Chapter 17

Automating System Administration Tasks

In This Chapter

- Starting SQL Server Agent
- Creating Jobs and Operators
- Alerts
One of the most important advantages of the Database Engine in relation to other relational DBMSs is its capability to automate administrative tasks and hence to reduce costs. The following are examples of some important tasks that are performed frequently and therefore could be automated:

- Backing up the database and transaction log
- Transferring data
- Dropping and re-creating indices
- Checking data integrity

You can automate all these tasks so that they occur on a regular schedule. For example, you can set the database backup task to occur every Friday at 8:00 p.m. and the transaction log backup task to occur daily at 10:00 p.m.

The components of the Database Engine that are used in automation processes include the following:

- SQL Server service (MSSQLSERVER)
- Windows Application log
- SQL Server Agent service

Why does the Database Engine need these three components to automate processes? In relation to automation of administration tasks, the MSSQLSERVER service is needed to write events to the Windows Application log. Some events are written automatically, and some must be raised by the system administrator (see the detailed explanation later in this chapter).

The Windows Application log is where all application and system messages of Windows operating systems and messages of their components are written. The role of the Windows Application log in the automation process is to notify SQL Server Agent about existing events.

SQL Server Agent is another service that connects to the Windows Application log and the MSSQLSERVER service. The role of SQL Server Agent in the automation process is to take an action after a notification through the Windows Application log. The action can be performed in connection with the MSSQLSERVER service or some other application. Figure 17-1 shows how these three components work together.
Chapter 17: Automating System Administration Tasks

Starting SQL Server Agent

SQL Server Agent executes jobs and fires alerts. As you will see in the upcoming sections, jobs and alerts are defined separately and can be executed independently. Nevertheless, jobs and alerts may also be complementary processes, because a job can invoke an alert and vice versa.

Consider an example: A job is executed to inform the system administrator about an unexpected filling of the transaction log that exceeds a tolerable limit. When this event occurs, the associated alert is invoked and, as a reaction, the system administrator may be notified by e-mail or pager.

Another critical event is a failure in backing up the transaction log. When this happens, the associated alert may invoke a job that truncates the transaction log. This reaction will be appropriate if the reason for the backup failure is an overflow (filling up) of the transaction log. In other cases (for example, the target device for the backup copy is full), such a truncation will have no effect. This example shows the close connection that may exist between events that have similar symptoms.

SQL Server Agent allows you to automate different administrative tasks. Before you can do this, the process has to be started. To start SQL Server Agent, right-click SQL Server Agent and choose Start.

As already stated, the invocation of an alert can also include the notification of one or more operators by e-mail using Database Mail. Database Mail is an enterprise solution for sending e-mail messages from the Database Engine. Using Database Mail, your applications can send e-mail messages to users. The messages may contain query results, and may also include files from any resource on your network.
Creating Jobs and Operators

Generally, there are three steps to follow if you want to create a job:

1. Create a job and its steps.
2. Create a schedule of the job execution if the job is not to be executed on demand.
3. Notify operators about the status of the job.

The following sections explain these steps using an example.

Creating a Job and Its Steps

A job may contain one or more steps. There are different ways in which a job step can be defined. The following list contains some of them.

- **Using Transact-SQL statements**  Many job steps contain Transact-SQL statements. For example, if you want to automate database or transaction log backups, you use the BACKUP DATABASE statement or BACKUP LOG statement, respectively.

- **Using the operating system (CmdExec)**  Some jobs may require the execution of a SQL Server utility, which usually will be started with the corresponding command. For example, if you want to automate the data transfer from your database server to a data file, or vice versa, you could use the `bcp` utility.

- **Invoking a program**  As another alternative, it may be necessary to execute a program that has been developed using Visual Basic or some other programming language. In this case, you should always include the path drive letter in the Command text box when you start such a program. This is necessary because SQL Server Agent has to find the executable file.

If the job contains several steps, it is important to determine which actions should be taken in case of a failure. Generally, the Database Engine starts the next job step if the previous one was successfully executed. However, if a job step fails, any job steps that follow will not be executed. Therefore, you should always specify how often each step should be retried in the case of failure. And, of course, it will be necessary to eliminate the reason for the abnormal termination of the job step. (Obviously, a repeated job execution will always lead to the same error if the cause is not repaired.)
Chapter 17: Automating System Administration Tasks

**NOTE**

The number of attempts depends on the type and content of the executed job step (batch, command, or application program).

You can create a job using the following:

- SQL Server Management Studio
- System stored procedures (`sp_add_job` and `sp_add_jobstep`)

SQL Server Management Studio is used in this example, which creates a job that backs up the `sample` database. To create this job, connect to an instance of the Database Engine in Object Explorer and then expand that instance. Expand SQL Server Agent, right-click Jobs, and choose New Job. (SQL Server Agent must be running.) The New Job dialog box appears (see Figure 17-2). On the General page, enter a name for the job in the Name box. (The name of the job for backing up the `sample` database will be `backup_sample`.)

For the Owner field, click the ellipsis (…) button and choose the owner responsible for performing the job. In the Category drop-down list, choose the category to which the job belongs. You can add a description of the job in the Description box, if you wish.

**NOTE**

If you have to manage several jobs, categorizing them is recommended. This is especially useful if your jobs are executed in a multiserver environment.

Check the Enabled check box to enable the job.

**NOTE**

All jobs are enabled by default. SQL Server Agent disables jobs if the job schedule is defined either at a specific time that has passed or on a recurring basis with an end date that has also passed. In both cases, you must re-enable the job manually.
Each job must have one or more steps. Therefore, in addition to defining job properties, you must create at least one step before you can save the job. To define one or more steps, click the Steps page in the New Job dialog box and click New. The New Job Step dialog box appears, as shown in Figure 17-3. Enter a name for the job step. (It is called backup in the example.) In the Type drop-down list, choose Transact-SQL script (T-SQL), because the backup of the sample database will be executed using the Transact-SQL statement BACKUP DATABASE.

In the Database drop-down list, choose the master database, because this system database must be the current database if you want to back up a database.
You can either enter the Transact-SQL statement directly in the Command box or invoke it from a file. In the former case, enter the following statements, after you change the path for the backup file:

```sql
EXEC sp_addumpdevice 'disk', 'backup_file1', 'C:\sample_backup'
BACKUP DATABASE sample TO backup_file1
```

As you probably guessed, the `sp_addumpdevice` system procedure adds a backup device to an instance of the Database Engine. To invoke the Transact-SQL statement from a file, click Open and select the file. The syntax of the statement(s) can be checked by clicking Parse.

**Creating a Job Schedule**

Each created job can be executed on demand (that is, manually by the user) or by using one or more schedules. A scheduled job can occur at a specific time or on a recurring schedule.
Each job can have multiple schedules. For example, the backup of the transaction log of a production database can be executed with two different schedules, depending on the time of day. This means that during peak business hours, you can execute the backup more frequently than during non-peak hours.

To create a schedule for an existing job using SQL Server Management Studio, select the Schedules page in the Job Properties dialog box and click New. (The Job Properties dialog box is the same dialog box as shown in Figure 17-2). If the Job Properties dialog box is not active, expand SQL Server Agent, expand Jobs, and click the job you want to process.

If you get the warning, “The On Access action of the last step will be changed from Get Next Step to Quit with Success,” click Yes.

The New Job Schedule dialog box appears (see Figure 17-4).
Chapter 17: Automating System Administration Tasks

For the sample database, set the schedule for the backup to be executed every Friday at 8:00 p.m. To do this, enter the name in the Name dialog box and choose Recurring in the Schedule Type drop-down list. In the Frequency section, choose Weekly in the Occur drop-down list, and check Friday. In the Daily Frequency section, click the Occurs Once At radio button, and enter the time (20:00:00). In the Duration section, choose the start date in the Start Date drop-down list, and then click the End Date radio button and choose the end date in the corresponding drop-down list. (If the job should be scheduled without the end date, click No End Date.)

Notifying Operators About the Job Status

When a job completes, several methods of notification are possible. For example, you can instruct the system to write a corresponding message to the Windows Application log, hoping that the system administrator reads this log from time to time. A better choice is to explicitly notify one or more operators using e-mail, pager, and/or the net send command.

Before an operator can be assigned to a job, you have to create an entry for it. To create an operator using SQL Server Management Studio, expand SQL Server Agent, right-click Operators, and choose New Operator. The New Operator dialog box appears (see Figure 17-5). On the General page, enter the name of the operator in the Name box. Specify one or more methods of notifying the operator (via e-mail, pager, or the net send address). In the Pager on Duty Schedule section, enter the working hours of the operator.

To notify one or more operators after the job finishes (successfully or unsuccessfully), return to the Job Properties dialog box of the job, select the Notifications page (see Figure 17-6), and check the corresponding boxes. (Besides e-mail, pager, or the net send command notification, in this dialog box you also have the option of writing the message to the Windows Application log and/or deleting the job.)

Viewing the Job History Log

The Database Engine stores the information concerning all job activities in the sysjobhistory system table of the msdb system database. Therefore, this table represents the job history log of your system. You can view the information in this table using SQL Server Management Studio. To do this, expand SQL Server Agent, expand Jobs, right-click the job, and choose View History. The Log File Viewer dialog box shows the history log of the job.
Figure 17-5  The New Operator dialog box

Figure 17-6  The Job Properties dialog box, Notifications page
Chapter 17: Automating System Administration Tasks

Each row of the job history log is displayed in the details pane, which contains, among other information, the following:

- Date and time when the job step occurred
- Whether the job step completed successfully or unsuccessfully
- Operators who were notified
- Duration of the job
- Errors or messages concerning the job step

By default, the maximum size of the job history log is 1000 rows, while the number of rows for a particular job is limited to 100. (The job history log is automatically cleared when the maximum size of rows is reached.) If you want to store the information about each job, and your system has several jobs, increase the size of the job history log and/or the number of rows per job. Using SQL Server Management Studio, right-click SQL Server Agent and choose Properties. In the SQL Server Agent Properties dialog box, select the History page and enter the new values for the maximum job history log size and maximum job history rows per job. You can also check Automatically Remove Agent History and specify a time interval after which logs should be deleted.

Alerts

The information about execution of jobs and system error messages is stored in the Windows Application log. SQL Server Agent reads this log and compares the stored messages with the alerts defined for the system. If there is a match, SQL Server Agent fires the alert. Therefore, alerts can be used to respond to potential problems (such as filling up the transaction log), different system errors, or user-defined errors. Before explaining how you create alerts, this section discusses system error messages and two logs, the SQL Server Agent error log and the Windows Application log, which are used to capture all system messages (and thus most of the errors).

Error Messages

System errors are grouped in four different groups. The Database Engine provides extensive information about each error. The information is structured and includes the following:

- A unique error message number
- An additional number between 0 and 25, which represents the error's severity level
Example 17.1 queries a nonexistent table in the sample database.

**EXAMPLE 17.1**

```
USE sample;
SELECT * FROM authors;
```

The result is

```
Msg 208, Level 16, State 1, Line 2
Invalid object name 'authors'.
```

To view the information concerning error messages, use the `sys.messages` catalog view. The three most important columns of this view are `message_id`, `severity`, and `text`. Each unique error number has a corresponding error message. (The error message is stored in the `text` column, and the corresponding error number is stored in the `message_id` column of the `sys.messages` catalog view.) In Example 17.1, the message concerning the nonexistent or incorrectly spelled database object corresponds to error number –208.

The severity level of an error (the `severity` column of the `sys.messages` catalog view) is represented in the form of a number between 0 and 25. The levels between 0 and 10 are simply informational messages, where nothing needs to be fixed. All levels from 11 through 16 indicate different program errors and can be resolved by the user. The values 17 and 18 indicate software and hardware errors that generally do not terminate the running process. All errors with a severity level of 19 and greater are fatal system errors. The connection of the program generating such an error is closed, and its process will then be removed.

The messages relating to program errors (that is, the levels between 11 and 16) are shown on the screen only. All system errors (errors with a severity level of 19 or greater) will also be written to the log.

In order to resolve an error, you usually need to read the detailed description of the corresponding error. You can also find detailed error descriptions in Books Online.
System error messages are written to the SQL Server Agent error log and to the Windows Application log. The following two sections describe these two components.

**SQL Server Agent Error Log**

SQL Server Agent creates an error log that records warnings and errors by default. The following warnings and errors are displayed in the log:

- Warning messages that provide information about potential problems
- Error messages that usually require intervention by a system administrator

The system maintains up to ten SQL Server Agent error logs. The current log is called **Current**, while all other logs have an extension that indicates the relative age of the log. For example, **Archive #1** indicates the newest archived error log.

The SQL Server Agent error log is an important source of information for the system administrator. With it, he or she can trace the progress of the system and determine which corrective actions to take.

To view the SQL Server Agent error logs from SQL Server Management Studio, expand the instance in Object Explorer, expand SQL Server Agent, and expand Error Logs. Click one of the files to view the desired log. The log details appear in the details pane of the Log File Viewer dialog box.

**Windows Application Log**

The Database Engine also writes system messages to the Windows Application log. The Windows Application log is the location of all operating system messages for the Windows operating systems, and it is where all application messages are stored. You can view the Windows Application log using the Event Viewer.

Viewing errors in the Windows Application log has some advantages compared to viewing them in the SQL Server Agent error log. The most important is that the Windows Application log provides an additional component for the search for desired strings.

To view information stored in the Windows Application log, choose Start | Control Panel | Administrative Tools | Event Viewer. In the Event Viewer window, you can choose between system, security, and application messages. For SQL Server system messages, click Application. SQL Server events are identified by the entry MSSQLSERVER.
Defining Alerts to Handle Errors

An alert can be defined to raise a response to a particular error number or to the group of errors that belongs to a specific severity code. Furthermore, the definition of an alert for a particular error is different for system errors and user-defined errors. (The creation of alerts on user-defined errors is described later in this chapter.)

The rest of this section shows how you can create alerts using SQL Server Management Studio.

Creating Alerts on System Errors

Example 13.5, in which one transaction was deadlocked by another transaction, will be used to show how to create an alert about a system error number. If a transaction is deadlocked by another transaction, the victim must be executed again. This can be done, among other ways, by using an alert.

To create the deadlock (or any other) alert, expand SQL Server Agent, right-click Alerts, and choose New Alert. In the New Alert dialog box (see Figure 17-7), enter the name of the alert in the Name box, choose SQL Server Event Alert in the Type drop-down list, and choose <all databases> from the Database Name drop-down list. Click the Error Number radio button, and enter 1205. (This error number indicates a deadlock problem, where the current process was selected as the “victim.”)

The second step defines the response for the alert. In the same dialog box, click the Response page (see Figure 17-8). First check Execute Job, and then choose the job to execute when the alert occurs. (The example here defines a new job called deadlock_all_db that restarts the victim transaction.) Check Notify Operators, and then, in the Operator List pane, select operators and choose the methods of their notifications (e-mail, pager, and/or the net send command).

NOTE

In the preceding example, it is assumed that the victim process will be terminated. Actually, after receiving the deadlock error 1205, the program resubmits the failed transaction on its own.

Creating Alerts on Error Severity Levels

You can also define an alert that will raise a response on error severity levels. As you already know, each system error has a corresponding severity level that is a number between 0 and 25. The higher the severity level is, the more serious the error. Errors with severity levels 20 through 25 are fatal errors. Errors with severity levels 19 through 25 are written to the Windows Application log.
Note

Always define an operator to be notified when a fatal error occurs.

As an example of how you can create alerts in relation to severity levels, here's how you use SQL Server Management Studio to create the particular alert for severity level 25. First, expand SQL Server Agent, right-click Alerts, and choose New Alert. In the Name box, enter a name for this alert (for example, **Severity 25 errors**). In the Type drop-down
list, choose SQL Server event alert. In the Database Name drop-down list, choose the sample database. Click the Severity radio button and choose 025 – Fatal Error.

On the Response page, enter one or more operators to be notified via e-mail, pager, and/or the net send command when an error of severity level 25 occurs.

**Creating Alerts on User-Defined Errors**

In addition to creating alerts on system errors, you can create alerts on customized error messages for individual database applications. Using such messages (and alerts), you can define solutions to problems that might occur in an application.

The following steps are necessary if you want to create an alert on a user-defined message:

1. Create the error message.
2. Raise the error from a database application.
3. Define an alert on the error message.
Chapter 17: Automating System Administration Tasks

An example is the best way to illustrate the creation of such an alert: the alert is fired if the shipping date of a product is earlier than the order date. (For the definition of the sales table, see Chapter 5.)

NOTE

Only the first two steps are described here, because an alert on a user-defined message is defined similarly to an alert on a system error message.

Creating an Error Message

To create a user-defined error message, you can use either SQL Server Management Studio or the sp_addmessage stored procedure. Example 17.2 creates the error message for the example using the sp_addmessage stored procedure.

**EXAMPLE 17.2**

```sql
sp_addmessage @msgnum=50010, @severity=16,
@msgtext='The shipping date of a product is earlier than the order date',
@lang='us_english', @with_log='true'
```

The `sp_addmessage` stored procedure in Example 17.2 creates a user-defined error message with error number 50010 (the `@msgnum` parameter) and severity level 16 (the `@severity` parameter). All user-defined error messages are stored in the `sysmessages` system table of the `master` database and can be viewed by using the `sys.messages` catalog view. The error number Example 17.2 is 50010 because all user-defined errors must be greater than 50000. (All error message numbers less than 50000 are reserved for the system.)

For each user-defined error message, you can optionally use the `@lang` parameter to specify the language in which the message is displayed. This specification may be necessary if multiple languages are installed on your computer. (When the `@lang` parameter is omitted, the session language is the default language.)

By default, user-defined messages are not written to the Windows Application log. On the other hand, you must write the message to this log if you want to raise an alert on it. If you set the `@with_log` parameter of the `sp_addmessage` system procedure to `TRUE`, the message will be written to the log.

Raising an Error Using Triggers

To raise an error from a database application, you invoke the RAISERROR statement. This statement returns a user-defined error message and sets a system flag in the `@@error` global variable. (You can also handle error messages using TRY/CATCH blocks.)
Example 17.3 creates the trigger **t_date_comp**, which returns a user-defined error of 50010 if the shipping date of a product is earlier than the order date.

**NOTE**
To execute Example 17.3, the table **sales** must exist (see Example 5.21).

**EXAMPLE 17.3**

USE sample;
GO
CREATE TRIGGER t_date_comp
  ON sales
  FOR INSERT AS
  DECLARE @order_date DATE
  DECLARE @shipped_date DATE
  SELECT @order_date=order_date, @shipped_date=ship_date FROM INSERTED
  IF @order_date > @shipped_date
    RAISERROR (50010, 16, -1)

Now, if you insert the following row in the **sales** table, the shipping date of a product is earlier than the order date:

```
INSERT INTO sales VALUES (1, '01.01.2007', '01.01.2006')
```

the system will return the user-defined error message:

```
Msg 50010, Level 16, State 1, Procedure t_date_comp, Line 8
```

**Summary**

The Database Engine allows you to automate and streamline many administrator tasks, such as database backups, data transfers, and index maintenance. For the execution of such tasks, SQL Server Agent must be running.

To automate a task, you have to execute several steps:

- Create a job
- Create operators
- Create alerts
Job and task are synonymous, so when you create a job, you create the particular task that you want to automate. The easiest way to create a job is to use SQL Server Management Studio, which allows you to define one or more job steps and create an execution schedule.

When a job (successfully or unsuccessfully) completes, you can notify one or more persons, using operators. Again, the general way to create an operator is to use SQL Server Management Studio.

Alerts are defined separately and can also be executed independently of jobs. An alert can handle individual system errors, user-defined errors, or groups of errors belonging to one of 25 severity levels.

The next chapter discusses data replication.

Exercises

E.17.1
Name several administrative tasks that could be automated.

E.17.2
You want to back up the transaction log of your database every hour during peak business hours and every four hours during nonpeak hours. What should you do?

E.17.3
You want to test performance of your production database in relation to locks and want to know whether the lock wait time is more than 30 seconds. How could you be notified automatically when this event occurs?

E.17.4
Specify all parts of a SQL Server error message.

E.17.5
Which are the most important columns of the sys.messages catalog view concerning errors?