SQL Server 2012 – SSIS Catalog

Creation and Deployment Automation

Contents

[Steps to Deploy a Package to a Catalog using Visual Studio 2015 3](#_Toc535586781)

[1. Introduction 3](#_Toc535586782)

[2. Use Proper SSDT 3](#_Toc535586783)

[3. Create a Catalog in SQL Server 2012 4](#_Toc535586784)

[3.1 Instructions to Create a SQL Server 2012 Catalog: 4](#_Toc535586785)

[4. Deployment of Package 5](#_Toc535586786)

[5. Executing the Package from the Catalog Area 9](#_Toc535586787)

[6. Running the Package from SQL Agent 10](#_Toc535586788)

[7. Deployment Automation 13](#_Toc535586789)

[7.1 Code Demo 14](#_Toc535586790)

[Introduction 14](#_Toc535586791)

[Data Preparation (Example) 14](#_Toc535586792)

[Package Setup and Automation (example) 16](#_Toc535586793)

[SQL Agent (Example) 19](#_Toc535586794)

[Error Handling 21](#_Toc535586795)

[Conclusion 24](#_Toc535586796)

# Steps to Deploy a Package to a Catalog using Visual Studio 2015

## Introduction

The Purpose of this document is to demonstrate how to use the SQL Server 2012 and 2014 SSIS deployment under the concept of “Project Deployment Model “ versus “Package Deployment Model” that was the standard for SQL Server 2005 and 2008.

While in SQL Server 2012, for compatibility reasons, we can use both methods, it appears the catalog shall be the proposed standard from now on.

From Microsoft: *“Integration Services supports two deployment models, the project deployment model and the legacy package deployment model. The project deployment model enables you to deploy your projects to the Integration Services server.”*

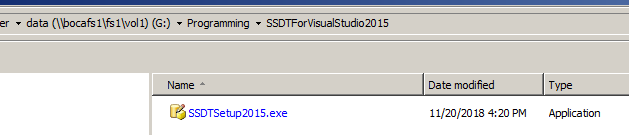
***While this document shows step by step for Project Deployment Model via package creation and deployment, the section “7. Deployment Automation” explains how to automate the deployment process so that connection string and parameters that are specific for each individual SQL Server may be included in a unique script and properly applied during deployment with no need to manual change or the use of configuration file(s).***

## Use Proper SSDT

There are a few SSDT versions on the internet.

Be aware that SQL Server 2012 will only work with compatible version. If you download the latest version of the SSDT, you will probably face version conflict when executing the package from the catalog.

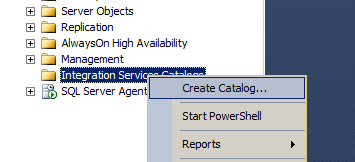
To avoid that, the version that is compatible with SQL Server 2012 was placed on the G: drive as below:



## Create a Catalog in SQL Server 2012

In order to be able to export a SSIS from Visual Studio 2015, you need to create a catalog in SQL Server 2012.

### Instructions to Create a SQL Server 2012 Catalog:

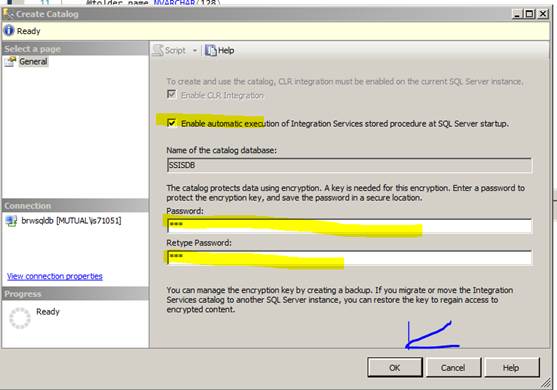


When you create the catalog, it will create a database named SSISDB which cannot be renamed.

For more information on the SSISDB, including naming convention for parameters and environments, please visit the link below:

<https://docs.microsoft.com/en-us/sql/integration-services/catalog/ssis-catalog?view=sql-server-2017>

When creating a catalog, a password is required and I am using “Sponsor123” for now. Also, the check box prompt on the screen must be checked.



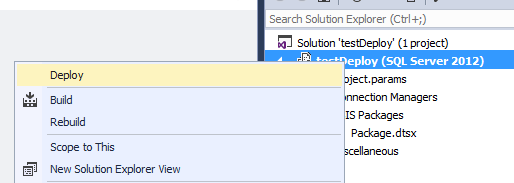
With the catalog created, than you are ready to export your package from Visual Studio 2015 to SQL Server 2012.

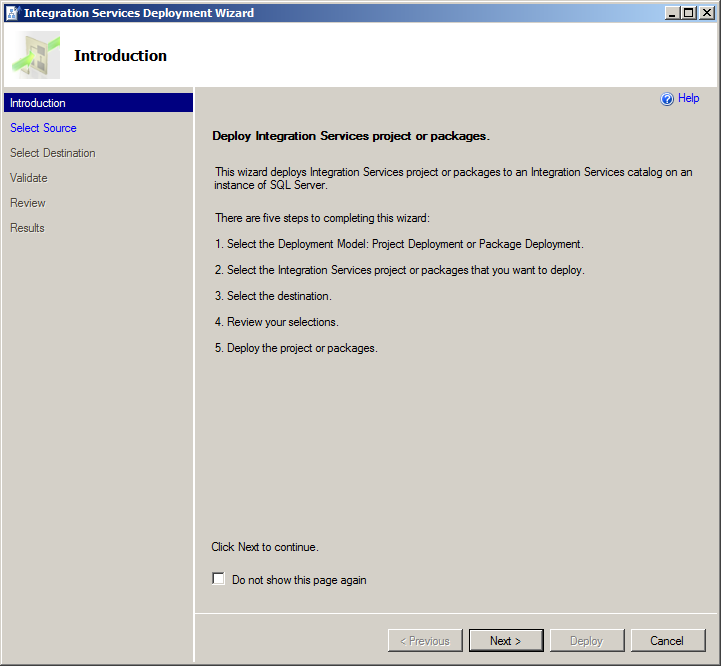
## Deployment of Package

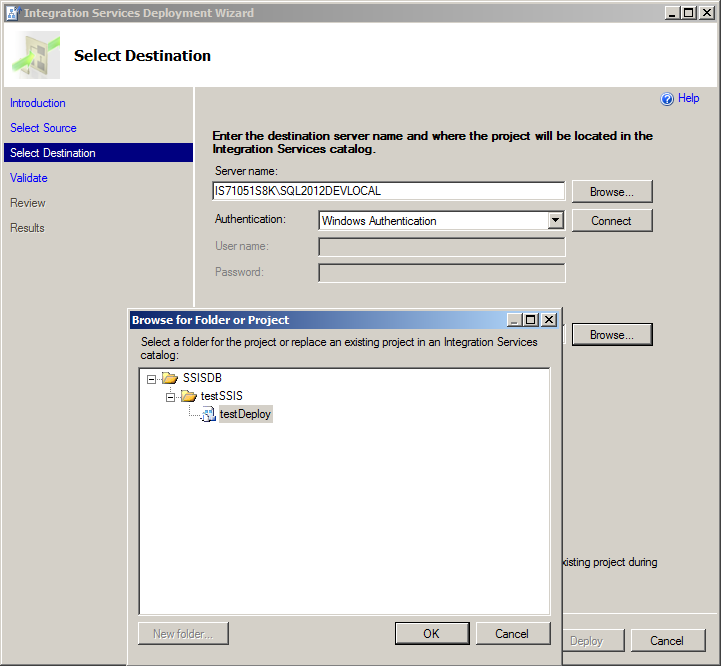
Below are a series of screen shots demonstrating how a package may be manually deployed.

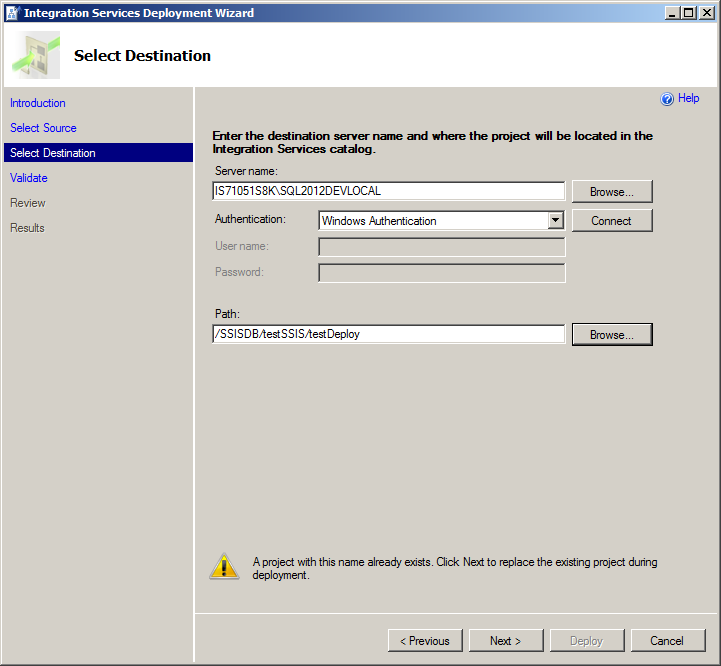
Nevertheless, **the section “8. Deployment Automation” of this document demonstrates how a created package (ISPAC file), may be deployed to all desired SQL Servers in Mutual of America (MOA) with the execution of one T-SQL per server.**

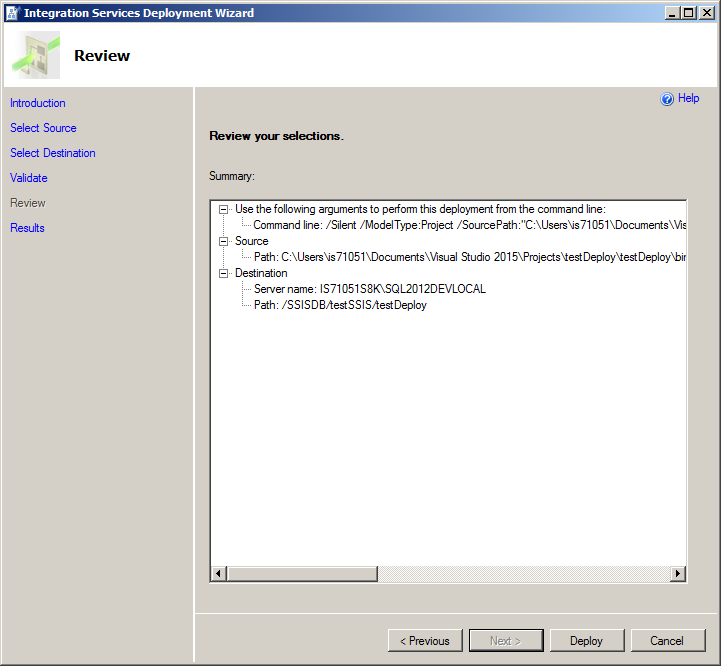
Follow the steps below to deploy your package to an existing catalog

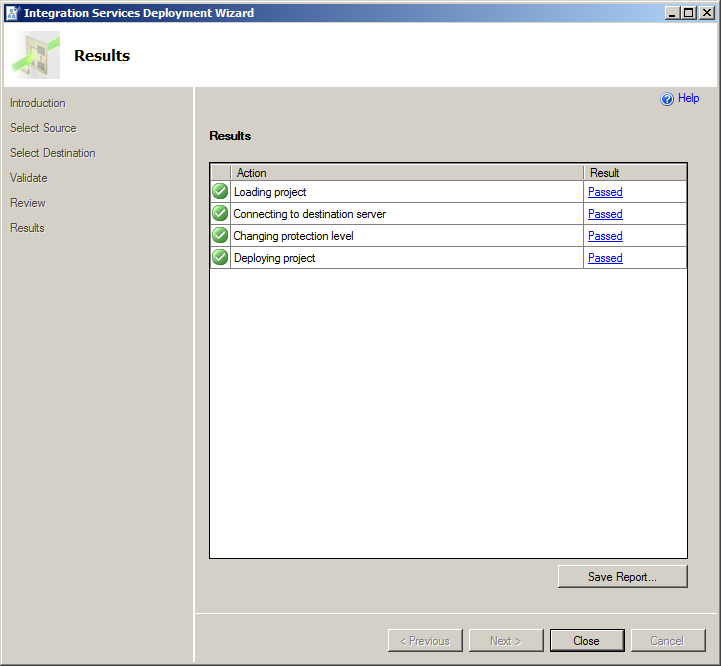








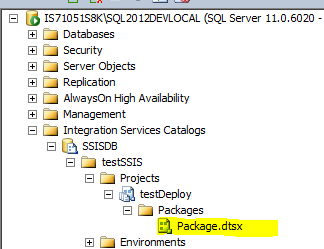


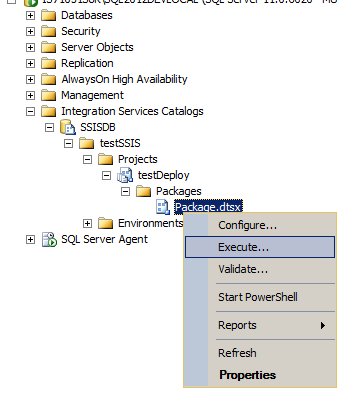


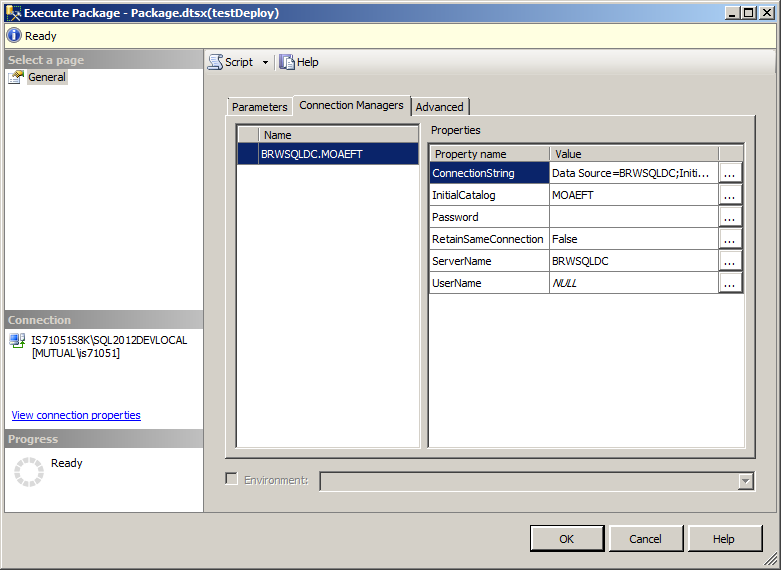
Once the package is exported to a catalog, you can execute it directly from the catalog folder in SSIS 2012.

When it is properly working you can create a job and schedule run time to execute that package.

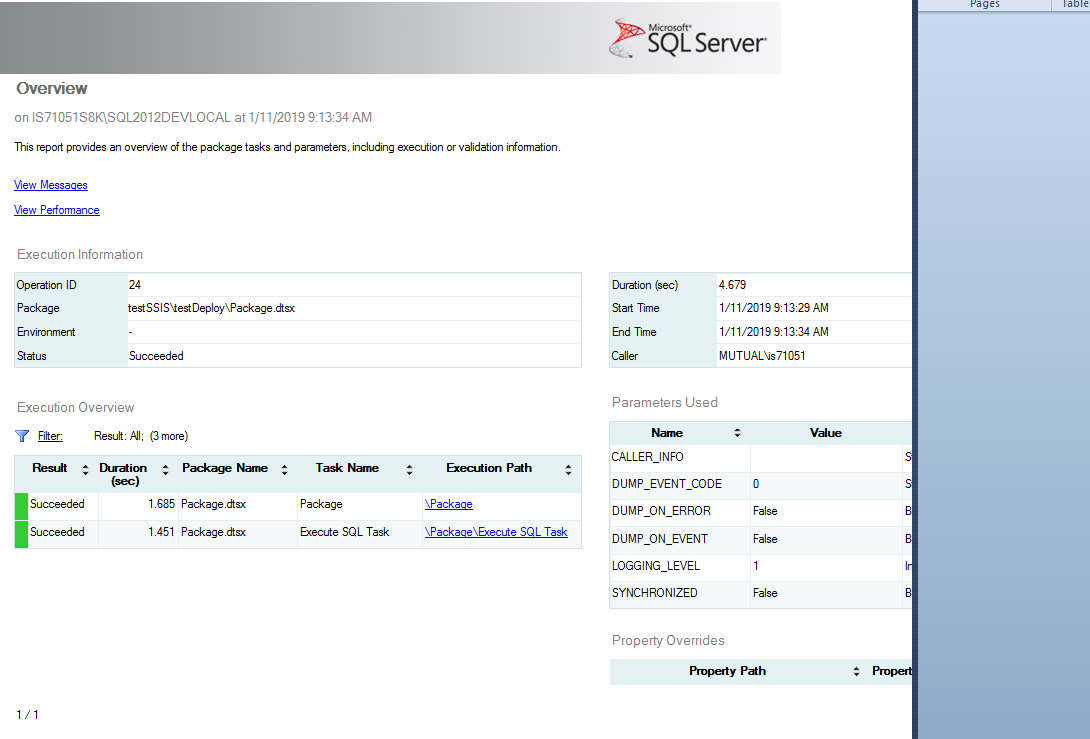
## Executing the Package from the Catalog Area



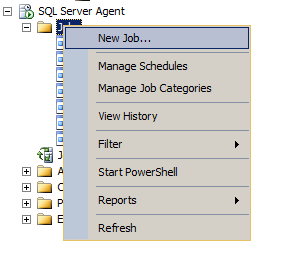


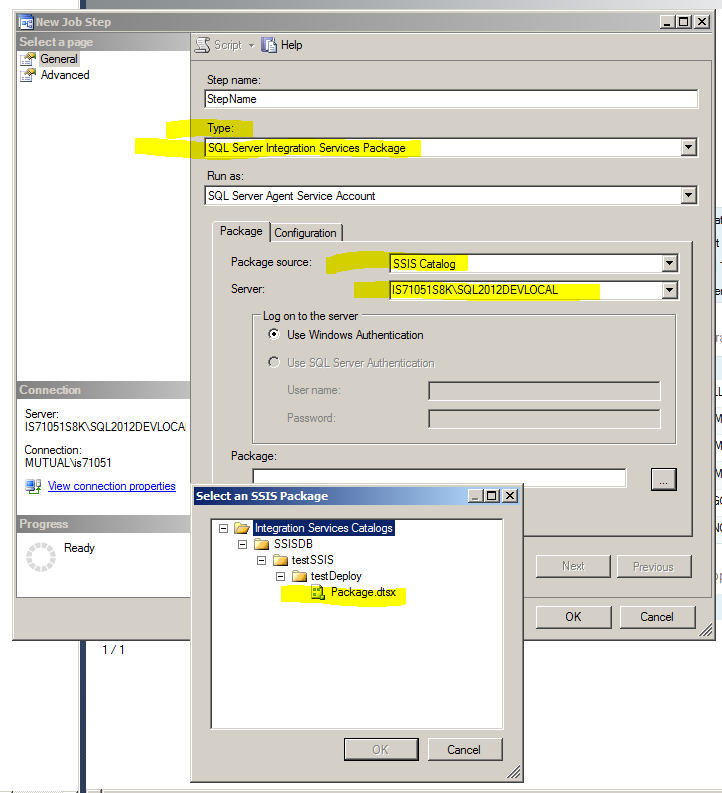


The result when positive or negative is available on the report.



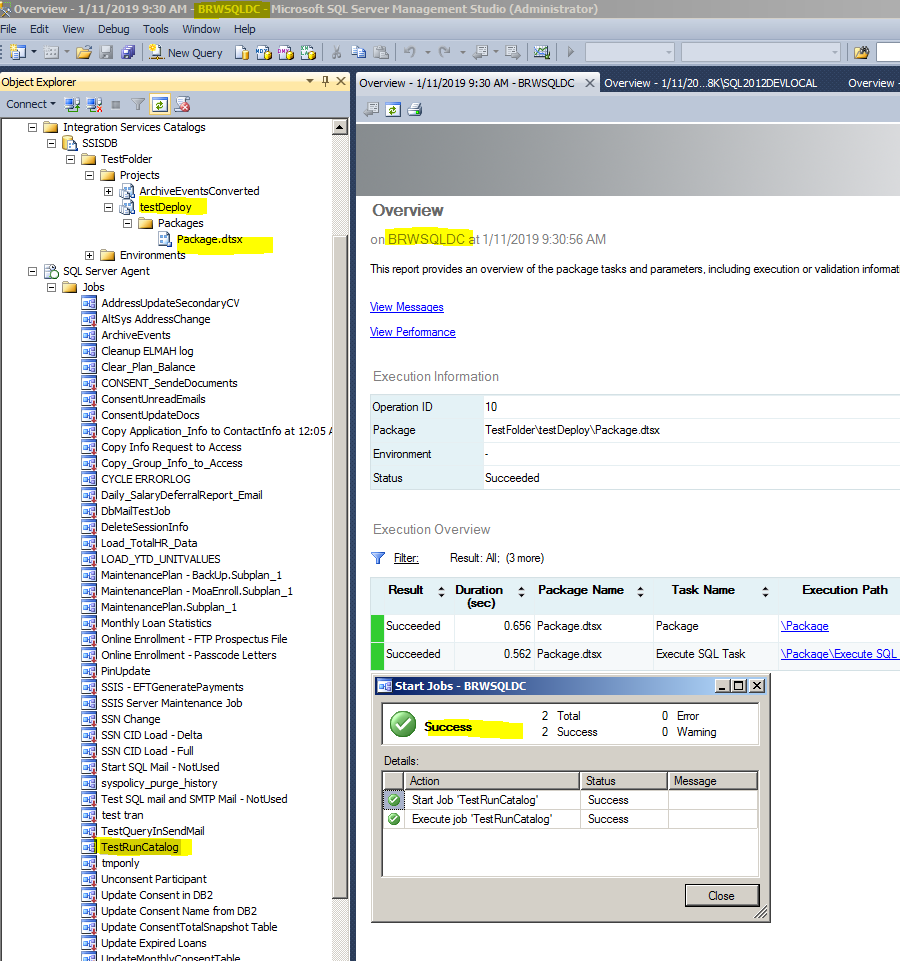
## Running the Package from SQL Agent



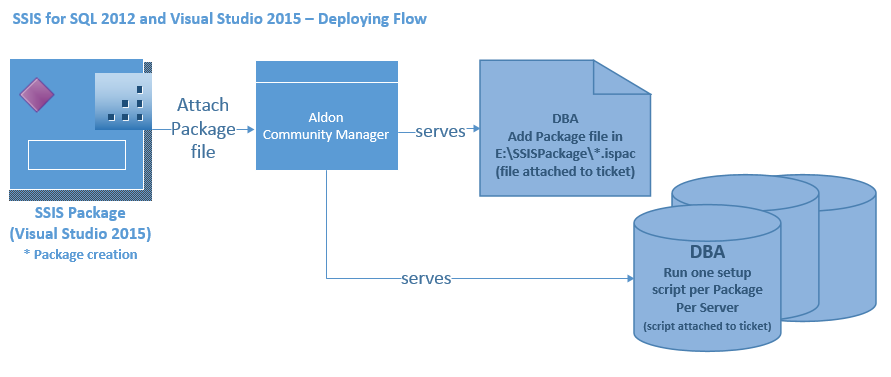


On the Screen below:

1. The Package was created in a developer local PC and from the local, it was exported to BRWSQLDC
2. In BRWSQLDC, the catalog now is ready to use and its information is in the SSISDB database.
3. We created a job named TestRunCatalog , pointing the job to the catalog.
4. Job run successfully as you may see below.
5. It was also executed from within the catalog and run fine as well (see below)



## Deployment Automation



The above diagram depicts:

1. The package is created and tested locally in the developer PC (properly saved in TFS).
2. Once completed and tested, the Package “.ispac” file is added to Aldon Community Manager.
3. A Script is created to install the package in the SQL Server where the script is running from.
4. The script first checks the server name and based on the server name it performs the following steps:
   1. The Script uses the “.ispac” file, the DBA placed in the E:\SSISPackage\ folder in the server and create proper folder in the SQL Server, Integration Services Catalog area.
   2. The Script installs the package inside of the created folder
   3. The Script then updates connection string and server specific parameters based on the server name and update the package with that information.
   4. The Script then creates a JOB for that package in the SQL Agent of that SQL Server.
   5. The Script then creates the step(s) for that package.
   6. DBA applies scheduling as per instructions on the ticket. (this may be automated as well)
5. JOB and steps are verified so the ticket can be closed.

Note: Be aware that the Catalog must be created in the SQL Server prior to executing the script. A Catalog is standard and when created, it creates the SSISDB database that are used by the SSIS 2012 in order for the Project Deployment model to work.

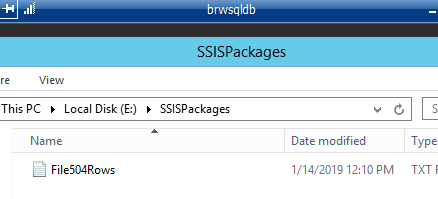
## Code Demo

### Introduction

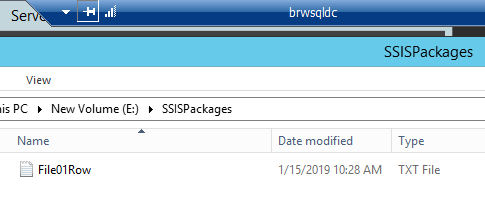
* The demo is developed based on an importation of a txt file to a table.
* The importation will be done in the BRWSQLDB and BRWSQLDC servers, respectively.

### Data Preparation (Example)

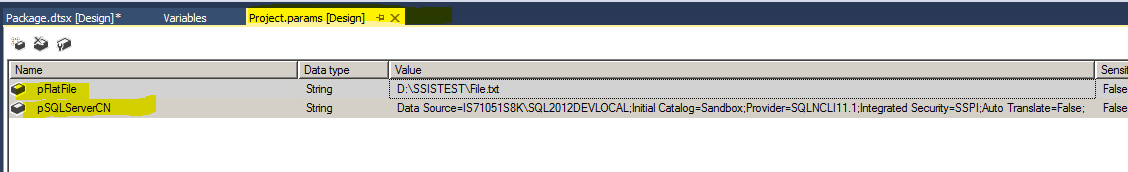
* As a part of the deployment process, Aldon Content Management will get a Package with extension “\*.ispac” and that package will be placed in the folder e:\SSISPackages, on both servers.
* For this demo purpose, we will add the file to be imported also on the servers and for demo purpose also in the e:\SSISPackages.
* Therefore, the BRWSQLDB now contains the file to be imported. (See below) the file has 504 rows so we will confirm that after package runs.



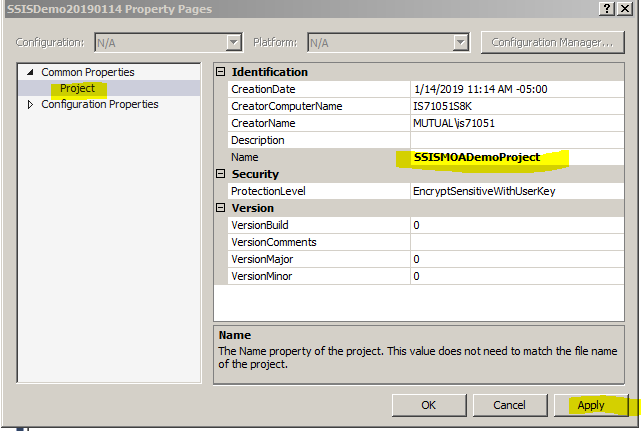
* The BRWSQLDC now contains the file to be imported. (See below) the file has 1 row so we will confirm that after package runs.



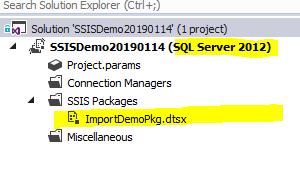
* The above two files portray a scenario where variable change from one server to another.
* The Package in Visual Studio has two project parameters, one for the file location and name and another one for the connection string. (See below)



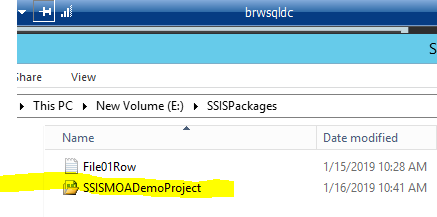
* The above parameters are the ones we will change based on the server name.
* The Project Name may be changed as new by renaming it in the project properties (See below)

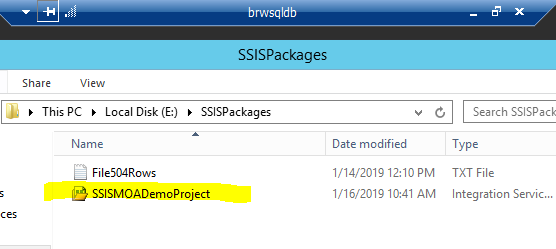


* Note that the package name will be the same for all servers. What is going to change will be the Project Parameters. We will show how to change the parameters very soon. For now, please see below the name of the package created. Be aware the package must be compiled to run in SQL Server 2012. It may not be your default and if it is not, you will need to change that as well.



* Now, simulating the work of the DBA, from Aldon, we will place the file with extension .ispac in the SQL server defined folder: (See Below)





* So far, we have prepared the SSIS Project and creating a file to import and place them in the SQL Servers. All that work will be performed from within Aldon Community Manager since the SSIS project file will be there and the file to be imported, well! That may be sitting in any place on the network and the script will specifically state where it is. Here, as written previously, is for demo purpose.

### Package Setup and Automation (example)

Code below Part 1 of 2 Will set the package and the configuration of the connection string and file location specifically to each server

-- Consider code commented only if needed

--According to KB3000850, this affects domain-joined Windows 8.1 computers after installing security update 2992611 or update 3000850.

--Add new string value named ProtectionPolicy with value = 1

--HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Cryptography\Protect\Providers\df9d8cd0-1501-11d1-8c7a-00c04fc297eb

--ProtectionPolicy = 1 (DWORD)

--Now I can connect.

--I found some other postings about this:

--https://blog.sqlauthority.com/2017/04/01/sql-server-system-security-cryptography-cryptographicexception-error-generating-xml-document/

--https://social.technet.microsoft.com/Forums/windows/en-US/47faab6b-d717-4068-bee4-c694811e0066/credential-manager-problems-error-0x80090345?forum=w8itpronetworking

-- above change not needed

USE SSISDB

GO

DECLARE @FolderName varchar(50)

SET @FolderName = 'ArchiveFolder'

-- \*\*\*

-- if folder was created, then delete it.

IF EXISTS

(

SELECT [name]

FROM [internal].[folders]

WHERE [name] = @FolderName

)

BEGIN

EXEC SSISDB.catalog.delete\_folder @FolderName

END

EXEC SSISDB.catalog.create\_folder @FolderName

-- \*\*\*

-- Set Default values and Project Name that is common to all SQL Servers by SSIS Project

-- Add those inside of the catalog and that will perform the package installation that will be common to all servers

DECLARE @ProjectBinary AS varbinary(max), @ProjectName varchar(80)

DECLARE @operation\_id AS bigint

SET @ProjectName = 'ArchiveConverted'

SET @ProjectBinary =

(SELECT \* FROM OPENROWSET(BULK 'E:\SSISPackages\ArchiveConverted.ispac', SINGLE\_BLOB) AS BinaryData)

EXEC catalog.deploy\_project @folder\_name = @FolderName,

@project\_name = @ProjectName,

@Project\_Stream = @ProjectBinary,

@operation\_id = @operation\_id out

-- \*\*\*

-- with folder and project created set parameter(s) that are specific to each SQL Server

IF (@@SERVERNAME = 'BRHLSQLDD')

BEGIN

EXEC catalog.set\_object\_parameter\_value 20, N'ArchiveFolder', N'ArchiveConverted', N'pMOAEventCN',

N'Data Source=BRHLSQLDD;User ID=hotplusu;Password=hotline2;Initial Catalog=MOAEVENT;Provider=SQLNCLI11.1;Auto Translate=False;RetainSameConnection=True;', N'ArchivePkg.dtsx', 'v'

EXEC catalog.set\_object\_parameter\_value 20, N'ArchiveFolder', N'ArchiveConverted', N'pMOAHistoryCN',

N'Data Source=BRHLSQLDD;User ID=hotplusu;Password=hotline2;Initial Catalog=MOAHistory;Provider=SQLNCLI11.1;Auto Translate=False;RetainSameConnection=True;', N'ArchivePkg.dtsx', 'v'

END

-- Above code reconfigure two parameters such as file name and location and connection string

-- Object\_type => 20

-- folder name => Folder Name

-- project name => Project Name

-- parameter name => Global Parameter Name

-- parameter value => value of parameter (change per server)

-- object name => Name of the package

-- value type => value default

-- below I simulating other servers (from other examples)

IF (@@SERVERNAME = 'BRWSQLDB')

BEGIN

EXEC catalog.set\_object\_parameter\_value 20, N'SSISDemoFolder', N'SSISMOADemoProject', N'pFlatFile', N'E:\SSISPackages\File504Rows.txt', N'ImportDemoPkg.dtsx', 'v'

EXEC catalog.set\_object\_parameter\_value 20, N'SSISDemoFolder', N'SSISMOADemoProject', N'pSQLServerCN',N'Data Source=BRWSQLDB;Initial Catalog=SSISDemo;Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto Translate=False;', N'ImportDemoPkg.dtsx', 'v'

END

IF (@@SERVERNAME = 'BRWSQLDC')

BEGIN

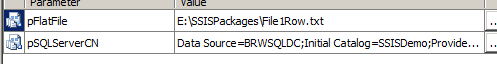
EXEC catalog.set\_object\_parameter\_value 20, N'SSISDemoFolder', N'SSISMOADemoProject', N'pFlatFile', N'E:\SSISPackages\File1Row.txt', N'ImportDemoPkg.dtsx', 'v'

EXEC catalog.set\_object\_parameter\_value 20, N'SSISDemoFolder', N'SSISMOADemoProject', N'pSQLServerCN', N'Data Source=BRWSQLDC;Initial Catalog=SSISDemo;Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto Translate=False;', N'ImportDemoPkg.dtsx', 'v'

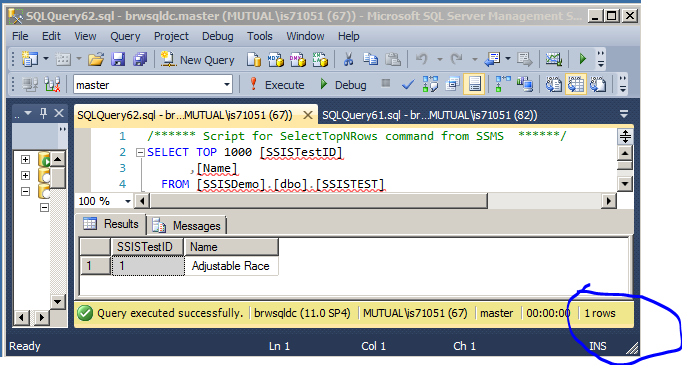
END

See results below from server:

***BRWSQLDC***

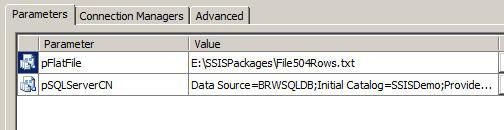


**When the package executes, results shows 1 row demonstrating the parameter from the T\_SQL script was applied as expected.**

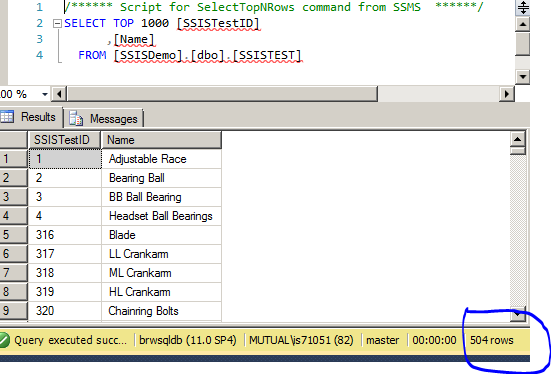


See results below from server:

***BRWSQLDB***



**When the package executes, results shows 504 rows demonstrating the parameter from the T\_SQL script was applied as expected.**



**With the above code and screen shot results, we demonstrated one T-SQL creating and configuring packages from server to server and modifying parameters as needed.**

**The following code may be deployed together with the above code.**

**We are separating them here so you can easily follow the logic in two parts. The First part refers to SSIS package setup, with parameters changed at run time and also with package execution directly on the server. That refers to the code above this paragraph.**

**The Following code (below) is associated with the SQL Agent. By executing it you may be able to create a store procedure to execute the SSIS package, create a job that executes the store procedure created.**

**Query may be dynamically improved, including the setup of schedule. As it is provided today, by executing the code below, it will create a job and a step to that job.**

**Upon successful execution of the query, the job will be created and will be ready to be executed.**

### SQL Agent (Example)

**Create a Store Procedure to Execute a Package**

USE SSISDB

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'P' AND name = 'execute\_ssis\_package\_sample')

DROP PROCEDURE execute\_ssis\_package\_sample

GO

create procedure dbo.execute\_ssis\_package\_sample

(

@FolderName varchar(50),

@ProjectName varchar(100),

@PackageName varchar(100),

@output\_execution\_id bigint output

)

as

begin

declare @execution\_id bigint

exec ssisdb.catalog.create\_execution

@folder\_name = @FolderName

,@project\_name = @ProjectName

,@package\_name = @PackageName

,@execution\_id = @execution\_id output

exec ssisdb.catalog.start\_execution @execution\_id

set @output\_execution\_id = @execution\_id

end

GO

-- \*\*\*

-- Uncoment below to Execute the package

--declare @execution\_id bigint, @output\_execution\_id bigint

--declare @FolderName varchar(50), @ProjectName varchar(100), @PackageName varchar(100)

--SELECT @FolderName = 'ArchiveFolder', @ProjectName = 'ArchiveConverted', @PackageName = 'ArchivePkg.dtsx'

--exec SSISDB.dbo. execute\_ssis\_package\_sample @FolderName, @ProjectName, @PackageName, @output\_execution\_id

-- Create a JOB that will call the store procedure

USE msdb ;

GO

-- Adds a new job that may be executed by the SQLServerAgent

EXEC dbo.sp\_add\_job

@job\_name = N'ArchiveConverted' ;

GO

EXEC dbo.sp\_add\_jobserver

@job\_name = N'ArchiveConverted' ;

GO

exec msdb..sp\_update\_job

@job\_name = 'ArchiveConverted',

@owner\_login\_name = N'hotplusu'

--EXEC dbo.sp\_manage\_jobs\_by\_login

-- @action = N'REASSIGN',

-- @current\_owner\_login\_name = N'MUTUAL\IS71051',

-- @new\_owner\_login\_name = N'hotplusu' ;

GO

EXEC sp\_add\_jobstep

@job\_name = N'ArchiveConverted',

@step\_name = N'Step1',

@subsystem = N'TSQL',

@command = N' DECLARE @output\_execution\_id bigint

exec SSISDB.dbo.execute\_ssis\_package\_sample ''ArchiveFolder'', ''ArchiveConverted'', ''ArchivePkg.dtsx'', @output\_execution\_id',

@retry\_attempts = 5,

@retry\_interval = 5 ;

GO

--Note:

--As it is done today, the created above job needs to be schedule. The scheduling instructions for it will be added in the Aldon Community Manager.

-- consider query below only if needed.

--USE MOAEVENT

--GO

--ALTER DATABASE MOAEVENT set TRUSTWORTHY ON;

--GO

--EXEC dbo.sp\_changedbowner @loginame = N'sa', @map = false

--GO

--sp\_configure 'show advanced options', 1;

--GO

--RECONFIGURE;

--GO

--sp\_configure 'clr enabled', 1;

--GO

--RECONFIGURE;

--GO

--USE MOAHISTORY

--GO

--ALTER DATABASE MOAHISTORY set TRUSTWORTHY ON;

--GO

--EXEC dbo.sp\_changedbowner @loginame = N'sa', @map = false

--GO

--sp\_configure 'show advanced options', 1;

--GO

--RECONFIGURE;

--GO

--sp\_configure 'clr enabled', 1;

--GO

--RECONFIGURE;

--GO

Note:

As it is done today, the created above job needs to be schedule. The scheduling instructions for it will be added in the Aldon Community Manager.

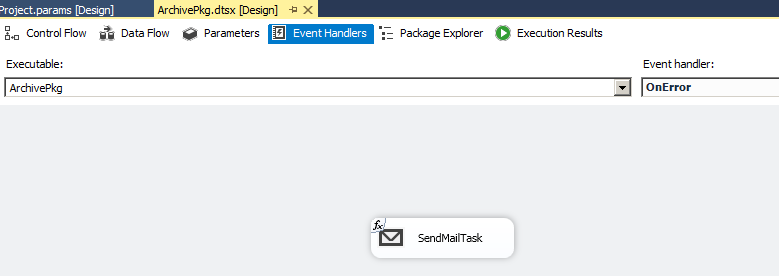
As an improvement of this flow, scheduling may be done in code as well and that may be included whenever possible since it does not take longer to create it and add it to the above script to be executed upon the creation of the package.

## Error Handling

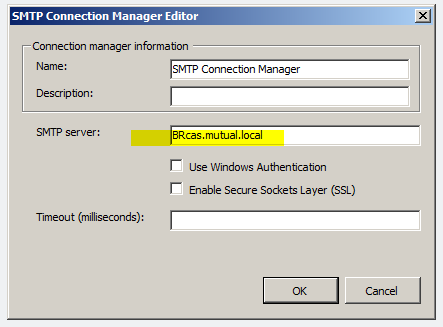
The created packages may have mails being sent to resolve specific particularities of each package. Above that, we recommend a global event handler, to handle package error to properly personal.

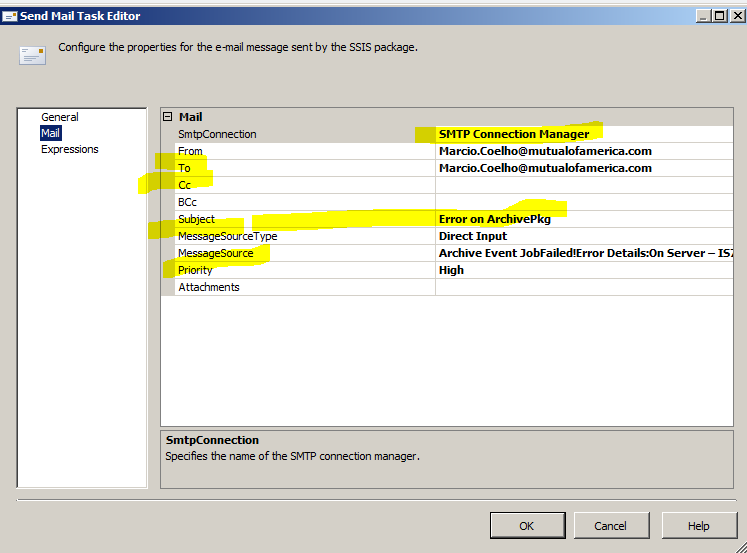
How to implement that?

* On the Event Handlers of the SSIS add a Send Mail Task and configure it to be used when error happens.

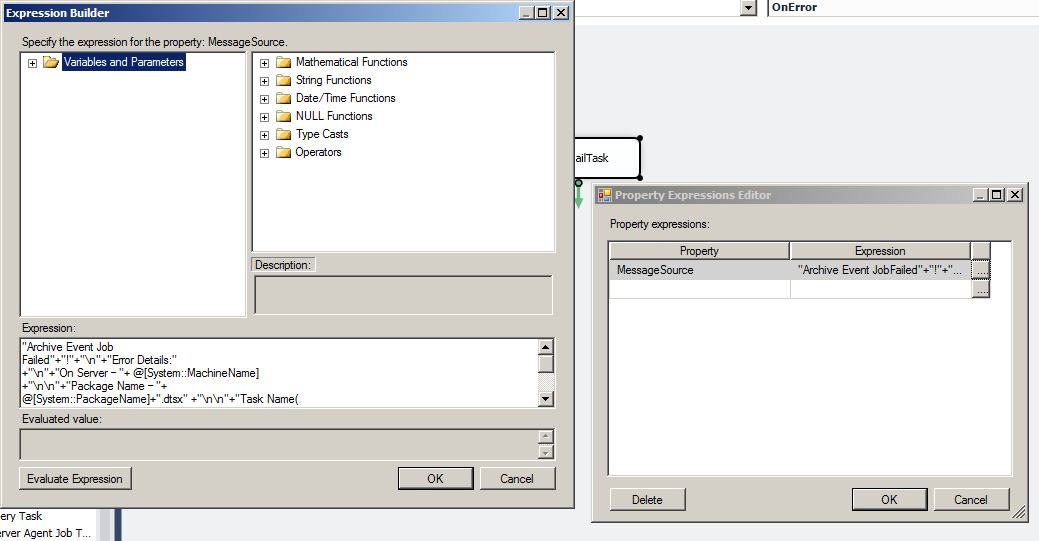


* Properly configure your send mail task after adding an SMTP connection manager on the package.





* Make sure to add an expression to capture the package error and to add it on the body of the email (Message Source).



Note: see code expression below:

"Archive Event Job

Failed"+"!"+"\n"+"Error Details:"

+"\n"+"On Server – "+ @[System::MachineName]

+"\n\n"+"Package Name – "+

@[System::PackageName]+".dtsx" +"\n\n"+"Task Name(

Task where job failed ) – "+ @[System::SourceName]

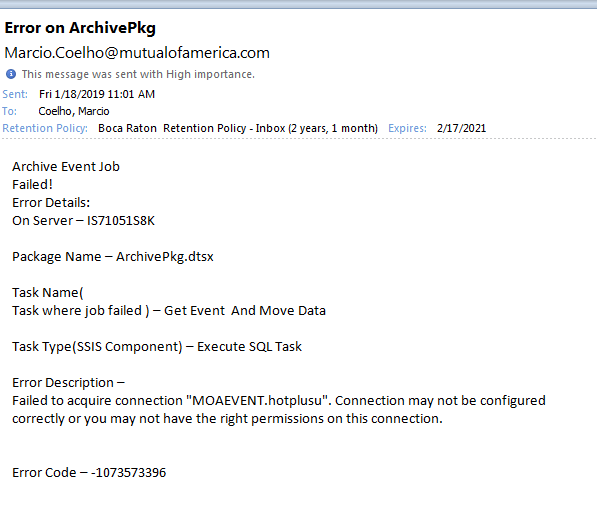
+"\n\n"+"Task Type(SSIS Component) – "+

@[System::SourceDescription]+"\n\n"+ "Error Description –

"+@[System::ErrorDescription]+"\n\n"+ "Error Code – "+

(DT\_WSTR, 12) @[System::ErrorCode]+"\n\n"

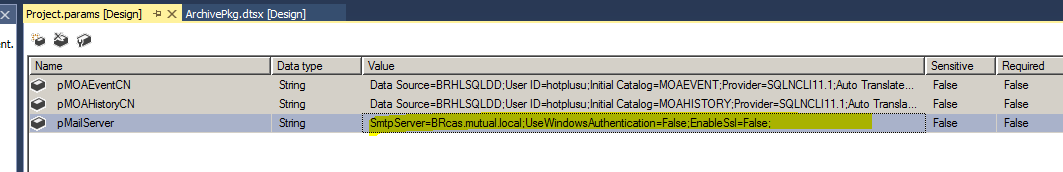
* See the email received below when an error happens:



Note:

The above email and previous descriptions and screen shots demonstrates how to add a global error handler to a package and send email with the error details when it happens.

Last but not least, a parameter is added as the SMTP server so we can change that at run time if needed.



## Conclusion

**All the above code and demonstration was done as a starting point for the SSIS for SQL Server 2012 for package setup and deployment automation. As more logic is involved on the process, this document may be updated and improved.**

**MOA – Technology (January 2019)**