SSIS and PowerShell in SQL Server 2012

Rate this article

Matt Masson - MSFT November 17, 2011

19

This post is from Parth Shah, a Software Development Engineer in Test on the SSIS Team.

Previously we have talked about SSIS Catalog Managed Object Model. For those of you don't remember what MOM is or have not heard of it before, think about MOM as a set of APIs that allow you to automate configuring, deploying, validating and executing your projects and packages in a seamless way. We showed an example of how to achieve this through C# but did you know you can achieve the same result through Windows PowerShell?

This post focuses on the usage of the SSIS Catalog Managed Object Model through Windows PowerShell. Over the course of this blog entry, we will go over a couple scenarios, so you can see how to achieve different tasks using PowerShell. For each scenario, we will go over a goal of what we are trying to achieve and then a simple plan of how we are going to achieve it. And finally, we will present the code that does the actual job.

Scenario #1 – Setting up Integration Services Server to execute a package

Plan:

1. Establish a connection to the server.
2. Retrieve the Integration Services object.
3. Create all objects needed to deploy our project.
4. Deploy project and then execute our package.

```
# Load the IntegrationServices Assembly
$loadStatus = [Reflection.Assembly]::Load("Microsoft"+
".SqlServer.Management.IntegrationServices" +
", Version=11.0.0.0, Culture=neutral" +
", PublicKeyToken=89845dcd8080cc91")

# Store the IntegrationServices Assembly namespace to avoid typing it every time
$ISNamespace = "Microsoft.SqlServer.Management.IntegrationServices"

Write-Host "Connecting to server ..."

# Create a connection to the server
$constr = "Data Source=localhost;Initial Catalog=master;Integrated Security=SSPI;"

$con = New-Object System.Data.SqlClient.SqlConnection $constr

# Create the Integration Services object
$ssis = New-Object $ISNamespace".IntegrationServices" $con

## Drop the existing catalog if it exists
# Write-Host "Removing previous catalog ..."
# if ($ssis.Catalogs.Count -gt 0)
# {
#    $ssis.Catalogs["SSISDB"].Drop()
# }

# Provision a new SSIS Catalog
Write-Host "Creating new SSISDB Catalog ..."
$cat = New-Object $ISNamespace".Catalog" ($ssis, "SSISDB", "#PASSWORD1")
$cat.Create()

# Create a new folder
Write-Host "Creating Folder ..."
$folder = New-Object $ISNamespace".CatalogFolder" ($cat, "Folder", "Description")
$folder.Create()

# Read the project file, and deploy it to the folder
Write-Host "Deploying ExecutionDemo project ..."
[byte[]] $projectFile = [System.IO.File]::ReadAllBytes("C:\Demos\Demo.ispac")
$folder.DeployProject("ExecutionDemo", $projectFile)

# Run the package
Write-Host "Running package ..."

# When executing, we need to specify two parameters
# 1 arg is a bool representing whether we want to run
# 32bit runtime on 64 bit server
# 2 arg is a reference to an environment if this package depends on it
$executionId = $package.Execute("false", $null)

Write-Host "Package Execution ID: " $executionId
```

Scenario #2 – Executing complex packages with parameters

Goal: To run a complex package that has parameters that need to filled in

Plan:

1. Repeat everything from previous scenario up to deploying project
2. Fill in the values for parameters by either specifying constants or creating environments

```
# Load the IntegrationServices Assembly
$loadStatus = [Reflection.Assembly]::Load("Microsoft"+
".SqlServer.Management.IntegrationServices" +
", Version=11.0.0.0, Culture=neutral" +
", PublicKeyToken=89845dcd8080cc91")

# Store the IntegrationServices Assembly namespace to avoid typing it every time
$ISNamespace = "Microsoft.SqlServer.Management.IntegrationServices"

Write-Host "Connecting to server ..."
```

```
# Create a connection to the server
$constr = "Data Source=localhost;Initial Catalog=master;Integrated Security=SSPI;"

$con = New-Object System.Data.SqlClient.SqlConnection $constr

# Create the Integration Services object
$ssis = New-Object $ISNamespace".IntegrationServices" $con

## Drop the existing catalog if it exists
# Write-Host "Removing previous catalog ..."
# if ($ssis.Catalogs.Count -gt 0)
# {
#    $ssis.Catalogs["SSISDB"].Drop()
# }

# Provision a new SSIS Catalog
Write-Host "Creating new SSISDB Catalog ..."
$cat = New-Object $ISNamespace".Catalog" ($ssis, "SSISDB", "#PASSWORD1")
$cat.Create()

# Create a new folder
Write-Host "Creating Folder ..."
$folder = New-Object $ISNamespace".CatalogFolder" ($cat, "Folder", "Description")
$folder.Create()

# Read the project file, and deploy it to the folder
Write-Host "Deploying ExecutionDemo project ..."
[byte[]] $projectFile = [System.IO.File]::ReadAllBytes("C:\Demos\Demo.ispac")
$folder.DeployProject("ExecutionDemo", $projectFile)

#### NEW STUFF STARTS FROM HERE ####

# we can specify the value of parameters to be either constants or
# to take the value from  environment variables

$package = $project.Packages[“ComplexPackage.dtsx”]

# setting value of parameter to constant
$package.Parameters["Servername"].Set(
   [Microsoft.SqlServer.Management.IntegrationServices.ParameterInfo+ParameterValueType]::Literal,
   "Foobar");
$package.Alter()

# binding value of parameter to value of an env variable is a little more complex
# 1) create environment
# 2) add variable to environment
# 3) make project refer to this environment
# 4) make package parameter refer to this environment variable
# These steps are shown below

# 1) creating an environment
$environment = New-Object $ISNamespace".EnvironmentInfo" ($folder, “Env1”, “Env1 Desc.”)
$environment.Create()

# 2) adding variable to our environment
# Constructor args: variable name, type, default value, sensitivity, description
$environment.Variables.Add(“Variable1”, [System.TypeCode]::Int32, “10”, “false”, “Desc.”)
$environment.Alter()

# 3) making project refer to this environment
$project = $folder.Projects[$SSISProjectName]
$project.References.Add($SSISEnv, $folder.Name)
$project.Alter()

# 4) making package parameter refer to this  environment variable
$package.Parameters["CoolParam"].Set(
   [Microsoft.SqlServer.Management.IntegrationServices.ParameterInfo+ParameterValueType]::Referenced,
   $SSISEnvVar)
$package.Alter()

# retrieving environment reference
$environmentReference = $project.References.Item($SSISEnv, $folder.Name)
```

```
$environmentReference = $project.References.Item($SSISEnv, $folder.Name)
$environmentReference.Refresh()

# executing with environment reference – Note: if you don't have any env reference,
# then you specify null as the second argument
$package.Execute("false", $environmentReference)

Write-Host "Package Execution ID: " $executionId
```

Tags Denali PowerShell

Comments (19)

Name *

Email *

Website

Post Comment

Ayyappan Thangaraj
November 18, 2011 at 12:37 pm
I like this integration of SSIS with Shell program.

can we use shell program to add datatap?

Reply

Rakesh Parida
November 24, 2011 at 1:22 am
The way to think about powershell API is anything that you can do with MOM can be done with powershell. Since you can add a data tap using MOM, you can do the same using powershell

Reply

Arthur
February 15, 2012 at 11:27 am
I think the example in scenario #1 is not going to run because the $package variable was not initialized and $executionId is not set in the 2nd.

Reply

Chuck
May 15, 2012 at 12:49 pm
Where is Microsoft.SqlServer.Management.IntegrationServices located? I've searched everywhere in a full installation of 2012 RTM and have not found that assembly

Reply

Matt Masson - MSFT
May 15, 2012 at 1:13 pm
You should be able to find it in the SSMS directory – C:Program Files (x86)Microsoft SQL Server110ToolsBinnManagementStudio

Reply

ArthurZ
May 16, 2012 at 8:47 am
It is not there. The DLL appears in GAC though, but not as a file anywhere. There are a few things to keep in mind: the target in your VS may not be .Net 4.0, and the reference you add directly to the project file, then it appears possible to use the IntergationServices class and the rest (as Catalog)

There was a discussion here: http://bit.ly/JTe3Ap that may be beneficial.

Thing is I get warnings switching to 3.5 as my target regarding the Microsoft.SQLServer.ManagedDTS reference in my VS 2010

Reply

Chuck
May 16, 2012 at 9:11 am
Yup – I'm the same Chuck from that thread 😮 Interesting that the dll appears nowhere but the GAC

Reply

Matt Masson - MSFT
May 16, 2012 at 2:07 pm
Ahh, sorry guys – it looks like we changed this in the RTM build. You're correct that it is only found under the 2.0/3.5 GAC (C:windowsassembly)

Reply

Prakash
September 27, 2012 at 5:53 am
Hi Matt,

How can we delete any folder inside catalog using powershell?

Reply

Matt Masson - MSFT
September 27, 2012 at 10:30 am
$ssis.Catalogs["SSISDB"].Folders["Folder"].Drop().

msdn.microsoft.com/.../microsoft.sqlserver.management.integrationservices.catalogfolder.aspx

Reply

Harish
December 3, 2012 at 9:09 pm
Thanks for the script and have got it working for my project. 🙂

One typo though in your example; when adding variables into environment, for sensitivity, instead of "false" it should be $false

Cheers!

Reply

Henrik
April 3, 2013 at 2:22 am
How is this related and compared to using Microsoft.SqlServer.Dts.Runtime namespace for executing SSIS packages? Is this for SQL 2012 only? What other fundamental differencies are they? Thanks!

Reply

Richard
September 26, 2013 at 8:13 am
You refer to variable $SSISEnvVar but this is not set anywhere?

Reply

Steven
November 19, 2013 at 5:34 am
My script runs fine upon the initial setup, but fails when I run script the second time. is there a way to execute the script so that if the project already exists and you deploy it again, it overrides the project, variables, ect and doesn't error out. when the deploy the project the second time, i always get the error, "cannot index into null array"

Reply

Steven
November 19, 2013 at 10:12 am
How do you remove an environment reference?

Reply

Andrew
August 4, 2014 at 7:10 pm
Hi Matt

How do I set a project parameter (rather than a package parameter) using a constant. Thanks.

Reply

Damian Reeves
January 7, 2015 at 10:27 pm
Is the Microsoft.SqlServer.Management.IntegrationServices assembly redistributable?

Reply

Martin
February 3, 2015 at 1:36 am
Hi, this is probably simple, i want to create a new folder. Thought it would be similar to drop. Any suggestions?

Reply

Ann
December 30, 2015 at 9:07 am
Hi Matt! Thanks for the article. Do you know if you will be able to use PowerShell to do an incremental package deployment in SQL 2016? The current BOL doesn't reference PowerShell as an option. I know you can always invoke the SQLCMD and call a SQL stored procedure, but are the direct PowerShell tie ins there?

Reply

Follow Us

Popular Tags

Conferences SSIS 30DaysOfSSIS Katmai Denali Lookup Connectivity Samples Performance API
Script Task SQL2012 Extensions ADO.Net Getting Started SQL Server Expressions Execute SQL Task Tutorial
SMO

Archives