64
Setting schema packaging options.....	65

## Getting started

---

SQL Packager enables you to package the structure and contents of a Microsoft® SQL Server™ database, database upgrade, or any SQL script, as a .NET executable file or a C# project.

You can use SQL Packager to package the database structure, data, or both, for installation or deployment. SQL Packager also enables you to compress your package for reduced storage overheads and faster deployment and distribution of databases.

If you package the database as a C# project, you can customize the package. For example, you can edit the forms created in the project to customize the appearance of the graphical user interface that is displayed when you run the package.

You can use SQL Packager to package SQL Server 2008, SQL Server 2005, and SQL Server 2000 databases.

### SQL Packager: step-by-step

1. Create or open a project (page 19) to package a database, database upgrade, or any SQL script.
2. Specify the database objects and tables to package. (page 21)
3. Preview the SQL scripts. (page 27)
4. Specify the package type and details. You can create an .EXE package (page 29), create a C# package (page 31), or choose to launch or save the package creation script. (page 27)
5. Run the package. (page 34)

### Worked examples

Learn more about SQL Packager by following one of these detailed examples:

- packaging a database as an .EXE (page 3)
- packaging an upgrade as a C# project (page 11) Worked example: Packaging a database as an .EXE

This worked example demonstrates how to package a database as an .EXE (.NET executable) and run the executable to create a copy of the database.

## Worked example: Packaging a database as an .EXE

---

In the example, the Magic Widget Company has a SQL Server database running on a live Web server. They have created a database of their products, which now needs to be packaged for deployment to the sales department.

You will see how to:


1. Set up the database if you want to follow the example on your own system.  
You will need access to a SQL Server to do this.
2. Specify the contents of the package.
3. Preview the SQL scripts.
4. Generate the package as an .EXE (.NET executable).
5. Run the package to create a copy of the database.

### Setting up the database

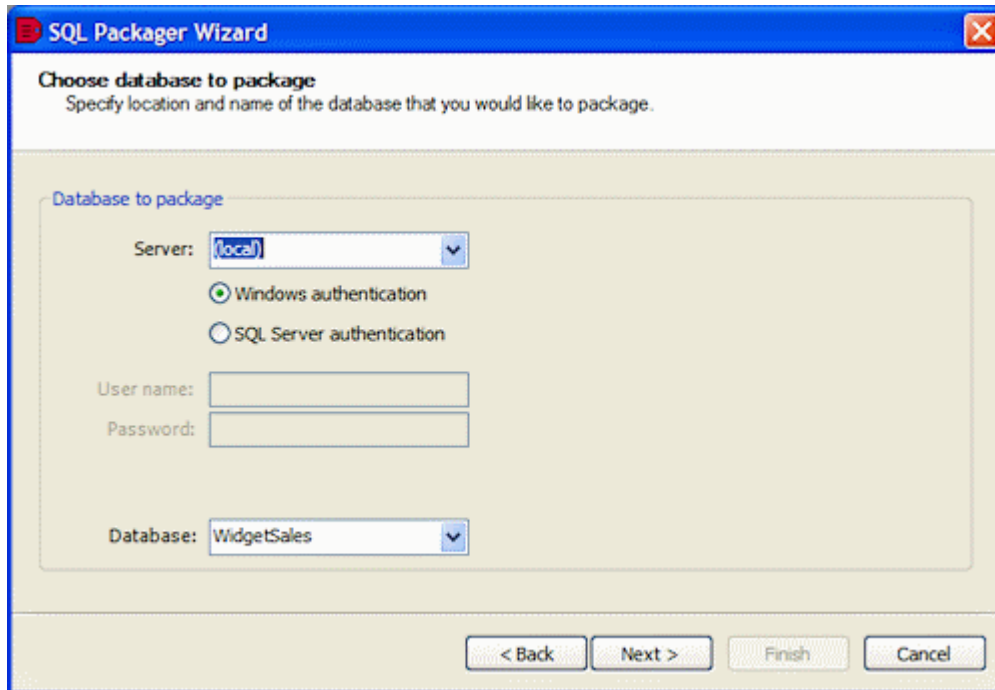
The worked example packages the **WidgetSales** database. To create this database on your SQL Server:

1. If it exists already, delete the database **WidgetSales** from your SQL Server.
2. Click here ([/support/SQL\\_Packager/help/5.5/SP\\_WECreatenET.sql](/support/SQL_Packager/help/5.5/SP_WECreatenET.sql)) to view the SQL creation script for the database.
3. Copy the script, paste it in your SQL application, and then run it.  
The database is created and populated with data.

### Specifying the package contents

1. If you have not yet started SQL Packager, select it from your **Start** menu; if SQL Packager is already running, click  **New Project**.
2. On the **Choose a project type** page of the Packager Wizard, select **Package a database**, and click **Next**.  
The **Choose database to package** page of the Packager Wizard is displayed.
3. In the **Server** box, under **Database to package**, type or select the name of the SQL Server on which you created the database.
4. Select the authentication method, and for **SQL Server authentication** enter the **User name** and **Password**.
5. In the **Database** box, type or select *WidgetSales*.

If *WidgetSales* is not displayed in the **Database** list, right-click in the **Database** box and click **Refresh**, or scroll to the top of the list and click **Refresh**.



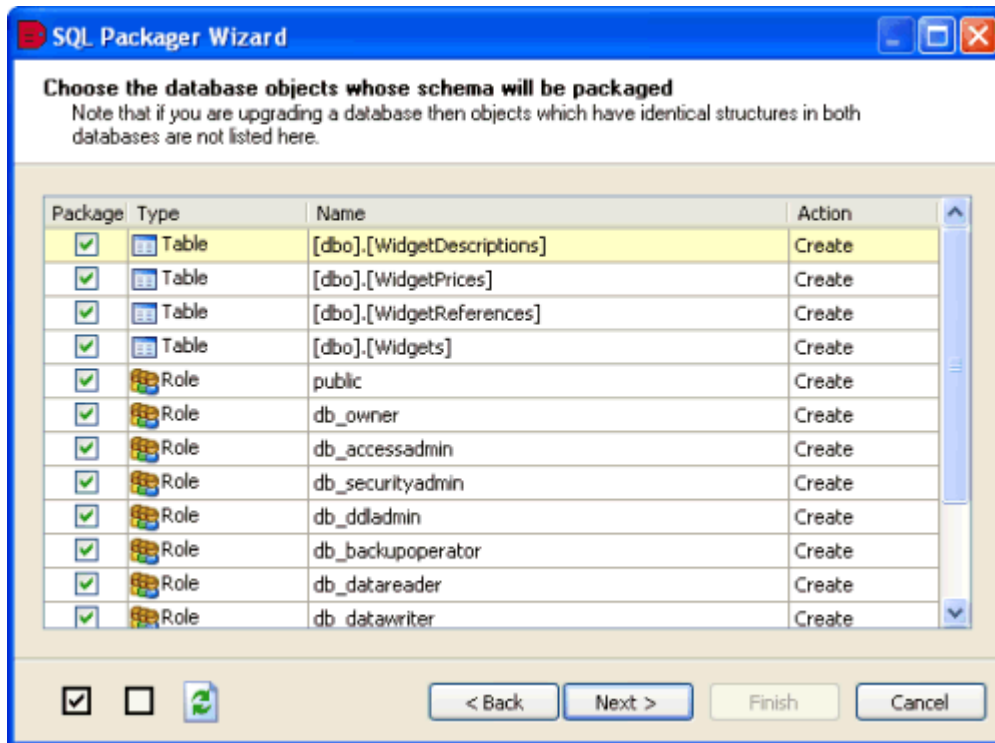
6. Click **Next**.

SQL Packager displays a message dialog box while it analyzes the database structure.

If you select the **Close message box on completion** check box, SQL Packager closes this message dialog box automatically the next time that you choose the database. For this example, leave the setting as it is.

7. Click **OK** to close the message box.

SQL Packager displays a list of the objects in the database.

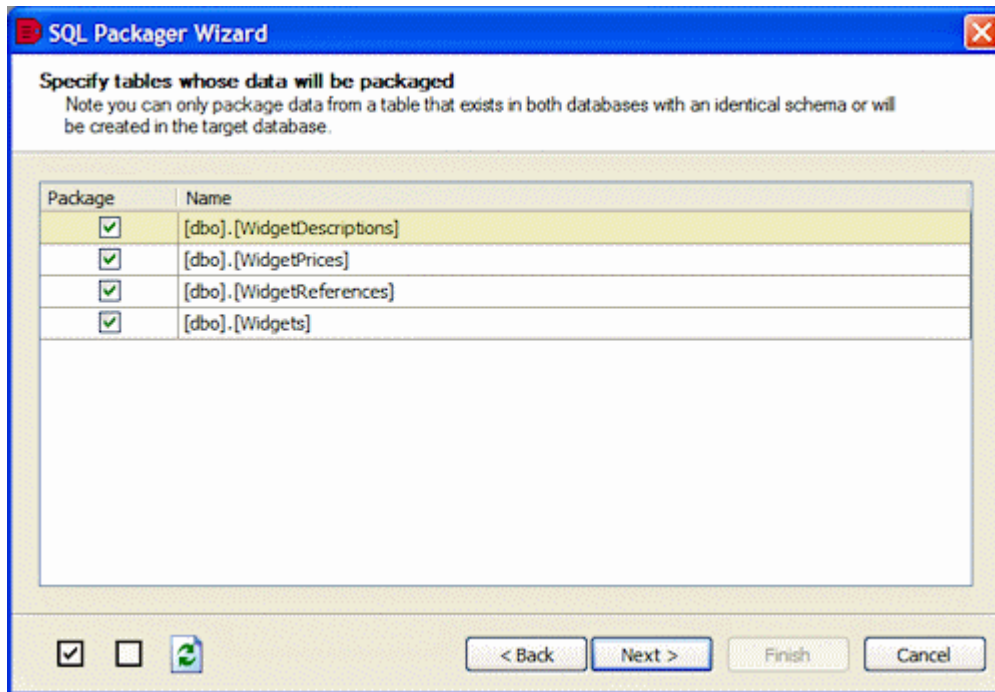


You can choose which objects to package. For more information, see Specifying the package contents (page 21).

For this example, leave all the check boxes selected so that all of the objects are created in the database when the package is run.

8. Click **Next**.

SQL Packager displays a list of the tables that contain data that you can package.



You can choose the tables for which you want to package data. For this example, leave all the check boxes selected so that you package all of the data.

9. Click **Next**.

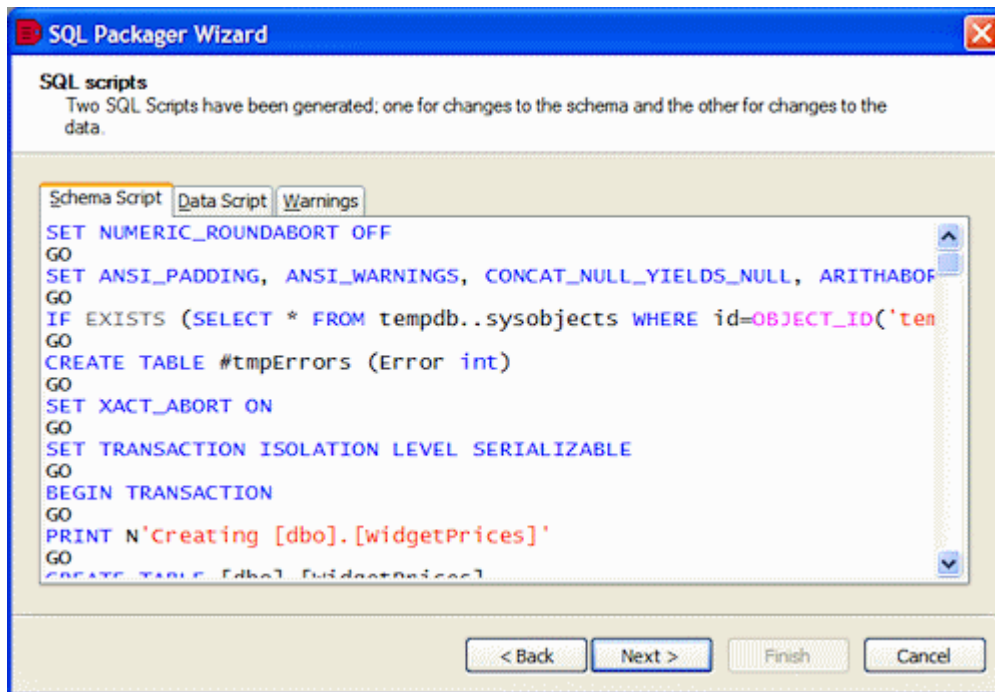
SQL Packager displays a message dialog box while it generates the SQL script.

If you select the **Close message box on completion** check box, SQL Packager closes this message dialog box automatically the next time that it generates the SQL script. For this example, leave the setting as it is.

10. Click **OK** to close the message box.

## Previewing the SQL Scripts

The **SQL scripts** page is displayed.



To see the SQL script for creating the data, click the **Data Script** tab. To see details about unexpected behavior that may occur when you run the package, click the **Warnings** tab.

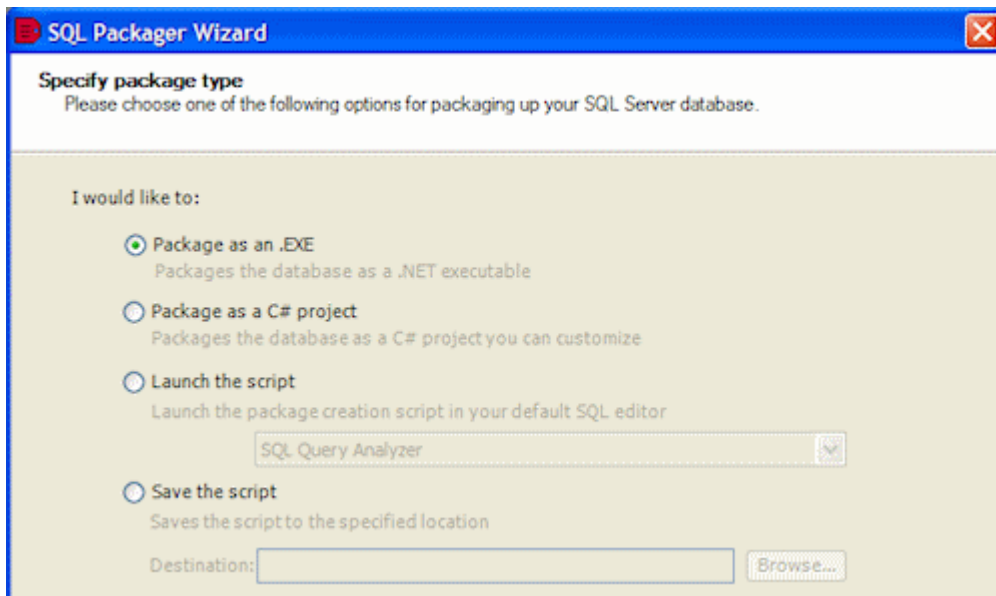
If required, you can save the scripts on the next page of the wizard.

When you have finished reviewing the SQL scripts, click **Next**.



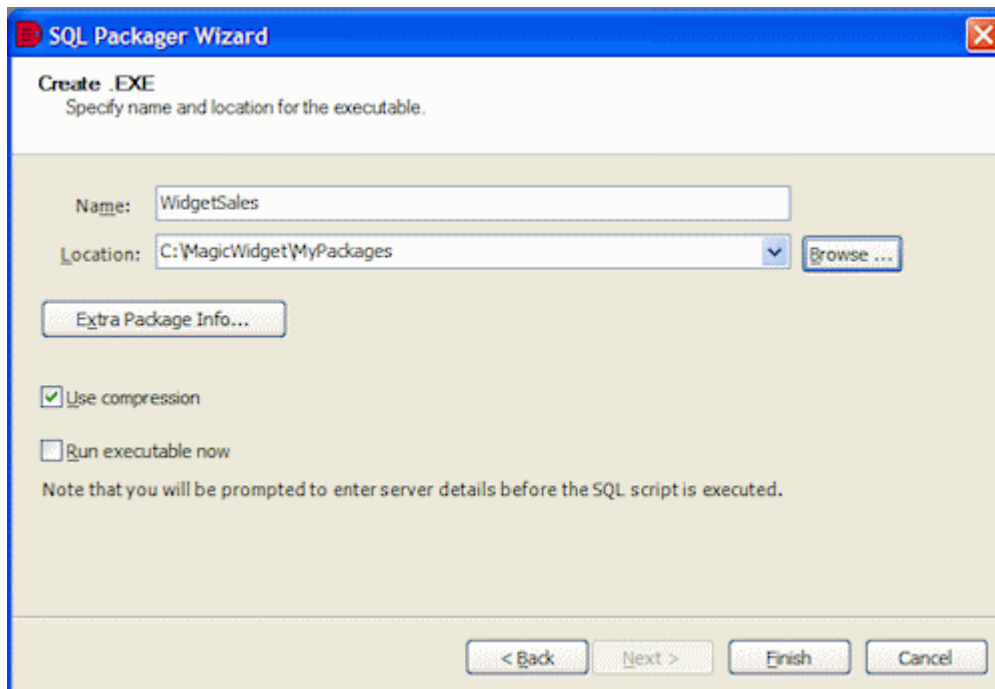
## Generating the package

The **Specify package type** page is displayed.



In this example, we will create an .EXE. For an example of how to create a C# project, see Packaging an upgrade as a C# project (page 11).

1. Ensure that **Package as an .EXE** is selected, and click **Next**.  
The **Create .EXE** page is displayed.

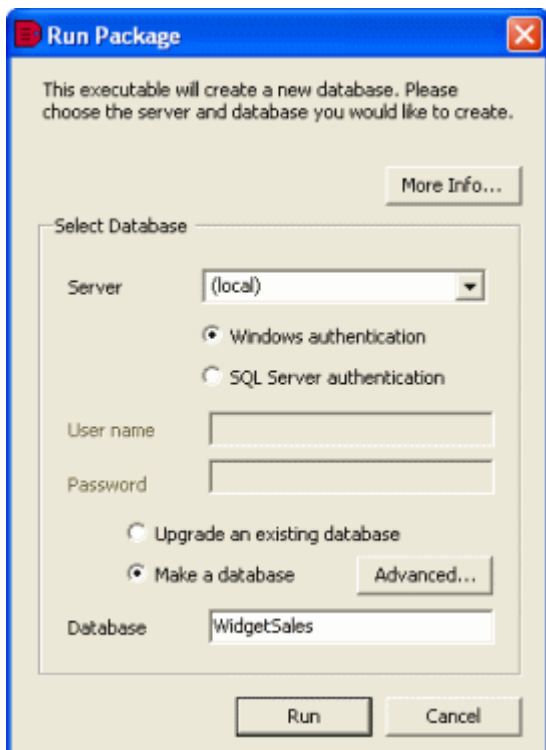


2. Enter a name and location for your package.

3. Leave the **Use compression** check box selected so that your package will be compressed.  
The generated files will be compressed to approximately 75% of their original size.
4. Select the **Run executable now** check box, and click **Finish**.  
A message dialog box informs you that the executable is created in the location you specified. Click **OK** to close it. For large databases, additional dynamic-link library (.dll) files are also created. The **Run Package** dialog box is displayed for you to run the executable immediately.

## Running the package

You use the **Run Package** dialog box to specify details of the database that will be created when the package is run.



Select the **Server** on which you want to create the database, and if required, enter the authentication details. Note that the SQL Server version must be compatible with the latest version database that you specified when you chose the database to package.

The **Advanced** options enable you to define properties for the database that will be created, such as the database location.

Type a name for the **Database** and click **Run**. A message dialog box is displayed for you to confirm that you want to continue. Click **Yes**.

When the database is created, a message dialog box confirms that the package has run successfully. Click **OK** to close it.

You can use your SQL application to check that the database has been created as you expect. If you have purchased Red Gate **SQL Compare** ([http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Compare&c=SQL\\_Compare/help/](http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Compare&c=SQL_Compare/help/)

8.0/SC\_Getting\_Started.htm&toc=SQL\_Compare/help/8.0/toc.htm) you can compare the databases' structure to confirm that they are identical; if you have purchased **SQL Data Compare** ([http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Data%20Compare&c=SQL\\_Data\\_Compare/help/7.0/SDC\\_GettingStarted.htm&toc=SQL\\_Data\\_Compare/help/7.0/toc.htm](http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Data%20Compare&c=SQL_Data_Compare/help/7.0/SDC_GettingStarted.htm&toc=SQL_Data_Compare/help/7.0/toc.htm)), you can compare the data to confirm it is identical.

## Worked example: Packaging an upgrade as a C# project

---

This worked example demonstrates how to package an upgrade to a database as a C# project, and run the package.

In the example, the sales department of the Magic Widget Company has a SQL Server database of their products. The development department have made a number of changes to the structure and content of the product database, which now need to be packaged for deployment to the sales department as an upgrade.

You will see how to:


1. Set up the databases if you want to follow the example on your own system.  
You will need access to a SQL Server to do this.
2. Specify the contents of the package.
3. Preview the SQL scripts.
4. Generate the package as a C# project.
5. Run the package to upgrade the existing database.  
You will need Microsoft® Visual Studio® .NET 2005 or later to compile the project.

### Setting up the databases

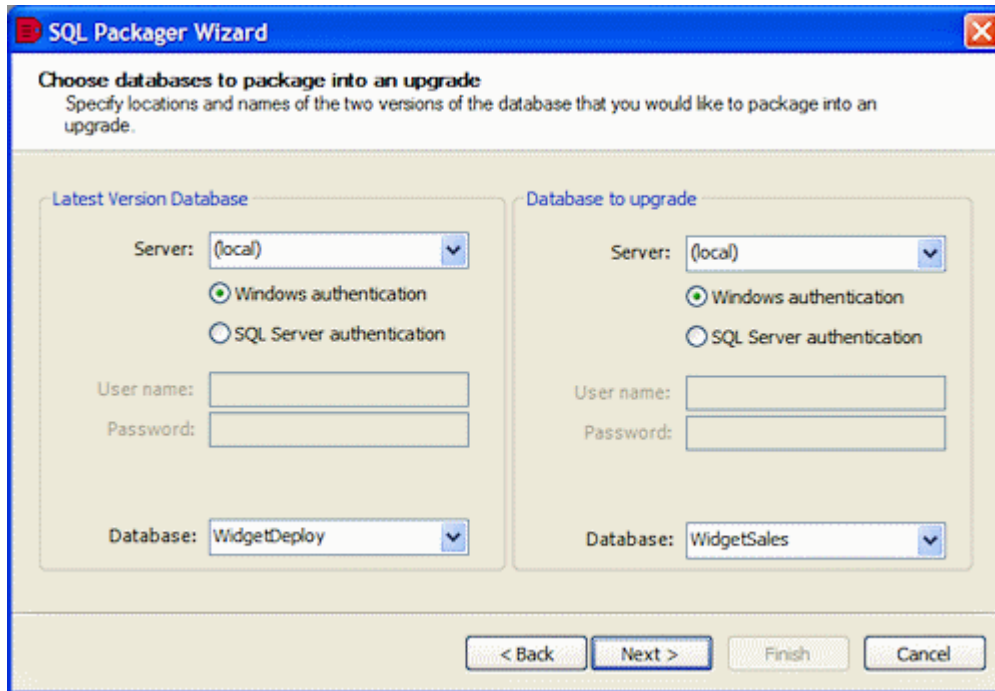
The worked example upgrades the product database, **WidgetSales**, with a later version of the database called **WidgetDeploy**. To create these databases on your SQL Server:

1. If they already exist, delete the databases **WidgetSales** and **WidgetDeploy** from your SQL Server.
2. Click here ([/support/SQL\\_Packager/help/5.5/SP\\_WEUpgradeC.sql](/support/SQL_Packager/help/5.5/SP_WEUpgradeC.sql)) to view the SQL creation script for the databases.
3. Copy the script, paste it in your SQL application, and then run it.  
The databases are created and populated with data.

### Specifying the package contents

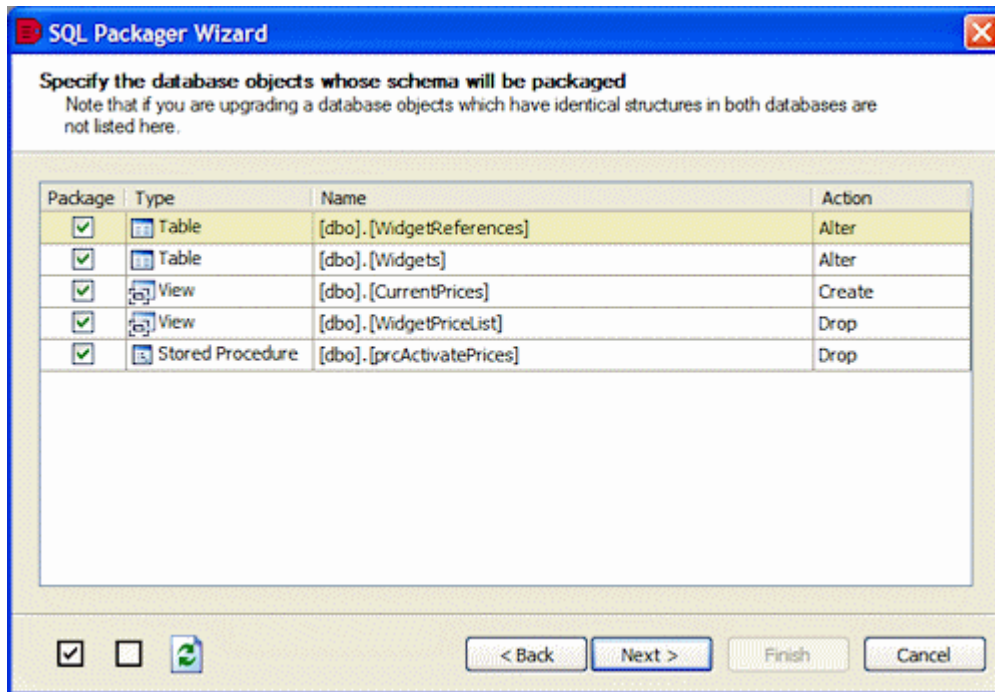
1. If you have not yet started SQL Packager, select it from your **Start** menu; if SQL Packager is already running, click  **New Project**.
2. On the **Choose a project type** page of the Packager Wizard, select **Package an upgrade to a database**, and click **Next**.  
The **Choose databases to package into an upgrade** page of the Packager Wizard is displayed.
3. In the **Server** box, under **Latest Version Database**, type or select the name of the SQL Server on which you created the databases.
4. Select the authentication method, and for **SQL Server authentication** enter the **User name** and **Password**.
5. In the **Database** box, type or select *WidgetDeploy*.  
If *WidgetDeploy* is not displayed in the **Database** list, right-click in the **Database** box and click **Refresh** or scroll to the top of the list and click **Refresh**.

6. Under **Database to upgrade**, in the **Server** box, type or select the name of the SQL Server.
7. Select the authentication method, and for **SQL Server authentication** enter the **User name** and **Password**.
8. In the **Database** box, type or select *WidgetSales*.  
If *WidgetSales* is not displayed in the **Database** list, right-click in the **Database** box and click **Refresh** or scroll to the top of the list and click **Refresh**.



9. Click **Next**.  
SQL Packager displays a message dialog box while it analyzes the database structure.  
If you select the **Close message box on completion** check box, SQL Packager closes this message dialog box automatically the next time that you choose the databases. For this example, leave the setting as it is.
10. Click **OK** to close the message box.

SQL Packager displays a list of objects whose structure differs in the databases.



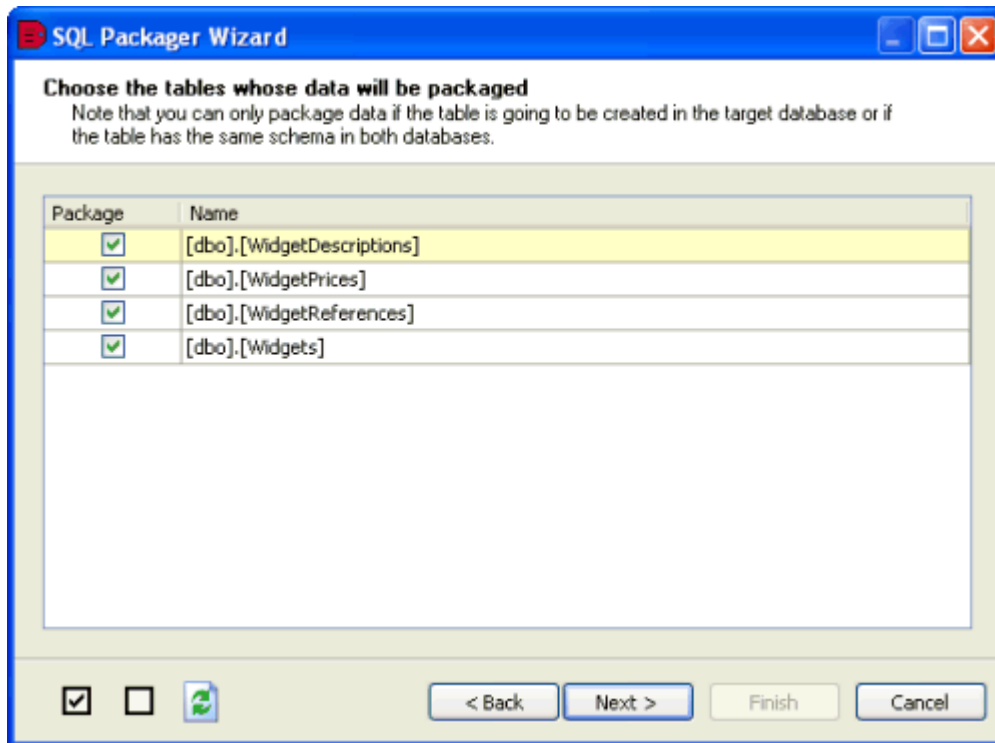
You can choose which objects to package. For more information, see Specifying the package contents (page 21).

For this example, leave all the check boxes selected so that all of the objects are updated when the package is run.

The **Action** column indicates the action that will be taken on WidgetSales to make it identical to WidgetDeploy. Note that the upgrade not only creates new objects in WidgetSales; it also alters and drops objects from WidgetSales to make it identical to WidgetDeploy. For example, the WidgetPriceList view will be dropped from WidgetSales.

11. Click **Next**.

SQL Packager displays a list of the tables that contain data that can be packaged.



You can choose the tables for which you want to package data. For more information, see Specifying the package contents (page 21).

For this example, leave all the check boxes selected so that all the data in the tables will be updated when the package is run.

12. Click **Next**.

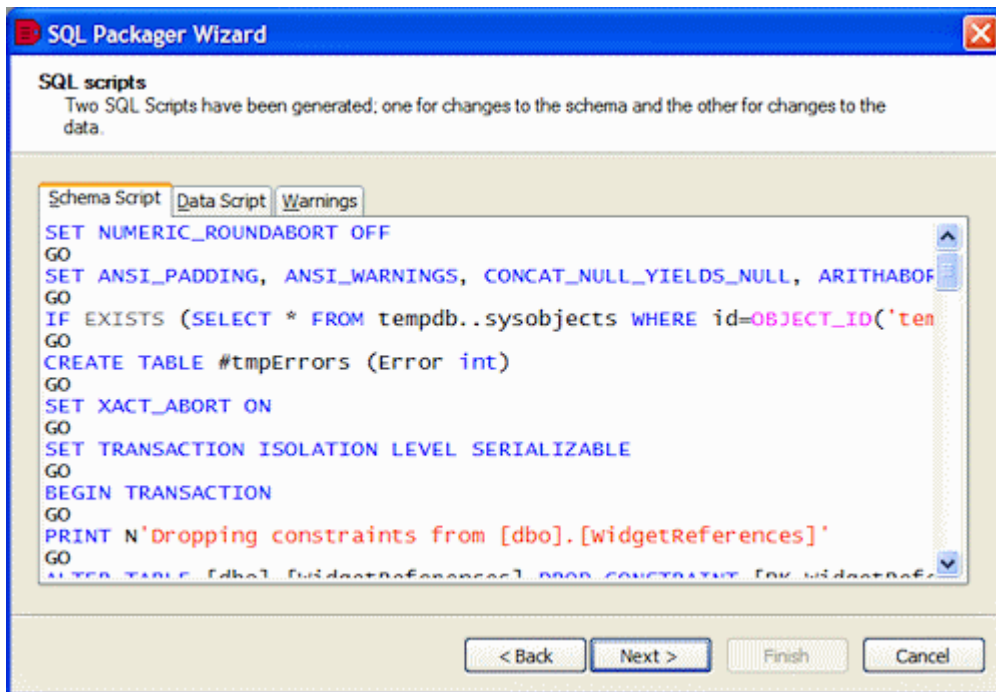
SQL Packager displays a message dialog box while it generates the SQL script.

If you select the **Close message box on completion** check box, SQL Packager closes this message dialog box automatically the next time that it generates the SQL script. For this example, leave the setting as it is.

13. Click **OK** to close the message box.

## Previewing the SQL Scripts

The **SQL scripts** page is displayed.



The **SQL scripts** page displays the following tabs:

- **Schema Script** displays the SQL code to update the structure in WidgetSales so that it is identical to WidgetDeploy
- **Data Script** displays the SQL code to update the data in WidgetSales so that it is identical to WidgetDeploy
- **Warnings** provides details about unexpected behavior that may occur when you run the package

In this example, SQL Packager displays a warning to inform you that it cannot use the ALTER TABLE command to change the IDENTITY column, so the package will rebuild the WidgetReferences table. Warnings are displayed whenever tables require rebuilding as these may be slow operations.

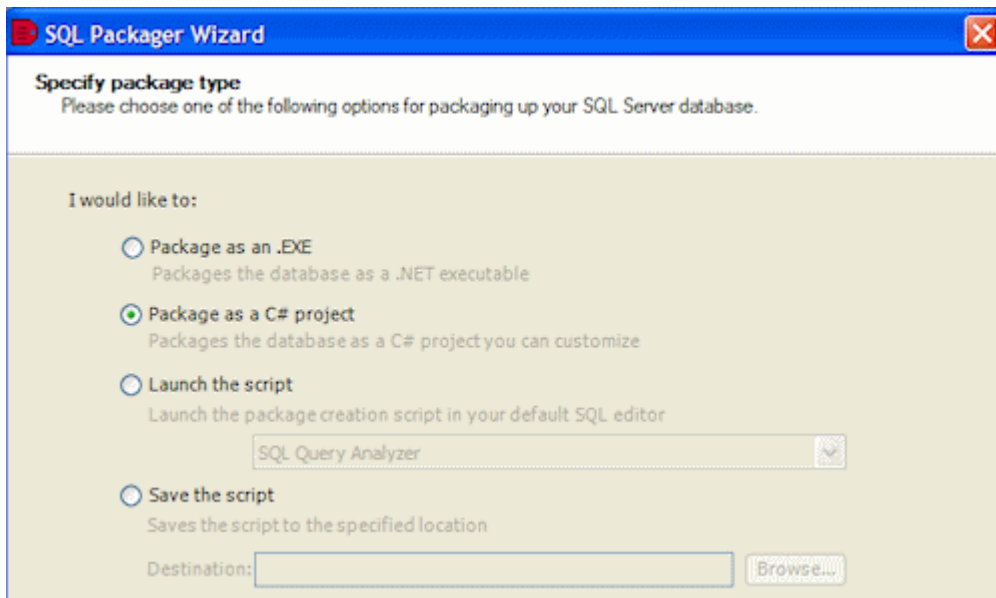
If required, you can save the scripts from the next page of the Packager Wizard.

When you have finished reviewing the scripts, click **Next**.



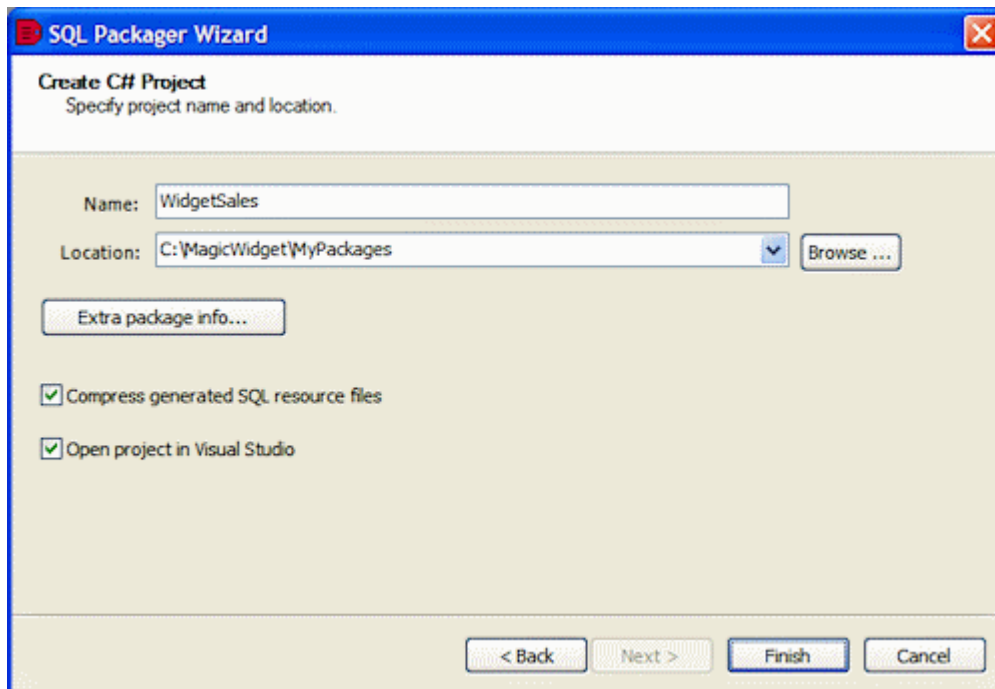
## Generating the package

The **Specify package type** page is displayed.



In this example, we will create a C# project. For an example of how to generate a .NET executable, see Packaging a database as an .EXE (page 2)

1. Ensure that **Package as a C# project** is selected, and click **Next**.  
The **Create C# Project** page is displayed.



2. Enter a name and location for your package.

3. Leave the **Compress generated SQL resource files** check box selected so that your package will be compressed.

The generated files will be compressed to approximately 75% of their original size.

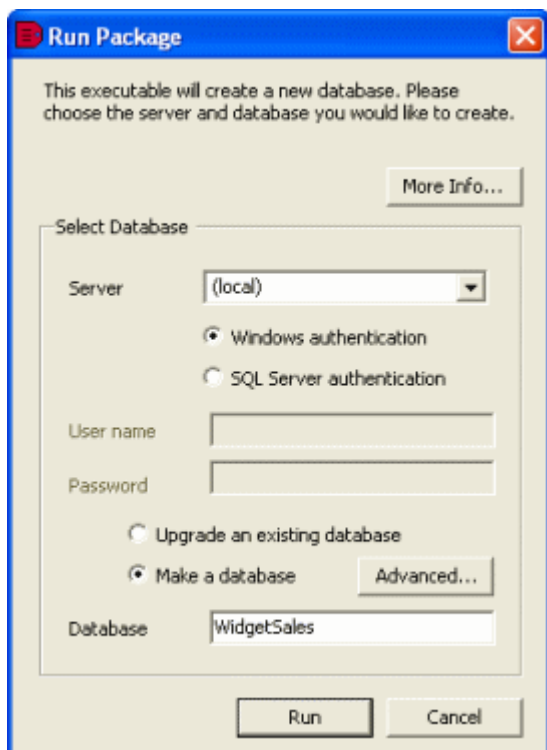
Note that compressing the resource files means that you cannot edit the resource files or add resource files to the C# project.

4. Ensure the **Open project in Visual Studio** check box is selected, and click **Finish**.

A message dialog box informs you that the project is created in the location you specified. Click **OK** to close it. Visual Studio .NET is launched with the project.

## Running the package

To compile and run the project, in Visual Studio .NET 2005 or later, press **F5**, or on the **Debug** menu, click **Start**. The **Run Package** dialog box is displayed.



You use this dialog box to specify details of the database that will be upgraded when the package is run.

Select the **Server** for *WidgetSales*, and if required, enter the authentication details.

Click **Run**. A message dialog box is displayed for you to confirm that you want to continue. Click **Yes**.

When *WidgetSales* is upgraded, a message dialog box confirms that the package has run successfully. Click **OK** to close it.

You can use your SQL application to check that the database has been changed as you expect. If you have purchased Red Gate **SQL Compare** ([http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Compare&c=SQL\\_Compare/help/8.0/SC\\_Getting\\_Started.htm&toc=SQL\\_Compare/help/8.0/toc.htm](http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Compare&c=SQL_Compare/help/8.0/SC_Getting_Started.htm&toc=SQL_Compare/help/8.0/toc.htm)) you can compare the databases' structure to confirm that they are identical; if you have

purchased **SQL Data Compare** ([http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Data%20Compare&c=SQL\\_Data\\_Compare/help/7.0/SDC\\_GettingStarted.htm&toc=SQL\\_Data\\_Compare/help/7.0/toc.htm](http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Data%20Compare&c=SQL_Data_Compare/help/7.0/SDC_GettingStarted.htm&toc=SQL_Data_Compare/help/7.0/toc.htm)), you can compare the data to confirm it is identical.

## Working with projects

---

Whenever you create a database package, you set up a *project*. A project contains:

- the information required to connect to the database you want to package
- information about which objects and data you want to include in the package
- the package details such as name, location, type, and compression
- optional deployment notes
- optional details about the database that will be created, such as the default name and size


You can create a new project each time you create a package. Alternatively, if you will be using the same settings repeatedly, you can save the current project to a file. You can then open it at a later date to make it the current project. You can also edit the current project if required.

You are recommended to run the project using the default packaging options (page 33) and review the results before you change any options. If you want to change your packaging options before you run the next project, if necessary click **Cancel** on the Packager Wizard, and set the options (page 33) before you proceed.

Note that packaging options are not saved as part of the project.

### Creating a new project

To create a new project:

1. If the Packager Wizard is not already displayed, click  **New Project**. The Packager Wizard opens.  
If you have unsaved changes in the current project, you are prompted to save it.
2. In the Packager Wizard, choose a project type.
3. If you are not packaging an existing SQL script, specify the package contents (page 21).
4. Review any generated SQL scripts (page 27) for the object and data creation.
5. Create an .EXE (.NET executable) (page 29) or a C# project (page 31).  
SQL Packager packages the database. Alternatively, you can launch the package creation script in your default SQL editor, or save the script.  
You can then save the project if required.

### Saving the current project

When you have packaged a database, you can save a new project so that you can re-use the settings at a later date, or you can save an existing project to a new name. Projects are saved with the extension .sqlpkg


- On the **File** menu, click **Save Project As**.  
A standard Windows® **Save As** dialog box is displayed.

If you have edited the current project, an asterisk (\*) is shown in the title bar; to save the edited project to the same name:


- On the **File** menu, click **Save Project**; alternatively, click 

### Editing the current project

You can edit the current project and package the database again.

1. Click  **Edit Project**.
2. In the Packager Wizard, change the package specification (page 21) as required.
3. Review the generated SQL scripts (page 27) for the object and data creation.
4. Generate an .EXE (page 29) or a C# project (page 31).  
SQL Packager packages the database. An asterisk (\*) is displayed in the title bar to indicate that the project has been edited.

### Opening an existing project

1. If the Packager Wizard is displayed, click **Cancel** to close it.
2. Click  **Open Project**.  
If you have unsaved changes in the current project, you are prompted to save it.
3. In the **Open** dialog box, choose the project and click **Open**.  
The Packager Wizard is displayed with the project's settings.

To open a recently-used project, on the **File** menu, click the name of the project.

## Specifying the package contents

---

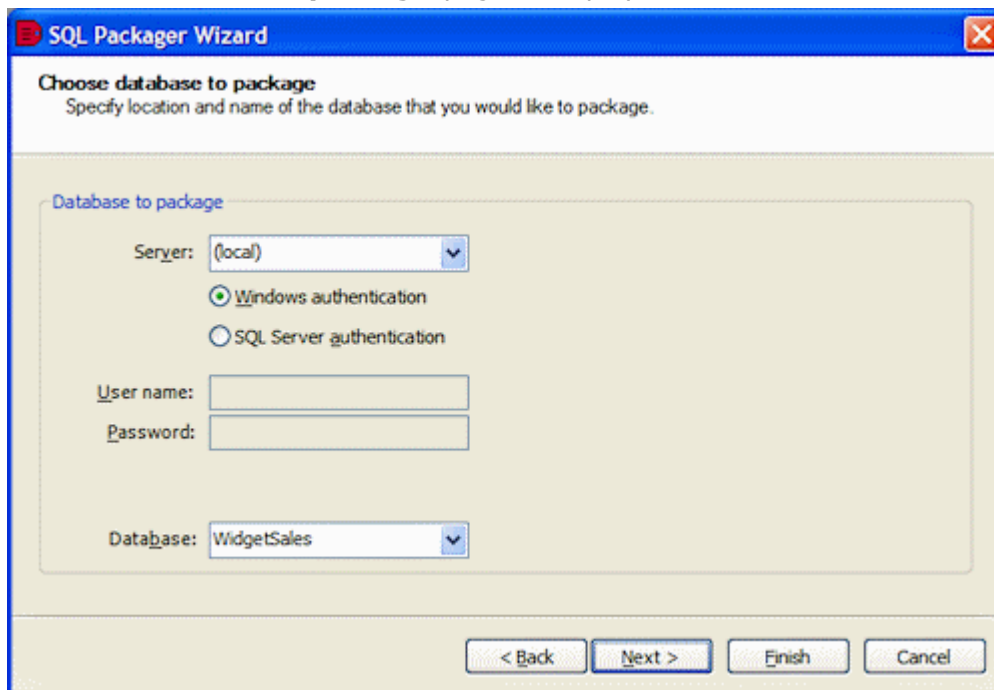
Choosing a project type > **Specifying contents** > Previewing scripts (page 27) > Specifying a package type (page 60) > Running (page 34)

Whenever you package a database, SQL Packager requires information to connect to the database you want to package, and information about the database objects and data you want to include in that package. You enter this information using the Packager Wizard.

The information you enter in the Packager Wizard is saved in the current project. For more information about projects, see Working with projects (page 19).

### Choosing the databases

1. If you have chosen to **Package a database** on the Choose a project type page, the **Choose database to package** page is displayed:



The screenshot shows the 'SQL Packager Wizard' dialog box with the 'Choose database to package' step. The title bar reads 'SQL Packager Wizard'. The main heading is 'Choose database to package' with the instruction 'Specify location and name of the database that you would like to package.' Below this, there is a section titled 'Database to package' containing the following fields and options:

- Server:** A dropdown menu currently showing '(local)'.
- Authentication:** Two radio buttons: 'Windows authentication' (which is selected) and 'SQL Server authentication'.
- User name:** An empty text input field.
- Password:** An empty text input field.
- Database:** A dropdown menu currently showing 'WidgetSales'.

At the bottom of the dialog box, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Alternatively, if you have chosen **Package an upgrade to a database**, the **Choose databases to package into an upgrade** page is displayed:

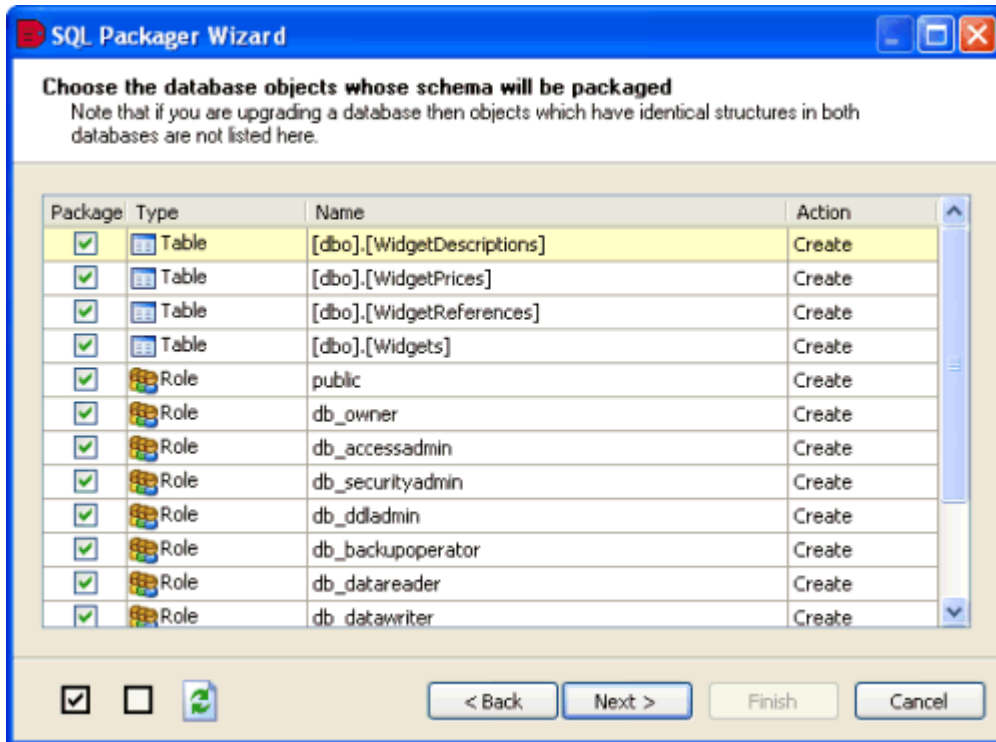
The screenshot shows the 'SQL Packager Wizard' dialog box with the title 'Choose databases to package into an upgrade'. The instruction reads: 'Specify locations and names of the two versions of the database that you would like to package into an upgrade.' The dialog is divided into two main sections: 'Latest Version Database' and 'Database to upgrade'. Each section contains a 'Server' dropdown menu (both set to '(local)'), two radio buttons for authentication (both 'Windows authentication' selected), 'User name' and 'Password' text boxes, and a 'Database' dropdown menu (set to 'WidgetDeploy' and 'WidgetSales' respectively). At the bottom, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

2. In the relevant **Server** box, type or select the name of the SQL Server.  
If you experience problems selecting a SQL Server that is not running on the LAN, for example if you are accessing the SQL Server via an internet connection, you may need to create an alias to the SQL Server using TCP/IP (refer to your SQL Server documentation for details). You can then type the alias name in the **Server** box to connect to the remote SQL Server.  
To refresh the **Server** list, right-click the box and click **Refresh**, or scroll to the top of the list and click **Refresh**.
3. Select the authentication method, and for **SQL Server authentication** enter the **User name** and **Password**.
4. In the **Database** box, type or select the database that you want to package.  
To refresh the **Database** list, right-click in the box and click **Refresh**, or scroll to the top of the list and click **Refresh**.
5. If you are packaging an upgrade, in **Database to upgrade** enter the SQL Server and database details.
6. Click **Next**.  
SQL Packager displays a message dialog box while it analyzes the database structure.  
If you select the **Close message box on completion** check box, SQL Packager closes this message dialog box automatically the next time that you choose the databases.
7. If necessary, click **OK** to close the message box.

### Choosing the database objects to package

If you are creating a package for a new database, SQL Packager lists the objects in the database that you can select for packaging.

If you are creating a package for upgrading an existing database, SQL Packager compares the previous version database with the latest version database to identify the changes to be made to the database structure to make the databases identical. SQL Packager lists the objects whose structure differs.



You select the objects to package by selecting or clearing the appropriate check boxes in the **Package** column. By default, the first time you run a project all objects are selected for packaging. To select all objects, click  ; to clear all of the check boxes, click



You may see the following types of object:



- Tables
- Views
- Stored Procedures
- Users
- Roles
- Rules
- Defaults
- User Defined Types
- Functions
- Full Text Catalogs

For SQL Server 2008 or SQL Server 2005, the following object types may also be shown:

- Assemblies
- Asymmetric Keys
- Certificates
- Contracts
- DDL Triggers
- Event Notifications
- Message Types
- Queues
- Routes
- Schemas
- Services
- Service Bindings
- Symmetric Keys
- Synonyms



 Partition Functions  
 Partition Schemes

 XML Schema Collections  
 Full Text Stoplist  
(SQL Server 2008 only)

Note that SQL Server 2008 and SQL Server 2005 severely restrict access to certificates, symmetric keys, and asymmetric keys. Consequently, SQL Packager can package only the permissions of certificates and asymmetric keys; symmetric keys cannot be packaged. To ignore all certificates, symmetric keys, and asymmetric keys, select the Ignore certificates, symmetric and asymmetric keys schema packaging option.

If you are upgrading a database, note that:

- Objects that are the same but have different owners are treated as different objects. For example, if a stored procedure exists in both databases and is identical except for its owner, it is considered to be a completely different object.
- For SQL Server 2000, differences in database-level permissions are not detected by SQL Packager. For example, if you have used SQL Server Enterprise Manager to set up permissions for your database, such as GRANT CONNECT or GRANT BACKUP, those permissions are not considered; however, permissions on objects are detected. If you want to include database-level permissions in your database package, you are recommended to use roles.


For SQL Server 2008 and SQL Server 2005 differences in database-level permissions are detected.

- If the database contains an encrypted user-defined function, stored procedure, trigger, or view and you have system administrator permissions, SQL Packager decrypts the object and you can view its internal SQL in the schema packaging script (page 27). If you do not have system administrator privileges, the encrypted object cannot be displayed or upgraded.

SQL Packager cannot decrypt views, stored procedures, functions, DML triggers, and DDL triggers that are encrypted in a SQL Server 2008 or SQL Server 2005 database. Therefore, SQL Packager cannot compare the objects; if an encrypted object exists in both databases, SQL Packager assumes that they are different, but will not be able to upgrade them.

- Stored procedures that are for replication are not compared or displayed.
- SQL Packager does not compare extended stored procedures.

For each object, the **Action** column indicates the action that will be taken on the object. If you are packaging a database to create a new one, the action will be **Create** for all objects. However, for an upgrade, note that the package does not only add new objects to the upgraded database; it may also **Alter** or **Drop** objects to make the databases identical.

If you are editing an existing project (page 19) and the structure of the database has changed since you previously packaged it, you may need to click  to update the page with the new structure.

When you have selected the objects that you want to package, click **Next**.

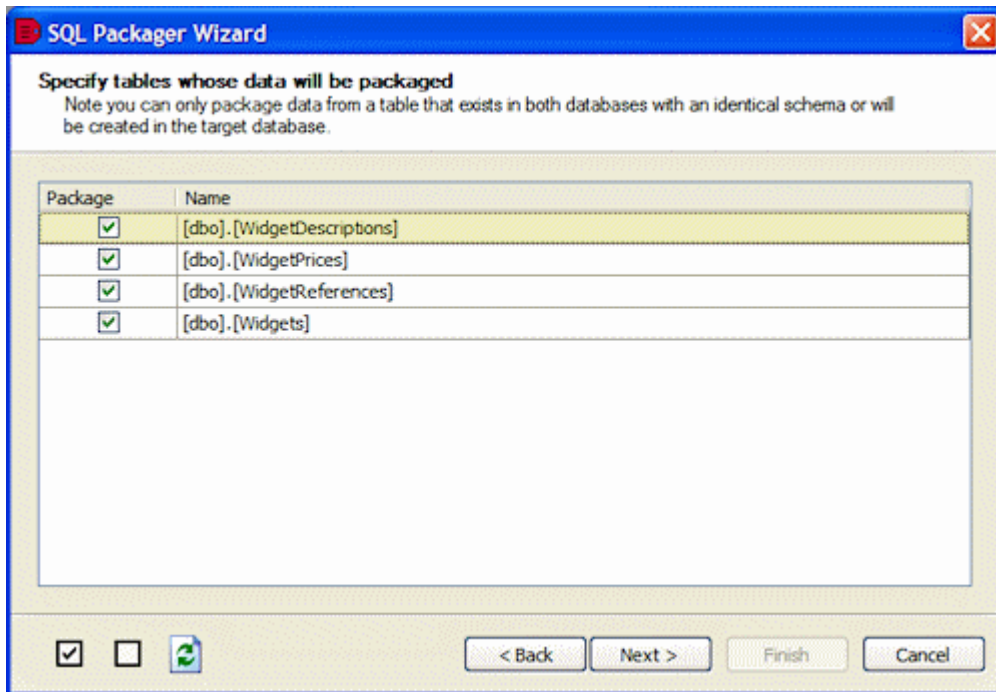
## Choosing the data to package

SQL Packager lists the tables in the database for you to select for data packaging.

If you are creating a package for upgrading an existing database, you can select tables for data packaging only if they have:


- similar names  
You can set SQL Packager so that it ignores the case of object names, and spaces or underscores in object names by using the data packaging options.
- the same owner (case-sensitive)
- similar structures
- a primary key, unique index, or unique constraint that matches in both databases  
SQL Packager uses the key, index, or constraint to determine which records correspond with each other. If more than one matching primary key, unique index, or unique constraint exists for a table, SQL Packager selects the most applicable (for example, a primary key is used in preference to a unique index).

You can also select a table for data packaging if it does not exist in the database you are upgrading, and you have chosen to package the table's structure in the previous page of the wizard.



You select the tables for data packaging by selecting or clearing the appropriate check boxes in the **Package** column. If you are creating a package for a new database, the package will insert data into the tables you have selected. However, for an upgrade, the package not only inserts data into tables, it may also update or delete rows to make the databases identical.

By default, the first time you run a project, all available tables are selected. To select all objects, click  ; to clear all of the check boxes, click

If you are editing an existing project (page 19) and the databases' data has changed since you previously packaged it, you may need to click  to update the page.

When you have selected the tables for data packaging, click **Next**. SQL Packager displays a message dialog box while it generates the SQL script.

If you select the **Close message box on completion** check box, SQL Packager closes this message dialog box automatically the next time that it generates the SQL script.

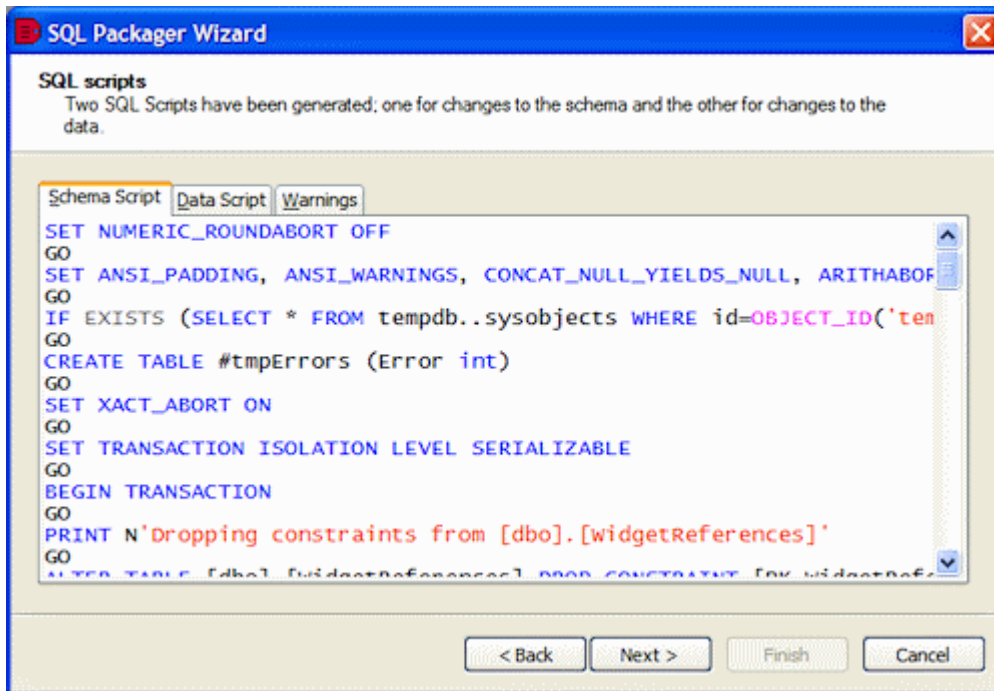
If necessary, click **OK** to close the message box. The **SQL scripts** page of the Packager Wizard is displayed; for details, see [Previewing the SQL scripts](#) (page 27).

## Previewing the SQL scripts

---

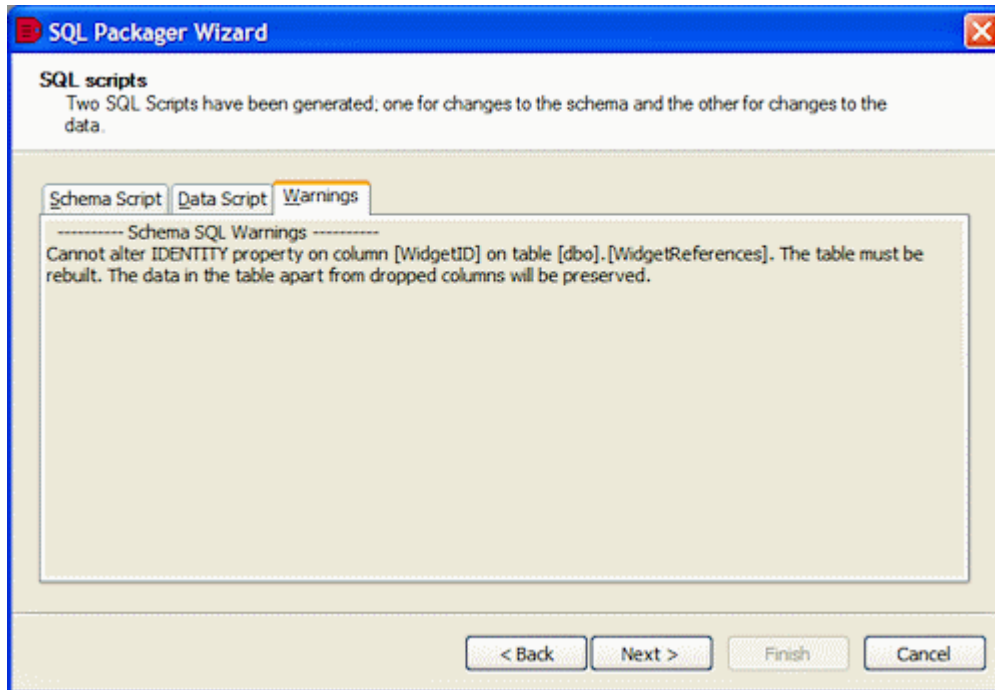
Choosing a project type > Specifying contents (page 21) > **Previewing scripts** > Specifying a package type (page 60) > Running (page 34)

The Packager Wizard displays the SQL scripts for creating or modifying the database structure and data.



The **SQL scripts** page displays the following tabs:

- **Schema Script** shows the SQL script to create or modify the database structure.
- **Data Script** shows the SQL script to create or modify the data.
- **Warnings** displays details about unexpected behavior that may occur when you run the package, for example:



You can:

- Search a SQL script:  
Click the **Schema Script** or **Data Script** tab, right-click the SQL code, and click **Find**; a standard Windows® **Find** dialog box is displayed.
- Copy the SQL scripts, summary details, or warning information for use in another application:  
Select the required text, right-click, and then click **Copy**.  
Alternatively, right-click the text, click **Select All**, then right-click, and click **Copy**.
- Save the scripts or launch them in a SQL editor on the next page of the Packager Wizard.

When you have reviewed the SQL scripts and any warnings, click **Next** to specify the package type, or choose to save or launch the scripts:

- To create an .EXE (.NET executable), see Generating an .EXE (page 29).
- To create a C# project, see Generating a C# project (page 31).  
If you will want to customize the package, you are recommended to create a C# project.

## Creating an .EXE

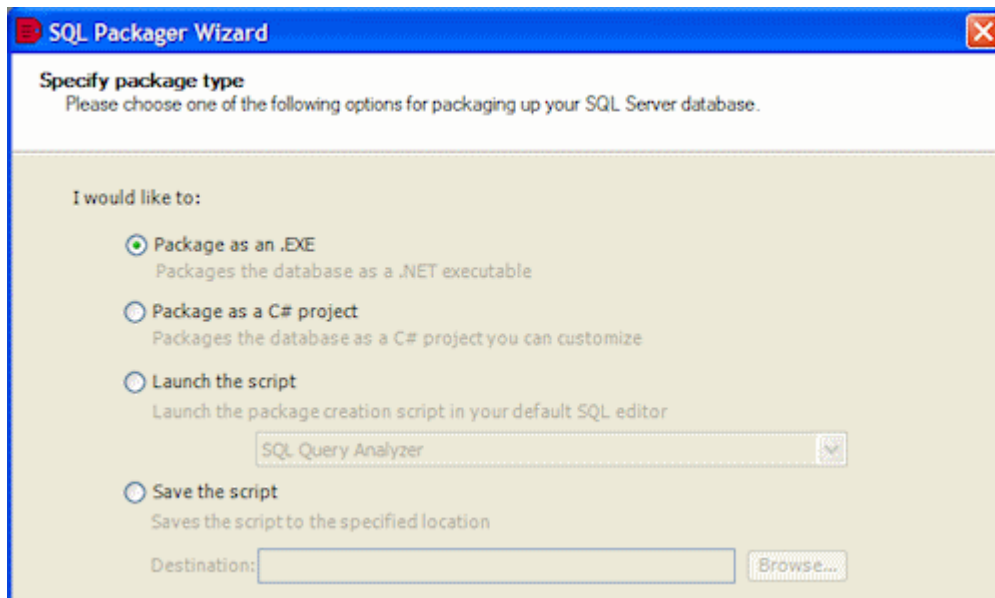
---

Choosing a project type > Specifying contents (page 21) > Previewing scripts (page 27) > **Specifying a package type** > Running (page 34)

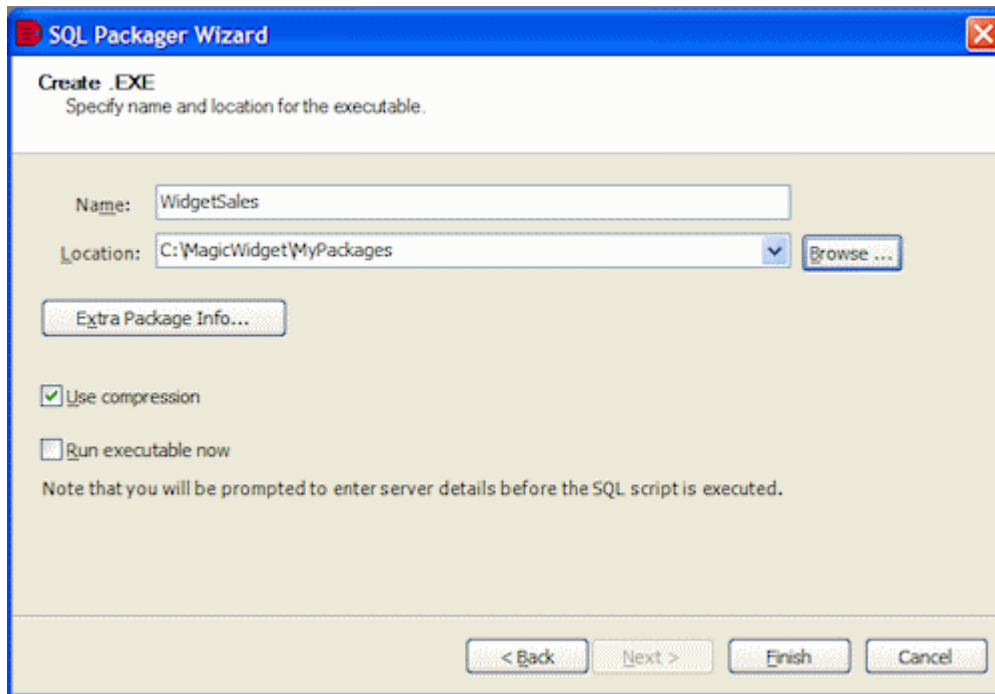
When you have chosen the database, objects, and data, that you want to package and previewed the SQL scripts, you can define the parameters for the .EXE.

To create an .EXE:

1. On the **Specify package type** page of the Packager Wizard, select **Package as an .EXE**.



2. Click **Next**.



3. On the **Create .EXE** page, in the **Name** box, type a name for the package. If you create a package with a file name that already exists, SQL Packager automatically assigns a different file name for the package. For example, if *SQLPackage* exists, the default name for the new package is *SQLPackage1*. You can turn off this feature by using the packaging options.
4. In the **Location** box, type or select the path for the package, or click **Browse** to choose the folder or create a new folder. You can change the default location by using the packaging options.
5. If you want to add information to be seen when the package is run, or if you are creating a package for a new database and you want to specify the database properties, click **Extra Package Info** (page 58) and enter the details as required.
6. To compress the generated files, select the **Use compression** check box. The package is usually compressed to approximately 75% of its original size.
7. To run the executable immediately, select the **Run executable now** check box; to create the executable without running it, clear the **Run executable now** check box.
8. Click **Finish**. A message dialog box is displayed to confirm that the executable has been created at the location you specified.
9. Click **OK** to close the message dialog box. If you chose to run it immediately, SQL Packager launches the **Run Package** dialog box. For more information, see *Running the package* (page 34).

Note that for large databases, additional dynamic-link library (.dll) files are also created.

## Creating a C# project

---

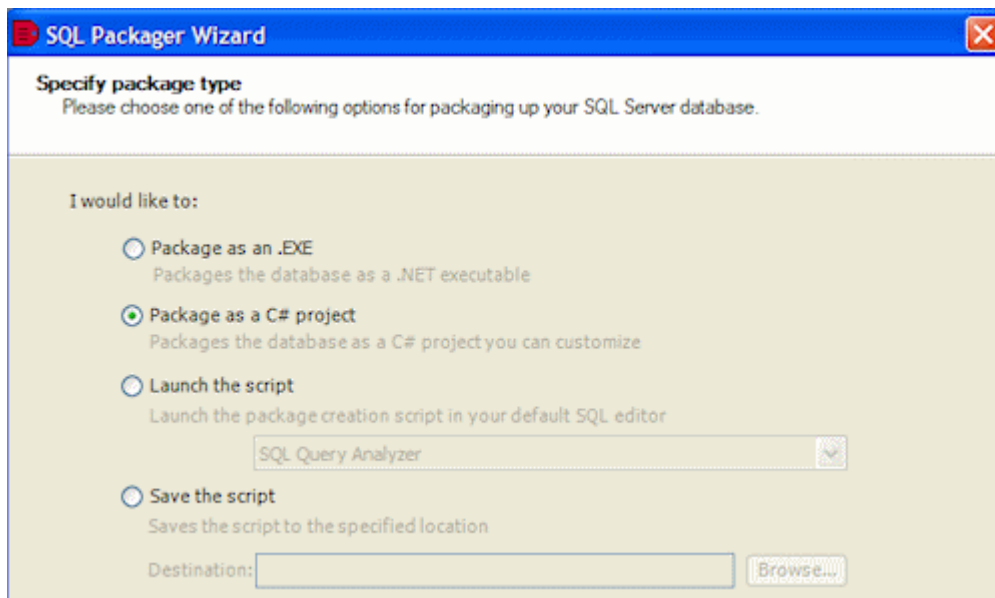
Choosing a project type > Specifying contents (page 21) > Previewing scripts (page 27) > **Specifying a package type** > Running (page 34)

When you have chosen the database, objects, and data that you want to package and previewed the SQL scripts, you can define the parameters for the C# project.

You should create a C# project if you want to customize the package. For example, you can edit the forms created in the project to customize the appearance of the graphical user interface that is displayed when you run the package (page 34).

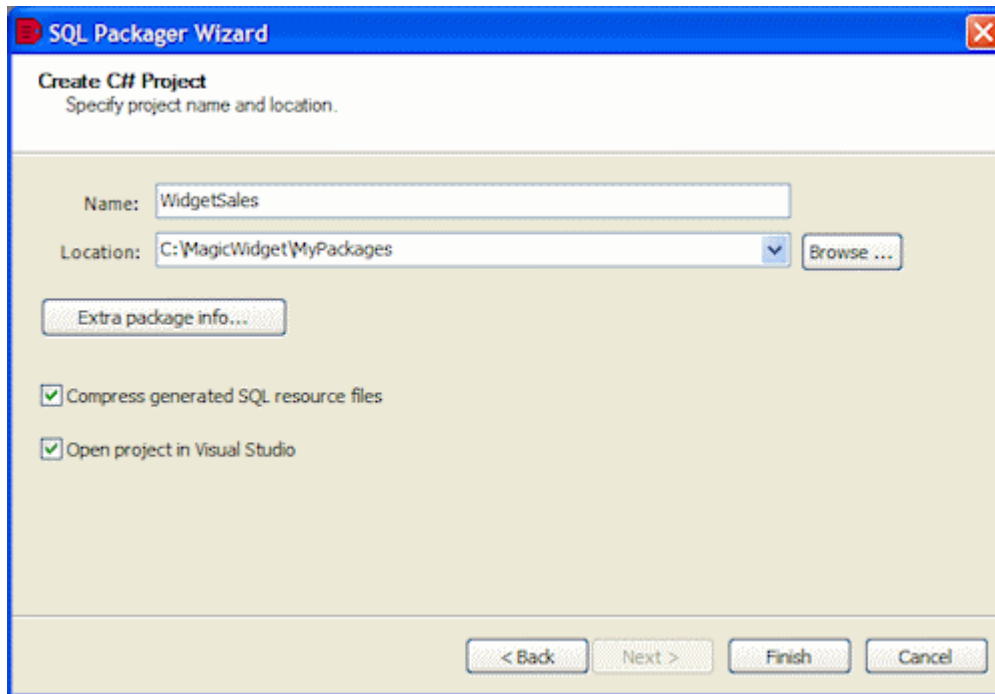
To create a C# project:

1. On the **Specify package type** page of the Packager Wizard, select **Package as a C# project**.





2. Click **Next**.



3. On the **Create C# Project** page, in the **Name** box, type a name for the project. If you create a package with a file name that already exists, SQL Packager automatically assigns a different file name for the package. For example, if *SQLPackage* exists, the default name for the new package is *SQLPackage1*. You can turn off this feature by using the packaging options.
4. In the **Location** box, type or select the path for the project, or click **Browse** to choose the folder or create a new folder. You can change the default location by using the packaging options.
5. If you want to add information to be seen when the package is run, or if you are creating a package for a new database and you want to specify the database properties, click **Extra package info** (page 58) and enter the details as required.
6. To compress the package, select **Compress generated SQL resource files**. The package is usually compressed to approximately 75% of its original size. Note that if you compress the package, you will not be able to add resource files to the project or edit the existing resource files.
7. To open the project in Microsoft® Visual Studio® immediately, select the **Open project in Visual Studio** check box; to create the project without opening it, clear the check box.
8. Click **Finish**. A message dialog box is displayed to confirm that the project has been created at the location you specified.
9. Click **OK** to close the message dialog box. If you chose to open it immediately, SQL Packager launches the project in Visual Studio. For more information, see *Running the package* (page 34).

## Setting packaging options

---

The packaging options are a set of advanced features that enable you to modify the behavior of SQL Packager. For example, you can set SQL Packager so that it ignores certain features, such as triggers.

To display the **SQL Packager Options** dialog box:

- On the **Packager** menu, click **Options**.

The options are divided into the following tabs:

- **Schema options** apply to the structure of the database
- **Data options** apply to the data for the selected objects
- **Packager options** apply to the .EXE or C# project

The packaging options are saved for each user. Therefore, if you change the options they will apply to all projects (page 19) run by the current user.

To reset all the options to their default values, click **Restore All Defaults**.

## Running the package

---

Choosing a project type > Specifying contents (page 21) > Previewing scripts (page 27) > Specifying a package type (page 60) > **Running**

When SQL Packager has generated the package, you can run it using a graphical user interface or from the command line (page 60).

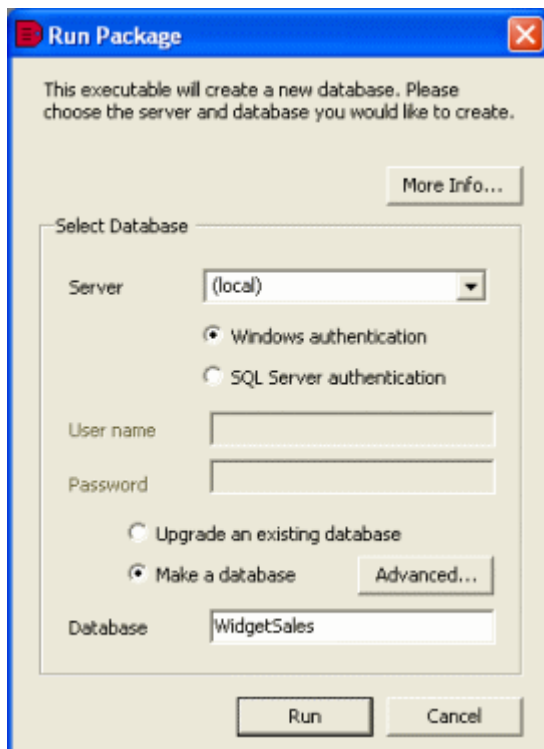
Note that the graphical user interface may differ from the interface described in this topic if it has been customized.

### Using the graphical user interface to create a database

To create a new database:

1. Display the **Run Package** dialog box:
  - ◆ for an .EXE package, run the executable in the usual way  
For example, double-click the .exe file in Microsoft® Windows® Explorer.
  - ◆ for a C# project, open the project in Visual Studio®, press **F5** to run it, or on the **Debug** menu, click **Start**  
Alternatively, compile the project and run the executable in the usual way.

The **Run Package** dialog box is displayed.



2. To view any notes that were added to the package when it was generated, or to see a summary of the SQL scripts and any warnings, click **More Info**.  
The **More Information** dialog box is displayed with the information. Click **OK** to close it.

3. In the **Server** box, type or select the name of the SQL Server on which you want to create the database.

If you experience problems selecting a SQL Server that is not running on the LAN, for example if you are accessing the SQL Server via an internet connection, you may need to create an alias to the SQL Server using TCP/IP (refer to your SQL Server documentation for details). You can then type the alias name in the **Server** box to connect to the remote SQL Server.

4. Select the authentication method, and for **SQL Server authentication** enter the **User name** and **Password**.

5. Do one of the following:

- ◆ To create a new database, click **Make a database** (the default) and type a name for the database in the **Database** box.

If you want to change the default database properties, click **Advanced** and enter the details as required. For more information about database properties, see Entering extra package information (page 58). Note that if you want your database to use filegroups or full-text processing you should create the database manually, and use the **Upgrade an existing database** option.

- ◆ To populate an empty database that you have already created, click **Upgrade an existing database** and in the **Database** box, type or select the name of the database.

6. Click **Run**.

A message dialog box is displayed for you to confirm that you want to continue. A second message dialog box confirms that the package has run successfully.

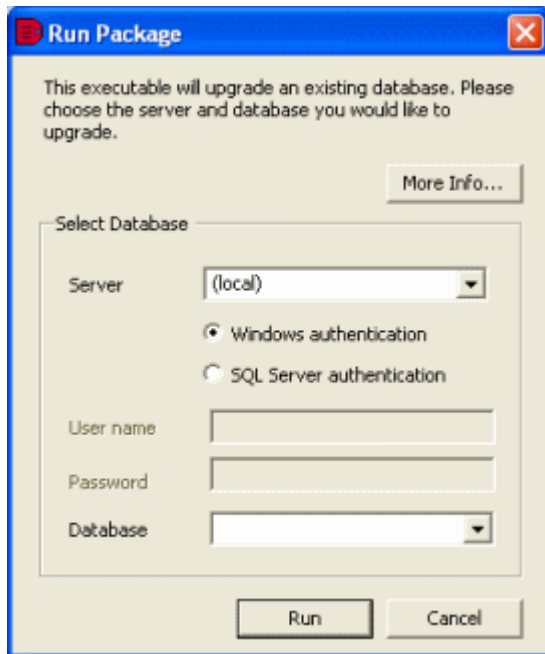
## Using the graphical user interface to upgrade a database

To upgrade an existing database:

1. Display the **Run Package** dialog box:

- ◆ for an .EXE package, run the executable in the usual way  
For example, double-click the .exe file in Windows Explorer.
- ◆ for a C# project, open the project in Visual Studio .NET, press **F5** to run it, or on the **Debug** menu, click **Start**  
Alternatively, compile the project and run the executable in the usual way.

The **Run Package** dialog box is displayed.



2. To view any notes that were added to the package when it was generated, or to see a summary of the SQL scripts and any warnings, click **More Info**.

The **More Information** dialog box is displayed with the information. Click **OK** to close it.

3. In the **Server** box, type or select the name of the SQL Server for the database you are upgrading.

If you experience problems selecting a SQL Server that is not running on the LAN, for example if you are accessing the SQL Server via an internet connection, you may need to create an alias to the SQL Server using TCP/IP (refer to your SQL Server documentation for details). You can then type the alias name in the **Server** box to connect to the remote SQL Server.

4. Select the authentication method, and for **SQL Server authentication** enter the **User name** and **Password**.
5. In the **Database** box, type or select the name of the database that you want to upgrade.
6. Click **Run**.

A message dialog box is displayed for you to confirm that you want to continue. A second message dialog box confirms that the package has run successfully.

## Using the command line

When you run the package from the command line, the following options are available:

```
/server:<server>
```

The name of the SQL Server. The default is (local).

```
/server:<server>\<instance>
```

The name of the SQL Server and instance. The default is (local).

```
/database:<database>
```

The name of the database that you want to create or upgrade.

```
/username:<username>
```

The user name for the database for SQL Server authentication.

```
/password:<password>
```

The password for the database for SQL Server authentication.

```
/quiet
```

Runs the package without displaying the graphical user interface.

```
/makedatabase
```

Creates the database on the specified SQL Server.

For example, to create a database called **MyDatabase** on SQL Server **MyServer** using SQL Server authentication, navigate to the folder that contains the package, and at the command prompt, type:

```
MyPackage.exe /server:MyServer /database:MyDatabase  
              /username:MyUserName /password:MyPassword
```

SQL Packager sets two return codes that you can use in batch files or scripts to determine whether the package ran successfully:

- **0** is returned if the package ran successfully
- **-1** is returned if an error occurred

If you have specified `/quiet` to run without the graphical user interface, the return code is stored in the `ERRORLEVEL` environment variable, and the error message is written to the console.

## Understanding the results

---

This topic provides information that may help you to understand the results when you use SQL Packager to create or upgrade a database. You may also wish to refer to Troubleshooting (page 44).

### Database diagrams

SQL Packager does not package or upgrade database diagrams.

### System tables

SQL Packager does not package or upgrade system tables.

### Encrypted database objects

If you are packaging a SQL Server 2000 database that contains an encrypted user-defined function, stored procedure, trigger, or view and you have system administrator permissions, SQL Packager decrypts the object and you can view its internal SQL in the schema packaging script (page 27). If you do not have system administrator privileges, you cannot package the encrypted object.

SQL Packager cannot decrypt views, stored procedures, functions, and DML triggers that are encrypted on a SQL Server 2008 or SQL Server 2005 database. Therefore, SQL Packager cannot display the SQL code for the encrypted objects, and cannot package them.

### Column order

If you are upgrading a database, column order is not forced unless you select the **Force table column order to be identical** schema packaging option.

For example, the latest version database has a table that contains *CoIA* and *CoIB*, in that order, and the previous version database has the same table but with *CoIB* then *CoIA*. If **Force table column order to be identical** is not selected, SQL Packager considers the tables to be identical. If the option is selected, SQL Packager considers the tables to be different and upgrades the table.

### Renamed columns

If you are upgrading a database, SQL Packager attempts to recognize renamed columns by the similarity of the names and the data types of the columns. When a renamed column is recognized as such, SQL Packager renames the column as appropriate.

However, if the names and data types are very different, SQL Packager may consider the renamed column to be a completely different column. In this case, if *CoIA* in the latest version database is renamed to *CoIB* in the previous version database, when SQL Packager creates the upgrade script, *CoIA* will be created in the previous version database as a new column and *CoIB* will be deleted. To avoid data loss, before you run the package you must take care to preserve any data in the two columns, and merge them following the upgrade.

## Updated views

Following an upgrade, if a view has not been updated by the package and it contains a **SELECT \*** statement, you must refresh it using **sp\_refreshview**, to reflect any changes that have been made to the underlying objects on which the view depends. Refer to your SQL Server documentation for more information.

Note that it is not best practice to use **SELECT \*** statements in views; you are recommended to specify an explicit column list.

## Replication

If objects that are used in replication are upgraded, errors may occur. For example, SQL Packager cannot drop a table if it is used for replication.

## Users

In Microsoft® Windows®, users are a composite of the domain name or computer name and the user name, for example *Computer1\WindowsUser1*. If you are upgrading a database, SQL Packager references only the user name, so that *Computer1\User1* and *Computer2\User1* would be considered as the same. Therefore, if you intend to upgrade users, ensure that their user names are different.

SQL Packager upgrades changes to users, such as changes to permissions. However, SQL Packager does not upgrade modifications to user passwords.

New users are created with the password: **p@ssw0rd**.

## Filegroups

SQL Packager supports the upgrade of databases that use multiple filegroups. However, you must ensure that the filegroups have been created on the target server prior to upgrade. If the filegroups do not exist, the upgrade will fail.

## CLR assemblies

When a CLR assembly is to be updated, if possible SQL Packager achieves this by using ALTER ASSEMBLY.

If SQL Packager determines that it would not be possible to use ALTER ASSEMBLY, the relevant table is rebuilt twice:

- in the first rebuild, the CLR type columns are converted to *nvarchar*  
The CLR type columns are dropped and recreated.
- in the second rebuild, the *nvarchar* data is converted to the final CLR type

Data is preserved. Note that the ToString representation of the CLR user-defined type must be the same for both the old and the new assembly, otherwise the upgrade may fail, or the data may be corrupted.

To force SQL Packager to use the double-table rebuild method, select the schema packaging option Do not use ALTER ASSEMBLY to change CLR types.



## Partition schemes and functions

In SQL Server 2008 and SQL Server 2005, partition schemes can be specified for tables so that the table is stored in several filegroups. By default, SQL Packager ignores filegroups. However, if you select the schema packaging option Consider next filegroups in partition schemes, SQL Packager upgrades the files. Note that for updates to partition schemes, a large amount of disk space may be required on the defined filegroups because partition ranges must be merged and split.

In certain cases, for example when a partition function changes from left range to right range, it is necessary to drop and recreate partition functions and partition schemes. In these cases, the table is rebuilt twice:

- in the first table rebuild, the content is saved to a temporary filegroup
- in the second table rebuild, the table is migrated from the temporary filegroup to a new partition scheme

Data is preserved. Note that if a CLR assembly upgrade also requires a table to be rebuilt twice, the CLR assembly and the partition scheme are upgraded at the same time.

## Certificates, symmetric keys, and asymmetric keys

SQL Server 2005 severely restricts access to certificates, symmetric keys, and asymmetric keys. Consequently, SQL Packager can package only the permissions of certificates and asymmetric keys; symmetric keys cannot be packaged. To ignore all certificates, symmetric keys, and asymmetric keys, select the Ignore certificates, symmetric and asymmetric keys schema packaging option.

## Extended properties on databases

Extended properties on databases are always packaged if they differ. If you do not want them to be packaged, select the Ignore extended properties schema packaging option.

## Upgrading databases on different SQL Server versions

---

This topic provides additional information for you if you are upgrading a database that is on a different version of Microsoft® SQL Server™ from the source database.

### SQL Server 2008, SQL Server 2005, and SQL Server 2000

You can upgrade to a SQL Server 2008 database from a SQL Server 2005 or SQL Server 2000 database. Upgrading from SQL Server 2005 requires no additional actions. Upgrading from SQL Server 2000 may require changes to the packaging options.

#### For upgrading the database structure:

If you are upgrading to a SQL Server 2008 or SQL Server 2005 database from a SQL Server 2000 database, you must not change the default schema packaging options (page 33).

However, if your database is on a SQL Server with case-sensitive sort order, you must select **Treat items as case sensitive** in the schema packaging options.

If you are moving changes to a SQL Server 2000 database from a SQL Server 2005 database, note the following:

- SQL Packager may be unable to upgrade all objects. Warnings (page 27) will be displayed where possible.
- SQL Packager cannot decrypt views, stored procedures, functions, DML triggers, and DDL triggers that are encrypted in a SQL Server 2008 or SQL Server 2005 database. Therefore, you cannot upgrade an object in a SQL Server 2000 database from an encrypted object in a SQL Server 2008 or SQL Server 2005 database.

When you create a default value or constraint in SQL Server 2008 or SQL Server 2005, the definitions of the default value or constraint are parsed and the parsed version is stored. The syntax of the SQL Server 2008 or SQL Server 2005 parsed version is not the same as the parsed version in SQL Server 2000. For example, in SQL Server 2005, **(1)** is parsed to **((1))**. If these are the only differences, SQL Packager considers the objects to be identical.

#### For upgrading the data:

- You can upgrade CLR data in a SQL Server 2008 or SQL Server 2005 database with values from a text or string data type in a SQL Server 2000 database. Ensure that the **Transport CLR data types as binary** data packaging option is not selected. SQL Packager considers the collation order for string data. Therefore, if the ordering is not the same as the CLR order, differences are reported.
- You can upgrade XML data in a SQL Server 2008 or SQL Server 2005 database with values from a text or string data type in a SQL Server 2000 database. SQL Packager will attempt to preserve white space. SQL Packager supports DTD (Document Type Definition), except for default attributes and entities.

Note that some data, such as XML encoding and DTD, cannot be stored in the SQL Server 2008 or SQL Server 2005 representation. Therefore, if you convert data from a string data type to an XML data type, and then you convert back to a string data type,

this information will be lost. SQL Packager considers the collation order for string data. Therefore, if the ordering is not the same as the XML order, differences are reported.

### **SQL Server 2008 and SQL Server 2005 compatibility level 80 databases**

If a SQL Server 2008 or SQL Server 2005 database has its compatibility level set to 80, it conforms to strict rules for views, stored procedures, functions, and DML triggers. Therefore, upgrades may fail.

## Upgrading the database structure and data

---

When you use SQL Packager to upgrade the data in a database, you can select data only for those tables whose structure is identical.

If both the schema and the data has been updated for a particular table, and the schema changes include new columns that do not allow null values, you will have to run two packages; the first package to update the schema, and the second to update the data.

For example, the previous version of the database is called **DatabaseOld**, and the latest is called **DatabaseNew**. To upgrade **DatabaseOld**:

1. Create a package to upgrade only the schema of **DatabaseOld**:
  - a. On the **Choose databases to package into an upgrade** page of the Packager Wizard, select **DatabaseOld** as the database to upgrade, and **DatabaseNew** as the latest version.
  - b. On the **Specify the database objects whose schema will be packaged** page, select all the objects to package their schema.
  - c. On the **Specify the tables whose data will be packaged** page, clear the selection for all of the tables so that no data is packaged.
  - d. Generate the package (for example **Package1**).
2. Run **Package1** on **DatabaseOld**.  
**DatabaseOld** now contains the upgraded schema, but still has the old data.
3. Create a package to upgrade the data in **DatabaseOld** with the data in **DatabaseNew**:
  - a. On the **Choose databases to package into an upgrade** page of the Packager Wizard, select **DatabaseOld** as the database to upgrade, and **DatabaseNew** as the latest version.
  - b. On the **Specify the database objects whose schema will be packaged** page, no objects will be available for schema packaging because they are now identical.
  - c. On the **Specify the tables whose data will be packaged** page, select all of the tables so that all data is packaged.
  - d. Generate the package (**Package2**).
4. Run **Package2** on **DatabaseOld** to complete the upgrade.

If you need to upgrade a number of databases, you should deploy both packages and run **Package 1** followed by **Package 2**.

## Troubleshooting

---

This topic provides information that may help you if you are having difficulties. You may also wish to refer to Understanding the results (page 38).

### SQL Packager sometimes creates DLL files with the executable

SQL Packager creates a dynamic-link library (DLL) file in addition to the executable when the package exceeds 100MB. For each additional 100MB, another DLL file is created. You must distribute the executable and all the DLL files to run the package.

### Insufficient disk space

SQL Packager may be unable to create or upgrade databases if there is insufficient disk space.

SQL Packager uses temporary files when it analyzes the databases to create the package. To successfully create a package, these temporary files require available disk space at least four times the size of the database you are packaging.

You can decrease the size of the temporary files by selecting the **Use checksum comparison** data packaging option.

The location of the temporary files is defined by the **RGTEMP** environment variable, or the **TMP** variable if **RGTEMP** does not exist (see your Windows® documentation for information about environment variables). Note that changing the **TMP** variable will affect other programs that use the variable.


### Missing tables in Packager Wizard

If you are creating a package to upgrade a database, you can select tables to package their data only if they have:

- similar names  
You can set SQL Packager so that it ignores the case of object names, and spaces or underscores in object names by using the data packaging options.
- the same owner (case-sensitive)
- similar schema
- a matching primary key, unique index, or unique constraint that has the same name in both databases

If the tables are very different, for example if primary keys or column data types are different, SQL Packager cannot upgrade the data. Red Gate Software Ltd offers **SQL Compare** ([http://www.red-gate.com/products/SQL\\_Compare/index.htm](http://www.red-gate.com/products/SQL_Compare/index.htm)), which will synchronize the schema of two databases. You can then use SQL Packager to package an upgrade to the data.

Alternatively, you could use SQL Packager to upgrade the schema, then generate a second package to upgrade the data. For details, see Upgrading the database structure and data (page 43).

In addition, if the structure of the databases that you have selected has changed while you are working on a project, those changes are not automatically shown in the **Choose the tables whose data will be packaged** page of the Packager Wizard. To refresh the page, click ; you can then select the tables. For more information, see Specifying the package contents (page 21).

### Identical CLR data is flagged as different

SQL Packager considers the collation order for CLR and XML data. Therefore, if the ordering is not identical, differences are reported.

### Identity columns created or upgraded even though they are excluded

If you clear the data packaging option **Include identity columns**, but an identity column is the key used to match records, the identity column is included in the package.

### Data has not been upgraded

This may occur if:

- there are triggers defined on the tables  
If you have a trigger defined on a table that inserts data into another table on INSERT, DELETE, or UPDATE, the data in the tables will change as the upgrade is run, which will cause unpredictable results. To avoid this, select the **Disable DML triggers** data packaging option before you generate the package.
- primary keys are defined on columns with differing collation order  
If you upgrade tables that have primary keys defined on columns that have different collation order, SQL Packager may produce unpredictable results.
- columns contain timestamp data  
SQL Packager cannot upgrade data in *timestamp* columns.

tables do not have identical schema

- If the structure of the tables you are upgrading is not identical, SQL Packager may produce unpredictable results.

### CLR data has not been upgraded

Data for CLR types can be stored as string or binary values. When CLR data is compared, SQL Packager always compares the binary representations. However, by default, when CLR data is upgraded, SQL Packager updates the string representations, because binary formats are not always compatible.

If you know that the binary formats are compatible, you can select the **Transport CLR data types as binary** data packaging option to force SQL Packager to update the binary representations.

### Primary keys, indexes, or unique constraints are not dropped

If you select the data packaging option **Drop primary keys, indexes, and unique constraints**, note that primary keys, indexes or unique constraints that are selected as comparison keys are not dropped.

## Rollback on script cancellation or failure

If a script fails, or if it is cancelled, in most circumstances changes are rolled back. SQL Packager uses *transactions* to do this. However, there are some circumstances in which this is not possible:

- if full-text information must be altered  
For example, within a transaction, catalogs cannot be dropped, and indexing cannot be added to a column.
- if users and roles need to be created, altered, or deleted  
For example, within a transaction, a user cannot be created, or added to a role.

In these cases, SQL Packager rolls back all the changes that it can. Your database will be in an undetermined state.

Note that if the data packaging script fails, only the data changes are rolled back; the changes made by the schema packaging script are not rolled back.

If you have selected the schema packaging option **Do not include plumbing for transactional synchronization scripts** to remove transactions from the schema packaging script, or cleared the data packaging option **Use transactions in SQL scripts** to remove transactions from the data packaging script, no changes are rolled back when the script is cancelled or fails. This may be useful if you want to run a script up to the point of failure, for example for debugging.

SQL Packager always warns you if it is unable to roll back changes.

## Common error messages

---

Some of the more common error messages are explained below.

### User already exists in database

In Microsoft® Windows®, users are a composite of the domain name or computer name, and the user name, for example *Computer1\WindowsUser1*. SQL Packager references both parts of the name. However, SQL Server references only the user name, so that *Computer1\User1* and *Computer2\User1* would be considered as the same. Therefore, if SQL Packager attempts to create a user for which the computer name is different but the user name is the same, SQL Server returns an error.

To avoid this error, ensure that the user names are different for users that you want to upgrade.

### SQL Server doesn't exist or access is denied

SQL Packager cannot connect to the SQL Server. Try the following to rectify the problem:

1. Verify that the SQL Server is online and that the SQL Server name is listed in your LAN by *pinging* the address.

For example, open a command prompt and run the following command:

```
ping <ServerName>
```

where *ServerName* is the name of your SQL Server.

2. If the SQL Server is online, verify that you are connecting to the correct port.

If your SQL Server is not running on the default port (1433), type the following in

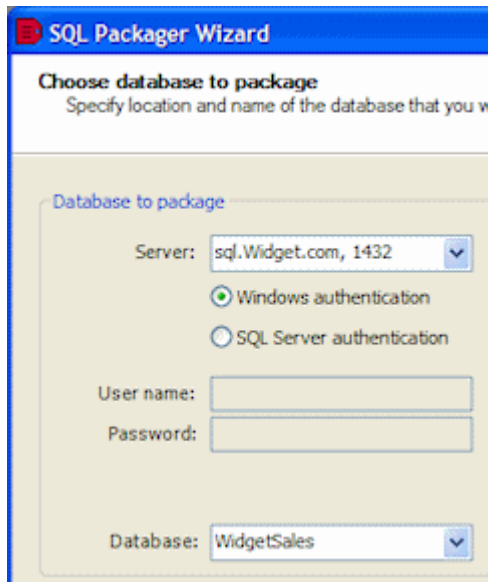
**Server:**

```
<ServerName>,<Port>
```

where *ServerName* is the name of your SQL Server and *Port* is the number of the port on which your SQL Server is running.



For example:



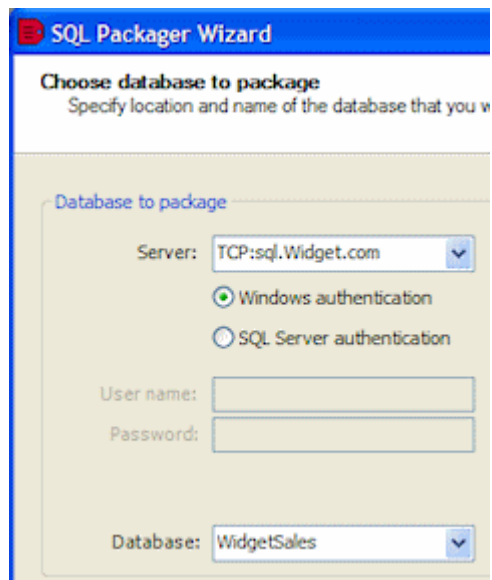
The screenshot shows the 'SQL Packager Wizard' dialog box with the title 'Choose database to package'. Below the title is the instruction 'Specify location and name of the database that you w'. The main area is titled 'Database to package' and contains the following fields:

- Server: sql.Widget.com, 1432 (dropdown menu)
- Authentication:  Windows authentication,  SQL Server authentication
- User name: (empty text box)
- Password: (empty text box)
- Database: WidgetSales (dropdown menu)

3. If you are sure that you are connecting to the correct port, force SQL Packager to use the TCP network protocol when it makes the connection, by typing the following:

`TCP:<ServerName>`

For example:



The screenshot shows the 'SQL Packager Wizard' dialog box with the title 'Choose database to package'. Below the title is the instruction 'Specify location and name of the database that you w'. The main area is titled 'Database to package' and contains the following fields:

- Server: TCP:sql.Widget.com (dropdown menu)
- Authentication:  Windows authentication,  SQL Server authentication
- User name: (empty text box)
- Password: (empty text box)
- Database: WidgetSales (dropdown menu)

### A duplicate object name has been found

SQL Packager displays this message when you connect to a database on a SQL Server that uses case-sensitive sort order and you have not selected the **Treat items as case sensitive** schema packaging option; you must select this schema packaging option.

## Using the command line interface

---

The command line interface provides access to the functionality provided by SQL Packager. For example, using the command line interface you can:

- automate the comparison, synchronization, and packaging of both database structures and data
- perform scheduled comparisons and synchronizations
- synchronize multiple databases
- upgrade customer databases without manual intervention

You invoke the command line either from a script, such as a batch script or VBScript, or by using the facilities provided by compiled languages such as C++ and C#.

This online Help provides a description of basic command line features (page 50) and examples (page 53) illustrating how you can use the command line interface. To display full help on all of the switches that are available for the command line, at the command prompt enter:

```
sqlpackager /help /verbose
```

where */help* displays the help message and can be used in conjunction with *verbose* for more detailed information. To output the help in HTML format, use the */html* switch, for example:

```
sqlpackager /help /verbose /html > filename.htm
```

If you specify any other switches, they are ignored.

### Prerequisites

To use the SQL Packager command line interface, you must have:

- a license for SQL Packager 6.0 (or later versions), a license for the Professional Edition of SQL Packager version 5.5 and earlier, or a SQL Toolbelt license  
If you do not have a license, you can use the command line for 14 days.
- .NET framework version 2.0 or later  
This is required to run the command line interface, but it is not required when you develop applications and scripts that use the interface.
- MDAC 2.8 or later

For information about distributing the command line interface with your application, see Integrating the command line with applications (page 54).

## Basic command line features

---

This topic describes how to use the basic features of the command line.

### Getting help from the command line

To display help on any of the tools from the command line, enter:

```
sqlpackager /help
```

This displays a brief description of the tool, and basic help on all the command line switches.

For more detailed help enter:

```
sqlpackager /help /verbose
```

This displays a detailed description of each switch and the values it can accept (where applicable), and all exit codes. To output the help in HTML format, enter:

```
sqlpackager /help /verbose /html
```

### Entering a command

When you enter a command line, the order of switches is unimportant. You are recommended to follow the Microsoft convention of separating a switch from its values using a colon as shown below.

```
/out:output.txt
```

(You can separate a switch that accepts a single value from its value using a space, but this is not recommended.) Values that include spaces must be delimited by double quotation marks ( " ). For example:

```
/out:"c:\output file.txt"
```

Note that if you delimit a path with double quotation marks, you must not terminate the path with the backslash character ( \ ), because the backslash will be interpreted as an escape character. For example:

```
Incorrect:    /location:"C:\Packages\"
```

```
Correct:     /location:"C:\Packages"
```

For switches that accept multiple values, use commas to separate the values. For example:

```
/options:IncludeDependencies,ForceColumnOrder
```

For switches that accept a compound value, separate each part of the value using a colon. For example, the */includeschema* and */excludeschema* switches are used to include and exclude database objects from the actions performed by the tool. For example:

```
/includeschema:table:Product
```

includes all tables for which the table name contains the word *Product*.

## Aliases

Many of the switches have an alias. The alias provides a convenient short-hand way to specify the switch. For example, `/?` is the alias for the `/help` switch, and `/v` is the alias for the `/verbose` switch. Note that switches and aliases are not case-sensitive.

## `/options` switch

You can use the `/options` switch to change your options. For example, by default, comparisons are not case-sensitive; to specify case-sensitive comparisons:

```
/options:CaseSensitiveObjectDefinition
```

However, note that if you set any options explicitly, all of the default options are switched off.

Refer to the full command line help for more information about which options are set by default, and all the options that are available.

## `/verbose` and `/quiet` switches

The standard output mode prints basic information about what the tool is doing while it is executing. You can specify verbose and quiet modes using the `/verbose` and `/quiet` switches, respectively: in verbose mode, detailed output is printed; in quiet mode, output is printed only if an error occurs.

## Redirecting command output

Output from all commands can be redirected to a file by one of several methods:

- Use the `/out` switch to specify the file to which you want output directed:

```
sqlpackager ... /out:outputlog.txt
```

where `outputlog.txt` is the name of the file. If the file exists already, you must also use the `/force` switch to force the tool to overwrite the file, otherwise an error will occur.

- Use the output redirection features that are provided by the shell in which you are executing the command.

From the standard command prompt provided by Windows, you can redirect output to a file as follows:

```
sqlpackager ... > outputlog.txt
```

Note that the redirection operator ( `>` ) and file name must be the last items on the command line.

If the specified file exists already, it will be overwritten. To append output from the tool to an existing file, for example to append to a log without losing the data already present in the log, enter the following:

```
sqlpackager ... >> existinglog.txt
```

If you are scripting using a language such as VBScript, JScript, PHP, Perl, or Python, or if you want to access the tool from Web pages using ASP.NET, refer to the documentation for the language.

- Specify command line arguments in an XML file that can be referenced using the `/argfile` switch.

For details, see Using XML to specify command line arguments (page 52).

## Using XML to specify command line arguments

---

You can use an XML file to specify the arguments for the command line interface. You may want to do this because:

- An XML file is easier to read than a long and complex command line, particularly where complex rules for including and excluding objects are specified.
- You can easily transform an XML file into other formats using XSLT.  
For example, you could transform your argument file to HTML for presentation on a Web page.
- Using an XML file overcomes some limitations that can be a problem when you want to specify regular expressions as command line arguments.  
For example, you may want to use the pipe character ( | ) as part of a regular expression, but it causes problems when it is used at the command prompt; if you use an XML file you can use the pipe character with no problems.
- Most programming languages support XML, through built-in or freely available third-party libraries.  
This makes it easy to generate and process the XML file.

Create the XML file in the following format:

```
<?xml version="1.0"?>
<commandline>
  <switch_name1/>
  <switch_name2>switch_value</switch_name2> ....
</commandline>
```

For example, for the */includeschema* and */excludeschema* switches, use the following format:

```
<includeschema>objecttype:RegularExpression</includeschema>
```

To execute the command line tools using an XML argument file as input, at the command prompt enter:

```
sqlpackager /argfile:XMLfilename.xml
```

When you use an XML file to supply the arguments, you cannot specify any other switches on the command line except */verbose* or */quiet*.

## Examples using the command line

---

This topic provides some simple examples of how to use the command line interface. You may also wish to refer to Frequently asked questions for the command line (page 55).

### Packaging databases

To package database *WidgetSales* on the local SQL Server, creating a package executable called *WidgetPackage.exe* in *C:\Packages*, which will create a new, identical database:

```
sqlpackager /database1:WidgetSales /location:"C:\Packages"  
            /name:WidgetPackage /makeexe
```

To create the same package and run it immediately (for example, if you want to test the package):

```
sqlpackager /database1:WidgetSales /location:"C:\Packages"  
            /name:WidgetPackage /makeexe /run
```

To package database *WidgetSales* on the local SQL Server, creating a C# project called *WidgetPackage* in *C:\Packages\Projects*, which will create a new, identical database:

```
sqlpackager /database1:WidgetSales /location:"C:\Packages\Projects"  
            /name:WidgetPackage /makeproject
```

To create the same package and open it immediately in Microsoft® Visual Studio®:

```
sqlpackager /database1:WidgetSales /location:"C:\Packages\Projects"  
            /name:WidgetPackage /makeproject /open
```

To package an upgrade from *WidgetDev* to *WidgetLive* creating a package executable called *WidgetPackage.exe* in *C:\Packages*:

```
sqlpackager /database1:WidgetDev /database2:WidgetLive  
            /location:"C:\Packages" /name:WidgetPackage /makeexe
```

In this example, when the package is run, *WidgetLive* will be updated.

### Using */presql* and */postsq*

Packages created by SQL Packager support the */presql* and */postsq* switches. These switches allow you to specify SQL scripts to run before and after the package executes.

Note that any SQL scripts specified by */postsq* are always run, even if errors occur when the package executes.

For example to run the script *WidgetPostScript.sql* after the package *WidgetPackage.exe*:

```
widgetpackage.exe /postsq:WidgetPostScript.sql
```

## Integrating the command line with applications

---

To integrate the SQL Packager command line tool with applications that you distribute to your customers, you must have one of the following licenses:

- SQL Packager 6.0 (or later)
- SQL Packager Professional Edition 5.5 (or earlier)
- SQL Toolbelt

When you have a license, and you execute the tool, the distribution files that you need to distribute the applications are generated; these files are marked with an asterisk (\*) below.

The files that you should bundle into your application installer are listed below. The files should be installed in the same folder in which your application is installed.

Note that to distribute a package created with the SQL Packager command line, you need to distribute only the package executable.

- SQLPackager.exe
- RedGate.CommandLine.Common.dll
- RedGate.SQLPackager.Distribution.dll\*
- RedGate.SQLPackager.Distribution.mod\*
- RedGate.Shared.SQL.dll
- RedGate.Shared.Utils.dll
- RedGate.SQLCompare.ASTParser.dll
- RedGate.SQLCompare.Rewriter.dll
- RedGate.SQLCompare.Engine.dll
- RedGate.SQLCompare.CommandLine.dll
- RedGate.SQLDataCompare.Engine.dll
- RedGate.SQLDataCompare.CommandLine.dll
- RedGate.SQLPackager.Engine.dll
- RedGate.SQLPackager.CommandLine.dll
- SQLPackager.exe.config
- RedGate.Compression.ZLib.dll
- SQL Packager Code Templates (folder)

## Frequently asked questions for the command line

---

### Licensing

#### How do I activate the command line tools?

To use the SQL Packager command line interface, you must have one of the following licenses:

- SQL Packager 6.0 (or later)
- SQL Packager Professional Edition 5.5 (or earlier)
- SQL Toolbelt

You can also redistribute your application if you have a valid license. For more information, see [Integrating the command line with applications](#) (page 54).

### Comparing databases

#### How can I include or exclude the object definition for specific tables?

You can use the */includeschema* or */excludeschema* switch with regular expressions to do this.

#### How can I include or exclude the data for specific tables?

You can use the */includedata* or */excludedata* switch with regular expressions to do this.

#### What is included or excluded when I use a project?

When you use a project, all objects and data that were selected for inclusion when the project was saved are automatically included; you do not need to explicitly include them using the */includeschema* or */includedata* switches. You can override the inclusion by specifying the */excludeschema* or */excludedata* switches as required.

### Integration

#### How do I integrate the command line tools with applications?

For information about how to integrate the command line tools with applications that you distribute to your customers, see [Integrating the command line with applications](#) (page 54).

#### How do I integrate database package creation with a build process?

You may want to install a package as part of your application installation process. You can create the package during the build, bundle it as part of your installer, and set up the installer to execute it at the appropriate point during the installation.



The build script example below shows how you would execute the SQL Packager command line tool from an NAnt (<http://nant.sourceforge.net/>) build script. You can modify this task for use with other build systems such as Visual Build.

```
...
<!-- Use your SQL command line installation directory -->
<property name="sqlCmdLineInstallDir" value="..."/>
...
<target name="PackageDatabase"
description="Create executable database package.">
  <exec
    basedir="${sqlCmdLineInstallDir}"
    program="sqlpackager.exe"
    commandline="/database1:FirstDatabaseName /makeexe
                /name:PackageName
                /location:TargetDirectoryName"/>
</target>
...
```

By default, NAnt captures the output of the program and incorporates it as part of the build log. Alternatively, you can use the *output* attribute of the *exec* task to specify a file to which the output is to be redirected.

## Acknowledgements

---

### Copyright information

© Red Gate Software Ltd 1999 - 2008.

### Trademarks and registered trademarks

Red Gate is a trademark of Red Gate Software Ltd registered in the U.S. Patent and Trademark Office. SQL Packager is a trademark of Red Gate Software Ltd.

Microsoft, Windows, Windows 98, Windows NT, Windows 2000, Windows 2003, Windows XP, Windows Vista, Visual Studio, and other Microsoft products referenced herein are either registered trademarks or trademarks of Microsoft Corporation.

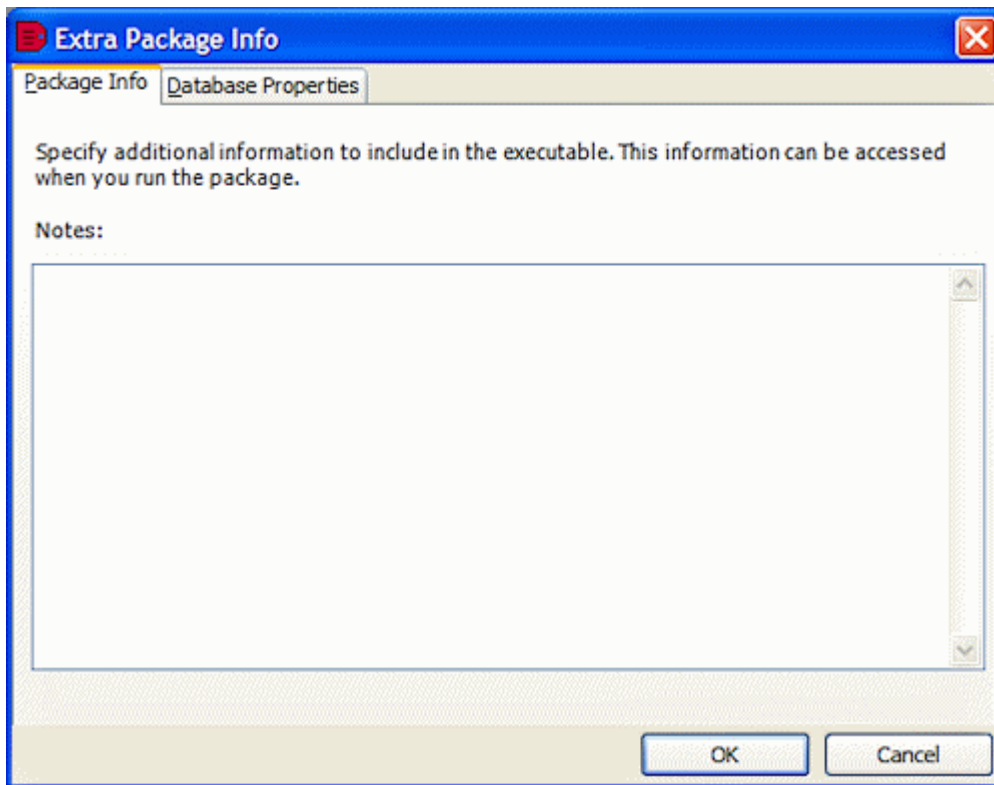
This product contains software that is Copyright © 1995 - 2005 Jean-loup Gailly and Mark Adler.

## Entering extra package information

---

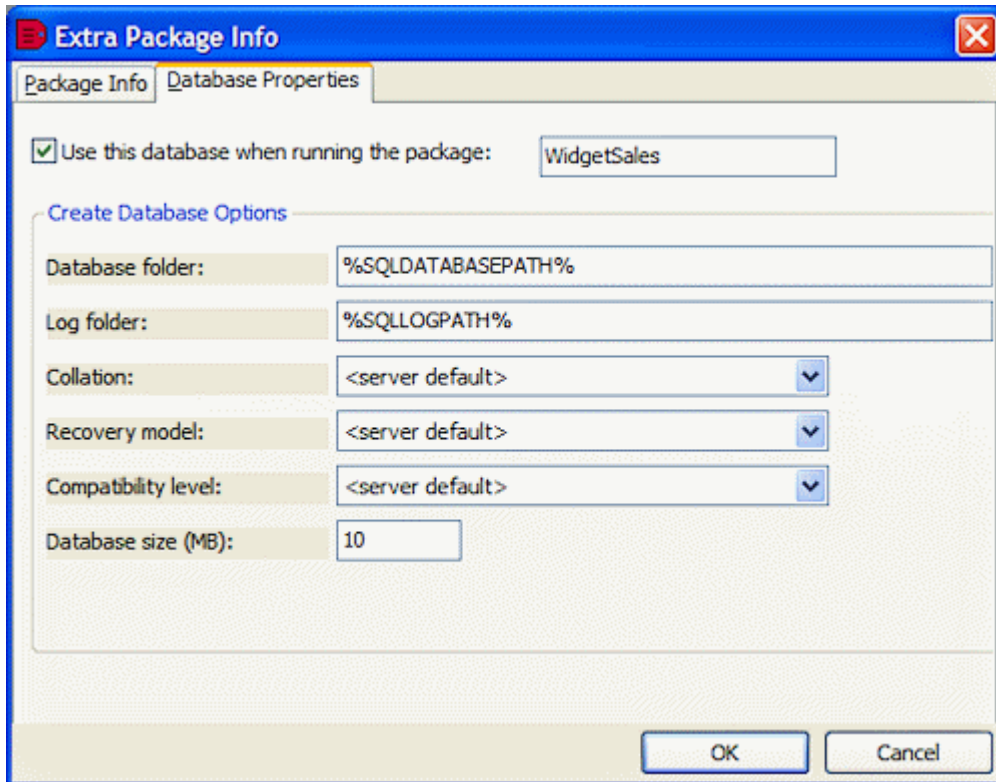
When you create an .EXE or a C# project, you can specify extra information about the package, which can be viewed when the package is run.

To display the **Extra Package Info** dialog box, on the Create .EXE (page 29) or Create C# Project (page 31) page of the Packager Wizard, click **Extra Package Info**.



You can type free form notes on the **Package Info** tab.

If you are creating a package for a new database, you can specify the database properties by clicking the **Database Properties** tab.



Enter the details as required. Note the following:

- **Collation** is dependent on the target system.
- **Compatibility level** must be compatible with the latest version database that you specified when you chose the database to package.

You can access these details at runtime from the **Run Package** dialog box, and amend them if required. For more information, see [Running the Package](#) (page 34).

Note that for upgrades, you can specify only the database name to be used when the package is run.

## Specifying a package type

---

Choosing a project type > Specifying contents (page 21) > Previewing scripts (page 27) > **Specifying a package type** > Running (page 34)

When you have finished reviewing the SQL scripts, specify the package type:

- package as an .EXE (page 29) (as a .NET executable)
- package as a C# project (page 31)

From the **Specify package type** page of the Packager Wizard, you can also launch the script in your default SQL editor, or save the script.

## Setting data packaging options

---

The data packaging options (page 33) are a set of advanced features that enable you to modify the behavior of SQL Packager when it packages data. For example, you can set options so that triggers are disabled when you upgrade a database and then, when the upgrade is complete, re-enable the triggers.

You can also set:

- schema options for packaging the structure of the database
- packager options for the .EXE or C# project

### Disable foreign keys

Disables foreign keys before creating or upgrading the database, and re-enables them on completion. Note that in some circumstances foreign keys will be dropped and recreated rather than disabled and re-enabled.

### Disable DML triggers

Disables DML triggers before creating or upgrading the database, and re-enables them on completion.

For example, you may want to disable triggers if you have a trigger defined on a table that inserts data into another table on INSERT, DELETE, or UPDATE; if you do not, the data in the tables will change as the package is run, which will cause unpredictable results.

### Drop primary keys, indexes and unique constraints

Drops then recreates primary keys, indexes (including XML indexes and partitioned indexes), and unique constraints before creating or upgrading the database, and re-enables them on completion. If the primary key, index, or unique constraint is the comparison key, it cannot be dropped.

Select this option to ensure that unique constraints are not violated when data is inserted into tables or modified. When the constraints are re-created, the data is verified to ensure that no constraints have been violated. (If they have, the script will fail.)

### Use transactions in SQL scripts

If this option is selected and the script fails, the script is rolled back to the start of the failed transaction. If this option is not selected, the script is not rolled back. This can be useful for detection of errors within a script.

### Transport CLR types as binary

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

If this option is selected, SQL Packager uses the binary representation of CLR types; if this option is not selected, SQL Packager uses the string representation. Note that string representations do not always contain the full information about the data.

## Disable DDL triggers

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

Disables DDL triggers before creating or upgrading the database, and re-enables them on completion.

## Trim trailing spaces

If the data in two columns differs only by the number of spaces at the end of the string, SQL Packager considers the data to be identical. This option does not apply to CLR or XML columns.

If this option is selected, trailing spaces are ignored when creating or upgrading databases.

## Force binary collation

SQL Packager uses keys to compare rows. If the comparison key is a string, this option forces a binary collation on all string sorting.

## Use checksum comparison

Performs a checksum prior to comparison. The data is compared only if the checksums differ. Note that if the data differs only in text or image columns, the checksums will be identical and SQL Packager will consider the data to be identical.

For SQL Server 2000, db\_owner permissions are required.

## Ignore case

When you are upgrading a database, SQL Packager will ignore the case of the object names, if this option is selected. For example, SQL Packager will consider [dbo].[Widget] to be the same as [dbo].[wIDgEt] and will compare the data in the two tables.

Note that if the databases that you are upgrading are running on a SQL Server that uses case-sensitive sort order, you should ensure that this option is cleared.

## Ignore spaces

When you are upgrading a database, SQL Packager will ignore spaces in object names, if this option is selected. For example, SQL Packager will consider [dbo].[Widget Prices] to be the same as [dbo].[WidgetPrices] and will compare the data in the two tables.

## Ignore underscores

When you are upgrading a database, SQL Packager will ignore underscores in object names, if this option is selected. For example, SQL Packager will consider [dbo].[Widget\_Prices] to be the same as [dbo].[WidgetPrices] and will compare the data in the two tables.

## Include indexed views

If this option is selected, SQL Packager includes indexed views in the upgrade. Generally, views can be updated only if the referenced rows are from a single table, and the

referenced columns are simple (for example, they must not include identity columns or computed columns).

### **Include timestamp columns**

If this option is selected, SQL Packager will compare timestamp columns when you are upgrading a database. Note that timestamp columns cannot be included in the data packaging script.

### **Include identity columns**

Includes identity columns in the package. Note that if you do not select this option and an identity column is used as the primary key for a table, it will be included in the packaging script.



## Setting SQL Packager options

---

The packager options (page 33) enable you to specify default settings for your packages. SQL Packager uses the packager options when you create an .EXE (page 29) or a C# project (page 31).

You can also set:

- schema options for packaging the structure of the database
- data options for packaging the data

### **Automatically increment package name (if already exists)**

If an executable file or project file with the same name already exists at the default package location, SQL Packager will automatically increment the package name when this option is selected.

### **Default package location**

You use this setting to specify the default location for your package files. Type or select the path for the **Default package location**, or click the browse button to choose the folder.

## Setting schema packaging options

---

The schema packaging options (page 33) are a set of advanced features that enable you to modify the behavior of SQL Packager when it packages the database structure. For example, you can set SQL Packager so that it ignores certain objects, or so that it does not script certain properties in the package (such as the collation order on columns).

You can also set:

- data options for packaging the data
- packager options for the .EXE or C# project

### **Include dependencies**

When this option is selected, SQL Packager checks for object dependencies; if you excluded objects and other objects that you selected are dependent on the excluded objects, the excluded objects are packaged. For example, if you select a stored procedure and it references a table, even if you excluded that table, the table is still packaged.

By default, SQL Packager will include dependencies in the package. Clear the option if you do not want to include the dependencies. Note that clearing this check box may produce unexpected results or the script may fail. Roles and users are always included in the package.

### **Ignore indexes**

Ignores indexes, unique constraints, and primary keys when packaging the database structure.

### **Ignore statistics**

Ignores statistics when packaging the database structure.

### **Ignore foreign keys**

Ignores foreign keys when packaging the database structure.

### **Ignore check constraints**

Ignores check constraints when packaging the database structure.

### **Ignore identity seed and increment values**

For identity properties, ignores only the identity seed and increment values when displaying the objects that are available for packaging. Note that they will be included in the schema packaging script.

### Ignore fill factor and index padding

Ignores the fill factor and index padding in indexes and primary keys when packaging the database structure.

### Ignore permissions

Ignores permissions on objects when packaging the database structure.

### Ignore DML triggers

Ignores DML triggers when packaging the database structure.

### Ignore INSTEAD OF triggers

Ignores INSTEAD OF DML triggers when packaging the database structure.

### Ignore bindings

Ignores bindings on columns and user-defined types. For example, **sp\_bindrule** and **sp\_bindefault** clauses will not be included in the schema packaging script.

### Ignore WITH NOCHECK on foreign keys and check constraints

Ignores the WITH NOCHECK argument on foreign keys and check constraints.

Note that foreign keys or constraints that are *disabled*, are not ignored.

### Ignore constraint and index names

Ignores the names of indexes, foreign keys, primary keys, and default, unique, and check constraints when displaying the objects that are available for packaging. Note that they will be included in the schema packaging script.

### Ignore filegroups, partition schemes and partition functions

Ignores filegroup clauses, partition schemes, and partition functions on tables and keys when packaging the database structure. Partition schemes and partition functions are not available for inclusion in the package when this option is selected.

### Force table column order to be identical

If additional columns are inserted into the middle of a table, this option forces a rebuild of the table so the column order is correct following upgrade. Data will be preserved.

### Ignore white space

Ignores white space (newlines, tabs, spaces, and so on) when displaying the objects available for packaging. Note that white space will not be ignored when the objects are created or upgraded.

### **Ignore comments**

Ignores comments when comparing views, stored procedures, and so on. Note that comments will be included in the schema packaging script.

### **Ignore extended properties**

Ignores extended properties on objects and databases when packaging databases.

### **Treat items as case sensitive**

For databases with case-sensitive collation, enables objects with case-sensitive names to be packaged. For example, considers object names such as *ATable* and *atable* as different and performs case-sensitive comparisons on stored procedures, and so on.

You should use this option only if you have databases with binary sort order or case-sensitive sort order.

### **Ignore SET QUOTED\_IDENTIFIER and SET ANSI\_NULLS statements**

Ignores these SET statements when displaying available views, stored procedures and so on. Note that these statements will be included in the schema packaging script.

### **Ignore collation order**

Ignores collation order on character data type columns when packaging databases.

### **Ignore full text indexing**

Ignores full-text indexes, catalogs, and so on when packaging databases.

### **Do not include plumbing for transactional synchronization scripts**

Removes transactions from the package to produce SQL code that is more readable.

If this option is not selected and the package fails, the script is rolled back to the start of the failed transaction. If this option is selected, the script is not rolled back. This can be useful for detection of errors within a script.

### **Add WITH ENCRYPTION option to stored procedures etc**

Adds WITH ENCRYPTION when stored procedures, functions, views, and triggers are included in the package.

Note that if you use ADD ENCRYPTION on a SQL Server 2008 or SQL Server 2005 database, SQL Packager will not subsequently be able to display, or package the encrypted objects.

### **Do not use ALTER ASSEMBLY to change CLR types**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

If CLR types are to be packaged, this option forces two rebuilds of the table with conversion to and from strings to update the CLR types, instead of using ALTER ASSEMBLY. For a detailed explanation, see Understanding the Results (page 38).

### **Consider next filegroups in partition schemes**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

When this option is selected, if a partition scheme contains a next filegroup, SQL Packager considers the next filegroup for the upgrade if the filegroup is extended. The next filegroup does not affect the way in which data is stored. For a detailed explanation, see Understanding the Results (page 38).

To ignore next filegroups, clear the check box.

### **Ignore certificates, symmetric and asymmetric keys**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

SQL Server 2005 severely restricts access to certificates, symmetric keys, and asymmetric keys. Consequently, SQL Packager can package only the permissions of certificates and asymmetric keys; symmetric keys cannot be packaged.

### **Ignore trigger order**

DML triggers can have an order specified, such as FIRST INSERT, LAST UPDATE, and so on. Select this option to ignore the trigger order for DML triggers when packaging databases. Note that the DDL trigger order is not affected.

### **Ignore event notifications on queues**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

Ignores the event notification on queues when packaging databases.

### **Ignore users' permissions and role memberships**

When role-based security is used, object permissions are assigned to roles, not users. If this option is selected, SQL Packager creates or upgrades object permissions only for roles, and members of roles that are roles. Users' permissions and role memberships are ignored.

### **Ignore users' properties in comparison**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

If this option is not selected, SQL Packager compares user properties, such as the type of user (SQL, Windows, certificate-based, asymmetric key based) and any schema to identify differences. If a user is selected to be upgraded, SQL Packager upgrades the properties where possible.

If you select this option, users' properties are ignored, and only the user name is packaged.

### **Disable and later re-enable DDL triggers**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

DDL triggers can cause problems when you run the packaging script. Select this option to disable any enabled DDL triggers before upgrading the databases, and re-enable those triggers on completion.

### **Ignore the order of WITH elements**

If a stored procedure, user-defined function, DDL trigger, DML trigger, or view contains multiple WITH elements (such as encryption, schema binding, and so on), select this option to ignore the order of the WITH elements when packaging databases.

### **Ignore the lock properties of indexes**

This option is used only for SQL Server 2008 and SQL Server 2005 databases.

Ignores index PAGE LOCK and ROW LOCK properties when packaging databases.

### **Ignore replication triggers**

Ignores replication triggers when packaging databases.

### **Ignores identity properties**

Ignores the identity property on columns when displaying the objects that are available for packaging. Note that the identity property will be included in the schema packaging script.