CHAPTER 5

SQL Server Command-Line Utilities

IN THIS CHAPTER

- What's New in SQL Server Command-Line Utilities
- The **sqlcmd** Command-Line Utility
- The **dta** Command-Line Utility
- The **tablediff** Command-Line Utility
- The **bcp** Command-Line Utility
- The **sqldiag** Command-Line Utility
- The **sqlservr** Command-Line Utility

This chapter explores various command-line utilities that ship with SQL Server. These utilities give administrators a different way to access the database engine and its related components. In some cases, they provide functionality that is also available with SQL Server's graphical user interface (GUI). Other command-line utilities provide functionality that is available only from the command prompt. For each utility, this chapter provides the command syntax along with the most commonly used options. For the full syntax and options available for the utility, see SQL Server Books Online.

NOTE

This chapter focuses on command-line utilities that are core to SQL Server and the SQL Server database engine. Several other command-line utilities that are used less frequently or geared toward other SQL Server services are not covered in this chapter. These utilities include `dtexec` and `dtutil`, which can be used with SQL Server Integration Services (SSIS). Reporting Services has the `rs`, `rsconfig`, and `rskeymgmt` command-line utilities. Lastly, there are several executable files documented as utilities in Books Online (such as `ssms`, which opens the SQL Server Management Studio) that have limited parameters and are basically used to launch their related applications.

Table 5.1 lists the command-line utilities discussed in this chapter. This table lists the physical location of each utility's

executable. The location is needed to execute the utility in most cases, unless the associated path has been added to the `Path` environmental variable.

TABLE 5.1 Command-Line Utility Installation Locations

Utility	Install Location
sqlcmd	x:\Program Files\Microsoft SQL Server\100\Tools\Binn
dta	x:\Program Files\Microsoft SQL Server\100\Tools\Binn
tablediff	x:\Program Files\Microsoft SQL Server\100\COM
bcp	x:\Program Files\Microsoft SQL Server\100\Tools\Binn
sqldiag	x:\Program Files\Microsoft SQL Server\100\Tools\Binn
sqlservr	x:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Binn

NOTE

The `tablediff` utility is installed when SQL Server replication is installed. If you can't find the tablediff.exe in the location specified in Table 5.1, check to see whether the replication was installed.

When you are testing many of these utilities, it is often easiest to set up a batch file (`.BAT`) that contains a command to change the directory to the location shown in Table 5.1. After you make this directory change, you can enter the command-line utility with the relevant parameters. Finally, you should enter a `PAUSE` command so that you can view the output of the utility in the command prompt window. Following is an example you can use to test the `sqlcmd` utility (which is discussed in more detail later in this chapter):

```
CD "C:\Program Files\Microsoft SQL Server\100\Tools\Binn"
SQLCMD -S(local) -E -Q "select @@servername"
pause
```

After you save the commands in a file with a `.BAT` extension, you can simply double-click the file to execute it. This approach is much easier than retyping the commands many times during the testing process.

What's New in SQL Server Command-Line Utilities

The SQL Server command-line utilities available in SQL Server 2008 are basically the same as those offered with SQL Server 2005. This has some key benefits for those who are familiar with the 2005 utilities. Very little has changed in the syntax, and batch files or scripts you have used with these utilities in the past should continue to work unchanged.

A few command-line utilities have been added in SQL Server 2008, however, and some have been removed. The `sqlps` utility is new to SQL Server 2008. This utility can be used to run PowerShell commands and scripts. The `sqlps` utility and the PowerShell Windows–based command-line management tool are discussed in detail in Chapter 17, "Administering SQL Server 2008 with PowerShell."

Utilities removed from SQL Server 2008 include `sac`. The `sac` utility can be used in SQL Server 2005 to import or export settings available in the graphical Surface Area Configuration (SAC) tool. Both the `sac` command-line utility and SAC graphical tool have been removed. Similar functionality is now available via policy-based management and the Configuration Manager tool.

The `sqlcmd` Command-Line Utility

The `sqlcmd` command-line utility is the next generation of the `isql` and `osql` utilities that you may have used in prior versions of SQL Server. It provides the same type of functionality as `isql` and `osql`, including the capability to connect to SQL Server from the command prompt and execute T-SQL commands. The T-SQL commands can be stored in a script file, entered interactively, or specified as command-line arguments to `sqlcmd`.

NOTE

The `isql` and `osql` command-line utilities are not covered in this chapter. The `isql` utility was discontinued in SQL Server 2005 and is not supported in SQL Server 2008. The `osql` utility is still supported but will be removed in a future version of SQL Server. Make sure to use `sqlcmd` in place of `osql` to avoid unnecessary reworking in the future.

The syntax for `sqlcmd` follows:

```
sqlcmd
[{ { -U login_id [ -P password ] } ¦ –E trusted connection }]
[ -z new password ] [ -Z new password and exit]
[ -S server_name [ \ instance_name ] ] [ -H wksta_name ] [ -d db_name ]
[ -l login time_out ] [ -A dedicated admin connection ]
[ -i input_file ] [ -o output_file ]
[ -f < codepage > | i: < codepage > [ < , o: < codepage > ] ]
[ -u unicode output ] [ -r [ 0 | 1 ] msgs to stderr ]
[ -R use client regional settings ]
[ -q "cmdline query" ] [ -Q "cmdline query" and exit ]
[ -e echo input ] [ -t query time_out ]
[ -I enable Quoted Identifiers ]
[ -v var = "value"...] [ -x disable variable substitution ]
[ -h headers ][ -s col_separator ] [ -w column_width ]
[ -W remove trailing spaces ]
```

```
[ -k [ 1 | 2 ] remove[replace] control characters ]
[ -y display_width ] [-Y display_width ]
[ -b on error batch abort ] [ -V severitylevel ] [ -m error_level ]
[ -a packet_size ][ -c cmd_end ]
[ -L [ c ] list servers[clean output] ]
[ -p [ 1 ] print statistics[colon format]]
[ -X [ 1 ] ] disable commands, startup script, environment variables [and exit]
[ -? show syntax summary ]
```

The number of options available for `sqlcmd` is extensive, but many of the options are not necessary for basic operations. To demonstrate the usefulness of this tool, we look at several different examples of the `sqlcmd` utility, from fairly simple (using few options) to more extensive.

Executing the `sqlcmd` Utility

Before we get into the examples, it is important to remember that `sqlcmd` can be run in several different ways. It can be run interactively from the command prompt, from a batch file, or from a Query Editor window in SSMS. When run interactively, the `sqlcmd` program name is entered at the command prompt with the required options to connect to the database server. When the connection is established, a numbered row is made available to enter the T-SQL commands. Multiple rows of T-SQL can be entered in a batch; they are executed only after the `GO` command has been entered. Figure 5.1 shows an example with two simple `SELECT` statements that were executed interactively with `sqlcmd`. The connection in this example was established by typing `sqlcmd` at the command prompt to establish a trusted connection to the default instance of SQL Server running on the machine on which the command prompt window is opened.

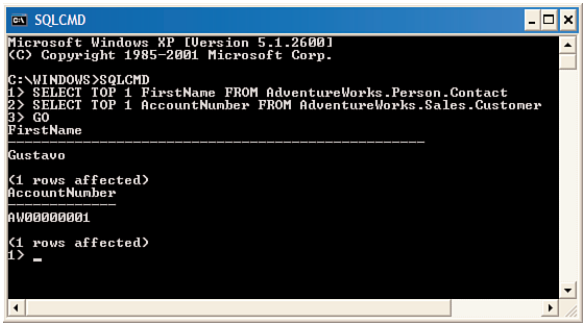

FIGURE 5.1 Executing **`sqlcmd`** interactively.

The capability to edit and execute `sqlcmd` scripts was added to SSMS with SQL Server 2005. A `sqlcmd` script can be opened or created in a Query Editor window within SSMS. To edit these scripts, you must place the editor in SQLCMD Mode. You do so by selecting Query, SQLCMD Mode or by clicking the related toolbar button. When the editor is put in SQLCMD Mode, it provides color coding and the capability to parse and execute the

commands within the script. Figure 5.2 shows a sample `sqlcmd` script opened in SSMS in a Query Editor window set to SQLCMD Mode. The shaded lines are `sqlcmd` commands.

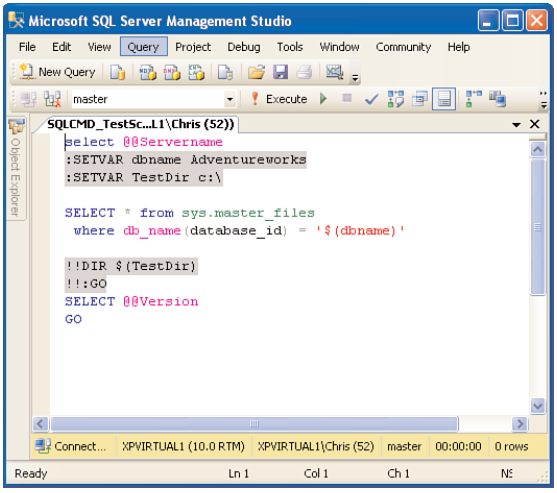

FIGURE 5.2 Executing and editing **`sqlcmd`** scripts in SSMS.

The most common means for executing `sqlcmd` utility is via a batch file. This method can provide a great deal of automation because it allows you to execute a script or many scripts by launching a single file. The examples shown in this section are geared toward the execution of `sqlcmd` in this manner. The following simple example illustrates the execution of `sqlcmd`, using a trusted connection to connect to the local database, and the execution of a simple query that is set using the `-Q` option:

```
sqlcmd -S (local) -E -Q"select getdate()"
```

You can expand this example by adding an output file to store the results of the query and add the `-e` option, which echoes the query that was run in the output results:

```
sqlcmd -S (local) -E -Q"select getdate()" -o c:\TestOutput.txt -e
```

The contents of the `c:\TestOutput.txt` file should look similar to this:

```
select getdate()
-----------------------
2008-09-10 20:29:05.645
 (1 rows affected)
```

Using a trusted connection is not the only way to use `sqlcmd` to connect to a SQL Server instance. You can use the `-U` and `-P` command-line options to specify the SQL Server user and password. `sqlcmd` also provides an option to specify the password in an environmental variable named `sqlcmdPASSWORD`, which can be assigned prior to the `sqlcmd` execution and eliminates the need to hard-code the password in a batch file.

sqlcmd also provides a means for establishing a dedicated administrator connection (DAC) to the server. The DAC is typically used for troubleshooting on a server that is having problems. It allows an administrator to get onto the server when others may not be able to. If the DAC is enabled on the server, a connection can be established with the -A option and a query can be run, as shown in the following example:

```
sqlcmd -S (local) -A -Q"select getdate()"
```

If you need to manage more complex T-SQL execution, it is typically easier to store the T-SQL in a separate input file. The input file can then be referenced as a sqlcmd parameter. For example, say that you have the following T-SQL stored in a file named C:\TestsqlcmdInput.sql:

```
BACKUP DATABASE Master
 TO DISK = 'c:\master.bak'

BACKUP DATABASE Model
 TO DISK = 'c:\model.bak'

BACKUP DATABASE MSDB
 TO DISK = 'c:\msdb.bak'
```

The sqlcmd execution, which accepts the C:\TestsqlcmdInput.sql file as input and executes the commands within the file, looks like this:

```
sqlcmd -S (local) -E -i"C:\TestsqlcmdInput.sql" -o c:\TestOutput.txt -e
```

The execution of the preceding example backs up three of the system databases and writes the results to the output file specified.

Using Scripting Variables with sqlcmd

sqlcmd provides a means for utilizing variables within sqlcmd input files or scripts. These scripting variables can be assigned as sqlcmd parameters or set within the sqlcmd script. To illustrate the use of scripting variables, let's change our previous backup example so that the database that will be backed up is a variable. A new input file named c:\BackupDatabase.sql should be created, and it should contain the following command:

```
BACKUP DATABASE $(DatabaseToBackup)
 TO DISK = 'c:\$(DatabaseToBackup).bak'
```

The variable in the preceding example is named DatabaseToBackup. Scripting variables are referenced using the $( ) designators. These variables are resolved at the time of execution, and a simple replacement is performed. This allows variables to be specified within quotation marks, if necessary. The -v option is used to assign a value to a variable at the command prompt, as shown in the following example, which backs up the model database:

```
sqlcmd -S (local) -E -i"C:\BackupDatabase.sql" -v DatabaseToBackup = model
```

If multiple variables exist in the script, they can all be assigned after the –v parameter. These variables should not be separated by a delimiter, such as a comma or semicolon. Scripting variables can also be assigned within the script, using the `:SETVAR` command. The input file from the previous backup would be modified as follows to assign the `DatabaseToBackup` variable within the script:

```
:SETVAR DatabaseToBackup Model
BACKUP DATABASE $(DatabaseToBackup)
 TO DISK = 'c:\$(DatabaseToBackup).bak'
```

Scripts that utilize variables, `sqlcmd` commands, and the many available options can be very sophisticated and can make your administrative life easier. The examples in this section illustrate some of the basic features of `sqlcmd`, including some of the features that go beyond what is available with `osql`.

The `dta` Command-Line Utility

`dta` is the command-line version of the graphical Database Engine Tuning Advisor. Both the command-line utility and graphical tool provide performance recommendations based on the workload provided to them. The syntax for `dta` is as follows:

```
       Dta [ -? ] |
[
     [ -S server_name[ \instance ] ]
     {
         { -U login_id [-P password ] }
         | –E             }
         { -D database_name [ ,...n ] }
             [-d database_name ]
             [ -Tl table_list | -Tf table_list_file ]
         { -if workload_file | -it workload_trace_table_name }
         { -s session_name | -ID session_ID }
             [ -F ]
                     [ -of output_script_file_name ]
                     [ -or output_xml_report_file_name ]
                     [ -ox output_XML_file_name ]
                     [ -rl analysis_report_list [ ,...n ] ]
                     [ -ix input_XML_file_name ]
                     [ -A time_for_tuning_in_minutes ]
                     [ -n number_of_events ]
             [ -m minimum_improvement ]
                     [ -fa physical_design_structures_to_add ]
                     [ -fp partitioning_strategy ]
                     [ -fk keep_existing_option ]
```

```
                [ -fx drop_only_mode ]
        [ -B storage_size ]
        [ -c max_key_columns_in_index ]
        [ -C max_columns_in_index ]
                [ -e | -e tuning_log_name ]
                [ -N online_option]
                [ -q ]
                [ -u ]
            [ -x ]
            [ -a ]
]
```

An extensive number of options is available with this utility, but many of them are not required to do basic analysis. At a minimum, you need to use options that provide connection information to the database, a workload to tune, a tuning session identifier, and the location to store the tuning recommendations. The connection options include `-S` for the server name, `-D` for the database, and either `-E` for a trusted connection or `-U` and `-P`, which can be used to specify the user and password.

The workload to tune is either a workload file or workload table. The `-if` option is used to specify the workload file location, and the `-it` option is used to specify a workload table. The workload file must be a Profiler trace file (`.trc`), SQL script (`.sql`) that contains T-SQL commands, or SQL Server trace file (`.log`). The workload table is a table that contains output from a workload trace. The table is specified in the form *`database_name.owner_name.table_name`*.

The tuning session must be identified with either a session name or session ID. The session name is character based and is specified with the `-s` option. If the session name is not provided, a session ID must be provided instead. The session ID is numeric and is set using the `-ID` option. If the session name is specified instead of the session ID, the `dta` generates an ID anyway.

The last options required for a basic `dta` execution identify the destination to store the `dta` performance recommendations, which can be stored in a script file or in XML. The `-of` option is used to specify the output script filename. XML output is generated when the `-or` or `-ox` option is used. The `-or` option generates a filename if one is not specified, and the `-ox` option requires a filename. The `-F` option can be used with any of the output options to force an overwrite of a file with the same name, if one exists.

To illustrate the use of `dta` with basic options, let's look at an example of tuning a simple `SELECT` statement against the `AdventureWorks2008R2` database. To begin, you use the following T-SQL, which is stored in a workload file named `c:\myScript.sql`:

```
USE AdventureWorks2008R2 ;
GO
select *
 from Production.transactionHistory
 where TransactionDate = '9/1/04'
```

The following example shows the basic `dta` execution options that can be used to acquire performance recommendations:

```
dta -S xpvirtual1 -E -D AdventureWorks2008R2 -if c:\MyScript.sql
-s MySessionX -of C:\MySessionOutputScript.sql -F
```

NOTE

`dta` and other utilities executed at the command prompt are executed with all the options on a single line. The preceding example and any others in this chapter that are displayed on more than one line should actually be executed at the command prompt or in a batch file on a single line. They are broken here because the printed page can accommodate only a fixed number of characters.

The preceding example utilizes a trusted connection against the `AdventureWorks2008R2` database, a workload file named `c:\MyScript.sql`, and a session named `MySessionX`, and it outputs the performance recommendations to a text file named `c:\MySessionOutputScript.sql`. The `-F` option is used to force a replacement of the output file if it already exists. The output file contains the following performance recommendations:

```
se [AdventureWorks2008R2]
go

CREATE NONCLUSTERED INDEX [_dta_index_TransactionHistory_5]
 ON [Production].[TransactionHistory]
(
    [TransactionDate] ASC
)
INCLUDE ( [TransactionID],
[ProductID],
[ReferenceOrderID],
[ReferenceOrderLineID],
[TransactionType],
[Quantity],
[ActualCost],
[ModifiedDate])
 WITH (SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
  DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go
```

In short, the `dta` output recommends that a new index be created on the `TransactionDate` column in the `TransactionHistory` table. This is a viable recommendation, considering that there was no index on the `TransactionHistory.TransactionDate` column, and it was used as a search argument in the workload file.

Many other options (that go beyond basic execution) can be used to manipulate the way `dta` makes recommendations. For example, a list can be provided to limit which tables the `dta` looks at during the tuning process. Options can be set to limit the amount of time that the `dta` tunes or the number of events. These options go beyond the scope of this chapter, but you can gain further insight into them by looking at the graphical DTA, which contains many of the same types of options. You can refine your tuning options in the DTA, export the options to an XML file, and use the `-ix` option with the `dta` utility to import the XML options and run the analysis.

The `tablediff` Command-Line Utility

The `tablediff` utility enables you to compare the contents of two tables. It was originally developed for replication scenarios to help troubleshoot nonconvergence, but it is also very useful in other scenarios. When data in two tables should be the same or similar, this tool can help determine whether they are the same, and if they are different, it can identify what data in the tables is different.

The syntax for `tablediff` is as follows:

```
tablediff
[ -? ] |
{
        -sourceserver source_server_name[\instance_name]
        -sourcedatabase source_database
        -sourcetable source_table_name
    [ -sourceschema source_schema_name ]
    [ -sourcepassword source_password ]
    [ -sourceuser source_login ]
    [ -sourcelocked ]
        -destinationserver destination_server_name[\instance_name]
        -destinationdatabase subscription_database
        -destinationtable destination_table
    [ -destinationschema destination_schema_name ]
    [ -destinationpassword destination_password ]
    [ -destinationuser destination_login ]
    [ -destinationlocked ]
    [ -b large_object_bytes ]
    [ -bf number_of_statements ]
    [ -c ]
    [ -dt ]
    [ -et table_name ]
    [ -f [ file_name ] ]
    [ -o output_file_name ]
    [ -q ]
    [ -rc number_of_retries ]
    [ -ri retry_interval ]
```

```
    [ -strict ]
    [ -t connection_timeouts ]
}
```

The `tablediff` syntax requires source and destination connection information to perform a comparison. This information includes the servers, databases, and tables that will be compared. Connection information must be provided for SQL Server authentication but can be left out if Windows authentication can be used. The source and destination parameters can be for two different servers or the same server, and the `tablediff` utility can be run on a machine that is neither the source nor the destination.

To illustrate the usefulness of this tool, let's look at a sample comparison in the `AdventureWorks2008R2` database. The simplest way to create some data for comparison is to select the contents of one table into another and then update some of the rows in one of the tables. The following `SELECT` statement makes a copy of the `AddressType` table in the `AdventureWorks2008R2` database to the `AddressTypeCopy` table:

```
select *
 into Person.AddressTypeCopy
 from Person.AddressType
```

In addition, the following statement updates two rows in the `AddressTypeCopy` table so that you can use the `tablediff` utility to identify the changes:

```
UPDATE Person.AddressTypeCopy
 SET Name = 'Billing New'
 WHERE AddressTypeId = 1

UPDATE Person.AddressTypeCopy
 SET Name = 'Shipping New',
  ModifiedDate = '20090918'
 WHERE AddressTypeId = 5
```

The `tablediff` utility can be executed with the following parameters to identify the differences in the `AddressType` and `AddressTypeCopy` tables:

```
tablediff -sourceserver "(local)" -sourcedatabase "AdventureWorks2008R2"
-sourceschema "Person"-sourcetable "AddressType"
-destinationserver "(local)" -destinationdatabase "AdventureWorks2008R2"
-destinationschema "Person" -destinationtable "AddressTypeCopy"
-f c:\TableDiff_Output.txt
```

The destination and source parameters are the same as in the previous example, except for the table parameters, which have the source `AddressType` and the destination `AddressTypeCopy`. The execution of the utility with these parameters results in the following output to the command prompt window:

```
User-specified agent parameter values:
-sourceserver (local)
-sourcedatabase AdventureWorks2008R2
-sourceschema Person
-sourcetable AddressType
-destinationserver (local)
-destinationdatabase AdventureWorks2008R2
-destinationschema Person
-destinationtable AddressTypeCopy
-f c:\TableDiff_Output

Table [AdventureWorks2008R2].[Person].[AddressType] on (local)
and Table [AdventureWorks2008R2].[Person].[AddressTypeCopy] on (local)
have 2 differences.
Fix SQL written to c:\TableDiff_Output.sql.
Err     AddressTypeID   Col
Mismatch        1       Name
Mismatch        5       ModifiedDate Name
The requested operation took 0.296875 seconds.
```

The output first displays a summary of the parameters used and then shows the comparison results. In this example, it found the two differences that are due to updates performed on `AddressTypeCopy`. In addition, the `-f` parameter used in the example caused the `tablediff` utility to output a SQL file that can be used to fix the differences in the destination table. The output file from this example looks as follows:

```
— Host: (local)
— Database: [AdventureWorks2008R2]
— Table: [Person].[AddressTypeCopy]
SET IDENTITY_INSERT [Person].[AddressTypeCopy] ON
UPDATE [Person].[AddressTypeCopy]
 SET [Name]='Billing'
 WHERE [AddressTypeID] = 1
UPDATE [Person].[AddressTypeCopy]
 SET [ModifiedDate]='2002-06-01 00:00:00.000',
 [Name]='Shipping' WHERE [AddressTypeID] = 5
SET IDENTITY_INSERT [Person].[AddressTypeCopy] OFF
```

NOTE

The `tablediff` utility requires the source table to have at least one primary key, identity, or `ROWGUID` column. This gives the utility a key that it can use to try to match a corresponding row in the destination table. If the `-strict` option is used, the destination table must also have a primary key, identity, or `ROWGUID` column.

Keep in mind that several different types of comparisons can be done with the `tablediff` utility. The `-q` option causes a quick comparison that compares only record counts and looks for differences in the schema. The `-strict` option forces the schemas of each table to be the same when the comparison is run. If this option is not used, the utility allows some columns to be of different data types, as long as they meet the mapping requirements for the data type (for example, `INT` can be compared to `BIGINT`).

The `tablediff` utility can be used for many different types of comparisons. How you use this tool depends on several factors, including the amount and type of data you are comparing.

The bcp Command-Line Utility

You use the `bcp` (bulk copy program) tool to address the bulk movement of data. This utility is bidirectional, allowing for the movement of data into and out of a SQL Server database.

`bcp` uses the following syntax:

```
bcp {[[database_name.][owner].]{table_name | view_name} | "query"}
    {in | out | queryout | format} data_file
    [-mmax_errors] [-fformat_file] [-x] [-eerr_file]
    [-Ffirst_row] [-Llast_row] [-bbatch_size]
    [-n] [-c] [-N] [-w] [-V (60 | 65 | 70 | 80)] [-6]
    [-q] [-C { ACP | OEM | RAW | code_page } ] [-tfield_term]
    [-rrow_term] [-iinput_file] [-ooutput_file] [-apacket_size]
    [-Sserver_name[\instance_name]] [-Ulogin_id] [-Ppassword]
    [-T] [-v] [-R] [-k] [-E] [-h"hint [,...n]"]
```

Some of the commonly used options—other than the ones used to specify the database, such as user ID, password, and so on—are the `-F` and `-L` options. These options allow you to specify the first and last row of data to be loaded from a file, which is especially helpful in large batches. The `-t` option allows you to specify the field terminator that separates data elements in an ASCII file. The `-E` option allows you to import data into SQL Server fields that are defined with identity properties.

TIP

The `BULK INSERT` T-SQL statement and SSIS are good alternatives to `bcp`. The `BULK INSERT` statement is limited to loading data into SQL Server, but it is an extremely fast tool for loading data. SSIS is a sophisticated GUI that allows for both data import and data export, and it has capabilities that go well beyond those that were available in SQL Server 2000's Data Transformation Services (DTS).

This section barely scratches the surface when it comes to the capabilities of `bcp`. For a more detailed look at `bcp`, refer to the section, "Using `bcp`" in Chapter 52, "SQL Server Integration Services."

The `sqldiag` Command-Line Utility

`sqldiag` is a diagnostic tool that you can use to gather diagnostic information about various SQL Server services. It is intended for use by Microsoft support engineers, but you might also find the information it gathers useful in troubleshooting a problem. `sqldiag` collects the information into files that are written, by default, to a folder named `SQLDIAG`, which is created where the file `sqldiag.exe` is located (for example, `C:\Program Files\Microsoft SQL Server\100\Tools\binn\SQLDIAG\`). The folder holds files that contain information about the machine on which SQL Server is running in addition to the following types of diagnostic information:

- SQL Server configuration information
- SQL Server blocking output
- SQL Server Profiler traces
- Windows performance logs
- Windows event logs

The syntax for `sqldiag` changed quite a bit in SQL Server 2005, but very little has changed in SQL Server 2008. Some of the options that were used in versions prior to SQL Server 2005 are not compatible with the current version. The full syntax for `sqldiag` is as follows:

```
sqldiag
     { [/?] }
     |
     { [/I configuration_file]
       [/O output_folder_path]
       [/P support_folder_path]
       [/N output_folder_management_option]
       [/C file_compression_type]
       [/B [+]start_time]
       [/E [+]stop_time]
       [/A SQLdiag_application_name]
       [/T { tcp [ ,port ] | np | lpc | via } ]
       [/Q] [/G] [/R] [/U] [/L] [/X] }
     |
     { [START | STOP | STOP_ABORT] }
     |
     { [START | STOP | STOP_ABORT] /A SQLdiag_application_name }
```

NOTE

Keep in mind that many of the options for `sqldiag` identify how and when the `sqldiag` utility will be run. The utility can be run as a service, scheduled to start and stop at a specific time of day, and it can be configured to change the way the output is generated. The details about these options are beyond the scope of this chapter but are covered in detail in SQL Server Books Online. This section is intended to give you a taste of the useful information that this utility can capture.

By default, the `sqldiag` utility must be run by a member of the Windows Administrators group, and this user must also be a member of the sysadmin fixed SQL Server role. To get a flavor for the type of information that `sqldiag` outputs, open a command prompt window, change the directory to the location of the `sqldiag.exe` file, and type the following command:

```
sqldiag
```

No parameters are needed to generate the output. The command prompt window scrolls status information across the screen as it collects the diagnostic information. You see the message "SQLDIAG Initialization starting..." followed by messages that indicate what information is being collected. The data collection includes a myriad of system information from `MSINFO32`, default traces, and SQLDumper log files. When you are ready to stop the collection, you can press Ctrl+C.

If you navigate to the `sqldiag` output folder, you find the files created during the collection process. In this output folder, you should find a file with `MSINFO32` in its name. This file contains the same type of information that you see when you launch the System Information application from Accessories or when you run `MSINFO32.EXE`. This is key information about the machine on which SQL Server is running. This information includes the number of processors, the amount of memory, the amount of disk space, and a slew of other hardware and software data.

You also find a file named *`xxx_sp_sqldiag_Shutdown.out`*, where *`xxx`* is the name of the SQL Server machine. This file contains SQL Server–specific information, including the SQL Server error logs, output from several key system stored procedures, including *`sp_helpdb`* and `sp_configure`, and much more information related to the current state of SQL Server.

You find other files in the `sqldiag` output directory as well. Default trace files, log files related to the latest `sqldiag` execution, and a copy of the XML file containing configuration information are among them. Microsoft documentation on these files is limited, and you may find that the best way to determine what they contain is simply to open the files and review the wealth of information therein.

The `sqlservr` Command-Line Utility

The `sqlservr` executable is the program that runs when SQL Server is started. You can use the `sqlservr` executable to start SQL Server from a command prompt. When you do that, all the startup messages are displayed at the command prompt, and the command prompt session becomes dedicated to the execution of SQL Server.

CAUTION

If you start SQL Server from a command prompt, you cannot stop or pause it by using SSMS, Configuration Manager, or the Services applet in the Control Panel. You should stop the application only from the command prompt window in which SQL Server is running. If you press Ctrl+C, you are asked whether you want to shut down SQL Server. If you close the command prompt window in which SQL Server is running, SQL Server is automatically shut down.

The syntax for the `sqlserver` utility is as follows:

```
sqlservr [-sinstance_name] [-c] [-dmaster_path] [-f]
     [-eerror_log_path] [-lmaster_log_path] [-m]
     [-n] [-Ttrace#] [-v] [-x] [-gnumber] [-h]
```

Most commonly, you start SQL Server from the command prompt if you need to troubleshoot a configuration problem. The `-f` option starts SQL Server in minimal configuration mode. This allows you to recover from a change to a configuration setting that prevents SQL Server from starting. You can also use the `-m` option when you need to start SQL Server in single-user mode, such as when you need to rebuild one of the system databases.

SQL Server functions when started from the command prompt in much the same way as it does when it is started as a service. Users can connect to the server, and you can connect to the server by using SSMS. What is different is that the SQL Server instance running in the command prompt appears as if it is not running in some of the tools. SSMS and SQL Server Service Manager show SQL Server as being stopped because they are polling the SQL Server service, which is stopped when running in the command prompt mode.

TIP

If you simply want to start the SQL Server service from the command prompt, you can use the `NET START` and `NET STOP` commands. These commands are not SQL Server specific but are handy when you want to start or stop SQL Server, especially in a batch file. The SQL Server service name must be referenced after these commands. For example, `NET START MSSQLSERVER` starts the default SQL Server instance.

Summary

SQL Server provides a set of command-line utilities that allow you to execute some of the available SQL Server programs from the command prompt. Much of the functionality housed in these utilities is also available in graphical tools, such as SSMS. However, the capability to initiate these programs from the command prompt is invaluable in certain scenarios.

Chapter 6, "SQL Server Profiler," covers a tool that is critical for performance tuning in SQL Server 2008. SQL Server Profiler provides insight by monitoring and capturing the activity occurring on a SQL Server instance. It is a "go-to" tool for many DBAs and developers because of the wide variety of information that it can capture.