http://www.sqlservercentral.com/articles/Stairway+Series/93403/
Printed 2014/05/06 09:56PM

Stairway to SQL PowerShell Level 4: Objects in SQL PowerShell

By Ben Miller, 2012/09/11

This far, we have learned about installation and setup of the PowerShell environment. We have built a simple profile to prepare for SQL Server PowerShell and learned about Input and Output functions. You should now have a foundation of SQL Server PowerShell. We now are ready to learn about Objects in SQL PowerShell. Everything in PowerShell is an object. Let's start out with what to expect in an Object by defining what that means in PowerShell.

Item	Description	Example
Property	This is the holder of the data. You use properties to hold information about a piece of the object. They can be defined as having a Name and Value.	$database.Name Simply returns the property name which in this case contains the name of the database.
Method	Methods are used to execute something against the object contents. Think of methods as instructions that use the data in the properties to accomplish tasks or output information that is in the properties. They also can get information related to the object from other sources.	$database.Alter() Method allowing the properties that were changed on the $database object to be persisted to the engine.
Event	Events are a specific definition inside an object. These are raised when a certain "event" happens to the object. This allows you to hook into event that is raised so that you can handle the event. Important to see and to know, but there will be few events that you will handle in most cases during this Stairway.	PropertyChanged Simply an event that the server object can raise when a property changes. The client script or application can handle the event and get the information from the event about what happened.

Table 4.1 Parts of Objects

Table 4.1 shows the main parts of Objects and what each part means. If you think about everything in your world as an object, you will be ahead of the game. Think of a table, and if you were to describe it to anyone, how would you do it? You may start by saying that this table has four legs, a top that is 48 inches by 48 inches and is a natural wood color. These are properties of the object, the Table, and they hold data. So a property could be Legs which is the name, and the value of the property would be 4. The Width property would be 48 inches, and the Depth property is 48 inches, etc.

When using SQL Server PowerShell you will need to understand what methods are and how to use them. This level will help you with that, and expand your knowledge of what they are and how to use them. In Level 2, you learned about the SQL Server objects that you installed called SMO (Shared Management Objects). These have Properties, Methods and Events. Using the Get-Member cmdlet that you learned about previously, if you run the command in Listing 4.1 you will see the results in Figure 4.1. Notice there are Methods, Properties and an Event for this object. It is a simple object of an Event Log item, but it illustrates the types well.

```
Get-EventLog -Logname System | Select -First 1 | Get-Member
```

Listing 4.1 Command to show pieces of an Object

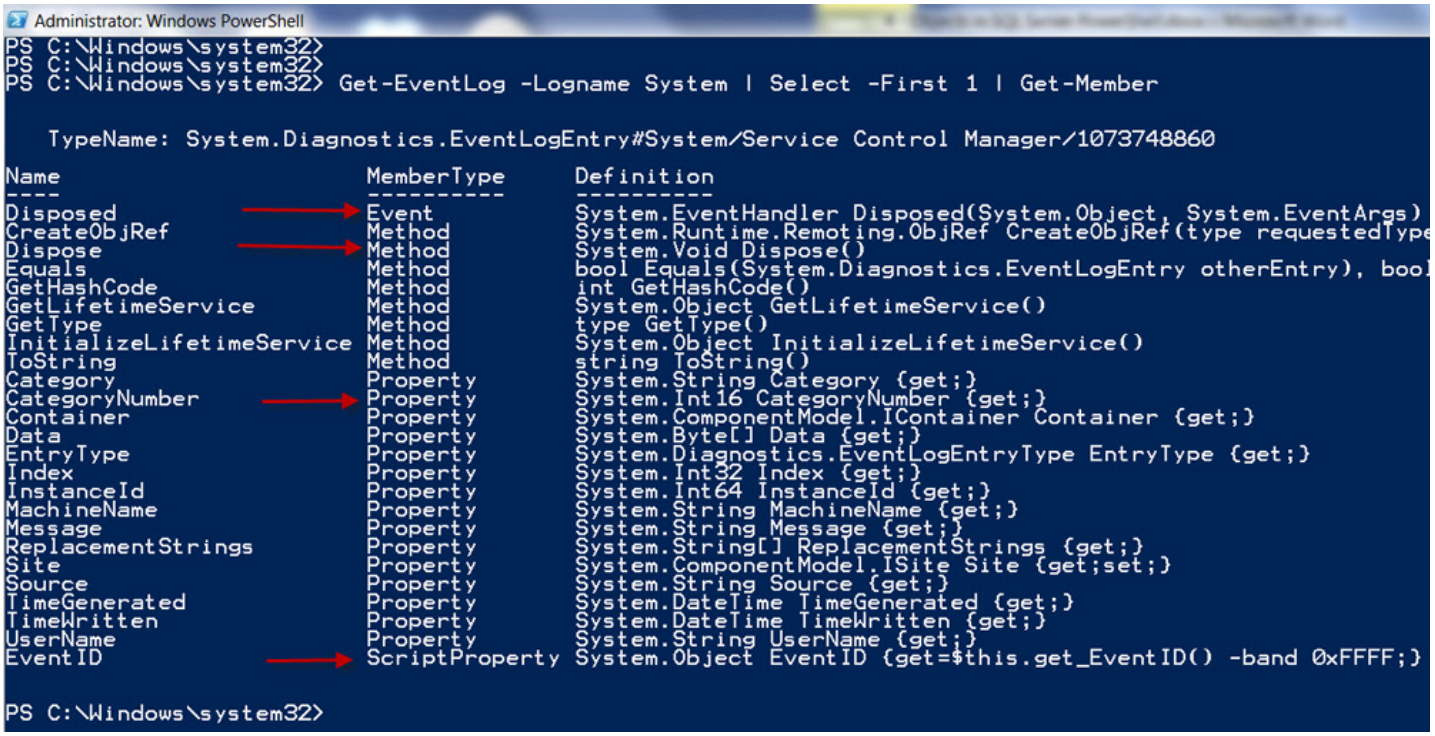

Figure 4.1 Using Get-Member to show different parts of an Object

Objects in SQL PowerShell

Now we will learn about the objects that you will encounter while using SQL PowerShell. The main focus will be on SMO and how these objects represent database objects that you are able to manage. You learned about loading assemblies when you built a simple profile in Level 1 and 2. The first object you will see in this level is the Server Object. This object is in the Microsoft.SqlServer.Smo assembly under the namespace of Microsoft.SqlServer.Management.Smo with the name of Server. Don't be too alarmed at the assembly and namespace words, as the assembly is another name for the DLL that the object is defined in and the namespace is simply a way to make homes or categories for the various objects so that you know what they are related to and reference them. Each period in the namespace can be a separate dividing line for objects to live in. In this case the namespace of Microsoft.SqlServer.Management.Smo is where the Server object lives as well as many other objects. Listing 4.2 is the script to create a server object that will use Integrated Security to connect to SQL Server and display the properties, etc. of this object.

NOTE: A point of clarification on the creation of a Server object. You are creating an object in memory that is of type Server, but a connection is not made immediately, it will be made when you access a property that needs to come from SQL Server. *Name* is one of those properties that will be populated by your parameter, not by connecting to SQL Server.

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.Smo")

$server = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Server -ArgumentList "localhost"

$server | Get-Member
```

Listing 4.2 Get SMO Server Object with Properties, Methods, Events

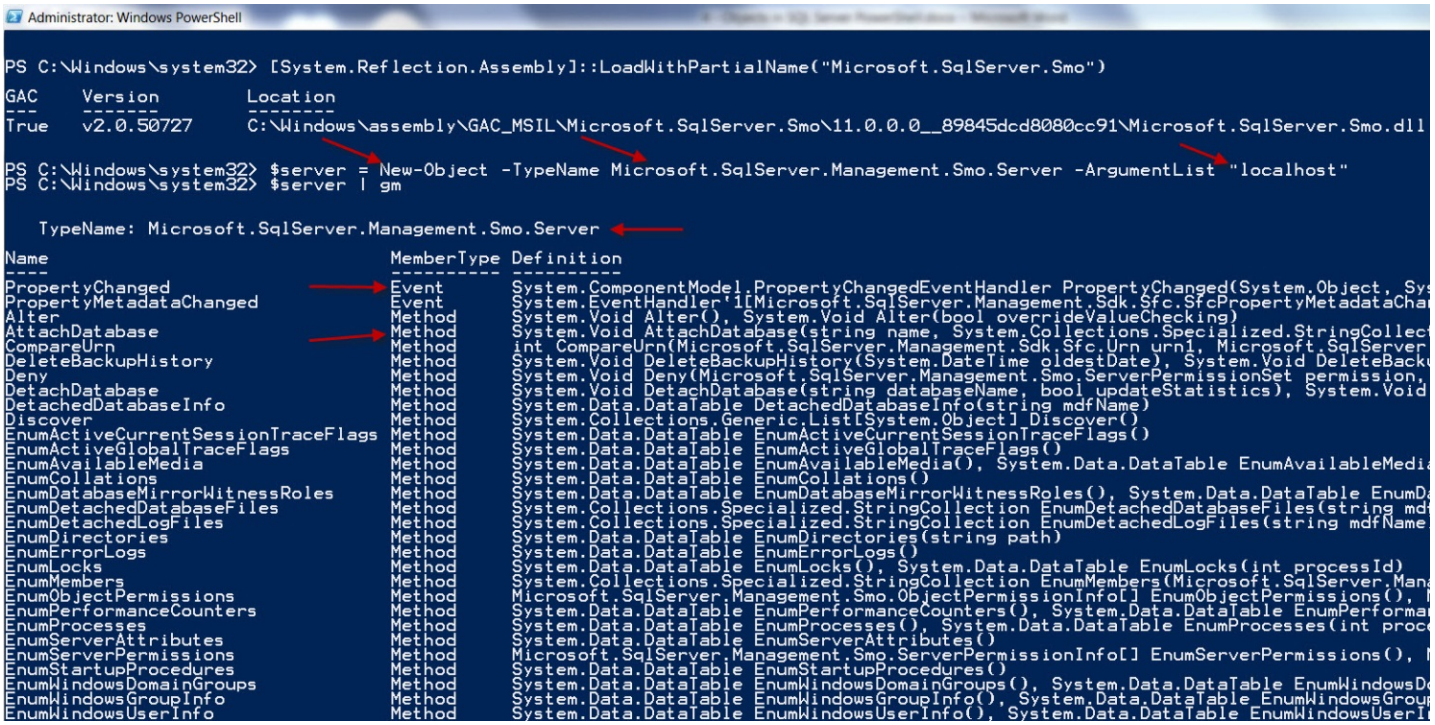

Figure 4.2 Output of Script for getting SMO Server Object

Notice that this server object is of type *Microsoft.SqlServer.Management.Smo.Server* and that the *Microsoft.SqlServer.Smo* assembly was loaded in order to get this SMO Server object. Also note that there are Events, Methods and Properties in this object. In the graphic you will only see Events and Methods, but the output from the script in Listing 4.2 gets you all the properties as well. Let's look at the Server object in more detail.

SMO Server Object

In Table 4.2 you will see an abbreviated list of properties that we can retrieve from the SQL Server inside this SMO Server Object.

Property	Object Notation	Description
Version	$server.Version	This is the version of the SQL Server. This property is actually another object of type System.Version which has Build, Major, MajorRevision, Minor, MinorRevision, Revision
Name	$server.Name	This is the name of the server as returned by SQL Server
Information	$server.Information	This is a collection of information about the Server.
Information.Language	$server.Information.Language	This tells of the Language of the server. It is in the Information object and referenced by adding Language to the end of the Information property.

Table 4.2 Abbreviated listing of Properties in the SMO Server object

This is a server object that contains many more things than can be listed in this level. But here is the reference for the SMO Server object. You can see from the page at the link that there are many properties and some that are objects instead of just strings or numbers. Other properties are collections which is simply a set of objects that are used to hold similar objects in a collection instead of in separate object properties. Knowing about objects will help you explore and work with PowerShell because everything in PowerShell is an object.

Now that you have seen the server object and some of its properties, we will look at methods. First, we will examine some of the methods associated with the SMO Server object. In using the Server object for information, you will notice that you use the dot notation, object.Property. In using methods, it begins the same way with one difference. The call consists of object.Method(parameters) which is a little different than the property call because it adds a set of parenthesis with the potential of parameters in between. The parameters will be defined by the object, and the types of those parameters are also defined by the object containing the method. The parameters can be passed into the method by variable or text or numbers, and if there are multiple parameters, they are separated by commas. Referring to the documentation is very important to understand how to use the methods correctly. The easy way to find documentation on the object you are using in SMO, is to go to your favorite search engine, and type SQL SMO ObjectName, such as SQL SMO Server. It should be in the first page of results, so you can be on your way.

Let's dive into some of the methods on the Server object and what they do so you can get an idea of how powerful and useful automation can be using PowerShell as your vehicle.

In the SMO Server object, there are a few methods that I will describe. This will give you an idea of how to use them and show that there is no magic here. The rest of the work is yours to experiment and find out what the others do. Table 4.3 illustrates the methods that we will be looking at in this level.

Method	**Explanation**

EnumActiveGlobalTraceFlags()	This method shows you the trace flags that are set globally. This method has no parameters.
PingSqlServerVersion("localhost")	Returns the version of SQL Server (BuildNumber, Major, Minor)
EnumProcesses($true | $false)	Returns a list of the SPIDs connected to SQL Server. $true excludes system SPIDs, and $false shows system SPIDs.

Table 4.3 Methods of the SMO.Server object

EnumActiveGlobalTraceFlags method

This method of the SMO Server object will enumerate or list the Active Global Trace Flags as the method is named. It will return a System.Data.DataTable and is called with no parameters. In Listing 4.3 you see the code to get the data from the Server object and display them. In Figure 4.3 you will see the results of my instance. This shows that you get a TraceFlag back as well as the Status of the flag. This could be a useful method in automation for your server environment. There are 2 other results that you don't see by default, the first is Global and it is either 1 or 0, and Session which is either 1 or 0. These values mean ON or OFF. Under the covers it executes DBCC TRACESTATUS(-1) WITH NO_INFOMSGS, so again, no magic.

```
$server = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Server -ArgumentList localhost

$server.EnumActiveGlobalTraceFlags()
```

Listing 4.3 Code to enumerate global trace flags in this instance

```
PS C:\Windows\system32> $server = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Server -ArgumentList localhost
PS C:\Windows\system32> $server.EnumActiveGlobalTraceFlags()

                                TraceFlag                                Status
                                ---------                                ------
                                     1204                                     1
                                     3604                                     1


PS C:\Windows\system32>
```

Figure 4.3 Output of EnumActiveGlobalTraceFlags method

PingSqlServerVersion method

This is a method on the SMO Server object. This will return a Major, Minor and BuildNumber of the SQL Server name that you specify in the call. The signature of this method is PingSqlServerVersion(string), or PingSqlServerVersion(string, string, string). In the first signature the string is the server name. In the second signature, it includes a second and third parameter and they are for a login and password to connect via SQL authentication. In Listing 4.4 and Figure 4.4 you see the command and the results. Signature simply means, the parameters and their types.

```
$server = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Server -ArgumentList localhost

$server.PingSqlServerVersion("localhost")
```

Listing 4.4 Using PingSqlServerVersion to get version from an instance

```
PS C:\Windows\system32> $server.PingSqlServerVersion("localhost")

                                Major                                Minor                            BuildNumber
                                -----                                -----                            -----------
                                   10                                   50                                   2500


PS C:\Windows\system32>
```

Figure 4.4 Output from PingSqlServerVersion method

EnumProcesses method

This method has three possible signatures, each with one parameter. Table 4.3 lists the one we will cover here. The other two are for you to experiment with. This particular method signature has a Boolean argument, which can be either $true or $false. The $true indicates to exclude System SPIDs and $false, does the opposite and includes System SPIDs. This is simple enough. The call is made the same way as shown in Listing 4.5, which returns properties that can be seen in Figure 4.. As you can see, the information is not exhaustive, but you can get some interesting information out. Also note that not all the columns that come from the output are listed in this Format-Table. You can use Format-List * in place of the Format-Table and see what other information you get.

```
$server = New-Object -TypeName Microsoft.SqlServer.Management.Smo.Server -ArgumentList localhost

$server.EnumProcesses($true)| Format-Table -Auto

$server.EnumProcesses($false) | Format-Table -Auto
```

Listing 4.5

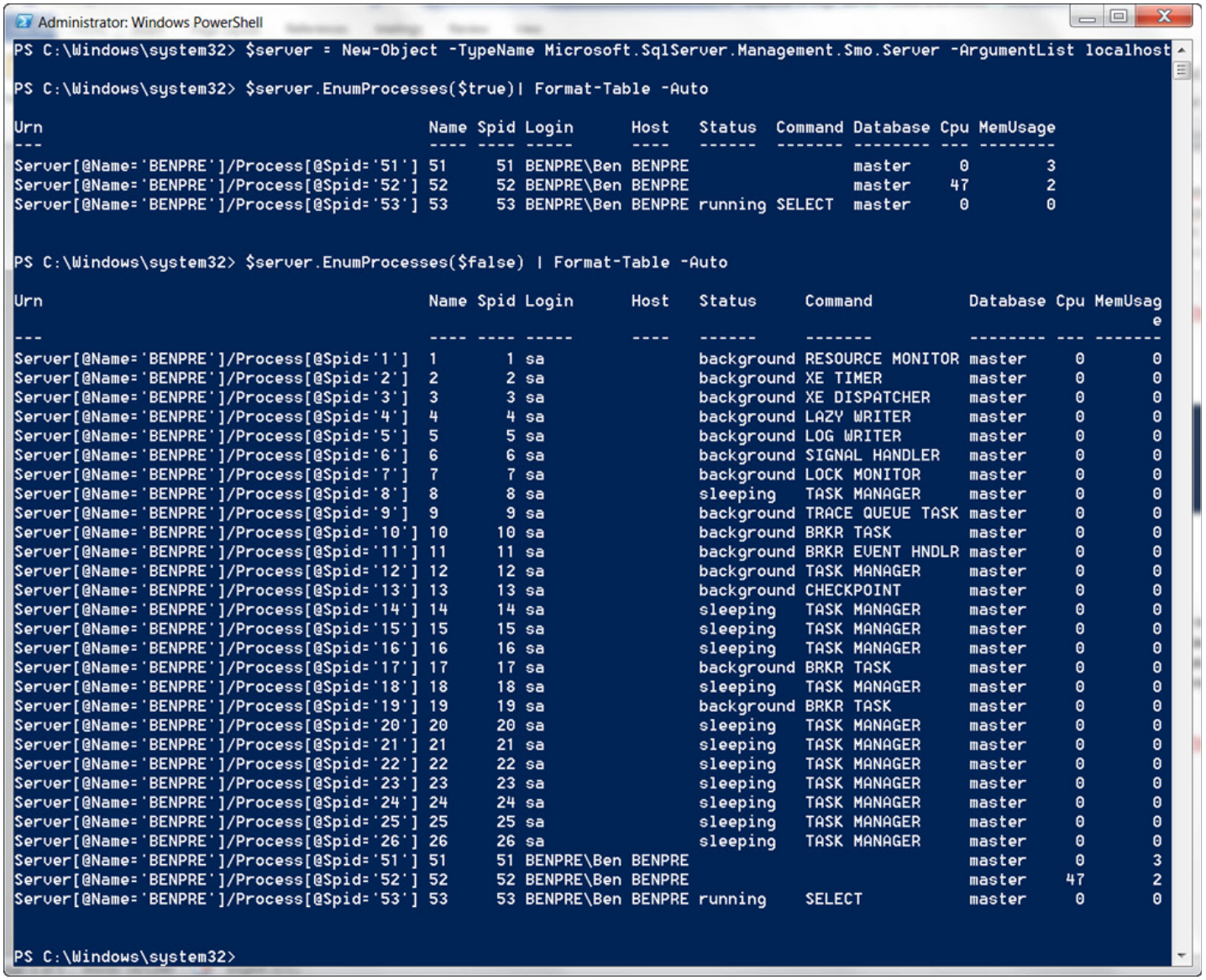

Figure 4.5 Output of EnumProcesses, $true and $false

This should give you a good idea about the use of objects in PowerShell and SQL Server. Remember that everything is an object and all objects will have Methods and Properties. Some objects have more Methods and Properties than others, but that is the nature of objects. Have some fun with methods and properties of these objects, there are a great number of things you can do with this new knowledge.

If you want to keep exploring SQL PowerShell, here are some things that you could experiment with until the next level. Use the knowledge you gained above in how to call a method on an object and learn more about the methods below to solidify what you learned in this level.

Try the method on the Server object called GetActiveDBConnectionCount with a parameter of the database name in a string. Use the Script method with no parameters, and see what you get when you use the Server object and try it against a Database object (hint: $db = Server.Databases["dbname"]).
