

SQL Data Generator

Version - 1.2

Contents

Getting started.....	3
Worked example: setting up data generation	4
About generators	9
Generic generators	13
Customizing existing generators	18
Creating new generators	21
Using existing sources	26
Mapping SQL tables or views	27
Mapping CSV files	29
Using the command line interface	32
Acknowledgements	34

Getting started

SQL Data Generator enables you to populate selected tables and entire databases with realistic data. You can populate empty tables, or add extra rows to your existing data.

This is useful when you have a table or database that you want to populate, and no live data that you can use. For example, you may want to create a large database of test data so that you can perform *nunit* or performance tests on a new product. Or you may have live data you do not want to use all of; for example, you may want to replace the data in a credit card number column with randomly-generated data. With SQL Data Generator, you can do this by importing some of the data from existing SQL tables or CSV files, while generating the other columns or tables.

You can use SQL Data Generator to populate SQL Server 2008 and 2008 R2, SQL Server 2005, and SQL Server 2000 databases.

You can use SQL Data Generator to:

- Generate data for all supported SQL Server data types, using the standard generators supplied.
- Customize the generators to suit your specific needs.
- Create a new generator based on one of the supplied generic data generators.

Technical notes

SQL Data Generator automatically assigns a generator to each column based on information such as table name, column name, data type, and any constraints; otherwise the Regular Expressions Generator is assigned.

There are over 80 pre-configured generators, supporting all SQL Server 2008 data types. These are detailed in the list of generators and the data types they support (<http://downloads.red-gate.com/HelpPDF/SupportedDataTypesByGenerator.pdf>).

You must have administrator privileges for the database that you want to populate.

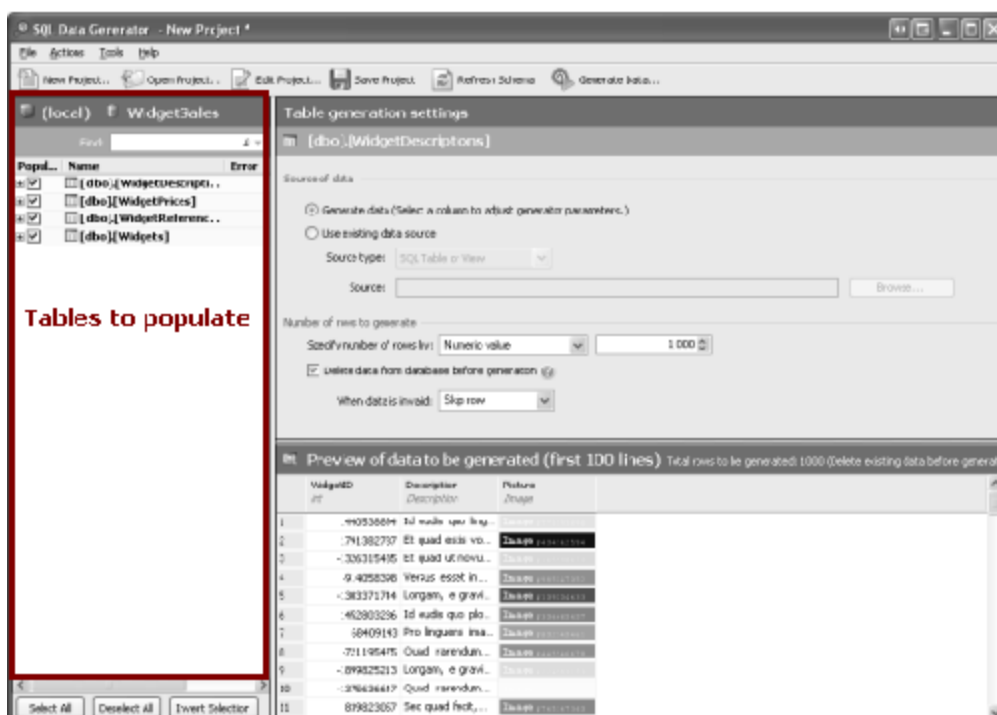
Worked example: setting up data generation

This topic provides an overview of how you set up SQL Data Generator to generate data.

You are recommended to back up the database that you are going to populate before you generate the data; you can then adjust the settings and repeat the data generation if you are not happy with the results.

To generate data, first create a project by selecting the SQL Server and database you want to populate. The project also defines some options for the data generation, and you can specify any number of SQL scripts that you want SQL Generator to run automatically before or after generating the data.

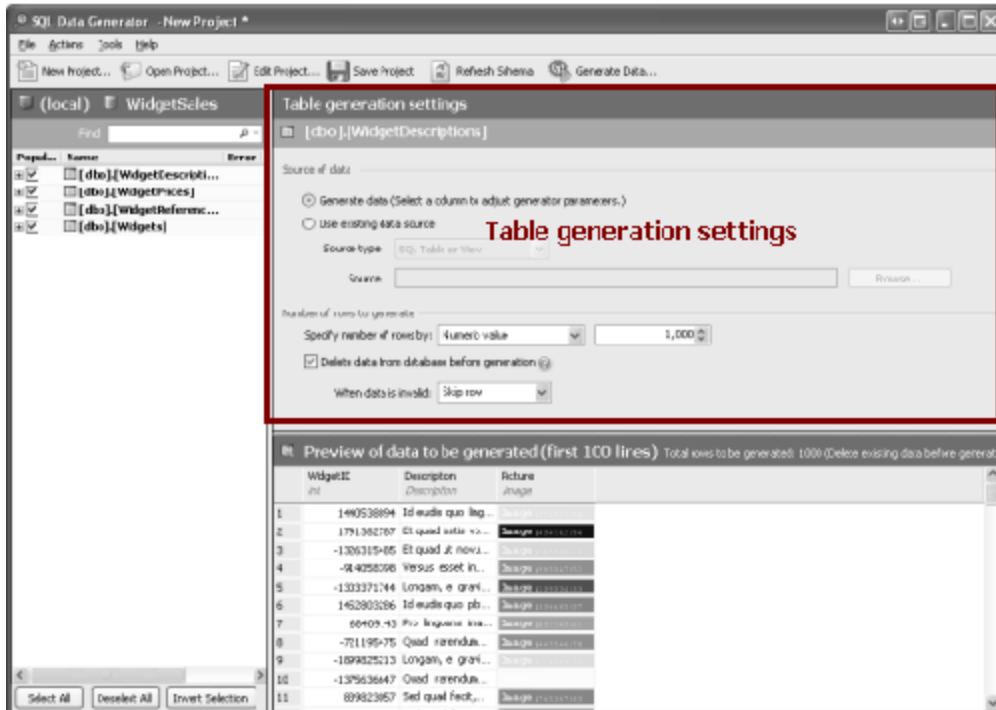
When you have created a project, the schema of the database you selected is listed in a tree view in the **Tables to populate** pane.



You specify the tables that you want to populate by selecting the **Populate** check box. By default, these are all selected, but you can change this option in your application options (accessed from the **Tools** menu).

To see the creation SQL script for a table, right-click the table or column name in the tree view and click **Show Schema Creation Script**.

When you have selected the **Populate** check box for a table, you can define how you want the data to be generated: click the table name in the **Tables to populate** pane, and specify the details in the **Table generation settings** pane.



You can choose to:

- create data using generators

SQL Data Generator automatically assigns a generator to each column based on its table name, column name, data type, and length. If the column has constraints, SQL Data Generator uses these to set the generator parameters for the column; if the constraints cannot be complied with in this way, the RegexpGenerator is assigned instead and an appropriate regular expression is set up. You can change the generator used by a particular column later if required. For detailed information, see About generators.

- import data

You can import a table or view from an existing database, or an existing CSV file. SQL Data Generator maps the columns based on name and data type. If any columns cannot be mapped, SQL Data Generator assigns a generator. You can change the mappings later if required. For detailed information, see Mapping SQL tables or views (page 27) and Mapping CSV files (page 29).

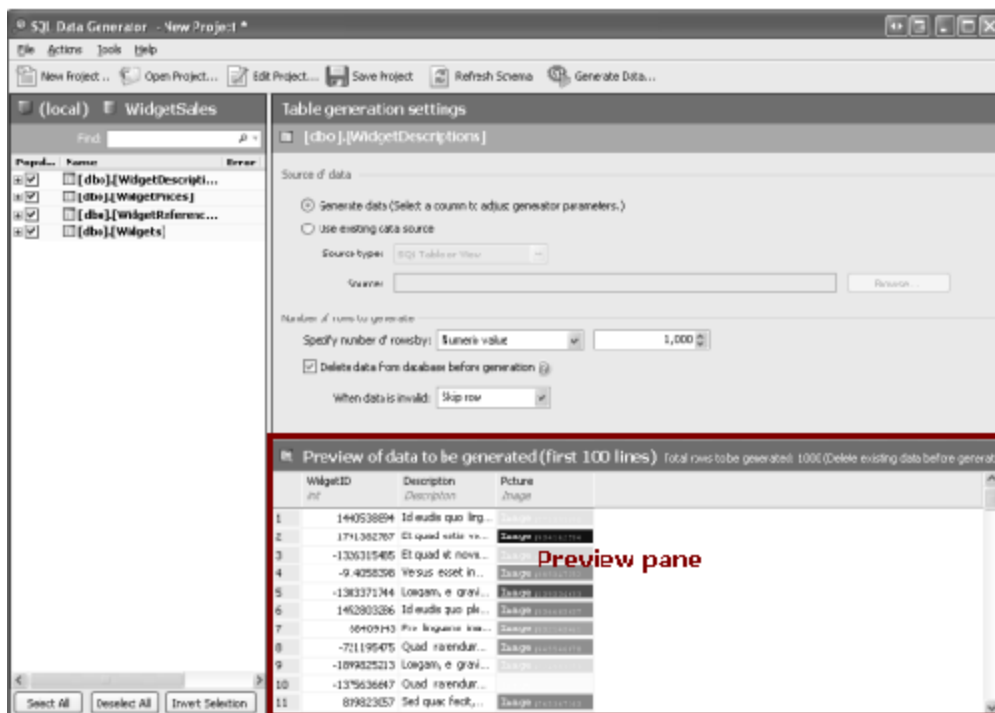
Note that:

- foreign keys are automatically assigned the Foreign Key generator; this cannot be changed to a different generator, but you can customize its settings
- columns for which data is auto-generated display *Server Assigned* in the **Generator** box; this cannot be changed




For example, identity columns and computed columns are server assigned.

In the table generation settings, you can also specify the number of rows that you want to be generated, and whether you want existing data to be deleted prior to populating the table.

You can preview the data that will be generated for each table in the **Preview** pane.




You may see the following icons when the values for a column cannot be previewed prior to generation:

-  server-assigned column
-  foreign key column
-  computed column or a manual foreign key column

You may also see warnings or errors in the preview.

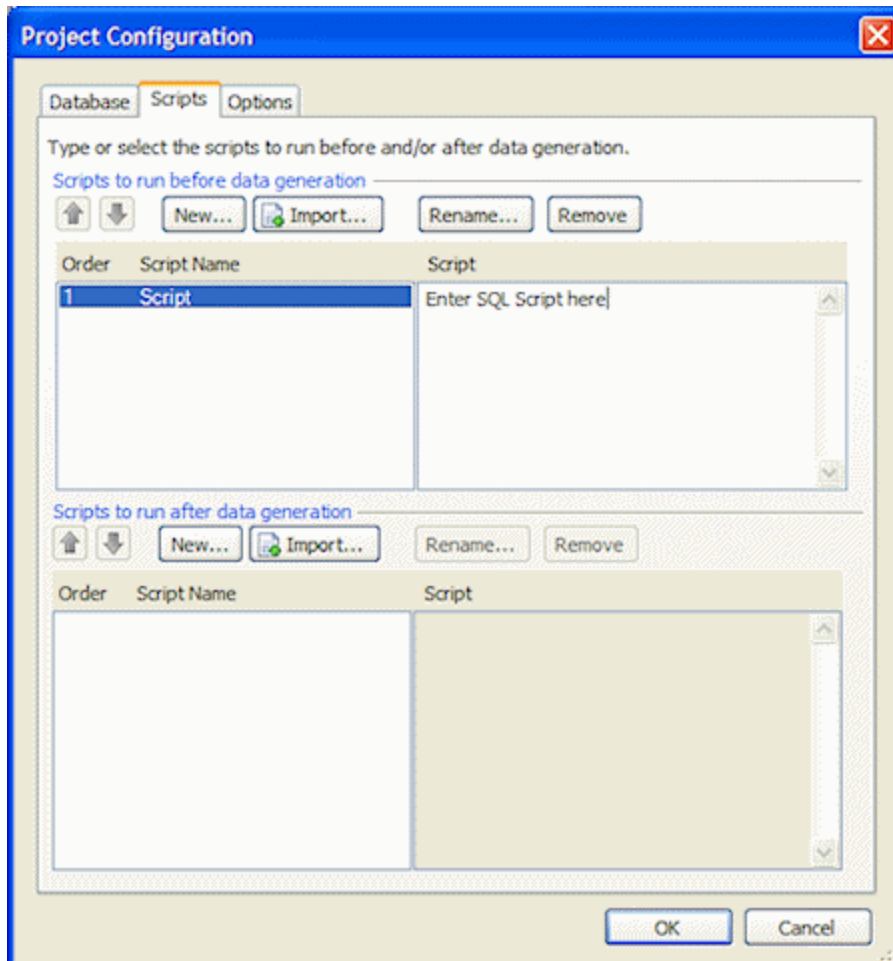
When you have set the table-level parameters, you can check the settings for each column in the table, and customize them if required. To select a column for customization, click the name of the column in the **Tables to populate** pane, or click the column in the **Preview** pane.

When you are happy with the settings for all the columns that you want to populate, click  **Generate Data**. An action plan provides a summary of the data generation for you to review before you generate.

Scripts

You can run SQL scripts before or after the data generation. For example, you could use scripts to add custom data to the database. SQL Data Generator can run these scripts automatically.

You set up the scripts you want to run in the **Project Configuration** dialog box (click  **Edit Project** and select the **Scripts** tab).







You can link to external script files, or you can embed scripts by typing in the **Project Configuration** dialog box, (or a combination of the two). Scripts are run in the order in which you list them in the project configuration.

If a script that is run before generation fails, the generation process is stopped.

If a script that is run after data generation fails, the process continues with the next script.




Warnings and errors

When you are setting up the tables and columns that you want to populate, SQL Data Generator displays warning and error messages to inform you when there may be a problem with the data generation.



-  Errors prevent you from selecting  **Generate Data**. For example, an error is displayed if there are circular dependencies.
-  Warnings inform you when a problem may arise during data generation. The problem does not prevent you from generating data, but it may stop the data generation for a particular table. For example, a warning is displayed if you have chosen to delete rows in a table, but another table references it.
-  Information messages provide you with information about issues that you may want to rectify, but which will not prevent data generation. For example, you may see an information message if SQL Data Generator cannot create enough rows for the preview.

To see the details of an error or warning, hover your mouse pointer over the icon to display a tooltip, or click the column name to see the details in the **Column generation settings** pane.

Refreshing the schema

When you open a project, if the database schema has changed since you created the project, the schema that has changed is flagged with  in the **Tables to populate** pane. For example, a changed column is indicated by  and a changed primary key column is indicated by .

To clear the flags, click  **Refresh Schema**.

You can also click  **Refresh Schema** to see any changes that have been made to the schema since you opened it. Click  **Refresh Schema** again to clear the flags.

Command line

When you have set up a SQL Data Generator project, you can use the command line to

Chapter

run the project. For more information, see Using the command line interface.

About generators

SQL Data Generator uses generators to create the data for the tables that you choose to populate. Different generators are used to create different types of values, and to enable you to define specific parameters for the values.

When you select a column in the **Tables to populate** pane, **Generators** lists only the generators that create data of the same data type as the column. For example, if the column type is *int*, only generators that create integer values are available in the list.

SQL Data Generator provides a number of pre-defined generators, such as *FirstName*, *WorkingAge*, *Country*, and so on. These generators are grouped by subject area in the **Generator** list. You can change the settings for these generators as required.

In addition, SQL Data Generator provides some non-specific generators for you to customize:

- *SQL Type* lists a generator for each SQL data type (except CLR)
- *Generic* lists some basic generators

For information about the generic generators, see [Generic generators](#).

For full details about customizing generators, see [Customizing generators](#).

For information about how you can create your own generators, see [Creating new generators](#).

Data types supported by the supplied generators

To see a matrix of the data types that are supported by the supplied generators, see the table of supported data types by generator (PDF) (<http://downloads.red-gate.com/HelpPDF/SupportedDataTypesByGenerator.pdf>)

Uniqueness

Many of the generators have a **Set unique** setting. When this check box is selected, SQL Data Generator makes the values that are generated for the column unique.

If the column schema has a uniqueness constraint (such as a unique index or primary key), **Set unique** is selected by default. However, you can override the uniqueness for the column by clearing the check box. For example, you may want to do this if the uniqueness constraint applies across multiple columns, and you know that another of the columns is unique. A warning is displayed, but you can proceed with the generation.


Generators that do not offer the **Set unique** option are not available for columns that have a uniqueness constraint, except for the SQL statement generator.

If **Set unique** is selected but there are not enough unique values to display in the preview, a warning is displayed. However, you can proceed with the generation. (Note that you can change the number of values to be displayed in the preview by changing your application options, which are available from the **Tools** menu.)

Check constraints

When SQL Data Generator automatically assigns generators to the columns in a new project or new schema, it sets the generator parameters to take account of any check constraints.

However, it is not always possible to set the generator parameters appropriately. When you generate data, if the values generated do not comply with a check constraint, data generation for that table is stopped and an error is reported.

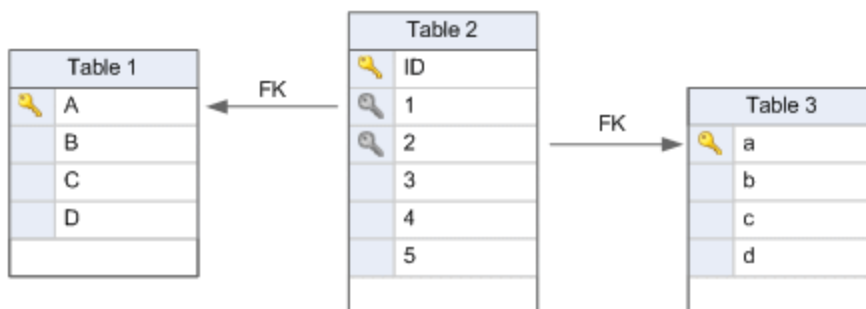
You can set up the project so that check constraints are not enforced when the data is generated. To do this, clear the **Enforce check constraints** check box in the project configuration options (click  **Edit Project** and select the **Options** tab).

Foreign keys

When SQL Data Generator automatically assigns generators to the columns in a new project or new schema, the Foreign Key generator is assigned to all columns that have foreign key constraints.

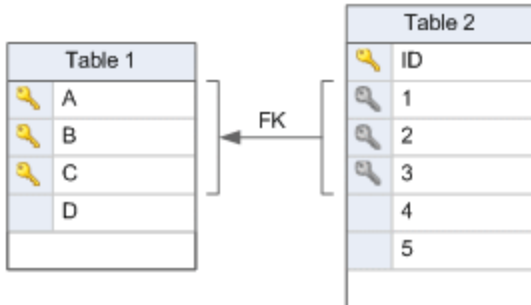
You cannot change the generator, but you can change the settings for the Foreign Key generator.

In the example below, Table 2 Column 1 references Table 1 Column A, and Table 2 Column 2 references Table 3 Column a.

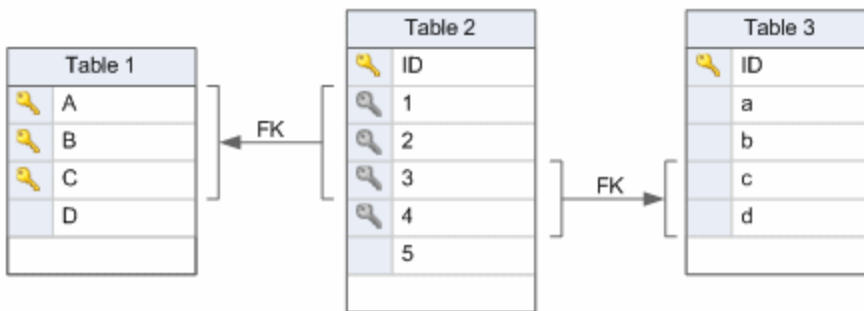


SQL Data Generator assigns the Foreign Key generator to Column 1 and Column 2. You can change the settings for these columns individually.

For a composite foreign key, the generator settings are the same for each of the columns; if you change the settings on one column, they are automatically changed on the others. In the example below, changing the generator settings for Column 2 in Table 2 also changes the settings for columns 1 and 3.



Similarly, if two or more composite foreign keys overlap, the generator settings are the same for each of the foreign keys. In the example below, changing the generator settings for Column 4 in Table 2 also changes the settings for Columns 1, 2, and 3 in Table 2.



In addition, when two composite foreign keys overlap, for the overlapping column(s) SQL Data Generator uses values that appear in both referenced tables; that is, if a value appears in one referenced table but not in the other, that value will not appear in the generated data. In the example above, only values that appear in both Table 1 Column C and Table 3 Column c will be used for Table 2 Column 3.

A NULL value in a composite foreign key is NULL across all of the columns in the foreign key.

SQL Data Generator cannot display preview values for the Foreign Key generator; 🔑 is displayed instead.


Foreign Key (manual) generator

You can create a single-column foreign key by using the Foreign Key (manual) generator, which is available under the SQL Types category.

There is no restriction on the data type of the column you select. However, if possible, you should select a column with the same data type. If you select a column with a different data type, SQL Data Generator attempts to convert the values when the data is



generated; if SQL Data Generator is unable to convert the data, the data generation may fail.

You cannot create a self-referential foreign key using this generator.

SQL Data Generator cannot display preview values for the Foreign Key (manual) generator;  is displayed instead.

Dependencies

SQL Data Generator takes dependencies into account when defining the order in which tables will be generated.

If there are any circular dependencies,  is displayed next to the relevant columns in the **Tables to populate** pane, and  **Generate Data** is not available.

Generating XML

There are a number of ways in which you can generate XML values:

- use the XML generator to generate XML strings
- use the RegExGenerator and write a regular expression that obeys the XML definition
- use the File Import generator to import XML files
- use the SQL Statement generator to retrieve values from another database that contains schema-validated XML

Generating real numbers

When you use the *real* SQL type generator, if you set **Min** or **Max** to be a large value, sequential distribution will not produce sequential values because the increment cannot be set high enough.

Generic generators

SQL Data Generator provides the following generators in the Generic category for you to customize:

- CSV (page 13)
- File Import (page 13)
- File List (page 14)
- RegexpGenerator
- SQL Statement
- Text Shuffler (page 16)
- Weighted List (page 17)

Information about each of these generators is provided below.

For information about how to customize the generators, see Customizing existing generators.

Data types supported by the supplied generators

To see a matrix of the data types that are supported by the supplied generators, see the table of supported data types by generator (PDF file) (<http://downloads.red-gate.com/HelpPDF/SupportedDataTypesByGenerator.pdf>)

CSV generator

Use the CSV generator when you want to import data from a CSV file into a single column. (If you want to import data from a CSV file into an entire table or multiple columns in a table, you can use the **Use existing data source** table generation setting instead; for details, see Mapping CSV files (page 29).)

Click **Browse** to select the CSV file you want to use; you then specify the delimiters to be used when importing the data, and select the column in the CSV file that you want to import.

Note that when you select **Shuffle data**, changing the **Seed** value in the CSV generator settings changes only the position of any null values.

File Import generator

Use the File Import generator to import the contents of files in a specified folder.

For example, if you specify a folder containing a number of images, each image is imported into a new row. You can specify a search string to identify the files within the specified folder you want to use.

If you specify large files, or if you specify a large number of files, performance will be reduced.

File List generator

Use the File List generator to import values from a text file.

You must first create a text file containing the list of values, with each value on a new line. The values will be imported from the list in a random order. You can then browse to this file when you select the File List generator.

If you have a very long list of values, you may want to consider creating a CSV file with the list of values and then importing the values using the CSV generator (page 29) to import the values.

Regexp generator

Use the Regexp generator to define the generated data using a regular expression.

In the basic syntax, most characters are treated as literals (for example, a generates "a"). Below is a list of syntax elements.

Syntax	Example	Generates
ordinary chars	bob	bob
[chars]character set	[A-Z0-9]	eg. 5 or G
individual chars	[FM]	F or M
initial] in char set	[]]
[x-y] range	[0-9]	eg. 3 or 9
complement	[^abc]	eg. d or #
* zero or more	abc*	eg. abcccccc or ab
+ 1 or more	abc+	eg. abcccc or abc
? Include or not	abc?	ab or abc
(regexp) grouping	(abc)*d	eg., abcabcd or d
{num} repeat	a{4}	aaaa

<code>{min,max}</code> repeat	<code>a{2,3}</code>	<code>aa or aaa</code>
<code>{min,}</code> at least min repeats	<code>a{3,}</code>	eg. <code>aaa</code> or <code>aaaaaaaaaa</code>
<code>()</code> empty string	<code>()</code>	
<code> </code> alternatives	<code>Yes No</code>	<code>Yes or No</code>
Empty Alternative	<code>(some- often-)time</code>	eg. <code>some-time</code> or <code>time</code>

Escapes:

Syntax	Generates
<code>\xdd</code> hex char (8- bit)	eg. <code>\x21</code> generates !
<code>\udddd</code> hex unicode	eg. <code>\u0021</code> generates !
<code>\\</code>	<code>\</code>
<code>\.</code>	<code>.</code>
<code>\^</code>	<code>^</code>
<code>\\$</code>	<code>\$</code>
<code>\{</code>	<code>{</code>
<code>\[</code>	<code>[</code>
<code>\]</code>	<code>\]</code>
<code>\(</code>	<code>(</code>
<code>\ </code>	<code> </code>
<code>\)</code>	<code>)</code>
<code>*</code>	<code>*</code>
<code>\+</code>	<code>+</code>
<code>\?</code>	<code>?</code>
<code>\a</code>	alarm character
<code>\b</code>	backspace
<code>\d</code>	digit
<code>\e</code>	escape
<code>\f</code>	formfeed
<code>\n</code>	newline
<code>\t</code>	tab

<code>\r</code>	carriage return
<code>\s</code>	space
<code>\v</code>	vertical tab
<code>\w</code>	[A-Za-z_0-9]

Use `$(file_name)` to import a text file containing a list of values.

Use `$(column_name)` to import the values from another column in the same table. Note that when you import values from another column:

- for repeating values, you must specify a fixed range of repetitions using `{1,10}`; you cannot use `*` when you import values from a column
- changing the **Seed** value in the RegexpGenerator settings changes only the position of any null values; to shuffle the order of the values, use the settings on the referenced column
- you should not select **Set unique** in the RegexpGenerator settings, because only one row will be generated

You can use the buttons below the **Regular Expression** box to add snippets of code, file lists, and table columns.

SQL Statement generator

Use the SQL Statement generator to define data to import from an external database using a SQL statement. If you want to import data from an external database into an entire table or multiple columns in a table, you can use the **Use existing data source** table generation setting instead; for details, see Mapping SQL tables or views (page 27).

Click **Edit** to specify the database and the SQL statement. The SQL statement must select a single column of values which are of the correct data type.

If the column you are populating has a unique constraint, you must ensure that the SQL statement returns unique values; if it does not, the data generation will fail.

Note that when you select **Shuffle data**, changing the **Seed** value in the SQL Statement settings changes only the position of any null values.

Text Shuffler generator

Use the Text Shuffler generator when you want to create values that contain words randomly selected from a pre-defined list. For example, you may want to use this to check the performance of a full text index.

You can type the text, or you can import it from a text file. You are recommended to use text that contains a high variety of words that are similar to your real data.

SQL Data Generator shuffles the text using the spaces as delimiters for the words. Therefore, if there is punctuation in the text, this will be included in the values.

Weighted List generator

Use this generator when you want to specify the percentage for the number of occurrences of each value in the column. For example, you may want to use this to check that your indexing strategy will work on the table.

Customizing existing generators

You can customize the generators in the following ways:

- change the settings for each column
- for generators that use lists, create a custom list of values

Changing the settings

When you select a generator for a column, you can customize the generator by changing the default settings.

For example, in the Personal category, *Working Age* is pre-configured to have a minimum value of 18 and a maximum value of 65. However, in your company the minimum age may be 21, so you can change this value in **Min**.

The screenshot shows the configuration interface for the 'Working Age' generator. At the top, it says 'Generator: Working Age - 29,30,35,18...'. Below this is the 'Generator settings' section. It contains several input fields and checkboxes: 'Seed' is set to 22531; 'Min' is set to 18; 'Max' is set to 65; 'Set unique' is unchecked; 'Allow null values' is unchecked; and '% null' is set to 1. Under the 'Distribution' section, 'Random' is selected with a radio button, and 'Sequential' is unselected. An 'Incremental value' field is set to 1.

Similarly, if you have selected the *Title* generator, you may want to add the title *Prof.* To do this, you can edit the expression in **Regular expression**. Change:

```
Mr|((Mrs|Miss|Ms)|Dr)(\.)?
```

To:

```
Mr|((Mrs|Miss|Ms)|Dr|Prof)(\.)?
```

For more information about regular expression generators, see [RegexpGenerator](#).

This is a good way to customize a generator for an individual column.

Creating custom lists

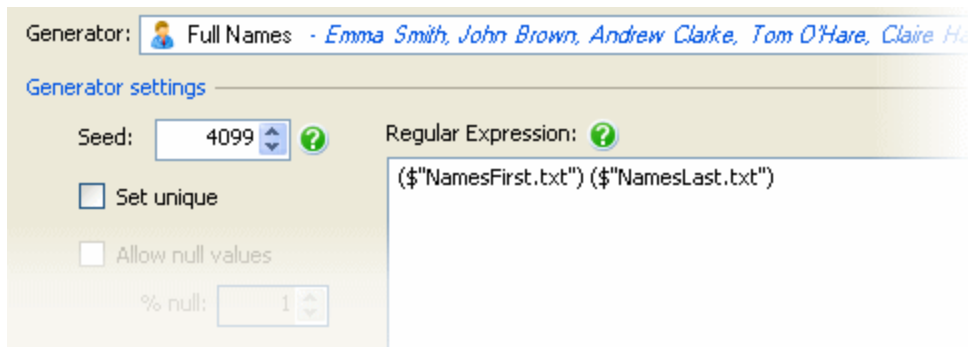
Some of the generators use lists of values (dictionaries) to supply the data. For example, *First Name (Female)* uses the list *NamesFirstFemale.txt*

The lists are located in:

Program Files\Red Gate\SQL Data Generator 1\Config

You can create your own lists by creating a text file with each value on a new line. You can then amend the generator settings so that it uses your file. For example, for *First Name (Female)*, click **Browse...** to select your file. Alternatively, you can use the generic File List generator.

For generators that use regular expressions such as *Full Names*, edit the regular expression to specify the name of the new text file.



For more information about regular expression generators, see RegexpGenerator.

Creating a custom generator

You may find that you want to make the same customizations for a number of columns. In this case, you can customize a generator and add it to the list of available generators so that you can re-use it.

To do this, find the SQL type or generic generator that you want to base the custom generator on, and then find its corresponding *.xml* file in:

Program Files\Red Gate\SQL Data Generator 1\UserExample\Config

Create a copy of the *.xml* file and name it appropriately, then edit the contents to specify the properties as required.

The example below shows the code for the *Working Age* generator which generates random integers between 18 and 65. It is based on *Int32Generator.xml*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<generators>
  <generator
    /* Specify the class. */
    type="RedGate.SQLDataGenerator.Generators.Number.
Int32Generator"
    /* The name and description to be displayed
in the generators list, */
```

```

/* and its category */
name="Working Age"
description="29,30,35,18..."
category="Personal">

/* Set the properties for the generator. */
<property name="MinValue">18</property>
<property name="MaxValue">65</property>

/* Specify the columns that match, and their score. */
<matches field="Age" score="50"/>

/* Define the data types for which the generator
is valid. */
<type type="Int64"/>
<type type="Int32"/>
<type type="Int16"/>
<type type="Byte"/>
</generator>

```

SQL Data Generator uses the matches field string when it allocates the generators to columns. You can specify a regular expression search string. You then specify a score; if more than one generator matches a particular column, SQL Data Generator uses the one with the highest score.

Creating new generators

You may want to create a new generator if the supplied generators do not meet your requirements and you cannot customize them to suit your needs.

To write your own generator, you must be proficient at a .NET 2.0 language, have a good understanding of .NET, and have access to SQL Data Generator on your computer.

The procedure is summarized below:

1. In Microsoft Visual Studio, create a Class Library .NET project.
2. Add references to RedGate.SQLDataGenerator.Engine and RedGate.SQLCompare.Engine
3. Create a public class that implements IGenerator.
4. Add the class attributes.
5. Implement the constructor.
6. Implement the method GetEnumerator.
7. Copy the output DLL to:

Program Files\Red Gate\SQL Data Generator\Generators

Example Microsoft® Visual Studio® 2005 project files are provided in:

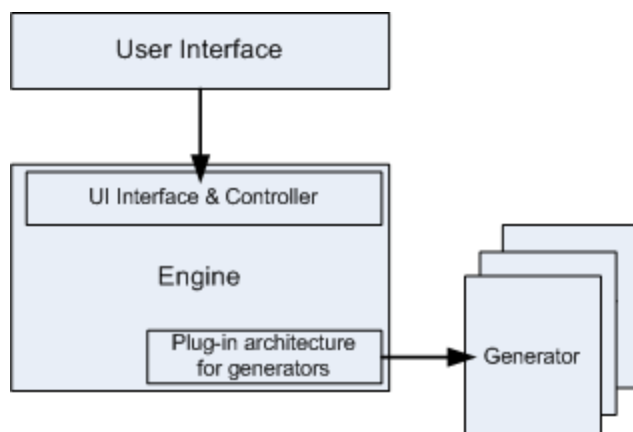
Program Files\Red Gate\SQL Data Generator\UserExample\Generator

The examples are written in C#. If you want to build the projects, you must first reset the project references for the SQL Compare Engine and the SQL Data Compare Engine. You may also want to change the output path. When you have built the project, copy the output DLL to:

Program Files\Red Gate\SQL Data Generator\Generators

Architecture

A simple diagram of the architecture is shown below.



The SQL Data Generator engine defines a series of interfaces. Each interface is very lightweight.

A generator must implement a series of interfaces in order that the engine considers it to be a generator. At startup, a specified folder is scanned for DLLs. Each DLL is loaded in turn, and reflection is used to check whether any public classes implement these interfaces. If they do, the class is considered to be a generator and is made accessible to the rest of the system.

By default, parameters for the generator are displayed in a standard Microsoft Grid Control. To override this default functionality, implement interfaces `IGeneratorUIStyle` and `IGeneratorUI`. See the *FullDemo* project for an example.

Basic interface: IGenerator

The `IGenerator` interface must be implemented for your class to be considered a generator.

The `IGenerator` interface is defined as:

```
namespace RedGate.SQLDataGenerator.Engine.Generators
{
    public interface IGenerator
    {
        IEnumerator GetEnumerator(
            GenerationSession session);
    }
}
```

You must also implement a special constructor that takes a single parameter of type `GeneratorParameters`. This parameter describes the SQL field in the Table that is being assigned. If necessary, your code can throw exceptions and so on.

To display your generator in the graphical user interface (GUI), you must add a simple `Generator` attribute to your class. The following example code produces random values between 0 and 1024 for the 8 times table.

```
namespace Basic
{
    [Generator(typeof(int), "Generic", "8 times table",
        "8, 16, 0, 256, ...")]
    public class Basic : IGenerator
    {
        public Basic(GeneratorParameters parameters)
```

```

    {
    }

    public System.Collections.IEnumerator GetEnumerator
    (GenerationSession session)
    {
        Random r = new Random(0);

        while (true)
        {
            yield return r.Next(0, 1024) * 8;
        }
    }
}

```

The Generator attribute defines the type of .NET result, the Category that the generator is to be placed in, and the name and description to be displayed in the GUI. It must be defined only once per class.

SQL Data Generator assigns the SQL data type that corresponds to the specified type of .NET result. To create a generator that supports multiple SQL data types, add SupportSQLType. SQL Data Generator will add SQL data types based on SqlTypes defined in the SQL Compare engine.

Note that you must ensure that the class is public, and the Generator class exists.

Constructor

GeneratorParameters provides access to the field. This enables your code to verify that lengths and types are consistent.

GetEnumerator

The easiest way to implement this code is by using the Yield statement; the above example never runs out of values. However, it is not always possible to do this. The engine is designed to cater for a limited number of values from the GetEnumerator. If necessary, the GetEnumerator can throw exceptions.

Interface: ISeedableGenerator

This interface enables the generator to specify a seed. The generator can then generate random data that is different each time.

The ISeedableGenerator is defined as:

```
public interface ISeedableGenerator
{
    int Seed { get; set; }
}
```

Use `public bool Unique { get { return m_Unique; } set { m_Unique = value; } }` the seed to initialize the Random class in the GetEnumerator.

The engine will automatically give a value to Seed at initialization. Each column will have its own seed, therefore the same generator can be assigned multiple times within a table and different values will be produced for each column.

A typical implementation is:

```
public int Seed
{
    get { return m_Seed; }
    set { m_Seed = value; }
}
```

For a complete example, see:

UserExample\Generator\Seedable\Seedable.cs

Interface: IUniqueableGenerator

This interface enables the generator to specify whether the data generated is unique so that the generator can then generate a unique value.

The IUniqueableGenerator is defined as follows:

```
public interface IUniqueableGenerator
{
    bool Unique { get; set; }
}
```

A typical implementation is:


```
public bool Unique

{
    get { return m_Unique; }
    set { m_Unique = value; }
}
```

For a complete example, see:

UserExample\Generator\Uniqueable\Uniqueable.cs

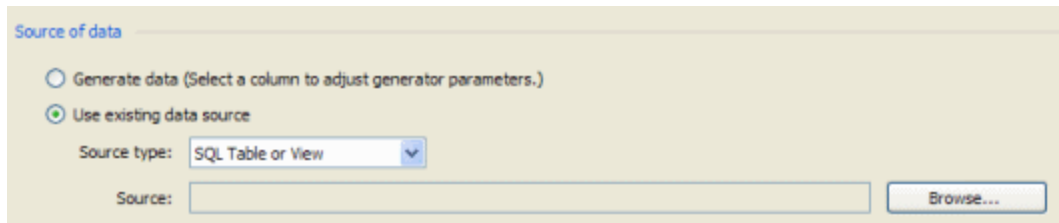
The generator can now be assigned to unique fields. The engine automatically configures the Unique flag as on when the generator is assigned.

Using existing sources

Mapping SQL tables or views

You can populate an entire table, or multiple columns in a table, by mapping to another SQL table or view. For example, this may be useful if you want to import master data or lookup data into your schema.

With the table selected in **Tables to populate**, click **Use existing data source**, select **SQL Table or View** and click **Browse**.

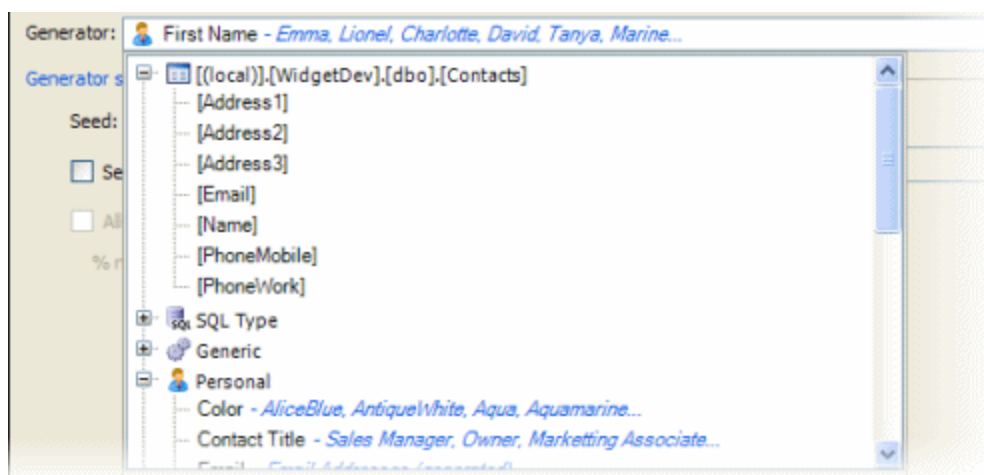


You can then select the SQL Server and database, and the table you want to use from the Select SQL Table or View wizard. When you click **Finish** on the wizard, SQL Data Generator matches columns in the two tables based on data type and column name.

Any columns in the table for which SQL Data Generator does not find a match will have a generator assigned to them instead. However, you can still map the schema column to a column in the external SQL table as long as the data types match:

1. Select the column in the **Tables to populate** pane.
2. Click the **Generator** list.

The list displays the table name, with any columns for which the data types match.



3. Click the name of the column you want to use.

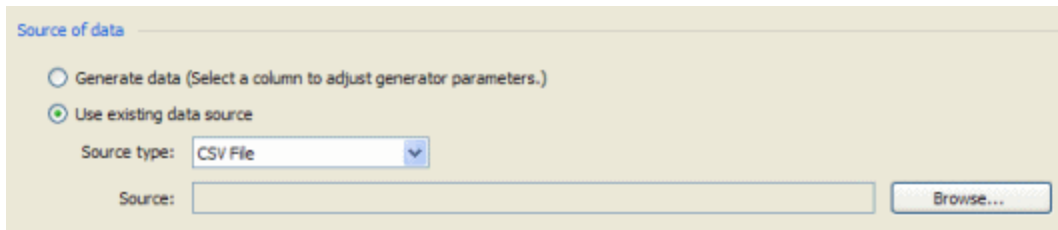
Similarly, if SQL Data Generator maps a schema column to an external SQL table column but you do not want to use it, you can select a different generator for that column.

If you want to define data to import from an external database using a SQL statement, use the SQL Statement generator.

Mapping CSV files

You can populate an entire table, or multiple columns in a table, by mapping an imported CSV file to the table. For example, this may be useful if you want to import master data or lookup data into your schema.

With the table selected in **Tables to populate**, click **Use existing data source**, select **CSV File** and click **Browse**.



Source of data

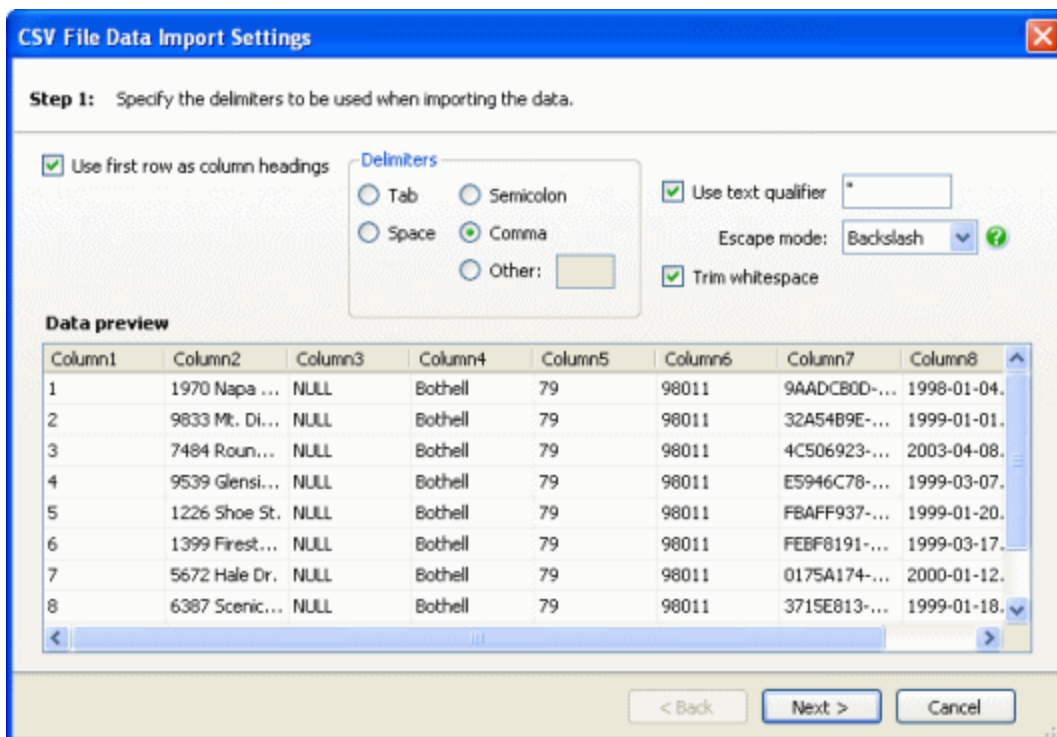
Generate data (Select a column to adjust generator parameters.)

Use existing data source

Source type: CSV File

Source:

You can then select the file, and define the import settings.



CSV File Data Import Settings

Step 1: Specify the delimiters to be used when importing the data.

Use first row as column headings

Delimiters

Tab Semicolon

Space Comma

Other:

Use text qualifier *

Escape mode: Backslash

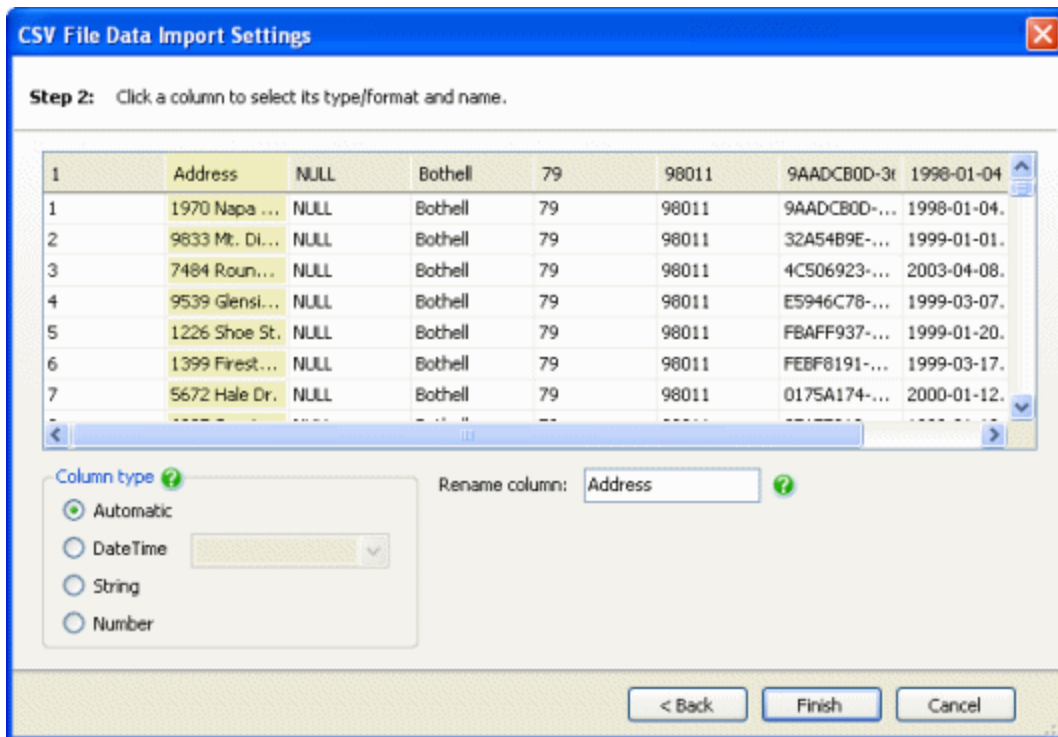
Trim whitespace

Data preview

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8
1	1970 Napa ...	NULL	Bothell	79	98011	9AADCB0D-...	1998-01-04.
2	9833 Mt. Di...	NULL	Bothell	79	98011	32A54B9E-...	1999-01-01.
3	7484 Roun...	NULL	Bothell	79	98011	4C506923-...	2003-04-08.
4	9539 Glensi...	NULL	Bothell	79	98011	E5946C78-...	1999-03-07.
5	1226 Shoe St.	NULL	Bothell	79	98011	FBAFF937-...	1999-01-20.
6	1399 Firest...	NULL	Bothell	79	98011	FEBF8191-...	1999-03-17.
7	5672 Hale Dr.	NULL	Bothell	79	98011	0175A174-...	2000-01-12.
8	6387 Scenic...	NULL	Bothell	79	98011	3715E813-...	1999-01-18.

< Back Next > Cancel

SQL Data Generator matches columns in the CSV file to columns in the table based on data type and column name. If the data type and/or column name for a column in the CSV file is not the same as in the table, you can specify these in the data import settings.



When you click **Finish**, SQL Data Generator maps the columns.

Any columns in the table for which SQL Data Generator does not find a match will have a generator assigned to them instead. However, you can still map the schema column to a column in the CSV file as long as the data types match:

1. Select the column in the **Tables to populate** pane.
2. Click the **Generator** list.

The list displays the CSV file path, with any columns for which the data types match.



3. Click the name of the column you want to use.

Similarly, if SQL Data Generator maps a schema column to a CSV file column but you do not want to use it, you can select a different generator for that column.

If you want to import data from a single column in a CSV file into a single column in a table, use the CSV generator.

Using the command line interface

SQL Data Generator provides a command line interface for you to use with SQL Data Generator projects that you have already created using the graphical user interface.

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

Getting help from the command line

To display help, enter:

```
SQLDataGenerator /help
```

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

For more detailed help enter:

```
SQLDataGenerator /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:

```
SQLDataGenerator /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"
```

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.

/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:

```
SQLDataGenerator ... /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:

```
SQLDataGenerator ... > 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:

```
SQLDataGenerator ... >> 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.

Acknowledgements

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