## Description

Runs a T-SQL script. Invoke-Sqlcmd2 only returns message output, such as the output of PRINT statements when -verbose parameter is specified.

.EXAMPLE

Invoke-Sqlcmd2 -ServerInstance "MyComputer\MyInstance" -Query "SELECT login\_time AS 'StartTime' FROM sysprocesses WHERE spid = 1"

This example connects to a named instance of the Database Engine on a computer and runs a basic T-SQL query.

StartTime

-----------

2010-08-12 21:21:03.593

.EXAMPLE

Invoke-Sqlcmd2 -ServerInstance "MyComputer\MyInstance" -InputFile "C:\MyFolder\tsqlscript.sql" | Out-File -filePath "C:\MyFolder\tsqlscript.rpt"

This example reads a file containing T-SQL statements, runs the file, and writes the output to another file.

.EXAMPLE

Invoke-Sqlcmd2 -ServerInstance "MyComputer\MyInstance" -Query "PRINT 'hello world'" -Verbose

This example uses the PowerShell -Verbose parameter to return the message output of the PRINT command.

### Use PowerShell to Collect Server Data and Write to SQL



[ScriptingGuy1](http://blogs.technet.com/ScriptingGuy1/ProfileUrlRedirect.ashx)

1 Nov 2010 12:01 AM

* [20](http://blogs.technet.com/b/heyscriptingguy/archive/2010/11/01/use-powershell-to-collect-server-data-and-write-to-sql.aspx#comments)

**Summary:** Use Windows PowerShell to collect server data and automatically store that information in a Microsoft SQL Server.

Hey, Scripting Guy! How is Windows PowerShell usage by database professionals different from the way that other IT Pros use Windows PowerShell?

-- MC

Hello MC, Microsoft Scripting Guy Ed Wilson here. Next week, Nov 8-11, 2010, is the annual [SQL PASS Summit](http://www.sqlpass.org/summit/na2010/) in Seattle. In honor of this event, we will have guest bloggers from the SQL side of life. Today we will begin with Chad Miller.

Chad Miller([Blog](http://sev17.com/)|[Twitter](http://www.twitter.com/cmille19)) is a SQL Server DBA and Senior Manager of Database Administration at Raymond James Financial. In his spare time, he is the Project Coordinator/Developer of the Codeplex project SQL Server PowerShell Extensions ([SQLPSX](http://sqlpsx.codeplex.com/)). Chad leads the [Tampa Powershell User Group](http://powershellgroup.org/tampa.fl) and is a frequent speaker at SQL Saturdays and Code Camps.

Take it away Chad!

We are data people and we believe data should exist in databases. We spend a good part of our day writing Transact-SQL scripts to query and load data from other sources. Therefore, not surprisingly when we use Windows PowerShell we want to run a command, capture the output and then store the data in a database. After the data is in a database, we can use Transact-SQL to do additional reporting and analysis. We might even use SQL Server Reporting Services to provide Web-based access to the data.

**Extract Data**

SQL Server 2008 and 2008 R2 provide the **Invoke-SqlCmd** cmdlet, but only on computers where the sqlps (the SQL Server mini-shell) is installed. As an alternative solution, you can implement your own function which does not require loading external snap-ins. I have copied the [invoke-sqlcmd2 function](http://gallery.technet.microsoft.com/ScriptCenter/en-us/7985b7ef-ed89-4dfd-b02a-433cc4e30894) to the Scripting Guys Script Repository. Save the code as**invoke-sqlcmd2.ps1** and then dot source our new function into your Windows PowerShell console by using the following command:

. ./invoke-sqlcmd2.ps1

The following example dot sources the **invoke-sqlcmd2.ps1** script that contains the **invoke-sqlcmd2** function, connects to the pubs database on a SQL server named SQL1 and runs a basic Transact-SQL query. The last command is a single command, but has wrapped to the third line in the output.

PS C:\> . C:\data\ScriptingGuys\2010\HSG\_11\_1\_10\invoke-Sqlcmd2.ps1

PS C:\> Invoke-Sqlcmd2 -ServerInstance sql1 -Database pubs -Query "Select \* from auth

ors"

au\_id : 172-32-1176

au\_lname : White

au\_fname : Johnson

phone : 408 496-7223

address : 10932 Bigge Rd.

city : Menlo Park

state : CA

zip : 94025

contract : True

<...OUTPUT Truncated>

This example reads a file that contains T-SQL statements, runs the file, and writes the output to another file.

Invoke-Sqlcmd2 -ServerInstance "MyComputer\MyInstance" -InputFile "C:\MyFolder\tsqlscript.sql" | Out-File -filePath "C:\MyFolder\tsqlscript.rpt"

This example uses the Windows PowerShell **-Verbose** parameter to return the message output of the **PRINT** command.

Invoke-Sqlcmd2 -ServerInstance "MyComputer\MyInstance" -Query "PRINT 'hello world'" -Verbose

VERBOSE: hello world

**Getting a Server List**

When you have to run a Windows PowerShell command across multiple servers, you will frequently see examples in which the list of servers are stored in a text file and read by using the **Get-Content** cmdlet. However, most database professionals maintain a list of SQL Servers they manage in either a table they create or they may use Central Management Server (CMS) introduced in SQL Server 2008. Instead of using a text file, for database professionals, it makes more sense to read a SQL table. Let's take a look at an example. For the purposes of this demonstration we'll create a table. However, you could just as easily substitute the **msdb.dbo.sysmanagement\_shared\_registered\_servers** view from your CMS server.

From SQL Server Management Studio, create a SQL table.

CREATE TABLE server\_instance

(server\_name varchar(255) NOT NULL);

Next populate the table with a list of SQL Servers:

INSERT server\_instance VALUES('Z001');

INSERT server\_instance VALUES('Z002\SQ2K8');

INSERT server\_instance VALUES('Z003\R2');

With our **server\_instance** table populated, we can use our **Invoke-SqlCmd2** function to retrieve a list of servers and then call Windows PowerShell command for each. The following example retrieves the version information.

Invoke-sqlcmd2 -ServerInstance "Z003\R2" -Database dbautility -Query "Select server\_name FROM server\_instance" | foreach-object {Invoke-SqlCmd2 -ServerInstance $\_.server\_name -Database master -Query "SELECT @@version"}

**Extract and Load data**

A common task for a database professional is collecting and loading data from multiple servers into central utility database. Using the invoke-sqlcmd2 function, we can extract data from a SQL Server data source, but to load data we have to introduce a new function called Write-DataTable. I have uploaded the [Write-DataTable Windows PowerShell function](http://gallery.technet.microsoft.com/ScriptCenter/en-us/2fdeaf8d-b164-411c-9483-99413d6053ae) to the Scripting Guys Script Repository. Save the Windows PowerShell function from the Script Repository as **write-datatable.ps1** and then dot source our new function into the Windows PowerShell console by using the following command.

. ./write-datatable.ps1

The **Write-DataTable** function uses the .NET [Data.SqlClient.SqlBulkCopy class](http://msdn.microsoft.com/en-us/library/30c3y597(v=VS.90).aspx) to load an in-memory **DataTable** or**DataRow** array into a SQL Server table. This works out well because the cmdlet **invoke-sqlcmd** or our function **invoke-sqlcmd2** returns a **datatable** object. Let us examine an example database named space usage collection for all databases in your environment to use in forecasting growth.

You have to create a SQL Server table using SQL Server Management Studio. To do this, run the following Transact-SQL script:

CREATE TABLE [dbo].[db\_space](

[server\_name] [varchar](128) NOT NULL,

[dbname] [varchar](128) NOT NULL,

[physical\_name] [varchar](260) NOT NULL,

[dt] [datetime] NOT NULL,

[file\_group\_name] [varchar](128) NOT NULL,

[size\_mb] [int] NULL,

[free\_mb] [int] NULL,

CONSTRAINT [PK\_db\_space] PRIMARY KEY CLUSTERED

(

[server\_name] ASC,

[dbname] ASC,

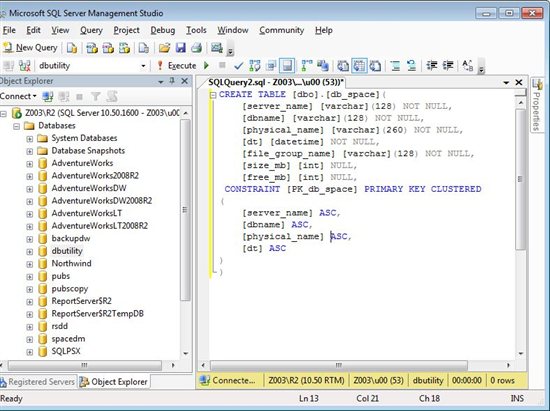
[physical\_name] ASC,

[dt] ASC

)

)

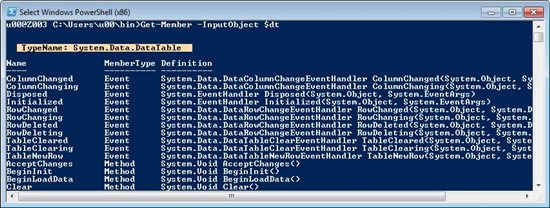
The T script and management studio are seen in the following figure.



Next we'll use a Transact-SQL query to collect the server name, database name and file information. We'll save the query to a plain old .sql file that is named [get-dbspace.sql](http://gallery.technet.microsoft.com/ScriptCenter/en-us/620f28ae-867e-4beb-b98b-d5640213e255). This is the same kind of SQL script that you would execute in SQL Server Management Studio. However, we will call the script from the **Invoke-SqlCmd2** function:

$dt = invoke-sqlcmd2 -ServerInstance "Z003\R2" -Database "master" -InputFile ./get-dbspace.sql -As 'DataTable'

Examining the type information about $dt variable we can see the type is of a DataTable as shown in the following figure.



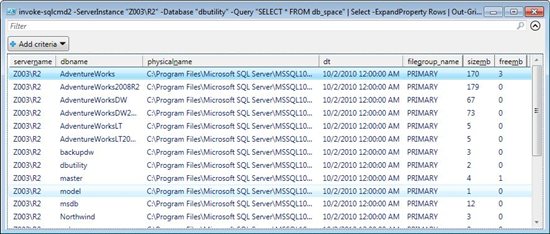
To load the **DataTable** into our SQL Server table we'll call the **Write-DataTable** function:

Write-DataTable -ServerInstance "Z003\R2" -Database "dbutility" -TableName "db\_space" -Data $dt

Using **invoke-sqlcmd2** and piping the output to **Out-GridView** we can see the data has in fact been loaded.

invoke-sqlcmd2 -ServerInstance "Z003\R2" -Database "dbutility" -Query "SELECT \* FROM db\_space" | Out-GridView

The results from this command are shown in the figure below.



**Extract, Transform, and Load Data**

As we've seen, by using a couple of simple functions - **invoke-sqlcmd2** and **write-datatable** - we can easily load data from any SQL Server data source, but what about any Windows PowerShell command? As an example, we want to collect disk space utilization by using **Get-WMIObject Win32\_LogicalDisk** across a group of SQL Servers into a central database for reporting trending. If we can convert the output of our WMI call into a **DataTable** then we can use our **Write-DataTable**function. Using a function called **Out-DataTable** adapted from [a script](http://thepowershellguy.com/blogs/posh/archive/2007/01/21/powershell-gui-scripblock-monitor-script.aspx) by Marc van Orsouw ([Blog](http://thepowershellguy.com/blogs/posh/default.aspx)|[Twitter](http://twitter.com/powershellguy)) we can do just that. I saved the [modified script](http://gallery.technet.microsoft.com/ScriptCenter/en-us/4208a159-a52e-4b99-83d4-8048468d29dd) to the Scripting Guys Script Repository.

Save the following code as **Out-DataTable.ps1** and source our new function

. ./ Out-DataTable.ps1

To load convert the output of a WMI call to a **DataTable**, we'll pipe to our newly created **out-datatable** function and assign the output to a *$dt*.

$dt = Get-WmiObject Win32\_LogicalDisk -filter "DriveType=3" | Select @{n='UsageDT';e={get-date -Format "yyyy-MM-dd"}}, '

SystemName, DeviceID, VolumeName, '

@{n='Size';e={$([math]::round(($\_.Size/1GB),2))}}, @{n='FreeSpace';e={$([math]::round(($\_.FreeSpace/1GB),2))}} | out-datatable

Having assigned the output to a datatable you can call the **write-datatable** to load the collected data into a SQL Server table. This makes **out-datatable** very powerful in that the output of any Windows PowerShell command can be easily loaded into a SQL Server table. Using **write-datatable** requires a SQL Server destination table to exist, but instead of manually creating a SQL Server table, we'll use a new function called **Add-SqlTable**. The **Add-SqlTable.ps1** script is also uploaded to the [Scripting Guys Script Repository.](http://gallery.technet.microsoft.com/ScriptCenter/en-us/c193ed1a-9152-4bda-b5c0-acd044e68b2c) Save the code as **Add-SqlTable.ps1** and source the new function.

The following command will create a new empty SQL Server table named **diskspace** based on the structure of our**DataTable** variable *$dt*:

Add-SqlTable -ServerInstance "Z003\R2" -Database dbutility -TableName diskspace -DataTable $dt

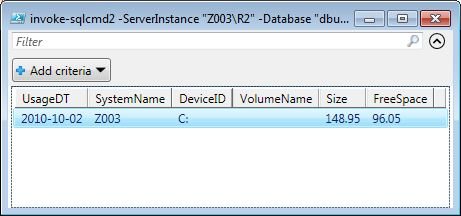
**Note:** Using this technique to create a SQL Server table is not as precise as manually creating a table as certain elements including defining primary keys and data types are not handled. **Add-SqlTable** works best for a quick data dump of Windows PowerShell data to a SQL Server table.

Finally, with the destination SQL Server table created, we can call our **write-datatable** function to load the collected data:

Write-DataTable -ServerInstance "Z003\R2" -Database "dbutility" -TableName "diskspace" -Data $dt

Using **invoke-sqlcmd2** we can see the data was loaded into the SQL Server table:

invoke-sqlcmd2 -ServerInstance "Z003\R2" -Database "dbutility" -Query "SELECT \* FROM diskspace" | Out-GridView



**Summary**

This post demonstrated how to query and load the output of any Windows PowerShell command into a SQL Server table. The functions **invoke-sqlcmd2**, **write-datatable**, **out-datatable** and **add-sqltable** can be used as building blocks for many of your Windows PowerShell-based data loading needs.

MC, that is all there is to using Windows PowerShell and SQL Server. SQL week will continue tomorrow when SQL guest blogger Aaron Nelson will talk about how to work with SQL snap-ins. Thank you Chad, for sharing your time and knowledge with us.

I invite you to follow me on [Twitter](http://bit.ly/scriptingguystwitter) or [Facebook](http://bit.ly/scriptingguysfacebook). If you have any questions, send email to me at scripter@microsoft.com or post them on the [Official Scripting Guys Forum](http://bit.ly/scriptingforum). See you tomorrow. Until then, peace.

Alternate Method

#define servers to be monitored  
$server = get-content "<path>.txt"  
  
#data table to hold results  
Function out-DataTable   
{  
 $dt = new-object Data.datatable   
 $First = $true   
  
 foreach ($item in $input){   
 $DR = $DT.NewRow()   
 $Item.PsObject.get\_properties() | foreach {   
 if ($first) {   
 $Col = new-object Data.DataColumn   
 $Col.ColumnName = $\_.Name.ToString()   
 $DT.Columns.Add($Col) }   
 if ($\_.value -eq $null) {   
 $DR.Item($\_.Name) = "[empty]"   
 }   
 elseif ($\_.IsArray) {   
 $DR.Item($\_.Name) =[string]::Join($\_.value ,";")   
 }   
 else {   
 $DR.Item($\_.Name) = $\_.value   
 }   
 }   
 $DT.Rows.Add($DR)   
 $First = $false   
 }   
  
 return @(,($dt))  
  
}  
  
#function to retrieve disk information  
Function Get-DisksSpace ([string]$Servername, $unit= "GB")  
{  
$measure = "1$unit"  
  
Get-WmiObject -computername $serverName -query "  
select SystemName, Name, DriveType, FileSystem, FreeSpace, Capacity, Label  
 from Win32\_Volume  
 where DriveType = 2 or DriveType = 3" `  
| select @{Label="SystemName";Expression={$serverName.ToUpper()}} `  
 , Name `  
 , @{Label="SizeIn$unit";Expression={"{0:n2}" -f($\_.Capacity/$measure)}} `  
 , @{Label="FreeIn$unit";Expression={"{0:n2}" -f($\_.freespace/$measure)}} `  
 , Label  
}  
  
#execute the functions  
foreach ($s in $server)  
{  
Get-DisksSpace $s  
  
$dataTable = Get-DisksSpace $s | where {$\_.name -like "E:\\*" -or $\_.name -like "C:\\*" -or $\_.name -like "F:\\*"} | out-DataTable  
$connectionString = "Data Source=<server\instance>; Integrated Security=True;Initial Catalog=<databaseName>;"  
$bulkCopy = new-object ("Data.SqlClient.SqlBulkCopy") $connectionString  
$bulkCopy.DestinationTableName = "<schema.table>"  
$bulkCopy.WriteToServer($dataTable)  
}