

How to Automatically Monitor Windows Event Log from SQL Server

Written By: Sadequul Hussain -- 1/15/2010

Problem

When troubleshooting server related issues like unexpected crashes or service restarts, SQL Server DBAs often browse through the Windows Event log looking for clues. This is a reactive approach where the DBA is investigating after an incident has been reported. The more proactive approach would be to regularly check the log for any warnings or errors. Many production DBAs are often responsible for large numbers of instances in the organization and don't have the time to go through each individual event log of every server. In this tip I will show an automated way of monitoring the application event log of SQL Server boxes. The system generates a summarized report of SQL Server errors and warnings from the target machines' event log and sends it to the DBA via e-mail. The DBA can then browse through this report to see which server needs to have an investigation.

Solution

We will discuss the event log monitoring system under a number of sections

System Architecture

At the center of the event log monitor is a simple T-SQL script file. This file will be stored in each SQL Server machine's local drive. We have kept this file under the C:\EventLogReport folder for simplicity and demonstration purpose. The script is executed by a batch file, also located in the same folder. The batch file in turn will be called by a Windows scheduled task every morning (or any time you prefer).

In our example, we have called this scheduled task "Event Log Report" (see below). It calls the EventLogReport.cmd command file every morning at 7:00AM.

Event Log Report [?] [X]

Task | Schedule | Settings | Security

C:\WINDOWS\Tasks\Event Log Report.job

Run: C:\EventLogReport\EventLogReport.cmd Browse...

Start in: C:\Windows\system32

Comments:

Run as: MYSERVER\Administrator Set password...

☐ Run only if logged on

☒ Enabled (scheduled task runs at specified time)

OK Cancel Apply

Event Log Report [?] [X]

Task | Schedule | Settings | Security

At 7:00 AM every day, starting 17/11/2009

Schedule Task: Start time: Advanced...

Daily 7:00 AM

Schedule Task Daily

Every 1 day(s)

☐ Show multiple schedules.

OK Cancel Apply

Now instead of having a Windows task calling a script file from the file system, you may want to create a stored procedure and call that procedure from a SQL Server job. We have not taken this approach because the stored procedure will have to reside in a database and unless there is a specific DBA / monitoring type of database in each of your target SQL Servers, you should not create it in either the master or any user database.

When the SQL script file executes, it looks at the application event log of the local server. The code only <http://www.mssqltips.com/tipprint.asp?t...>

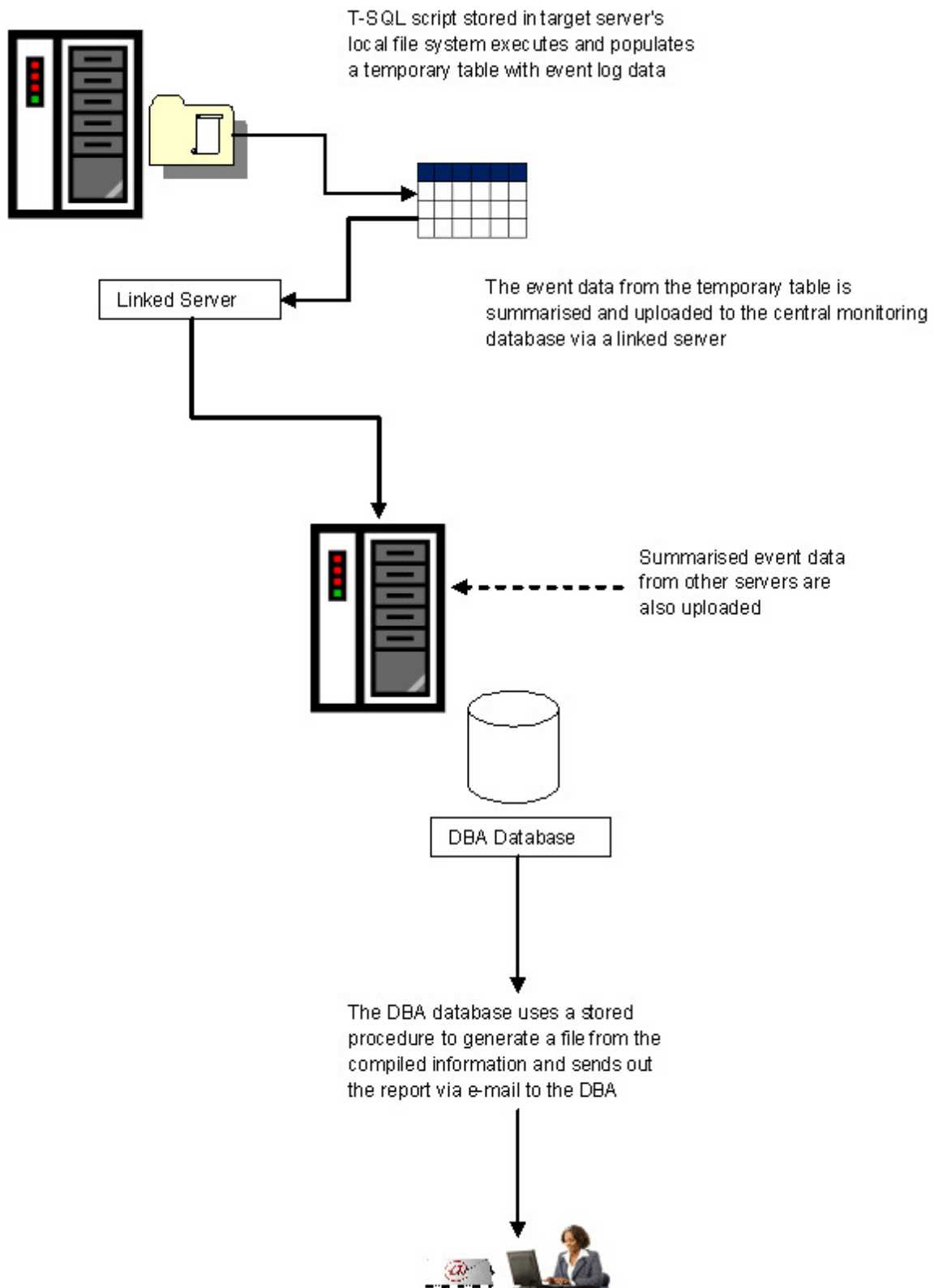
looks at the events of the last twenty-four hours. If any error or warning messages are logged for either the SQL or the Agent service, those entries are copied to a temporary table. A summary level message is then generated from these records. Finally the summary message is uploaded to a table in a central database.

The database where the summary message is uploaded can be hosted in a special "monitoring" instance. I try to call this type of monitoring database the "DBA" database. Its sole purpose will be to act as a centralized repository of information for all the SQL Servers in the network. It can also have a number of procedures, functions and views for monitoring different SQL Servers running in the enterprise.

If you think about this scenario, each SQL Server machine will execute a Windows task during a particular time of the day. Each task will add a record in a centralized log table. The script executed by the task will run in the context of a local SQL Server instance running on the machine and it will access the remote table via a linked server.

The monitoring SQL Server instance will have a scheduled job that will send the contents of this log table to the database administrator via e-mail. This SQL job will have to run well after all the Windows tasks have finished updating the central table. This is because the SQL job will have no way of knowing when all the target machines have completed posting their summarized data. Usually the Windows task takes less than a minute to finish and it would be safe to schedule the SQL job to run after an hour. So if your target machines are gathering event log reports at 7:00 AM in the morning, you can schedule the SQL Server job to run at 8:00 AM.

The following diagram shows how the system works.



A word about eventquery.vbs

The T-SQL script makes use of a VBScript program called **eventquery.vbs** to extract information from the event log. This VBScript file is a system supplied component and by default is located under the **<system_root_drive>:\Windows\system32** folder of a Windows Server 2003 system. You can run eventquery.vbs from the command prompt and specify one or more parameters as filter criteria. The script is a simple ASCII text file: you can open it in Notepad and see its inner workings.

The command syntax for eventquery.vbs with some of its important parameters is:

eventquery.vbs
 [/s server-name

```

[/u username]
[/p password]
[/fi Filter Condition]
[/fo TABLE | LIST | CSV]
[/nh]
[/I [APPLICATION] [SYSTEM] [SECURITY]]

```

Of these, /s , /u and /p switches are self explanatory. You do not need to provide a username and password if you are executing the code in the local server. The /fo parameter specifies the output format: you can view the event log in tabular fashion with table and column headers; you can view it as a list and also in a comma separated format. If you are using CSV or TABLE format and specify the /nh parameter, column headers will be suppressed. Finally the /I switch specifies the log you are extracting the information from: it can be the application, system or security event log. Errors and warnings related to SQL Server are logged under the application log.

The interesting parameter is the /fi switch which specifies the filter conditions to be applied. Eventquery.vbs only accepts filter conditions in a predefined format and if you want to know what they are, you would probably want to have a look at the code behind the script. The general format of a filter condition is like the following:

<field> <operator> <value>

The field can be any of the following column headers you find in a Windows Server event log:

- DATETIME
- ID
- TYPE
- SOURCE
- COMPUTER
- USER
- SOURCE

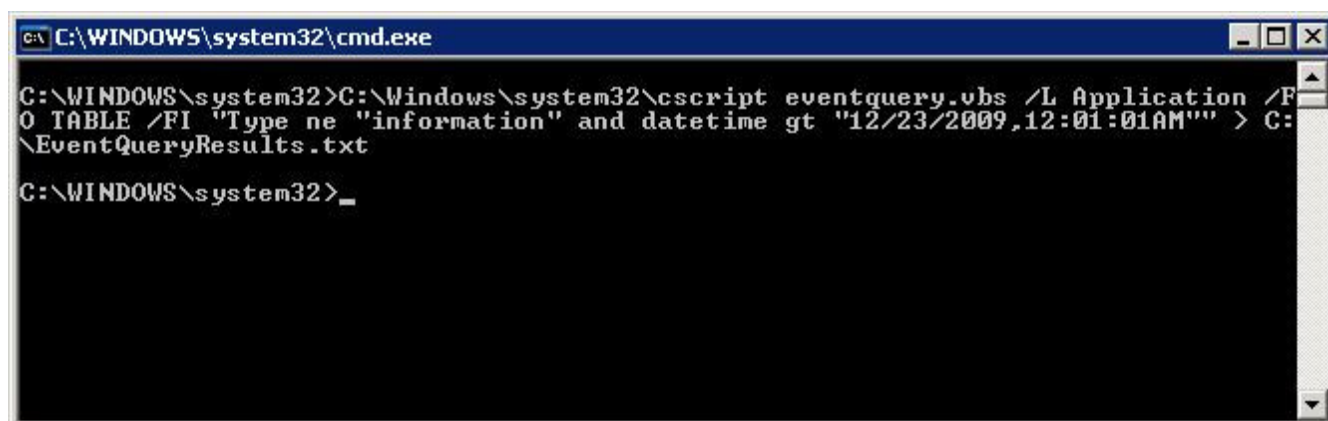
The comparison operator can be

- **eq** (equal to),
- **ne** (not equal to),
- **ge** (greater than or equal to),
- **le** (less than or equal to),
- **gt** (greater than) and
- **lt** (less than).

Not all fields can accept all comparison operators: for example, DATETIME and ID can accept all operators while COMPUTER can accept only "eq" or "ne".

As for the value part, this will be supplied by you. Except for ID field values, values have to be delimited by double quote marks ("). The DATETIME value has to be in the form of MM/dd/yy(yyyy),hh:mm:ssAM(/PM). Finally, the whole condition following the /fi switch will need to be delimited by double quote marks.

To give an example, let's say we are interested about all application event log entries that are not of "information" type and were logged after a particular date and time. We can issue a command like the following



```

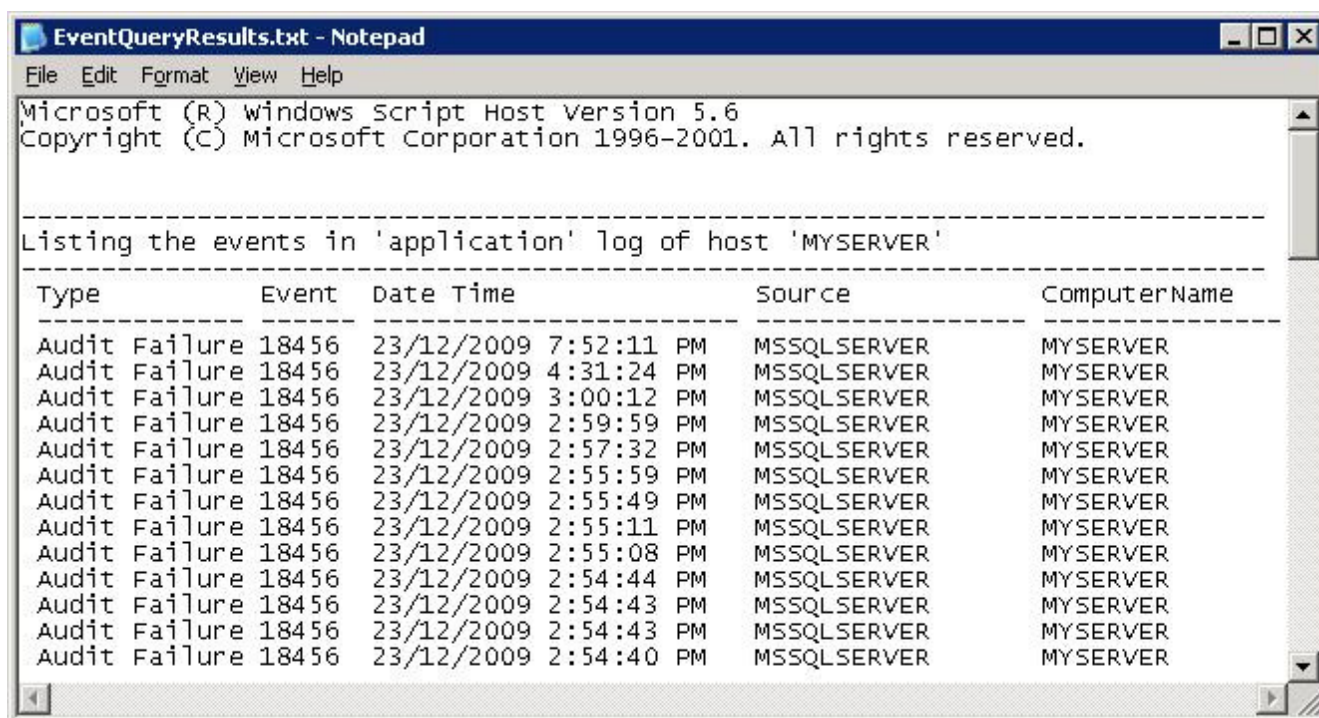
C:\WINDOWS\system32\cmd.exe

C:\WINDOWS\system32>C:\Windows\system32\cscript eventquery.vbs /L Application /FO
TABLE /FI "Type ne 'information' and datetime gt '12/23/2009,12:01:01AM'" > C:
\EventQueryResults.txt

C:\WINDOWS\system32>_

```

As you can see, the csript.exe application (also located under the Windows\system32 folder) is executing the eventquery.vbs script. We have specified the whole filter condition after the /FI switch and it is delimited by double quote marks. Also, the conditions are combined using an AND operator. The output format is to be tabular and we are capturing the results in a text file using the redirection symbol (">"). When the output file is opened in Notepad, it looks like the following:



The T-SQL Script File

Now let's look at the T-SQL script that makes use of eventquery.vbs. We have saved this script file under the C:\EventLogReport folder of the monitored box.

```

/*****
Name: scr_EventLogReport.sql
Type: T-SQL Script File
Copyright: 2009, Sadequl Hussain

```

Purpose: This script is used to query the event logs of SQL Server machines. It runs once every day and queries the application event log for SQL / Agent service related critical and warning messages. It then uploads the results to a centralised DBA database in a monitoring server. This script is called by a Windows scheduled task in the monitored server .

Assumptions:

- a) The server where the code runs is a Windows Server 2003 system
- b) The eventquery.vbs script is located under the default location of C:\Windows\system32
- c) The SQL Server instance whose context it will run against has a linked server called "DBA", pointing to the remote monitoring server.

```

*****/

```

```
SET NOCOUNT ON
```

```
-- Create some temporary tables before results are uploaded to the centralised DBA database...
```

```

IF (OBJECT_ID('tempdb.dbo.EventQuery') IS NOT NULL)
BEGIN
DROP TABLE tempdb.dbo.EventQuery
END
GO

```

```

IF (OBJECT_ID('tempdb.dbo.SystemName') IS NOT NULL)
BEGIN
DROP TABLE tempdb.dbo.SystemName
END
GO

CREATE TABLE tempdb.dbo.EventQuery
(
EventType nvarchar(50) NOT NULL,
EventID nvarchar(10) NOT NULL,
EventDateTime nvarchar(100) NOT NULL,
EventSource nvarchar(100) NOT NULL,
Computer nvarchar(50) NOT NULL
)
GO

CREATE TABLE tempdb.dbo.SystemName
(
SystemName sysname NULL
)
GO

DECLARE @ServerName sysname
DECLARE @EventQueryCmd nvarchar(2000)
DECLARE @CurrentDateTime datetime
DECLARE @BeginDateTime datetime
DECLARE @CurrentDate nvarchar(50)
DECLARE @CurrentTime nvarchar(50)
DECLARE @BeginDate nvarchar (50)
DECLARE @BeginTime nvarchar(50)
DECLARE @CurrentDateTimeText nvarchar(100)
DECLARE @BeginDateTimeText nvarchar(100)
DECLARE @CountSQLServerError int
DECLARE @CountSQLServerWarnings int
DECLARE @CountSQLServerAuditFailure int
DECLARE @CountSQLAgentErrors int
DECLARE @CountSQLAgentWarnings int
DECLARE @FinalMessage nvarchar(4000)

-- Getting the name of the current server...
SELECT @EventQueryCmd = 'hostname'
INSERT INTO tempdb.dbo.SystemName EXEC master..xp_cmdshell @EventQueryCmd
SELECT @ServerName = SystemName FROM tempdb.dbo.SystemName WHERE SystemName IS NOT
NULL

-- We have to do a bit of a juggling around with dates and times because the required
command expects the datetime parameter in a definite format....

-- Date and time when the code is running
SELECT @CurrentDateTime = GETDATE()
-- Starting date and time - 24 hours back...
SELECT @BeginDateTime = DATEADD(dd, -1, @CurrentDateTime)

-- Current date in mm/dd/yyyy format
SELECT @CurrentDate = CONVERT(nvarchar(100), @CurrentDateTime, 101)

-- Date and time in mmm dd yyyy hh:mm:ss:nnnAM/PM format
SELECT @CurrentTime = CONVERT(nvarchar(100), @CurrentDateTime, 109)
SELECT @CurrentTime = LTRIM(REVERSE(LEFT(REVERSE(@CurrentTime), CHARINDEX(' ',
REVERSE(@CurrentTime)))))
SELECT @CurrentTime = LEFT(@CurrentTime, (LEN(@CurrentTime) - CHARINDEX(':',
REVERSE(@CurrentTime)) + RIGHT(@CurrentTime, 2)

```

```

SELECT @CurrentDateTimeText = @CurrentDate + ',' + @CurrentTime

-- Start date in mm/dd/yyyy format
SELECT @BeginDate = CONVERT(nvarchar(100), @BeginDateTime, 101)

-- Date and time in mmm dd yyyy hh:mm:ss:nnnAM/PM format
SELECT @BeginTime = CONVERT(nvarchar(100), @BeginDateTime, 109)
SELECT @BeginTime = LTRIM(REVERSE(LEFT(REVERSE(@BeginTime), CHARINDEX(' ',
REVERSE(@BeginTime))))))
SELECT @BeginTime = LEFT(@BeginTime, (LEN(@BeginTime) - CHARINDEX(':',
REVERSE(@BeginTime)))) + RIGHT(@BeginTime, 2)
SELECT @BeginDateTimeText = @BeginDate + ',' + @BeginTime

-- Building the event query...
-- Please note that it expects the cscript and the eventquery.vbs to be in -- the
default location of C:\Windows\system32 folder. We are interested
-- about the application event log entries that are NOT information
-- and the output format should be in CSV

SELECT @EventQueryCmd = 'C:\Windows\system32\cscript eventquery.vbs /L Application /FO
CSV /FI "Type ne "Information" and datetime gt "' + @BeginDateTimeText + '" >
C:\EventLogReport\eventquery.txt'

EXEC master..xp_cmdshell @EventQueryCmd, no_output

-- Get the data into the temporary table
BULK INSERT tempdb.dbo.EventQuery FROM 'C:\EventLogReport\eventquery.txt' WITH
(
DATAFILETYPE = 'char',
FIRSTROW = 2,
ROWTERMINATOR = '\n',
FIELDTERMINATOR = ','
)

-- Get rid of all the double quote marks from field values...
UPDATE tempdb.dbo.EventQuery
SET EventType = REPLACE(EventType, '"', ''),
EventID = REPLACE(EventID, '"', ''),
EventDateTime = REPLACE(EventDateTime, '"', ''),
EventSource = REPLACE(EventSource, '"', ''),
Computer = REPLACE(Computer, '"', '')

-- Getting a summarised view of the data...
SELECT @CountSQLServerErrors = COUNT(*)
FROM tempdb.dbo.EventQuery
WHERE EventSource
LIKE 'MSSQL%'
AND EventType = 'Error'

SELECT @CountSQLServerWarnings = COUNT(*)
FROM tempdb.dbo.EventQuery
WHERE EventSource
LIKE 'MSSQL%'
AND EventType = 'Warning'

SELECT @CountSQLServerAuditFailure = COUNT(*)
FROM tempdb.dbo.EventQuery
WHERE EventSource
LIKE 'MSSQL%'
AND EventType = 'Audit Failure'

```



```

SELECT @CountSQLAgentErrors = COUNT(*)
FROM tempdb.dbo.EventQuery
WHERE EventSource
LIKE 'SQLSERVERAGENT%' OR EventSource LIKE 'SQLAGENT%'
AND EventType = 'Error'

SELECT @CountSQLAgentWarnings = COUNT(*)
FROM tempdb.dbo.EventQuery
WHERE EventSource
LIKE 'SQLSERVERAGENT%' OR EventSource LIKE 'SQLAGENT%'
AND EventType = 'Warning'

-- Aggregate all the information to give a summarised picture...

SELECT @FinalMessage = 'The application event log of ' + @ServerName + ' shows that
between ' + @BeginDateTimeText + ' and ' + @CurrentDateTimeText + ' : ' + CHAR(13)+
CHAR(10) + CHAR(13)+ CHAR(10) +
CONVERT(nvarchar(10), @CountSQLServerErrors) + ' critical error(s) and ' +
CONVERT(nvarchar(10), @CountSQLServerWarnings) + ' warning(s) related to SQL Server
services were logged.' + CHAR(13) + CHAR(10) +
CONVERT(nvarchar(10), @CountSQLAgentErrors) + ' critical error(s) and ' +
CONVERT(nvarchar(10), @CountSQLAgentWarnings) + ' warning(s) related to SQL Server
Agent services were logged.' + CHAR(13) + CHAR(10) +
CONVERT(nvarchar(10), @CountSQLServerAuditFailure) + ' failed login attempts related
SQL Server services were logged.' + CHAR(13) + CHAR(10) +
'-----'
-----' + CHAR(13) + CHAR(10)

-- Upload the data to the remote, centralised DBA database...
INSERT INTO DBA.DBA.Monitor.EventLogCheck_Log VALUES(@FinalMessage)

-- Get rid of the temporary tables...
IF (OBJECT_ID('tempdb.dbo.EventQuery') IS NOT NULL)
BEGIN
DROP TABLE tempdb.dbo.EventQuery
END
GO

IF (OBJECT_ID('tempdb.dbo.SystemName') IS NOT NULL)
BEGIN
DROP TABLE tempdb.dbo.SystemName
END
GO
SET NOCOUNT OFF

```

A Walk through the above Code

At the very beginning, we have created two tables in the tempdb database: one will be used for holding the server's machine name (dbo.SystemName) and the other will be used to hold the output of the eventquery command (dbo.EventQuery). We then declare a number of variables.

The name of the machine is found by calling the Windows *hostname* command. Next, we define the start and end date time for the filter condition. The @CurrentDateTimeText variable holds the current date and time while @BeginDateTimeText holds the date and time of a day before. We define these variables because we are interested in the events of the last twenty-four hours. The eventquery.vbs expects date and time to be in a specific format and since none of the CONVERT function output styles caters for this format, this block of code is used to manually generate the date and time in required form.

The next block of code is used to dynamically build the parameters for eventquery. We are interested to trap warnings, errors or audit failures: in other words any event other than the type "information". Also, the output of the query will be in comma separated (CSV) format and saved in a file called "eventquery.txt". The file will be created in the same folder where the script is located.

The contents of the output file is then loaded into dbo.EventQuery table using the BULK INSERT command. The table's structure reflects the contents of the file.

Once the data is loaded, the rest is easy. Each data field is delimited by double quote marks, so the first thing we do is get rid of those. Next, we count the total number of warnings, critical errors and audit failures for SQL Server and Agent services. This aggregate values are saved in local variables.

Once the summary values are obtained, a message is constructed with all the different pieces of information. This includes the machine name, the date and time range as well as the total number of events for each of the services. The message is then uploaded into a table called Monitor.EventCheck_Log. This is the centralised table located in our DBA database's Monitor schema. In the code above, the linked server is called DBA and it is pointing to the database called DBA.

Once the data is uploaded in the central repository, the tempdb tables are dropped.

The Windows Command File

The scr_EventLogReport.sql file is called by a Windows command file. The contents of this file is pretty straightforward. It calls the *sqlcmd* utility to run the script file in the context of a SQL Server instance running in the local machine. Note that we have not specified the username /password in the sqlcmd command: it will be specified in the Windows schedule task properties. In our example, the command file runs under the local administrator credentials.

```
REM This command file is called from a Windows Server Scheduled Task. It calls a T-SQL script file.
```

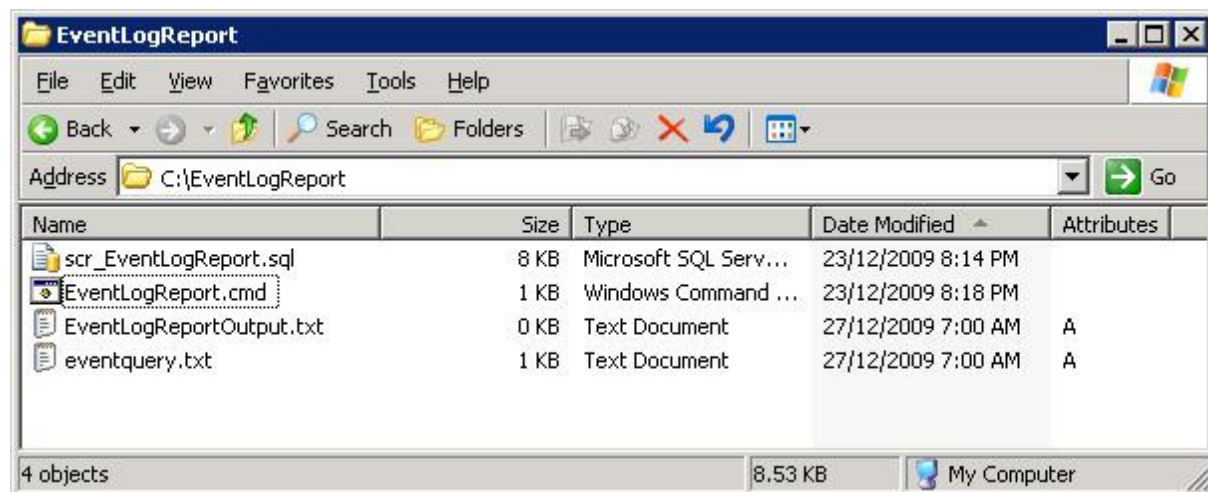
```
REM This command file and the T-SQL script both needs to be in the same location
```

```
REM The Server name needs to change to reflect the target system
```

```
REM The scheduled task runs every morning at 7:00 AM
```

```
sqlcmd -S MYSERVER -d tempdb -i C:\EventLogReport\scr_EventLogReport.sql -o  
C:\EventLogReport\EventLogReportOutput.txt
```

The contents of the EventLogReport folder will look like the following when the task runs. You can have a look at the EventLogReportOutput.txt file to see if the job or script ran successfully. It is a log file: if there were any issues, the contents of this file can help you find the reason.



Central Monitoring Database: the Stored Procedure

Once all the servers have uploaded their event log summaries to the central table, it needs to be sent to the DBA via e-mail. The DBA database will have two tables and one stored procedure for this purpose. We have already seen the central table Monitor.EventLogCheck_Log; the other table is EventLogCheck_Summary_Report. Both tables belong to a schema called "Monitor" and are almost identical in structure:

```
CREATE TABLE Monitor.EventLogCheck_Log(  
EventLogCheckReport nvarchar(4000) NULL  
)  
GO
```

```
CREATE TABLE Monitor.EventLogCheck_Summary_Report (
EventLogCheck_Summary nvarchar(4000) NULL
)
GO
```

The reason we created the second table (Monitor.EventLogCheck_Summary_Report) is because we would like to add a header section in our final report. The stored procedure for creating the report is shown below:

```
USE DBA
GO
```

```
IF EXISTS (SELECT * FROM DBA.sys.objects WHERE name = 'usp_MonitorEventLogs' AND type =
'P')
BEGIN
DROP PROCEDURE Monitor.usp_MonitorEventLogs
END
GO
```

```
CREATE PROCEDURE Monitor.usp_MonitorEventLogs
```

```
AS
```

```
/*****
Type: Stored Procedure
```

```
Copyright: 2009, Sadequl Hussain
```

Purpose: This stored procedure runs against the Monitor.EventLogCheck_Log table and sends out a DBA report about Event Log entries in different SQL Servers.

```
Version: 1.0
```

```
*****/
```

```
DECLARE @EventLogCheckReportFile nvarchar(50)
DECLARE @MonitorServer nvarchar(20)
DECLARE @DBAEMail nvarchar(255)
DECLARE @SAPassword nvarchar(255)
DECLARE @Query nvarchar(1000)
DECLARE @SQLCmd nvarchar(1000)
DECLARE @Header nvarchar(1000)
```

```
-----
-- Clear the report table...
-----
```

```
TRUNCATE TABLE DBA.Monitor.EventLogCheck_Summary_Report
```

```
-----
-- Create the report with a header and copy the summary values
-----
```

```
INSERT INTO DBA.Monitor.EventLogCheck_Summary_Report
VALUES ('=====')
SELECT @Header = '===== Event Log Monitor Report: ' +
CONVERT(nvarchar(20), GETDATE(), 103) + ' ====='
```

```
INSERT INTO DBA.Monitor.EventLogCheck_Summary_Report VALUES (@Header)
INSERT INTO DBA.Monitor.EventLogCheck_Summary_Report
VALUES ('=====')
INSERT INTO DBA.Monitor.EventLogCheck_Summary_Report VALUES (' ')
INSERT INTO DBA.Monitor.EventLogCheck_Summary_Report VALUES (' ')
```

```
INSERT INTO DBA.Monitor.EventLogCheck_Summary_Report
SELECT * FROM DBA.Monitor.EventLogCheck_Log
```

```

-----
-- Now output the report
-----

SELECT @EventLogCheckReportFile = 'D:\EventLogReports\Event_Log_Monitor_Report.txt'
SELECT @MonitorServer = @@SERVERNAME
SELECT @SAPassword = 'sa_Password'
SELECT @Query = 'SELECT * FROM DBA.Monitor.EventLogCheck_Summary_Report'
SELECT @DBAEmail = 'dba@your_organisation.com'

SELECT @SQLCmd = 'bcp "' + @Query + '" queryout ' + @EventLogCheckReportFile + ' -S' +
@MonitorServer + ' -Usa -P' + @SAPassword + ' -c'

EXEC master.dbo.xp_cmdshell @SQLCmd, no_output

-----
-- Now e-mail the report to DBA
-----

EXEC msdb.dbo.sp_send_dbmail
@profile_name = 'DB Mail Profile',
@recipients = @DBAEmail,
@subject = 'Event Log Monitor Report',
@body = 'Please find the attached monitor report on SQL Server event logs',
@file_attachments = @EventLogCheckReportFile

-----
-- Clear the event log monitor log
-----

TRUNCATE TABLE DBA.Monitor.EventLogCheck_Log

GO

```

As you can see, Monitor.usp_MonitorEventLogs is also a straightforward procedure. We add a header section into the Monitor.EventLogCheck_Summary_Report table and then simply copy the data from the central table into it. Next, some variables are initialized. We have used hard-coded values here for demonstration purpose - in your actual system they can be dynamically populated from metadata tables. Next, the contents of the Monitor.EventLogCheck_Summary_Report table is output using the *bcp* command. Finally the bcp output file is sent to the DBA via e-mail. Once the process is completed, the central table is cleared for the next day's event summary data. Also, when the stored procedure starts, it clears up the report table.

The SQL Server Job

This stored procedure is invoked from a SQL Server job. The properties for this job is shown below:

Job Properties - Monitor Database Server Event Logs

Select a page: General, Steps, Schedules, Alerts, Notifications, Targets

Script Help

Name: Monitor Event Logs

Owner: sa

Category: DBA_MONITOR

Description: This job sends out an e-mail with summary level report on the event logs of database servers...

Job Properties - Monitor Database Server Event Logs

Select a page: General, Steps, Schedules, Alerts, Notifications, Targets

Script Help

Script Action to New Query Window

Job step list:

St...	Name	Type	On Success	On Failure
1	Send Event Log Monitoring Summary Report	Transact...	Quit the jo...	Quit the job...

Job Step Properties - Send Event Log Monitoring Summary Report

Select a page: General, Advanced

Script Help

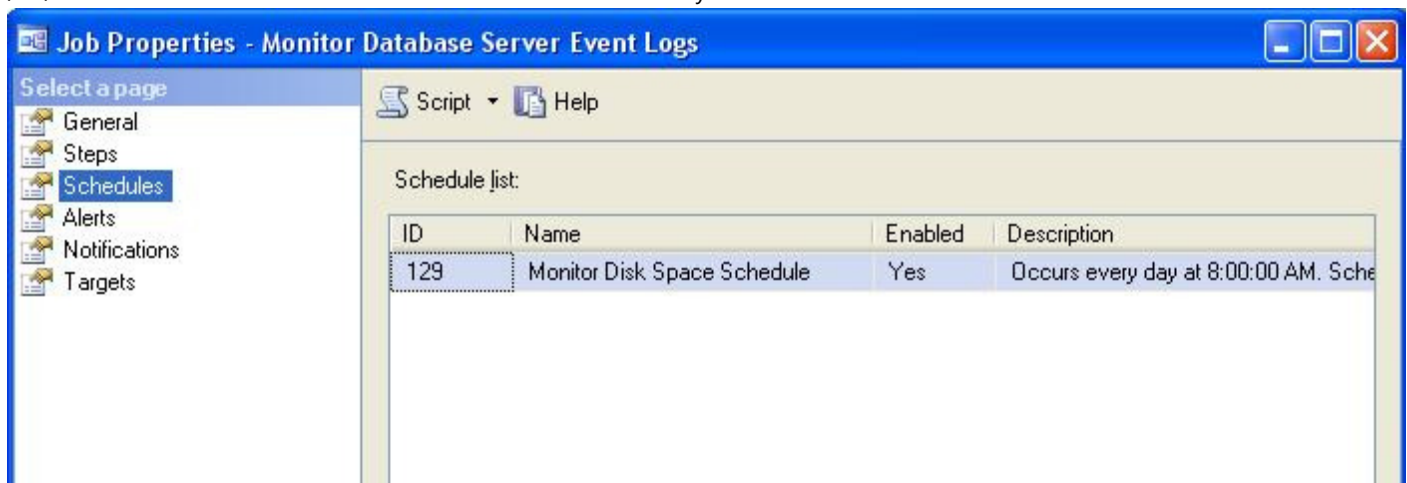
Step name: Send Event Log Monitoring Summary Report

Type: Transact-SQL script (T-SQL)

Run as:

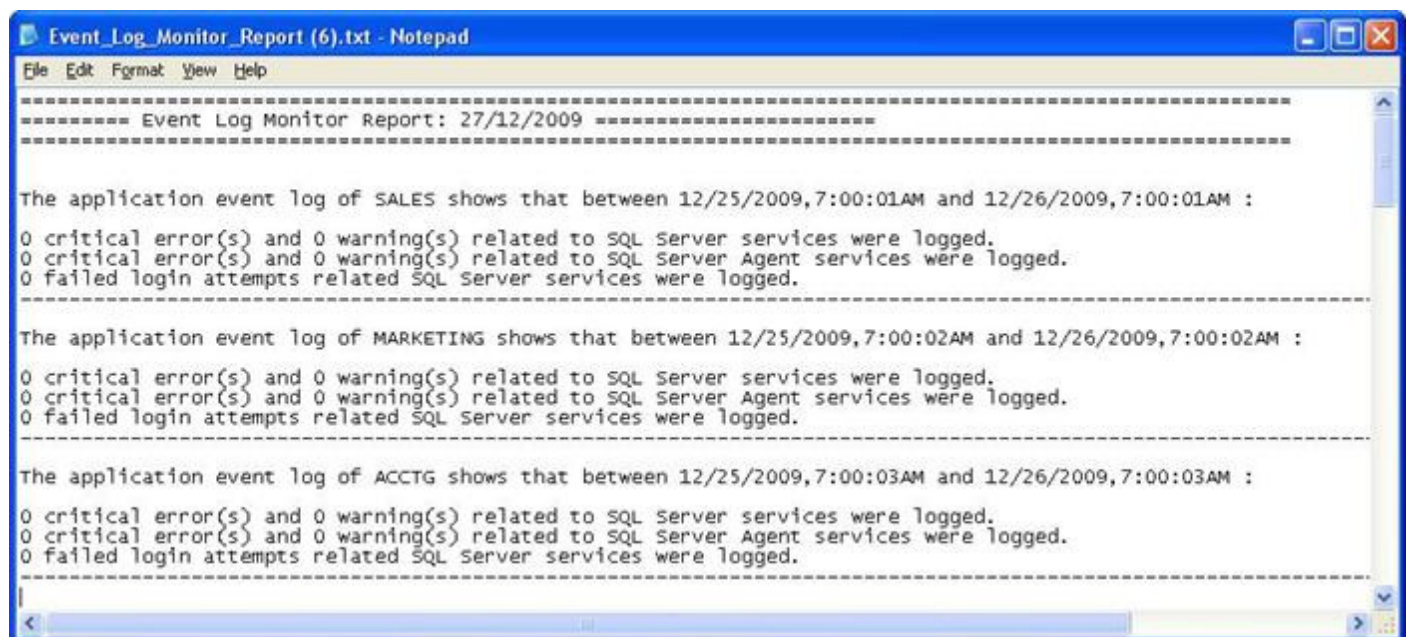
Database: DBA

Command: EXEC DBA.Monitor.usp_MonitorEventLogs



The Final Output

When the job executes, the DBA is sent an e-mail with an attachment. The contents of the attached file will look something like the following:



Here, the event log monitoring report is showing the status of three servers: SALES, MARKETING and ACCTG. Ideally we would want to see zero errors and warnings; however if there are errors and warnings reported, the DBA can have a look at the server involved for further analysis.

Assumptions, Limitations and Points to Note

There are a number of points to note about this monitoring system:

- The code provided in this tip will work with **Windows Server 2003 only**. Unfortunately Windows Server 2008 no longer comes with the eventquery.vbs script file. There is a way to trap Windows Server 2008 event log entries, but that will be the topic of another tip. We also assume that you are not running your SQL Servers on Windows Server 2000.
- The T-SQL script file in each target machine will be running under the context of a SQL Server instance installed on that machine. We assume that instance
 - is running when the script executes
 - has a linked server called "DBA" pointing to the central monitoring server
 - has xp_cmdshell enabled (for versions 2005 and above)
- There can be multiple SQL Server instances running in a machine; the report does not show which instance any errors or warnings are associated with. All it shows is an aggregated number.

- The central monitoring server hosting the DBA database is either a SQL Server 2005 or a SQL Server 2008 instance and has Database Mail configured.
- The Windows command file uses the sqlcmd utility. If the T-SQL script runs under the context of a SQL Server 2000 instance, we have to use *osql* instead.
- The account used to run the Windows scheduled task (MYSERVER\Administrator in our example) will need to have sufficient privileges in the SQL Server instance involved. If this account does not have CREATE TABLE permission in that instance, messages like the following will be logged in the EventLogReportOutput.txt file:

```
Msg 262, Level 14, State 1, Server MYSERVER, Line 2
CREATE TABLE permission denied in database 'tempdb'.
Msg 262, Level 14, State 1, Server MYSERVER, Line 3
CREATE TABLE permission denied in database 'tempdb'.
Msg 208, Level 16, State 1, Server MYSERVER, Line 25
Invalid object name 'tempdb.dbo.SystemName'.
```

- The account will also need to have sufficient privilege to create files in the local operating system . Otherwise a message like the following will be logged:

```
Msg 4860, Level 16, State 1, Server MYSERVER, Line 59
Cannot bulk load. The file "C:\EventLogReport\eventquery.txt" does not exist.
```

This happens because the file was not generated in the first place due to an "Access is denied" error.

- Finally, the system will need a little tweaking if your central monitoring server has a higher version of SQL Server (2005 or 2008) than your monitored servers (2000 or 2005 respectively). The problem becomes apparent when you create the linked server in the monitored server. For example, if you create a linked server in a SQL Server 2000 system pointing to a SQL Server 2005 instance, the following error will be generated when the linked server is accessed:

```
Error 17: SQL Server does not exist or access denied.
```

You can take one of the following steps in such situations:

- Create a separate monitoring instance that runs SQL Server 2000 (you will need to use SQL Mail instead of Database Mail)
- Send out individual e-mails from the monitored servers.

Next Steps

- Have a look at the code of eventquery.vbs in a Windows Server 2003 system. Try running the command using different filter conditions.
- Try to implement the system described in your own work environment. Make modifications and enhancements to suit business needs.
- Learn about the new look and feel Event Viewer of Windows Server 2008. Find out about the Windows Logs, Application and Services Logs, Forwarded Events and Subscriptions.
- Learn more about [Windows Power Shell](#).

Copyright (c) 2006-2010 [Edgewood Solutions, LLC](#) All rights reserved

[privacy statement](#) | [disclaimer](#) | [copyright](#)

Some names and products listed are the registered trademarks of their respective owners.