Load Data to an SQL Table from a SharePoint List Using SSIS

Introduction

According to MSDN, SSIS Services is a Microsoft Integration Service and a platform for building enterprise-level data integration and data transformations solutions. Use Integration Services to solve complex business problems by copying or downloading files, loading data warehouses, cleaning and mining data, and managing SQL Server objects and data.

Integration Services can extract and transform data from a wide variety of sources, such as XML data files, flat files, and relational data sources. They can then load the data into one or more destinations.

Prerequisites

- Visual Studio
- SharePoint Online account.
- Basic knowledge of SharePoint.

To achieve our functionality, we have to follow these steps:

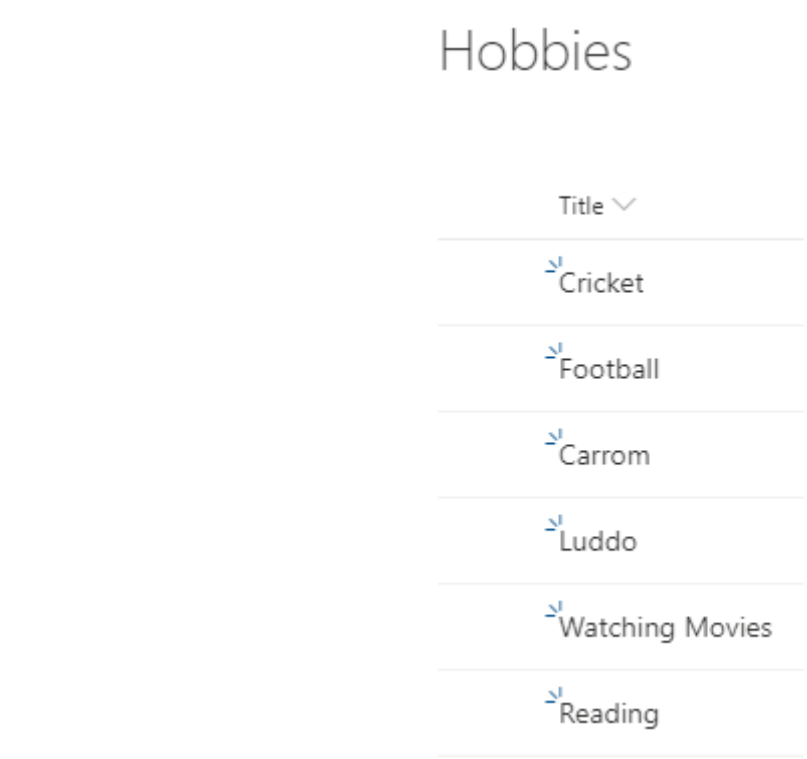
- Create a list and add items to the list.
- Create a database, table, and procedures in SQL Server
- Create an SSIS Project in Visual Studio
- Build and Deploy the solution
- Schedule the SSIS packages.

Let begin.

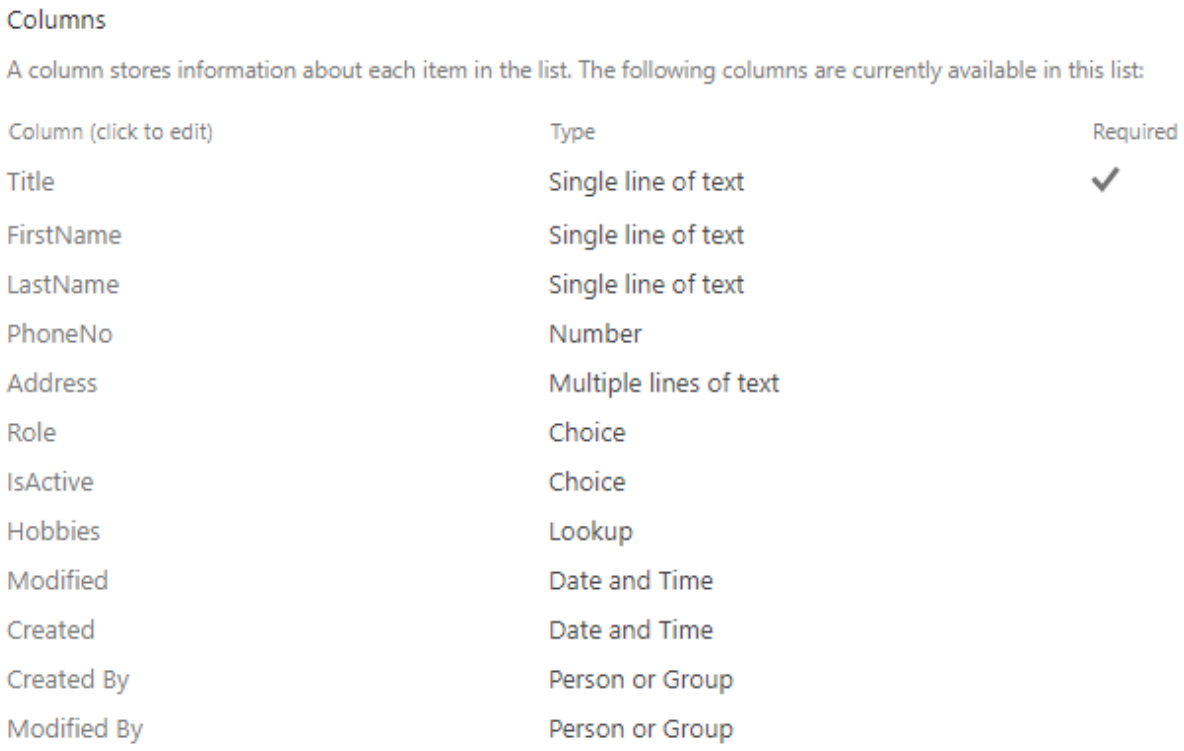
Create a list and add an item in the list

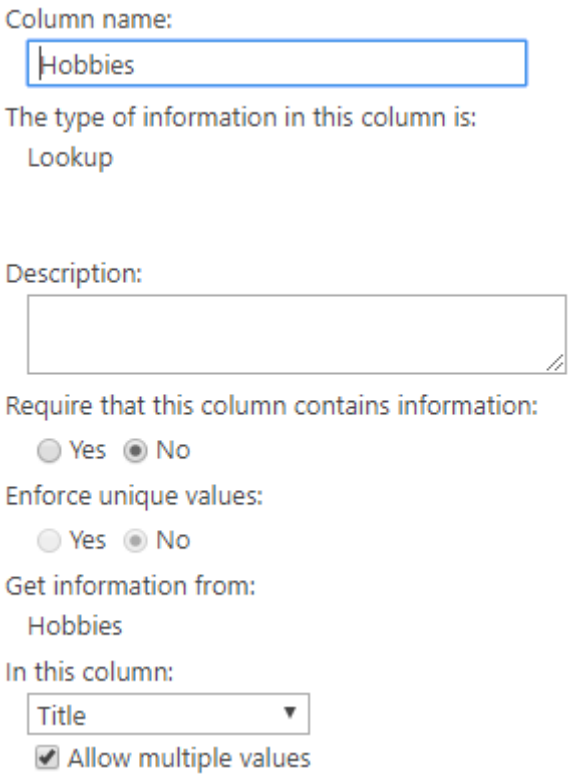
For demonstration purposes, I have created two lists i.e Hobbies and Employee in SharePoint Online.

Hobbies - This list is a Master List



Employee - This list consists of the lookup column (Hobbies) from the Hobbies master list.





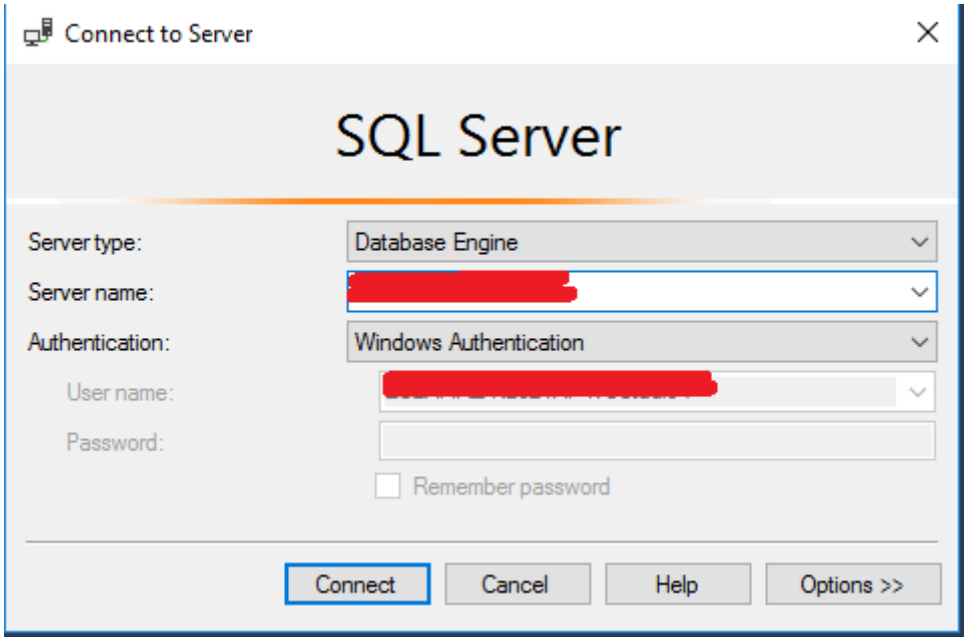
This list consists of the following items.

ID	Title		FirstName	LastName	PhoneNo	Address	Role	IsActive	Hobbies
79	Pravin Kushwaha	...	Pravin	Kushwaha	459,800	Bhopal	Tester	Yes	Cricket; Football; Luddo
87	Arvind Kushwaha	...	Arvind	Kushwaha	90,834,093	Mumbai Maharashtra	Developer	Yes	Cricket; Reading; Watching Movies
88	Randeep Singh	...	Randeep	Singh	3,990,348	Delhi	Developer	Yes	Carrom

Now, we have the records in the list, let's create the database and table to load these records.

Create database, table, and procedures in SQL Server

To create a database, log into the SQL server with the proper credentials.



Step 1

Select *New Query* from the menu and add this script (To execute script press F5)

create database SP_POC

Step 2

Once the database is created successfully, create two new tables for each list to holds employee and hobby records.

Employee List

Employee_Stage

This table will truncate weekly to holds all records for *EmployeeFullPackage.dtsx* as well as daily after every 15 min to holds new records for

EmployeeIncrementalPackage.dtsx

Employee

This table will truncate on weekly basis to holds all records for EmployeeFullPackage and updates daily after every 15 mins for *EmployeeIncrementalPackage.dtsx*

```
1. CREATE TABLE Employee_Stage (
2.     ItemId int,
3.     FullName nvarchar(max),
4.     FirstName nvarchar(max),
5.     LastName nvarchar(max),
6.     PhoneNum int,
7.     Address nvarchar(max),
8.     Role nvarchar(max),
9.     IsActive nvarchar(max),
10.         Hobbies nvarchar(max),
11.         Created datetime,
12.         Modified datetime,
13.         CreatedById int,
14.         ModifiedById int,
15.         CreatedBy nvarchar(max),
16.         ModifiedBy nvarchar(max)
17.     );
18.
19.     CREATE TABLE Employee (
20.         ItemId int,
21.         FullName nvarchar(max),
22.         FirstName nvarchar(max),
23.         LastName nvarchar(max),
24.         PhoneNum int,
25.         Address nvarchar(max),
26.         Role nvarchar(max),
27.         IsActive nvarchar(max),
28.         Hobbies nvarchar(max),
29.         Created datetime,
30.         Modified datetime,
31.         CreatedById int,
32.         ModifiedById int,
33.         CreatedBy nvarchar(max),
34.         ModifiedBy nvarchar(max)
35.     );
```

Hobbies List

Hobbies_Stage

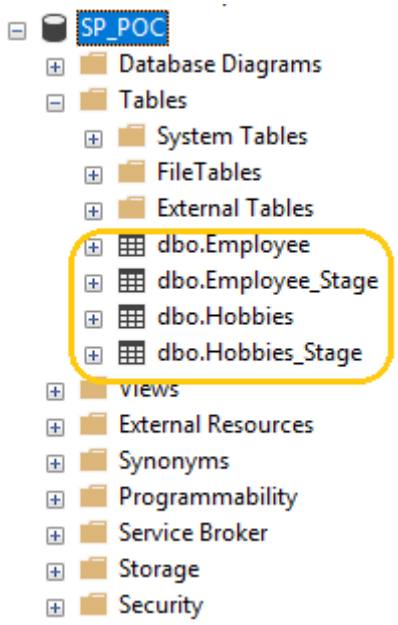
This table will truncate weekly to holds all records for *EmployeeFullPackage.dtsx* as well as daily after every 15 min to holds new records for

EmployeeIncrementalPackage.dtsx

Hobbies

This table will truncate on weekly basis to hold all records for EmployeeFullPackage and updates daily after every 15 mins for *EmployeeIncrementalPackage.dtsx*

```
1. CREATE TABLE Hobbies_Stage (
2.     ItemId int,
3.     Title nvarchar(max),
4.     Created datetime,
5.     Modified datetime,
6.     CreatedById int,
7.     ModifiedById int,
8.     CreatedBy nvarchar(max),
9.     ModifiedBy nvarchar(max)
10.      );
11.
12.      CREATE TABLE Hobbies (
13.          ItemId int,
14.          Title nvarchar(max),
15.          Created datetime,
16.          Modified datetime,
17.          CreatedById int,
18.          ModifiedById int,
19.          CreatedBy nvarchar(max),
20.          ModifiedBy nvarchar(max)
21.      );
```



Step 3

We required multiple stored procedures to insert or update records from stage to main table one for employee (i.e. From dbo.Employee_Stage to Employee) and another for hobbies.

EmployeeFullPackage - It will insert the records into the Employee table.

EmployeeIncrementalPackage - If the item is present in the Employee table, then it will update the records else will create the records.

Stored Procedures

A stored procedure is a group of one or more Transact-SQL statements into logical units, so that the statement can be reused over and over again.

Instead of writing an SQL query again and again, save it as a stored procedure, then just call it to execute it.

Write a new store procedure by right click on *Stored Procedures -> New -> Store Procedure* and paste the below scripts.

Stored Procedure for Employee - *usp_MergeEmployee* (Execute it by pressing F5).

```
1. USE [SP_POC]
2. GO
3. -- ============================================
4. -- Template generated from Template Explorer using:
5. -- Create Procedure (New Menu).SQL
6. --
7. -- Use the Specify Values for Template Parameters
8. -- command (Ctrl-Shift-M) to fill in the parameter
9. -- values below.
10.      --
11.      -- This block of comments will not be included in
12.      -- the definition of the procedure.
13.      -- ============================================
14.      SET ANSI_NULLS ON
15.      GO
16.      SET QUOTED_IDENTIFIER ON
17.      GO
18.      -- =============================================
19.      -- Author:      <Author,,Name>
20.      -- Create date: <Create Date,,>
21.      -- Description: <Description,,>
22.      -- =============================================
23.      CREATE PROCEDURE [dbo].[usp_MergeEmployee]
24.      AS
25.      BEGIN
26.          -
   - SET NOCOUNT ON added to prevent extra result sets from
27.          -- interfering with SELECT statements.
28.          SET NOCOUNT ON;
29.
30.          -- Insert statements for procedure here
31.          BEGIN TRAN
32.          MERGE dbo.[Employee] AS dest
33.          USING dbo.[Employee_Stage] AS sour
34.          ON (dest.ItemID = sour.ItemID)
35.          WHEN MATCHED
36.              THEN UPDATE SET
37.                          dest.[FullName] =  sour.[FullName],

38.                         dest.[FirstName] = sour.[FirstName],
39.                          dest.[LastName] = sour.[LastName],
40.                         dest.[PhoneNum] = sour.[PhoneNum],
41.                          dest.[Address] = sour.[Address],
42.                         dest.[Role]  = sour.[Role],
43.                          dest.[IsActive] = sour.[IsActive],
44.                         dest.[Hobbies] = sour.[Hobbies],
```

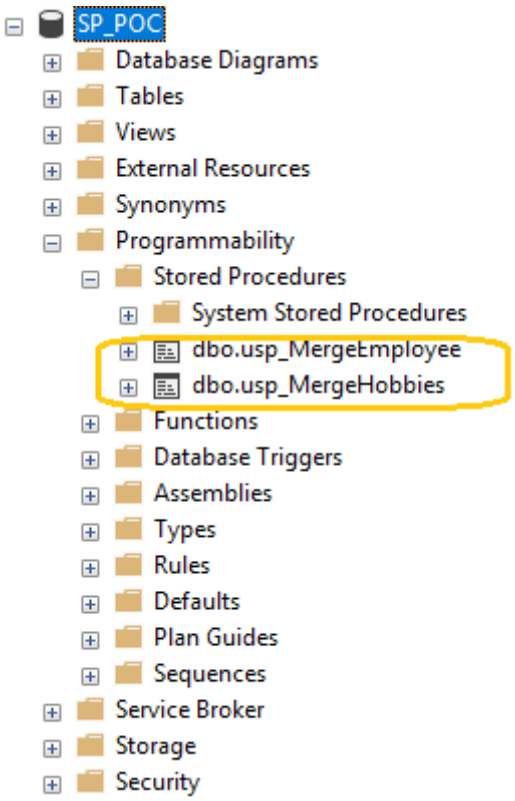
```
                          dest.[Created] = sour.[Created],
                        dest.[CreatedById] = sour.[CreatedById],
                          dest.[Modified] = sour.[Modified],
                        dest.[ModifiedById] = sour.[ModifiedById],
                          dest.[CreatedBy] = sour.[CreatedBy],
                        dest.[ModifiedBy] = sour.[ModifiedBy]
        WHEN NOT MATCHED THEN
            INSERT (
                          [ItemId]
                        ,[FullName]
                          ,[FirstName]
                        ,[LastName]
                          ,[PhoneNum]
                        ,[Address]
                          ,[Role]
                        ,[IsActive]
                          ,[Hobbies]
                        ,[Created]
                          ,[CreatedById]
                        ,[Modified]
                          ,[ModifiedById]
                        ,[CreatedBy]
                          ,[ModifiedBy]
                    )
                VALUES (   sour.[ItemId]
                           ,sour.[FullName]
                             ,sour.[FirstName]
                           ,sour.[LastName]
                             ,sour.[PhoneNum]
                           ,sour.[Address]
                             ,sour.[Role]
                           ,sour.[IsActive]
                             ,sour.[Hobbies]
                           ,sour.[Created]
                             ,sour.[CreatedById]
                           ,sour.[Modified]
                             ,sour.[ModifiedById]
                           ,sour.[CreatedBy]
                             ,sour.[ModifiedBy]
                       )
        OUTPUT $action, Inserted.*, Deleted.*;
        COMMIT TRAN
    END
```

Stored Procedure for Hobbies - *usp_MergeHobbies* (Execute it by pressing F5).

```
USE [SP_POC]
```

```
GO
-- ================================================
-- Template generated from Template Explorer using:
-- Create Procedure (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
      --
      -- This block of comments will not be included in
      -- the definition of the procedure.
      -- ================================================
      SET ANSI_NULLS ON
      GO
      SET QUOTED_IDENTIFIER ON
      GO
      -- =============================================
      -- Author:      <Author,,Name>
      -- Create date: <Create Date,,>
      -- Description: <Description,,>
      -- =============================================
      CREATE PROCEDURE [dbo].[usp_MergeHobbies]
      AS
      BEGIN
          -
  - SET NOCOUNT ON added to prevent extra result sets from
          -- interfering with SELECT statements.
          SET NOCOUNT ON;

          -- Insert statements for procedure here
          BEGIN TRAN
          MERGE dbo.[Hobbies] AS dest
          USING dbo.[Hobbies_Stage] AS sour
          ON (dest.ItemID = sour.ItemID)
          WHEN MATCHED
              THEN UPDATE SET
                          dest.[Title] =  sour.[Title],
                         dest.[Created] = sour.[Created],
                          dest.[CreatedById] = sour.[CreatedById
  ],
                         dest.[Modified] = sour.[Modified],
                          dest.[ModifiedById] = sour.[ModifiedBy
  Id],
                         dest.[CreatedBy] = sour.[CreatedBy],
                          dest.[ModifiedBy] = sour.[ModifiedBy]

          WHEN NOT MATCHED THEN
               INSERT (
                         [ItemId]
                          ,[Title]
```

```
                        ,[Created]
                          ,[CreatedById]
                        ,[Modified]
                          ,[ModifiedById]
                        ,[CreatedBy]
                          ,[ModifiedBy]
                    )
                VALUES (    sour.[ItemId]
                          ,sour.[Title]
                            ,sour.[Created]
                          ,sour.[CreatedById]
                            ,sour.[Modified]
                          ,sour.[ModifiedById]
                            ,sour.[CreatedBy]
                          ,sour.[ModifiedBy]
                        )
        OUTPUT $action, Inserted.*, Deleted.*;
        COMMIT TRAN
    END
```



In this article, we have completed the first two major steps.

- Visual Studio
- SharePoint Online account.
- Basic knowledge of SharePoint.

In my previous article we have completed the first two major steps, now we will target the remaining ones.

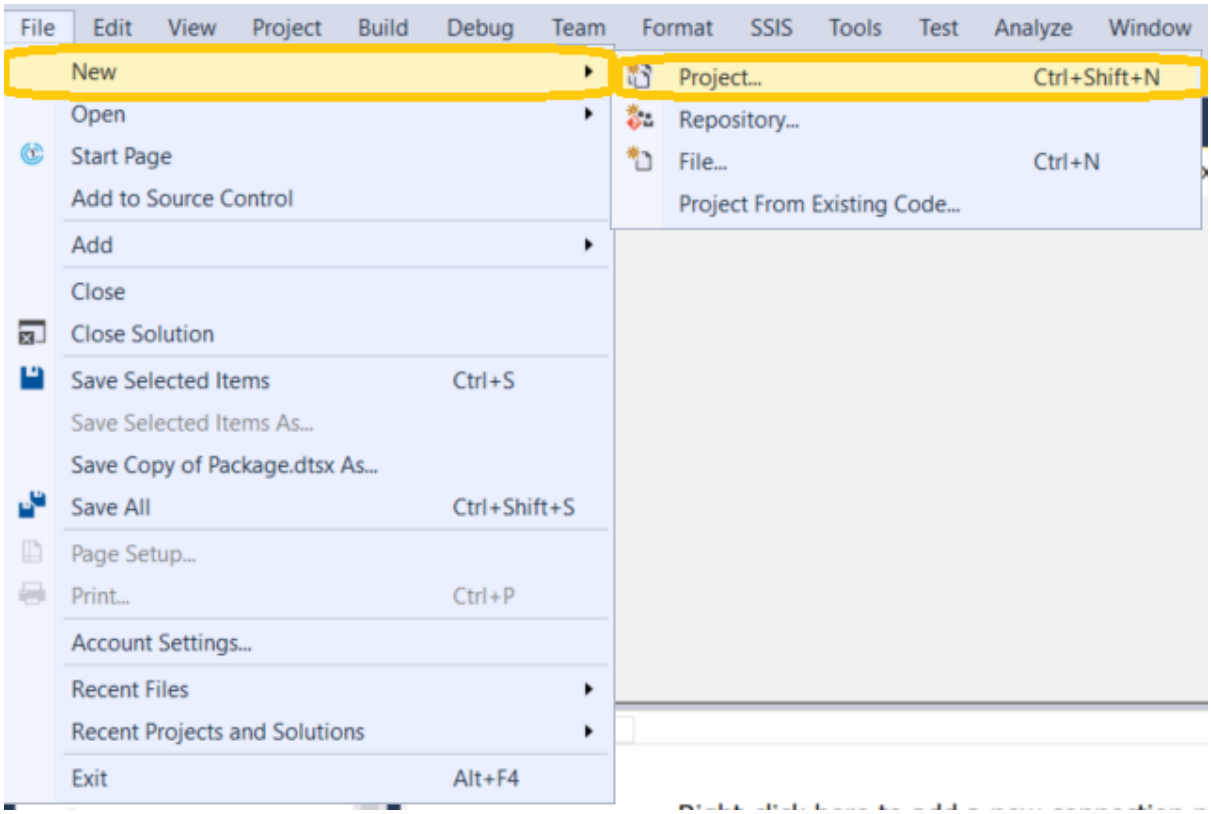
- Create SSIS Profontject in visual studio
- Build and Deploy the solution
- Schedule the SSIS packages.

Create SSIS Project in Visual Studio

Let's start by opening the Visual Studio and running as administrator.

Step 1

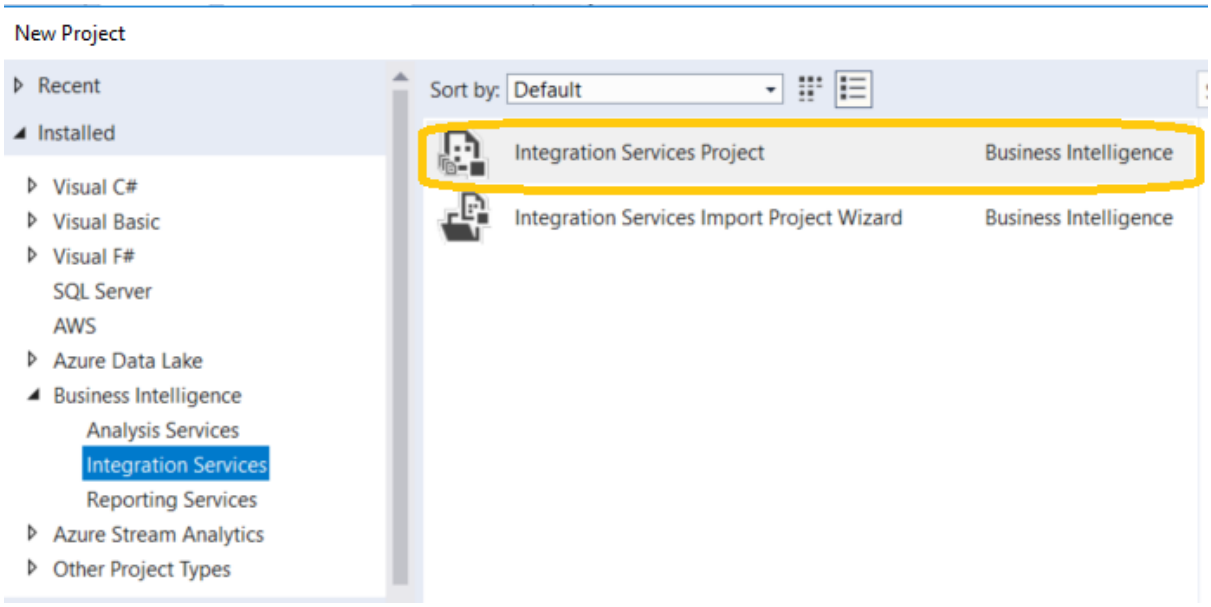
From file menu select *New -> Project*.



Step 2

Under New Project window, select *Business Intelligence -> Integration Services -> Integration Services Project*.

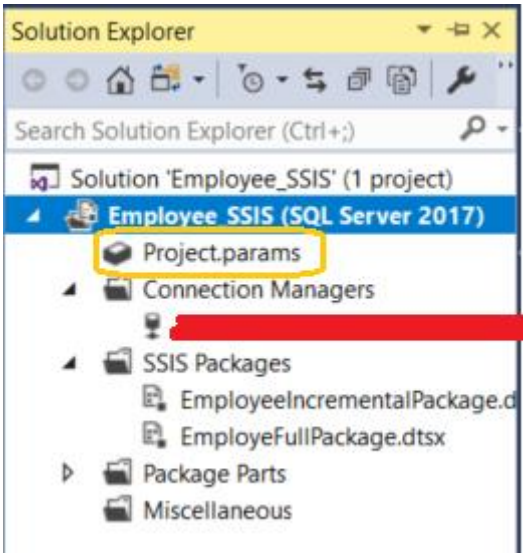
Enter the name of your project (e.g., Employee_SSIS) and select a folder where you want to save the project.



Once the project gets created, let's create the project params and connection in the next two steps.

Step 3

To create project params, click on solution explorer -> Project.params

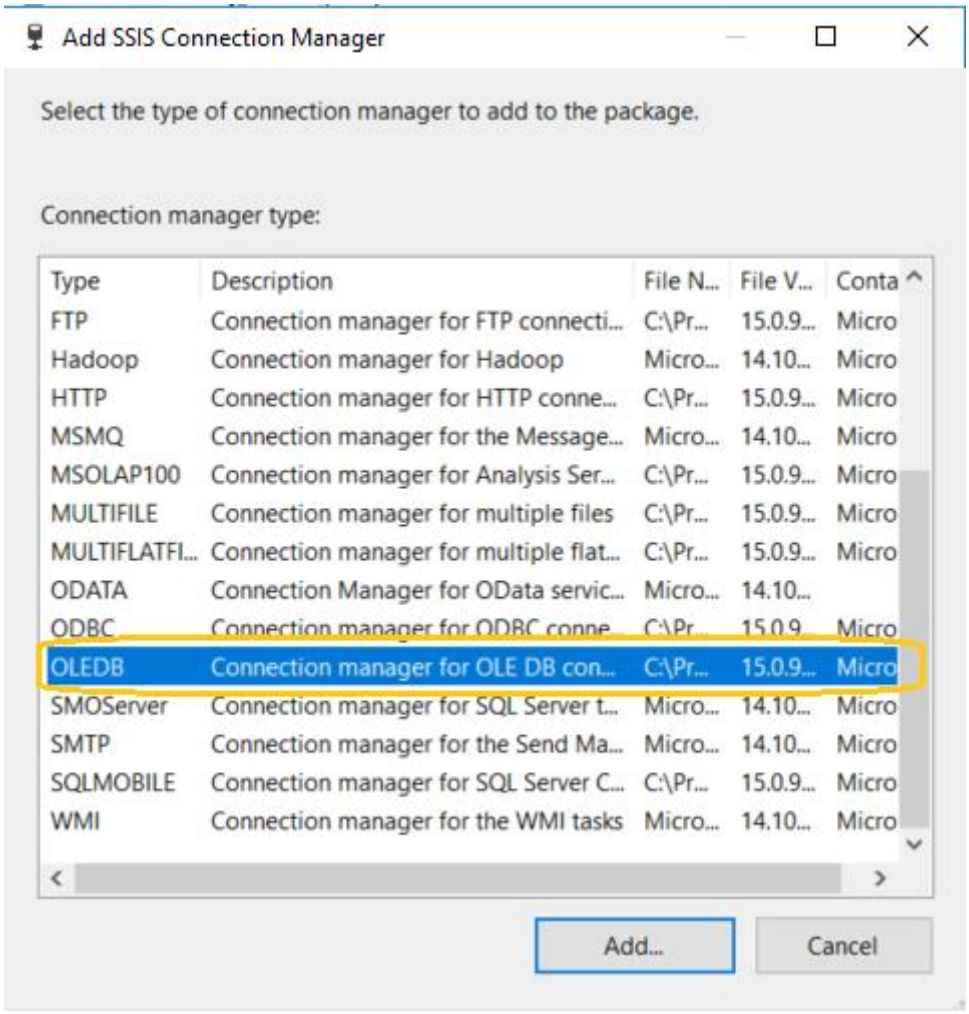


Create at least these four parameters which are required in our projects.

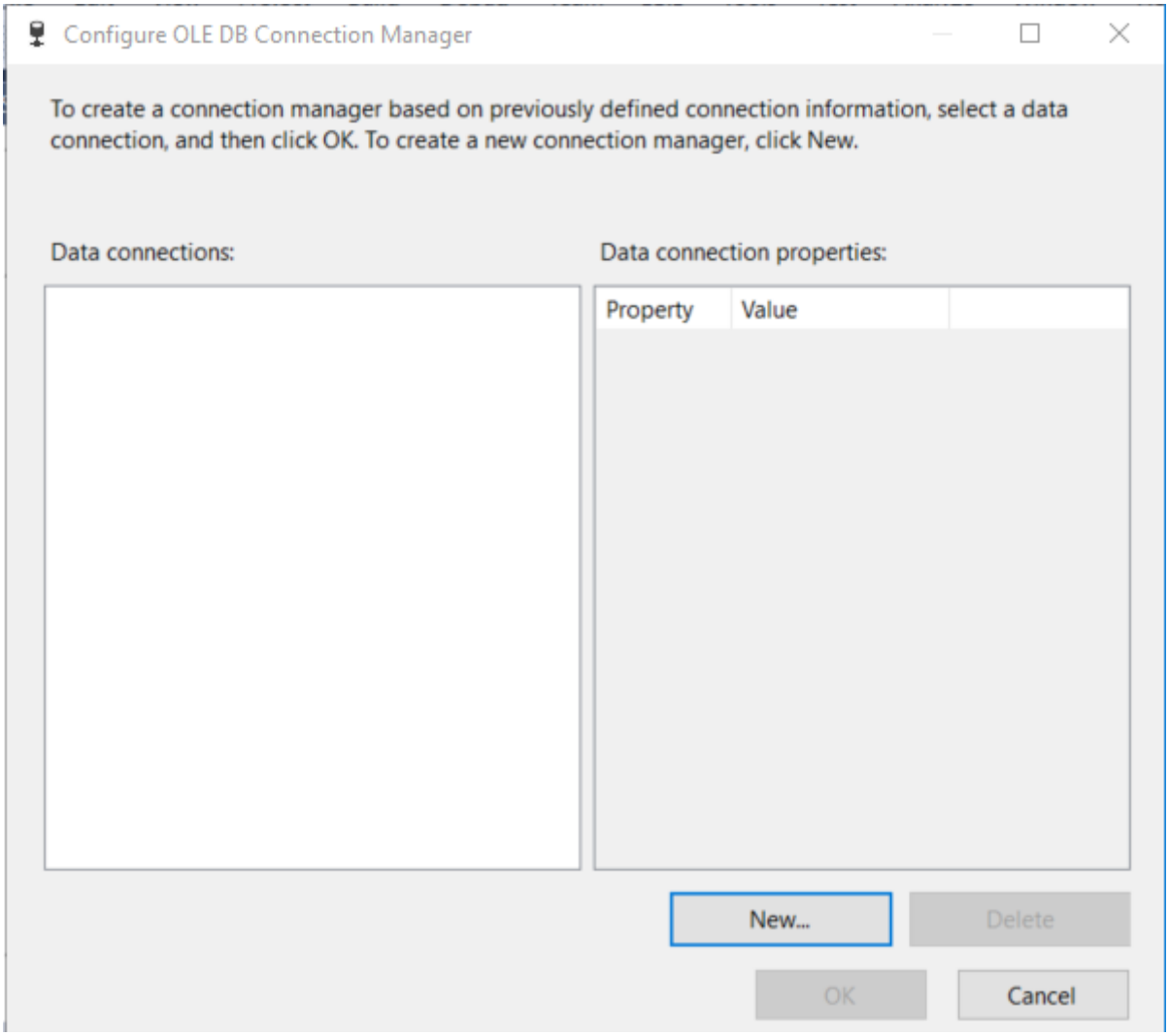
Step 4

Now create the new connection, right click on *Connection Managers- > New Connection Manager*

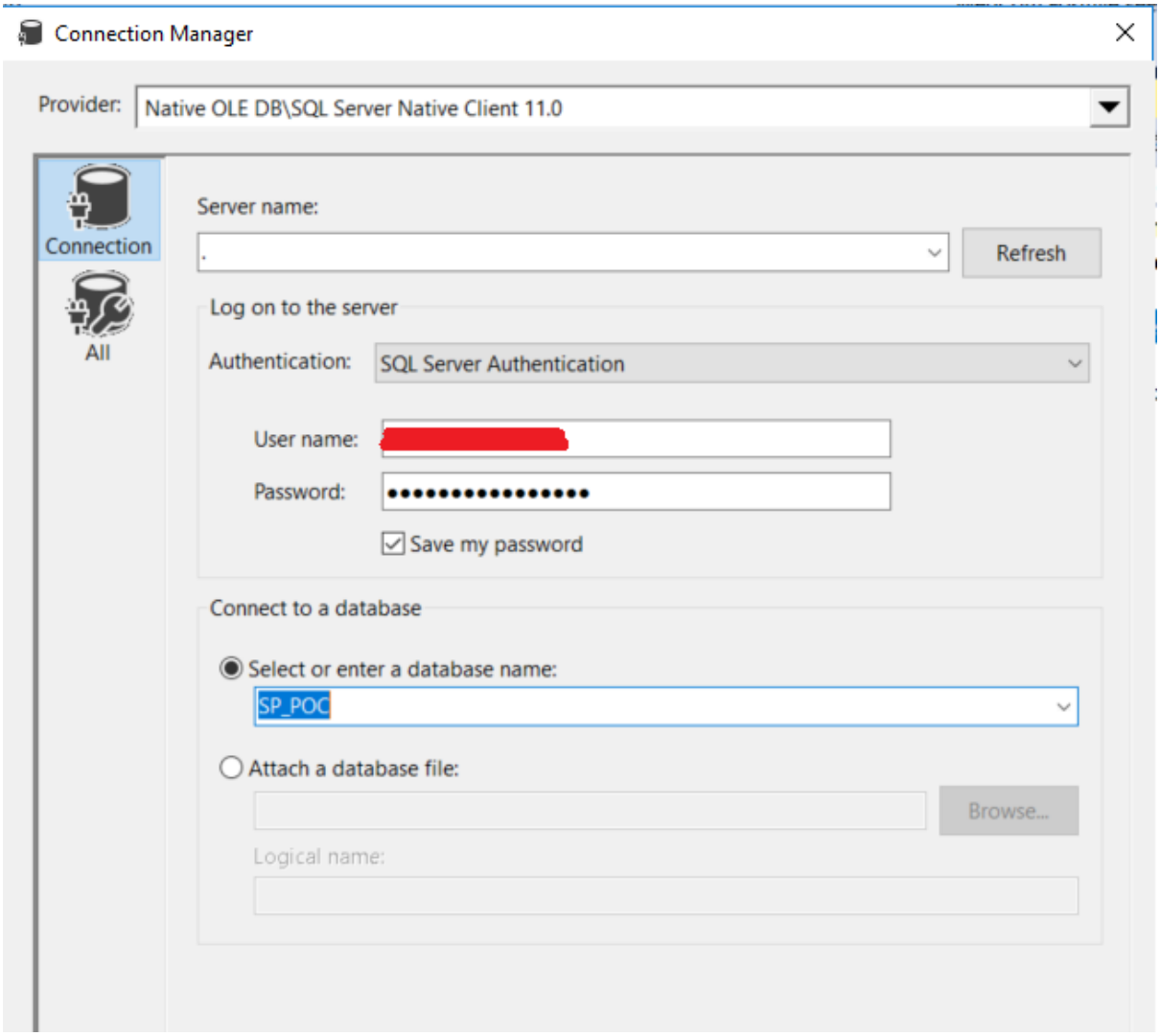
Select *OLEDB* option from type and click on Add button



Click on New button



Select the appropriate Server name, authentication and database, check the connection by clicking on test connection button.



Now we are ready to create SSIS packages

EmployeeFullPackage

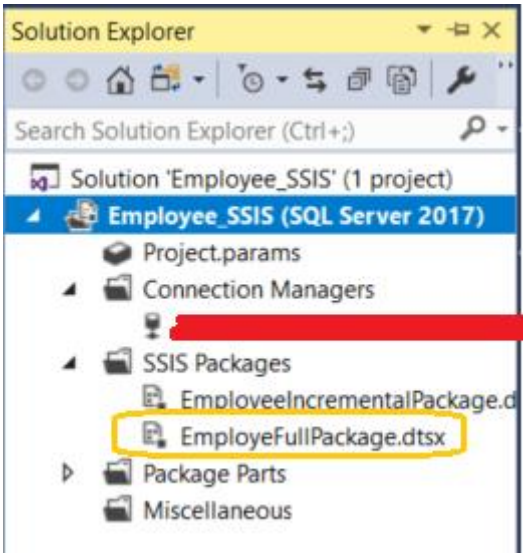
This package will run on a weekly basis (although you can configure it differently).

Functionality

- In the first step, it will truncate all tables; i.e Employee_Stage, Employee, Hobbies_Stage and Hobbies.
- In the second step, it will copy all items from Employee and Hobbies list and store into Employee and Hobbies buffer.
- In the third step, it will add all items from Employee buffer to Employee_Stage table and from Hobbies buffer to Hobbies_Stage table.

- In the last step, it will call the stored procedures *usp_MergeEmployee* and *usp_MergeHobbies* (created in previous article) to copy records from stage to main table.

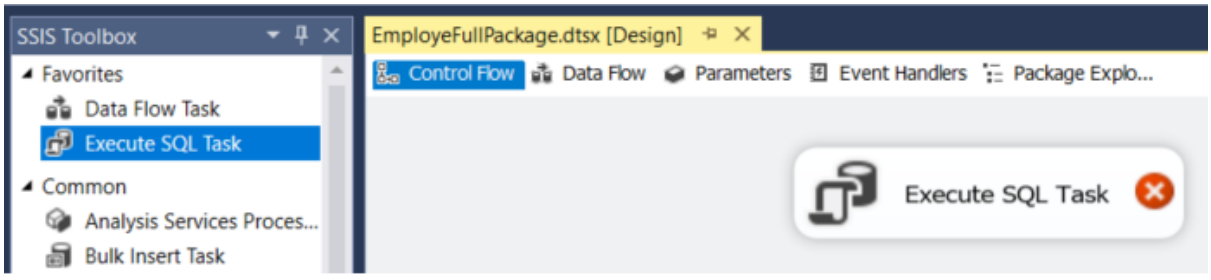
Right click on *SSIS package -> New SSIS package* and name it as EmployeFullPackage.dtsx



Open the EmployeeFullPackage.dstx and drag the relevent component from SSIS Toolbox to control flow.

Control Flow

The control flow task consists of three main tasks; i.e Execute SQL Task, Data Flow Task and Execute SQL Task 1



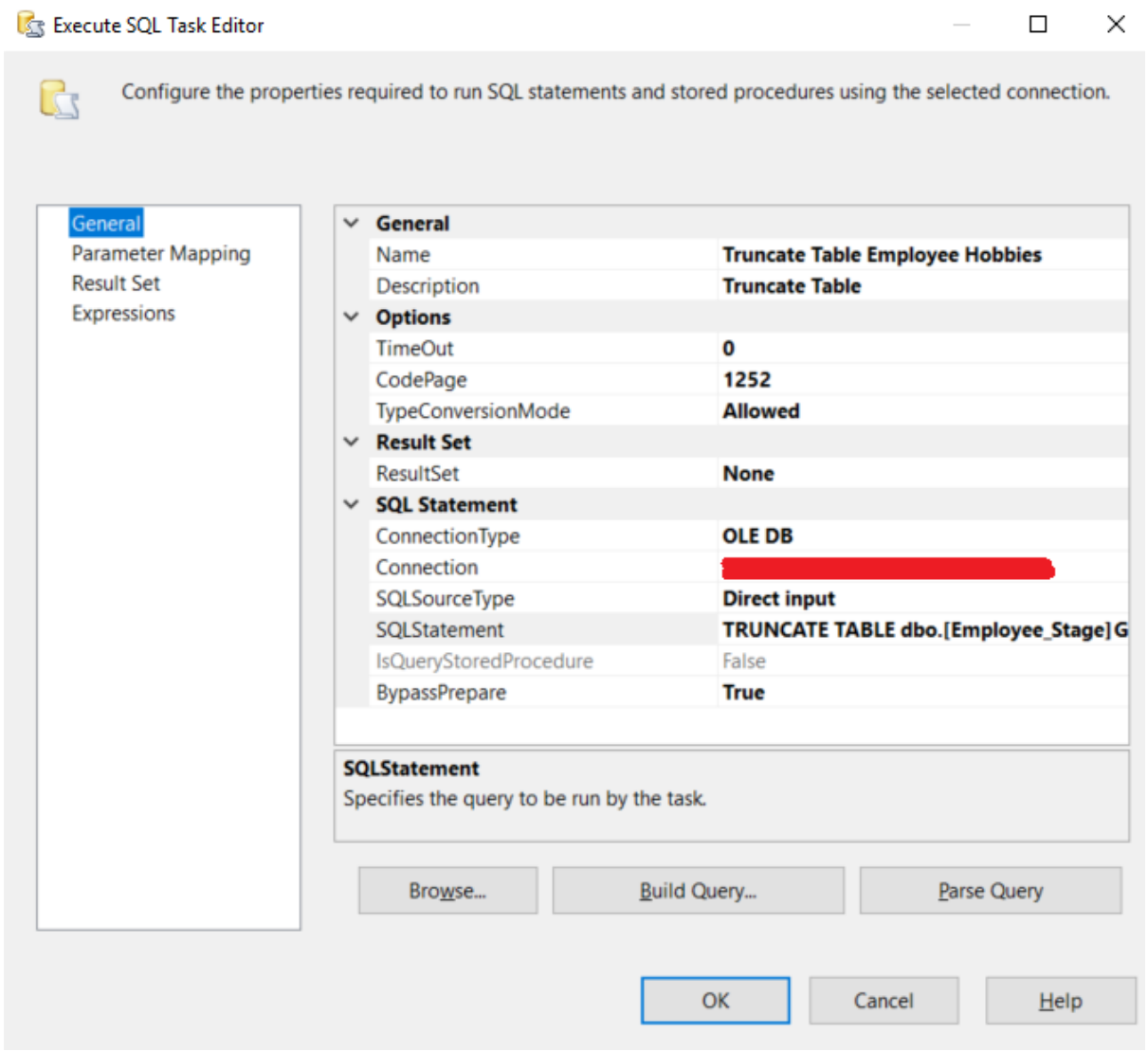
We have selected three components [Data Flow Task(1) and Execute SQL Task(2)] from SSIS toolbox and added into the EmployeeFullPackage.dstx control flow tab.

Execute SQL Task

It is used to execute the SQL statement.

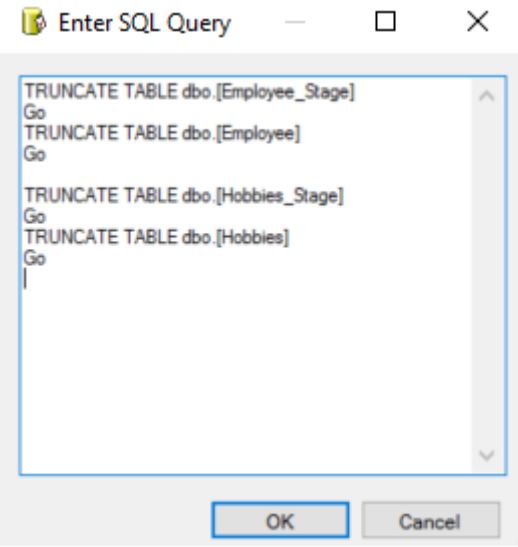
In this task, we will truncate our table i.e. *Employee_Stage, Employee, Hobbies_Stage and Hobbies.*

Open the task, rename the task as *Truncate Table Employee Hobbies* and configure in this way.



Open SQL Statement and paste the following script.

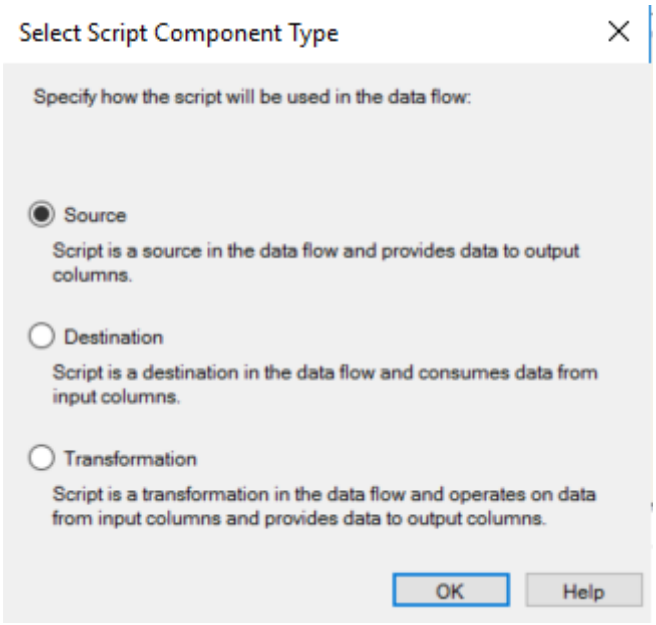
```
TRUNCATE TABLE dbo.[Employee_Stage]
Go


TRUNCATE TABLE dbo.[Employee]
Go

TRUNCATE TABLE dbo.[Hobbies_Stage]
Go


      TRUNCATE TABLE dbo.[Hobbies]
      Go
```

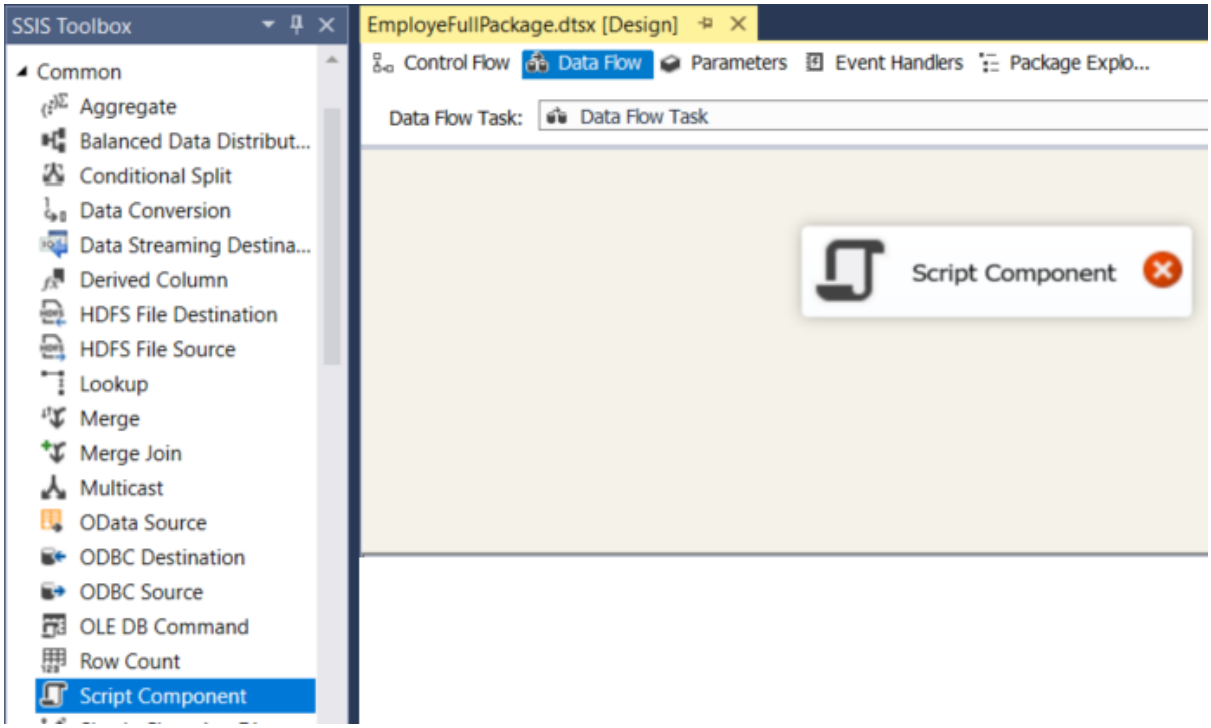


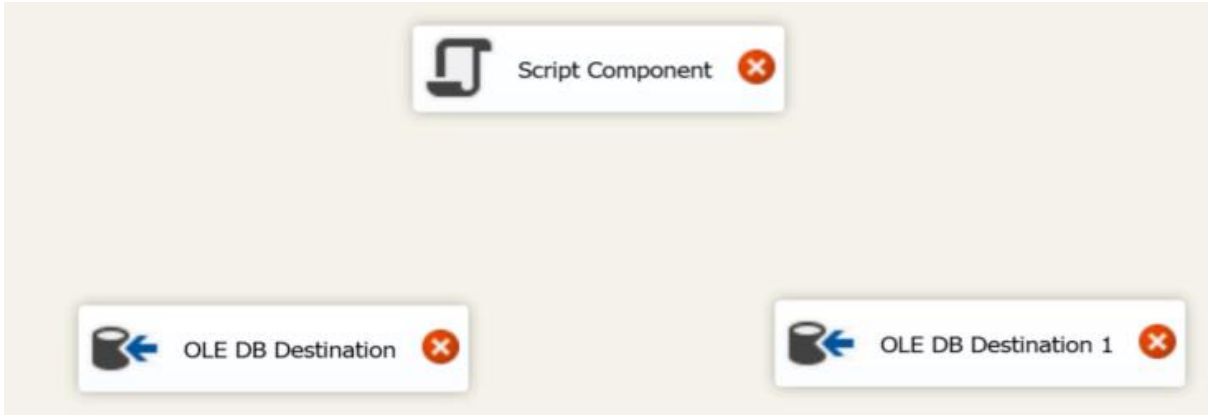
Data Flow Task

In this task, we will get all records from Employee and Hobbies list and add into Employee_Stage and Hobbies_Stage table.

Click on Data flow task and select the relevant component from SSIS toolbox by dragging and dropping to data flow.







For demo purposes, we have selected only three components [Script Component(1) and OLE DB Destination(2)] in data flow task.

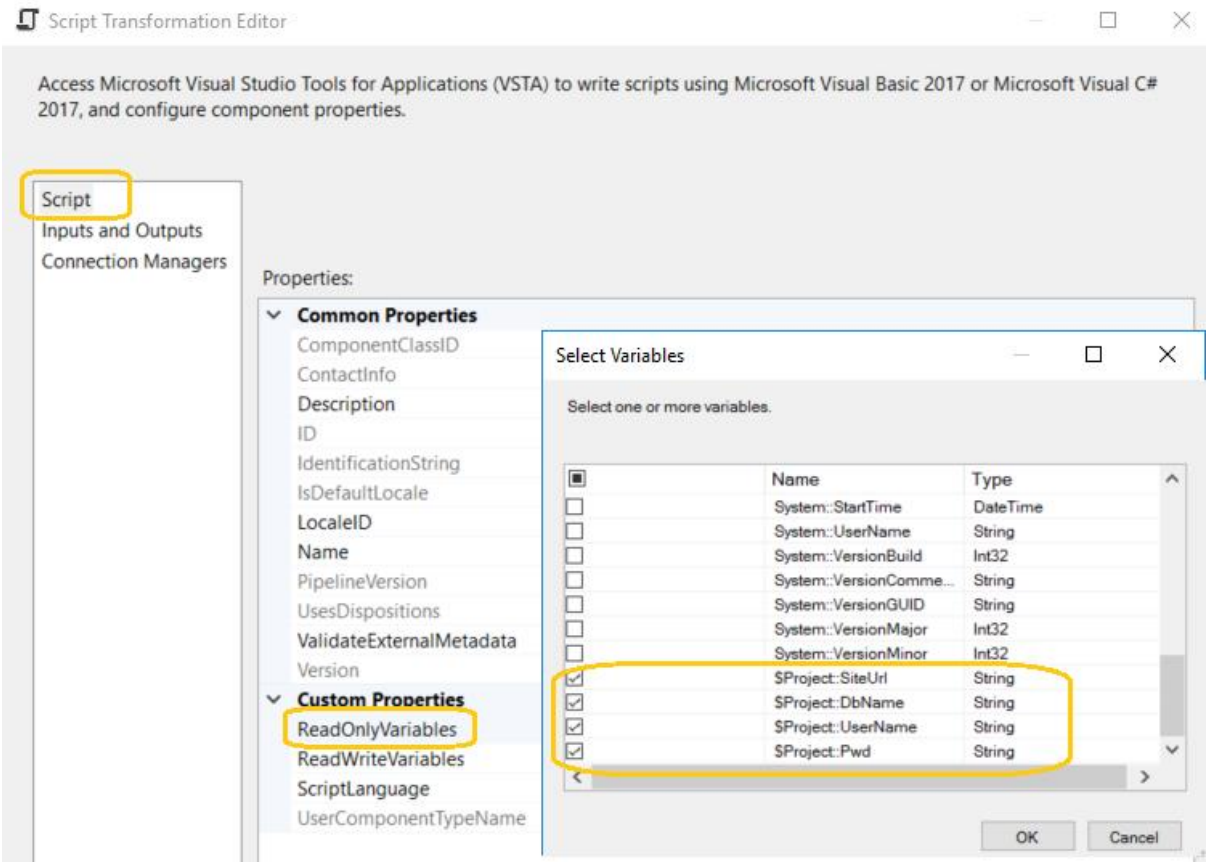
Let's configure each component in data flow task.

Script Component

Basically, the script component consists of three sub tasks; i.e Script, Inputs and Outputs, Connection Managers.

Script

In this task, we will add the project variables under *Custom Properties - > ReadOnlyVariables*.



Inputs and Outputs

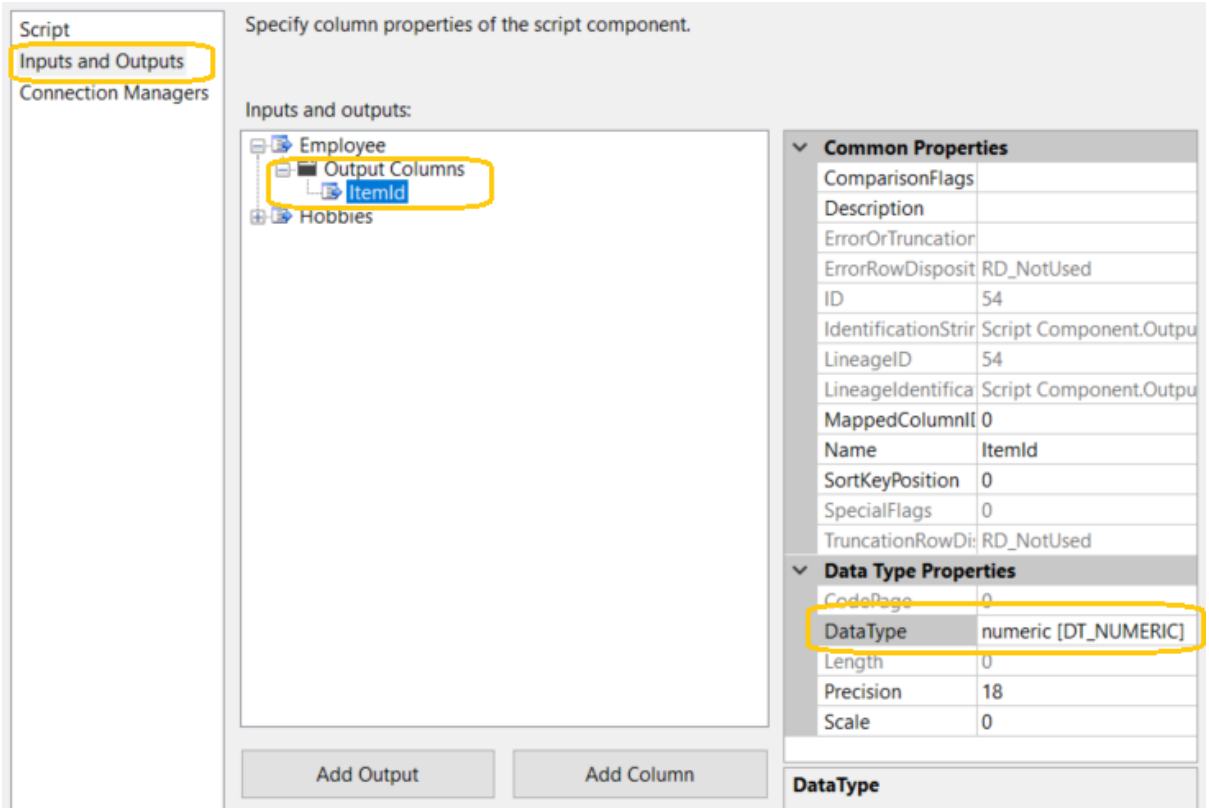
In this task, we will create the ouputs (by clicking on Add Ouput) and their columns (by clicking Add Column).

Employee

Create column ItemId for *Employee* Output in this way.

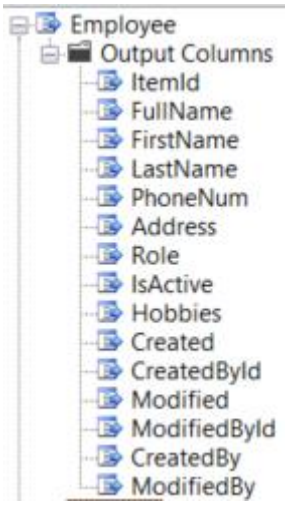
Note

Create the same number of output columns with datatype that exists in employee table.



Now, create the remaining output columns with proper datatype in the following ways.

- FullName (DataType - Unicode string [DT_WSTR] and Length - 500)
- FirstName (DataType - Unicode string [DT_WSTR] and Length - 500)
- LastName (DataType - Unicode string [DT_WSTR] and Length - 500)
- PhoneNum (DataType - numeric [DT_NUMERIC])
- Address (DataType - Unicode string [DT_WSTR] and Length - 4000)
- Role (DataType - Unicode string [DT_WSTR] and Length - 500)
- IsActive(DataType - Unicode string [DT_WSTR] and Length - 50)
- Hobbies(DataType - Unicode string [DT_WSTR] and Length - 4000)
- Created (DataType - date [DT_DATE])
- CreatedById (DataType - numeric [DT_NUMERIC])
- Modified (DataType - date [DT_DATE])
- ModifiedById (DataType - numeric [DT_NUMERIC])
- CreatedBy (DataType - Unicode string [DT_WSTR] and Length - 500)
- ModifiedBy (DataType - Unicode string [DT_WSTR] and Length - 500)

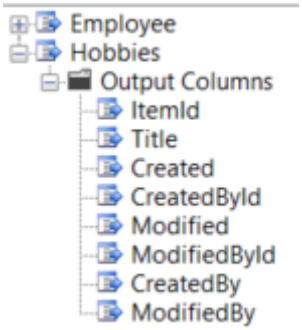


In the same way create the output Hobbies and their columns.

Hobbies

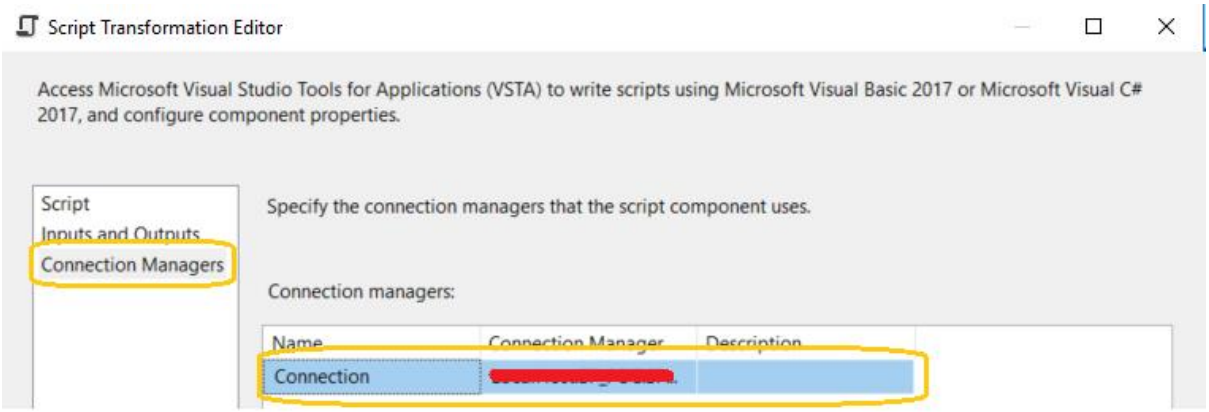
Output Hobbies columns are:

- ItemId (DataType - numeric [DT_NUMERIC])
- Title (DataType - Unicode string [DT_WSTR] and Length - 500)
- Created (DataType - date [DT_DATE])
- CreatedById (DataType - numeric [DT_NUMERIC])
- Modified (DataType - date [DT_DATE])
- ModifiedById (DataType - numeric [DT_NUMERIC])
- CreatedBy (DataType - Unicode string [DT_WSTR] and Length - 500)
- ModifiedBy (DataType - Unicode string [DT_WSTR] and Length - 500)

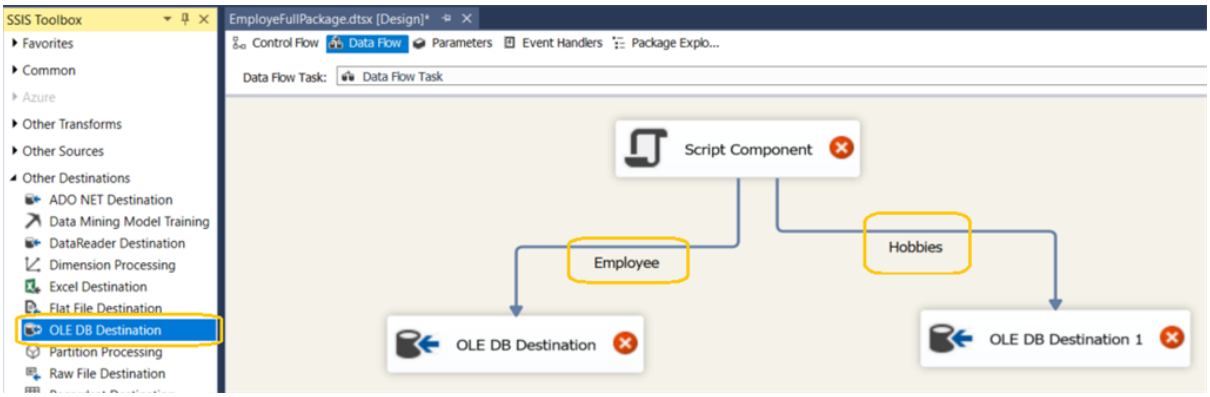


Connection Managers

Open the connection manager in script component and add the appropriate connection.

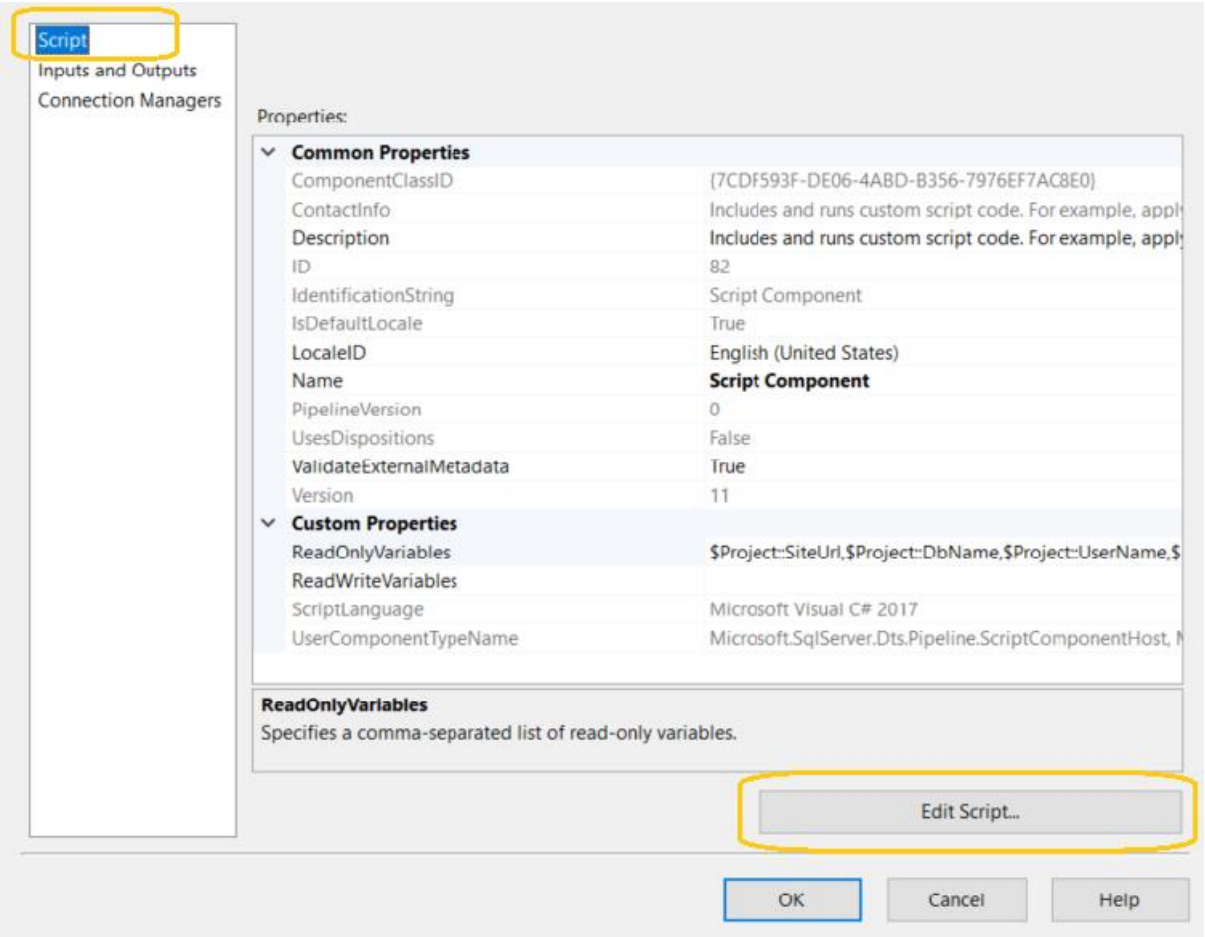


Add the link from Script component to OLE DB Destination and OLE DB Destination1 for full package.



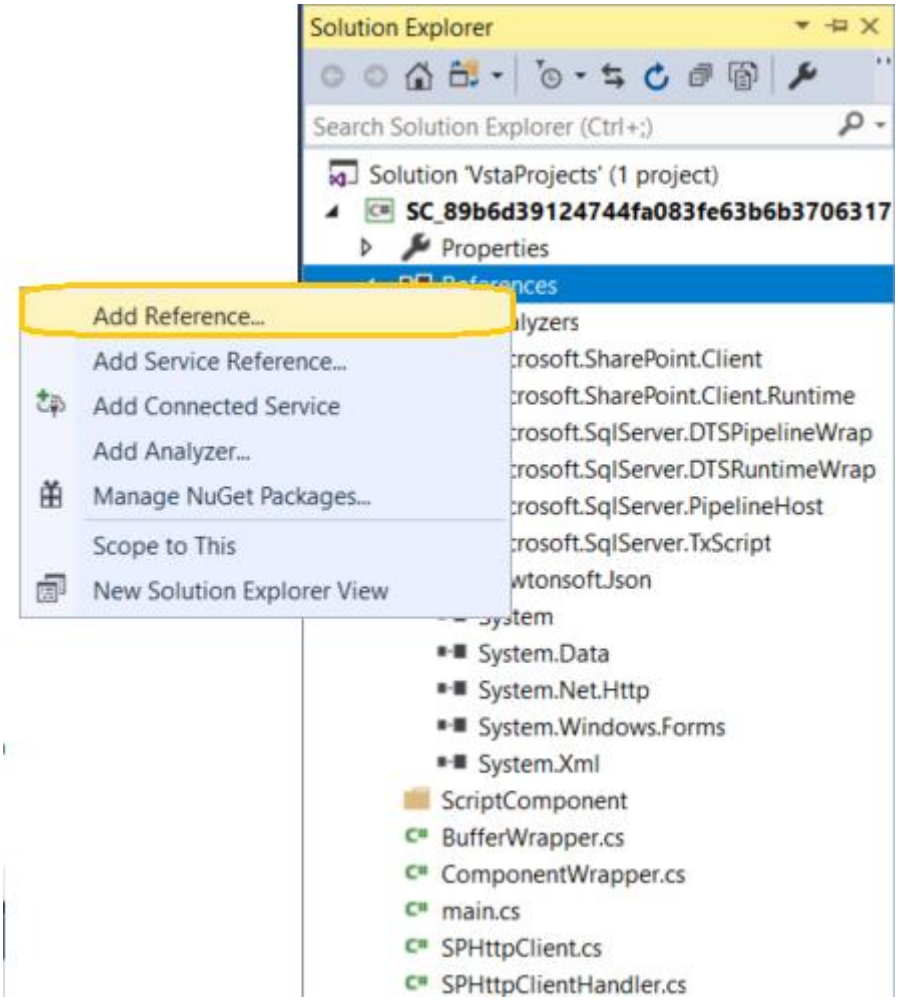
Now, we will write code to fetch the item from the list and insert into table.

Go back to the script section and click on Edit Script button.



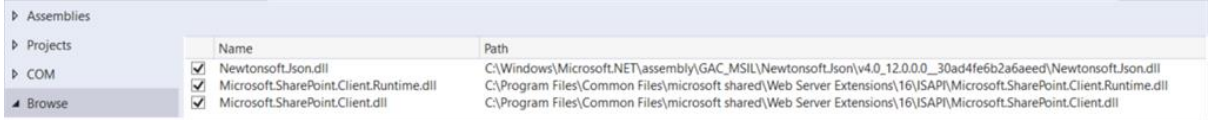
It will open the code in a new window solution.

Add the references in a new window solution in this way.



Add these four references from the following paths:

- System.Net.Http - Assemblies - > Framework or C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\System.Net.Http.dll
- Microsoft.SharePoint.Client - C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\16\ISAPI\Microsoft.SharePoint.Client.dll
- Microsoft.SharePoint.Client.Runtime - C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\16\ISAPI\Microsoft.SharePoint.Client.Runtime.dll
- Newtonsoft.Json- C:\Windows\Microsoft.NET\assembly\GAC_MSIL\Newtonsoft.Json\v4.0_12.0.0.0__30ad4fe6b2a6aeed\Newtonsoft.Json.dll



Create two helper classes and paste the following code in it.

Note

The namespace can be different because it is automatically generated by Visual Studio.

SPHttpClient.cs

```
1. using Newtonsoft.Json;
2. using Newtonsoft.Json.Linq;
3. using System;
4. using System.Collections.Generic;
5. using System.Linq;
6. using System.Net.Http;
7. using System.Net.Http.Headers;
8. using System.Text;
9. using System.Threading.Tasks;
10.
11.      namespace SC_89b6d39124744fa083fe63b6b3706317
12.      {
13.
14.          public class SPHttpClient : HttpClient
15.          {
16.              public SPHttpClient(Uri webUri, string userName, strin
   g password) : base(new SPHttpClientHandler(webUri, userName, passwor
   d))
17.              {
18.                  BaseAddress = webUri;
19.              }
20.              /// <summary>
21.               /// Execure request method
22.              /// </summary>
23.               /// <param name="requestUri"></param>
24.              /// <param name="method"></param>
25.               /// <param name="headers"></param>
26.              /// <param name="payload"></param>
27.               /// <returns></returns>
28.              public JObject ExecuteJson<T>(string requestUri, HttpM
   ethod method, IDictionary<string, string> headers, T payload)
29.              {
30.                  HttpResponseMessage response;
31.                   switch (method.Method)
32.                  {
33.                       case "POST":
34.                          var requestContent = new StringContent(Jso
   nConvert.SerializeObject(payload));
35.                            requestContent.Headers.ContentType =
   MediaTypeHeaderValue.Parse("application/json;odata=verbose");

36.                          DefaultRequestHeaders.Add("X-
   RequestDigest", RequestFormDigest());
37.                            if (headers != null)
38.                          {
```

```
                                foreach (var header in headers)
                            {
                                    DefaultRequestHeaders.Add(hea
der.Key, header.Value);
                            }
                        }
                    response = PostAsync(requestUri, requestCo
ntent).Result;
                        break;
                case "GET":
                        response = GetAsync(requestUri).Resul
t;
                    break;
                default:
                    throw new NotSupportedException(string.For
mat("Method {0} is not supported", method.Method));
                }

                response.EnsureSuccessStatusCode();
                var responseContent = response.Content.ReadAsStrin
gAsync().Result;
                return String.IsNullOrEmpty(responseContent)
? new JObject() : JObject.Parse(responseContent);
            }


            public JObject ExecuteJson<T>(string requestUri,
HttpMethod method, T payload)
            {
                return ExecuteJson(requestUri, method, null,
payload);
            }

            public JObject ExecuteJson(string requestUri)
            {
                return ExecuteJson(requestUri, HttpMethod.Get, nul
l, default(string));
            }


            /// <summary>
            /// Request Form Digest
            /// </summary>
            /// <returns></returns>
            public string RequestFormDigest()
            {
                var endpointUrl = string.Format("{0}/_api/contexti
nfo", BaseAddress);
                var result = this.PostAsync(endpointUrl, new
StringContent(string.Empty)).Result;
```

```
                result.EnsureSuccessStatusCode();
                 var content = result.Content.ReadAsStringAsync().Result;
                var contentJson = JObject.Parse(content);
                 return contentJson["d"]["GetContextWebInformation"]["FormDigestValue"].ToString();
            }
        }
    }
```

SPHttpClientHandler.cs

```
using Microsoft.SharePoint.Client;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Security;
using System.Text;
using System.Threading;
    using System.Threading.Tasks;

    namespace SC_89b6d39124744fa083fe63b6b3706317
    {
        public class SPHttpClientHandler : HttpClientHandler
        {
            public SPHttpClientHandler(Uri webUri, string userName, string password)
            {
                CookieContainer = GetAuthCookies(webUri, userName, password);
                 FormatType = FormatType.JsonVerbose;
            }


            protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)
            {
                 request.Headers.Add("X-FORMS_BASED_AUTH_ACCEPTED", "f");
                if (FormatType == FormatType.JsonVerbose)
                 {
                    //request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json;odata=verbose"));
                     request.Headers.Add("Accept", "application/json;odata=verbose");
                }
```

```
                return base.SendAsync(request, cancellationToken);
            }


             /// <summary>
            /// Retrieve SPO Auth Cookies
             /// </summary>
            /// <param name="webUri"></param>
             /// <param name="userName"></param>
            /// <param name="password"></param>
             /// <returns></returns>
            private static CookieContainer GetAuthCookies(Uri webUri, string userName, string password)
             {
                var securePassword = new SecureString();
                 foreach (var c in password) { securePassword.AppendChar(c); }
                var credentials = new SharePointOnlineCredentials(userName, securePassword);
                 var authCookie = credentials.GetAuthenticationCookie(webUri);
                var cookieContainer = new CookieContainer();
                 cookieContainer.SetCookies(webUri, authCookie);
                return cookieContainer;
             }
            public FormatType FormatType { get; set; }
        }

        public enum FormatType
        {
            JsonVerbose,
            Xml
        }
    }
```

main.cs

Update the main.cs file.

```
#region Help:  Introduction to the Script Component
/* The Script Component allows you to perform virtually any operation that can be accomplished in
 * a .Net application within the context of an Integration Services data flow.
 *
 * Expand the other regions which have "Help" prefixes for examples of specific ways to use
```

```
 * Integration Services features within this script component. */
#endregion

#region Namespaces
      using System;
      using System.Collections.Generic;
      using System.Data;
      using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
      using Microsoft.SqlServer.Dts.Runtime.Wrapper;
      using Newtonsoft.Json.Linq;
      using SC_89b6d39124744fa083fe63b6b3706317;
      #endregion

      /// <summary>
      /// This is the class to which to add your code.  Do not change the name, attributes, or parent
      /// of this class.
      /// </summary>
      [Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]
      public class ScriptMain : UserComponent
      {
          #region Help:  Using Integration Services variables and parameters
          /* To use a variable in this script, first ensure that the variable has been added to
           * either the list contained in the ReadOnlyVariables property or the list contained in
           * the ReadWriteVariables property of this script component, according to whether or not your
           * code needs to write into the variable.  To do so, save this script, close this instance of
           * Visual Studio, and update the ReadOnlyVariables and ReadWriteVariables properties in the
           * Script Transformation Editor window.
           * To use a parameter in this script, follow the same steps. Parameters are always read-only.
           *
           * Example of reading from a variable or parameter:
           *  DateTime startTime = Variables.MyStartTime;
           *
           * Example of writing to a variable:
           *  Variables.myStringVariable = "new value";
           */
          #endregion

          #region Help:  Using Integration Services Connnection Managers
          /* Some types of connection managers can be used in this script component.  See the help topic
```

```
45.          * "Working with Connection Managers Programatically"
    for details.
46.          *
47.          * To use a connection manager in this script, first
   ensure that the connection manager has
48.          * been added to either the list of connection managers on
    the Connection Managers page of the
49.          * script component editor.  To add the connection ma
   nager, save this script, close this instance of
50.          * Visual Studio, and add the Connection Manager to the li
   st.
51.          *
52.          * If the component needs to hold a connection open while
   processing rows, override the
53.          * AcquireConnections and ReleaseConnections methods.

54.          *
55.          * Example of using an ADO.Net connection manager to
   acquire a SqlConnection:
56.          *  object rawConnection = Connections.SalesDB.AcquireConn
   ection(transaction);
57.          *  SqlConnection salesDBConn = (SqlConnection)rawCon
   nection;
58.          *
59.          * Example of using a File connection manager to acqu
   ire a file path:
60.          *  object rawConnection = Connections.Prices_zip.AcquireC
   onnection(transaction);
61.          *  string filePath = (string)rawConnection;
62.          *
63.          * Example of releasing a connection manager:
64.          *  Connections.SalesDB.ReleaseConnection(rawConnection);
65.          */
66.         #endregion
67.
68.         #region Help:  Firing Integration Services Events
69.         /* This script component can fire events.
70.          *
71.          * Example of firing an error event:
72.          *  ComponentMetaData.FireError(10, "Process Values", "Bad
    value", "", 0, out cancel);
73.          *
74.          * Example of firing an information event:
75.          *  ComponentMetaData.FireInformation(10, "Process Va
   lues", "Processing has started", "", 0, fireAgain);
76.          *
77.          * Example of firing a warning event:
78.          *  ComponentMetaData.FireWarning(10, "Process Values", "N
   o rows were received", "", 0);
79.          */
```

```
80.          #endregion
81.
82.          /// <summary>
83.          /// This method is called once, before rows begin to
   be processed in the data flow.
84.          ///
85.          /// You can remove this method if you don't need to d
   o anything here.
86.          /// </summary>
87.          public override void PreExecute()
88.          {
89.              base.PreExecute();
90.              /*
91.                * Add your code here
92.               */
93.          }
94.
95.          /// <summary>
96.          /// This method is called after all the rows have passed t
   hrough this component.
97.          ///
98.          /// You can delete this method if you don't need to do any
   thing here.
99.          /// </summary>
100.         public override void PostExecute()
101.         {
102.             base.PostExecute();
103.              /*
104.              * Add your code here
105.               */
106.         }
107.
108.         public override void CreateNewOutputRows()
109.         {
110.             /*
111.                Add rows by calling the AddRow method on the me
   mber variable named "<Output Name>Buffer".
112.               For example, call MyOutputBuffer.AddRow() if your ou
   tput was named "MyOutput".
113.              */
114.             string siteURL = Variables.SiteUrl;
115.              string userName = Variables.UserName;
116.             string password = Variables.Pwd;
117.              CallDataEmployee(siteURL, userName, password, "ID
   ");
118.             //CallDataArchieveEmployee(archiveURL, userName, passw
   ord, "SourceID");
119.
120.             CallDataHobbies(siteURL, userName, password, "ID");
```

```
121.             //CallDataArchieveHobbies(archiveURL, userName, p
   assword, "SourceID");
122.         }
123.
124.         /// <summary>
125.         /// This method is used to get the data from Employee
    list.
126.         /// </summary>
127.         /// <param name="siteURL"></param>
128.         /// <param name="userName"></param>
129.         /// <param name="password"></param>
130.         /// <param name="IDCol"></param>
131.         public void CallDataEmployee(string siteURL, string u
   serName, string password, string IDCol)
132.         {
133.             var webUri = new Uri(siteURL);
134.             using (var client = new SPHttpClient(webUri, userName,
    password))
135.             {
136.                 try
137.                 {
138.                     var listTitle = "Employee";
139.                     string queryCall = string.Empty;
140.                     string callQuery = "ID&$top=1&$orderby=ID desc
   ";
141.                     int chunkSize = 3999;
142.                     int initial = 0;
143.                     int final = 0;
144.                     var endpointUrlTopCall = string.Format("{0}/_a
   pi/web/lists/getbytitle('{1}')/items?$select=" + callQuery, webUri,
   listTitle);
145.                     var data = client.ExecuteJson(endpointUrl
   TopCall);
146.                     JToken item = data["d"]["results"][0];
147.                     int maxNumber = GetInt(item, "ID", -1);
148.                     //maxNumber = 99;
149.                     string colNames = IDCol + ",Title,FirstNa
   me,LastName,PhoneNo,Role,Address,IsActive,Hobbies/ID,Hobbies/T
   itle,Author/Title,Editor/Title," +
150.                         "Created,Modified,AuthorId,EditorId";
151.                     do
152.                     {
153.                         initial = final + 1;
154.                         final = initial + chunkSize - 1;
155.                         queryCall = "&$top=" + chunkSize + "&
   $filter=ID ge " + initial + " and ID le " + final + "&$orderby
   =ID asc&$expand=Hobbies,Author,Editor";
156.                         var endpointUrl = string.Format("{0}/_api/
   web/lists/getbytitle('{1}')/items?$select=" + colNames + queryCall,
   webUri, listTitle);
```

```
157.                         var returndata = client.ExecuteJson(e
    ndpointUrl);
158.                       try
159.                         {
160.                           CallDataEmployee(returndata, IDCol);
161.                         }
162.                       catch (Exception e)
163.                         {
164.
165.                         }
166.
167.                     } while (maxNumber > final);
168.                 }
169.                  catch (Exception e)
170.                 {
171.
172.                 }
173.             }
174.         }
175.
176.         /// <summary>
177.         /// This method is used to iterate the result and sto
    red into Employee Buffer Output
178.         /// </summary>
179.         /// <param name="data"></param>
180.         /// <param name="IDCol"></param>
181.         public void CallDataEmployee(JToken data, string IDCo
    l)
182.         {
183.             foreach (var item in data["d"]["results"])
184.             {
185.                 try
186.                 {
187.                     EmployeeBuffer.AddRow();
188.                     EmployeeBuffer.ItemId = GetDecimal(item, IDCol
    , -1);
189.                     EmployeeBuffer.FullName = GetString(item,
     "Title");
190.                     EmployeeBuffer.FirstName = GetString(item, "Fi
    rstName");
191.                     EmployeeBuffer.LastName = GetString(item,
     "LastName");
192.                     EmployeeBuffer.PhoneNum = GetInt(item, "PhoneN
    o", -1);
193.                     EmployeeBuffer.Role = GetString(item, "Ro
    le");
194.                     EmployeeBuffer.Address = GetString(item, "Addr
    ess");
195.                     EmployeeBuffer.IsActive = GetString(item,
     "IsActive");
```

```
196.                     EmployeeBuffer.Hobbies = GetMultipleComplex(it
   em, "Hobbies", "Title", ",");
197.                      EmployeeBuffer.Created = GetDateTime(item
   , "Created");
198.                     EmployeeBuffer.Modified = GetDateTime(item, "M
   odified");
199.                      EmployeeBuffer.CreatedById = GetDecimal(i
   tem, "AuthorId", -1);
200.                     EmployeeBuffer.ModifiedById = GetDecimal(item,
    "EditorId", -1);
201.                      EmployeeBuffer.CreatedBy = GetComplex(ite
   m, "Author", "Title");
202.                     EmployeeBuffer.ModifiedBy = GetComplex(item, "
   Editor", "Title");
203.                 }
204.                 catch (Exception e1)
205.                 {
206.
207.                 }
208.             }
209.
210.         }
211.
212.         /// <summary>
213.         /// This method is used to get data from Hobbies list
   .
214.         /// </summary>
215.         /// <param name="siteURL"></param>
216.         /// <param name="userName"></param>
217.         /// <param name="password"></param>
218.         /// <param name="IDCol"></param>
219.         public void CallDataHobbies(string siteURL, string us
   erName, string password, string IDCol)
220.         {
221.             var webUri = new Uri(siteURL);
222.             using (var client = new SPHttpClient(webUri, userName,
    password))
223.             {
224.                 try
225.                 {
226.                     var listTitle = "Hobbies";
227.                      string queryCall = string.Empty;
228.                     string callQuery = "ID&$top=1&$orderby=ID desc
   ";
229.                      int chunkSize = 3999;
230.                     int initial = 0;
231.                      int final = 0;
232.                     var endpointUrlTopCall = string.Format("{0}/_a
   pi/web/lists/getbytitle('{1}')/items?$select=" + callQuery, webUri,
   listTitle);
```

```
233.                     var data = client.ExecuteJson(endpointUrl
   TopCall);
234.                   JToken item = data["d"]["results"][0];
235.                    int maxNumber = GetInt(item, "ID", -1);
236.                   //maxNumber = 99;
237.                    string colNames = IDCol + ",Title,Author/
   Title,Editor/Title," +
238.                       "Created,Modified,AuthorId,EditorId";
239.                    do
240.                   {
241.                        initial = final + 1;
242.                       final = initial + chunkSize - 1;
243.                        queryCall = "&$top=" + chunkSize + "&
   $filter=ID ge " + initial + " and ID le " + final + "&$orderby
   =ID asc&$expand=Author,Editor";
244.                       var endpointUrl = string.Format("{0}/_api/
   web/lists/getbytitle('{1}')/items?$select=" + colNames + queryCall,
   webUri, listTitle);
245.                        var returndata = client.ExecuteJson(e
   ndpointUrl);
246.                       try
247.                        {
248.                          CallDataHobbies(returndata, IDCol);
249.                        }
250.                       catch (Exception e)
251.                        {
252.
253.                        }
254.
255.                    } while (maxNumber > final);
256.               }
257.                catch (Exception e)
258.               {
259.
260.               }
261.            }
262.        }
263.
264.        /// <summary>
265.        /// This method is used to iterate the result and sto
   re into Hobbies buffer.
266.        /// </summary>
267.        /// <param name="data"></param>
268.        /// <param name="IDCol"></param>
269.        public void CallDataHobbies(JToken data, string IDCol
   )
270.        {
271.            foreach (var item in data["d"]["results"])
272.           {
273.                try
```

```
274.                 {
275.                     HobbiesBuffer.AddRow();
276.                     HobbiesBuffer.ItemId = GetDecimal(item, IDCol,
    -1);
277.                     HobbiesBuffer.Title = GetString(item, "Ti
   tle");
278.                     HobbiesBuffer.Created = GetDateTime(item, "Cre
   ated");
279.                     HobbiesBuffer.Modified = GetDateTime(item
   , "Modified");
280.                     HobbiesBuffer.CreatedById = GetDecimal(item, "
   AuthorId", -1);
281.                     HobbiesBuffer.ModifiedById = GetDecimal(i
   tem, "EditorId", -1);
282.                     HobbiesBuffer.CreatedBy = GetComplex(item, "Au
   thor", "Title");
283.                     HobbiesBuffer.ModifiedBy = GetComplex(ite
   m, "Editor", "Title");
284.                 }
285.                  catch (Exception e1)
286.                 {
287.
288.                 }
289.             }
290.
291.         }
292.         /// <summary>
293.         /// This method is used to Get the string from item.

294.         /// </summary>
295.         /// <param name="token"></param>
296.         /// <param name="key"></param>
297.         /// <returns></returns>
298.         public string GetString(JToken token, string key)
299.         {
300.             string value = string.Empty;
301.
302.             try
303.             {
304.                 value = Convert.ToString(token[key]);
305.             }
306.             catch (Exception e)
307.             {
308.
309.             }
310.
311.             return value;
312.         }
313.         /// <summary>
314.         ///
```

```
315.         /// </summary>
316.         /// <param name="token"></param>
317.         /// <param name="key"></param>
318.         /// <param name="var"></param>
319.         /// <returns></returns>
320.         public string GetComplex(JToken token, string key, string
   var)
321.         {
322.             string value = string.Empty;
323.
324.             try
325.              {
326.                 JToken tok = token[key];
327.                  value = GetString(tok, var);
328.             }
329.              catch (Exception e)
330.             {
331.
332.             }
333.
334.             return value;
335.         }
336.
337.         public Decimal GetComplexID(JToken token, string key,
    string var)
338.         {
339.              Decimal dec = -1;
340.
341.              try
342.             {
343.                  JToken tok = token[key];
344.                 dec = GetDecimal(tok, var, -1);
345.              }
346.             catch (Exception e)
347.              {
348.
349.              }
350.
351.              return dec;
352.         }
353.         /// <summary>
354.         /// This method is used to get the decimal from item.
355.         /// </summary>
356.         /// <param name="token"></param>
357.         /// <param name="key"></param>
358.         /// <param name="defaultVal"></param>
359.         /// <returns></returns>
360.         public Decimal GetDecimal(JToken token, string key, Decima
   l defaultVal)
361.         {
```

```
362.             Decimal dec = -1;
363.
364.             bool complete = Decimal.TryParse(GetString(token, key)
   , out dec);
365.
366.             if (complete)
367.              {
368.                 return dec;
369.              }
370.             else
371.              {
372.                 return defaultVal;
373.              }
374.         }
375.         /// <summary>
376.         /// This method is used to return int from item.
377.         /// </summary>
378.         /// <param name="token"></param>
379.         /// <param name="key"></param>
380.         /// <param name="defaultVal"></param>
381.         /// <returns></returns>
382.         public Int32 GetInt(JToken token, string key, Int32 defaul
   tVal)
383.         {
384.             Int32 dec = -1;
385.
386.             bool complete = Int32.TryParse(GetString(token, key),
   out dec);
387.
388.             if (complete)
389.              {
390.                 return dec;
391.              }
392.             else
393.              {
394.                 return defaultVal;
395.              }
396.         }
397.         /// <summary>
398.         /// This method is used to return the datetime from item.

399.         /// </summary>
400.         /// <param name="token"></param>
401.         /// <param name="key"></param>
402.         /// <returns></returns>
403.         public DateTime GetDateTime(JToken token, string key)

404.         {
405.              DateTime dec;
```

```
406.             bool complete = DateTime.TryParse(GetString(token, key
   ), out dec);
407.              return dec.ToLocalTime();
408.         }
409.         /// <summary>
410.         /// This method is used to return the multiple string valu
   e from item.
411.         /// </summary>
412.         /// <param name="token"></param>
413.         /// <param name="key"></param>
414.         /// <param name="objectKey"></param>
415.         /// <param name="seperater"></param>
416.         /// <returns></returns>
417.         public string GetMultipleComplex(JToken token, string
    key, string objectKey, string seperater)
418.         {
419.              string tempString = string.Empty;
420.             List<string> strList = new List<string>();
421.              try
422.             {
423.                  JToken stringTokens = token[key]["results"];

424.
425.                  foreach (var user in stringTokens)
426.                 {
427.                      string tempStr = GetString(user, objectKe
   y);
428.                     strList.Add(tempStr);
429.                  }
430.                 if (strList.Count > 0)
431.                  {
432.                     tempString = string.Join(seperater, strList);

433.                  }
434.             }
435.              catch (Exception ex)
436.             {
437.
438.             }
439.
440.             return tempString;
441.         }
442.     }
```

No need to change the BufferWrapper.cs and ComponentWrapper.cs classes.

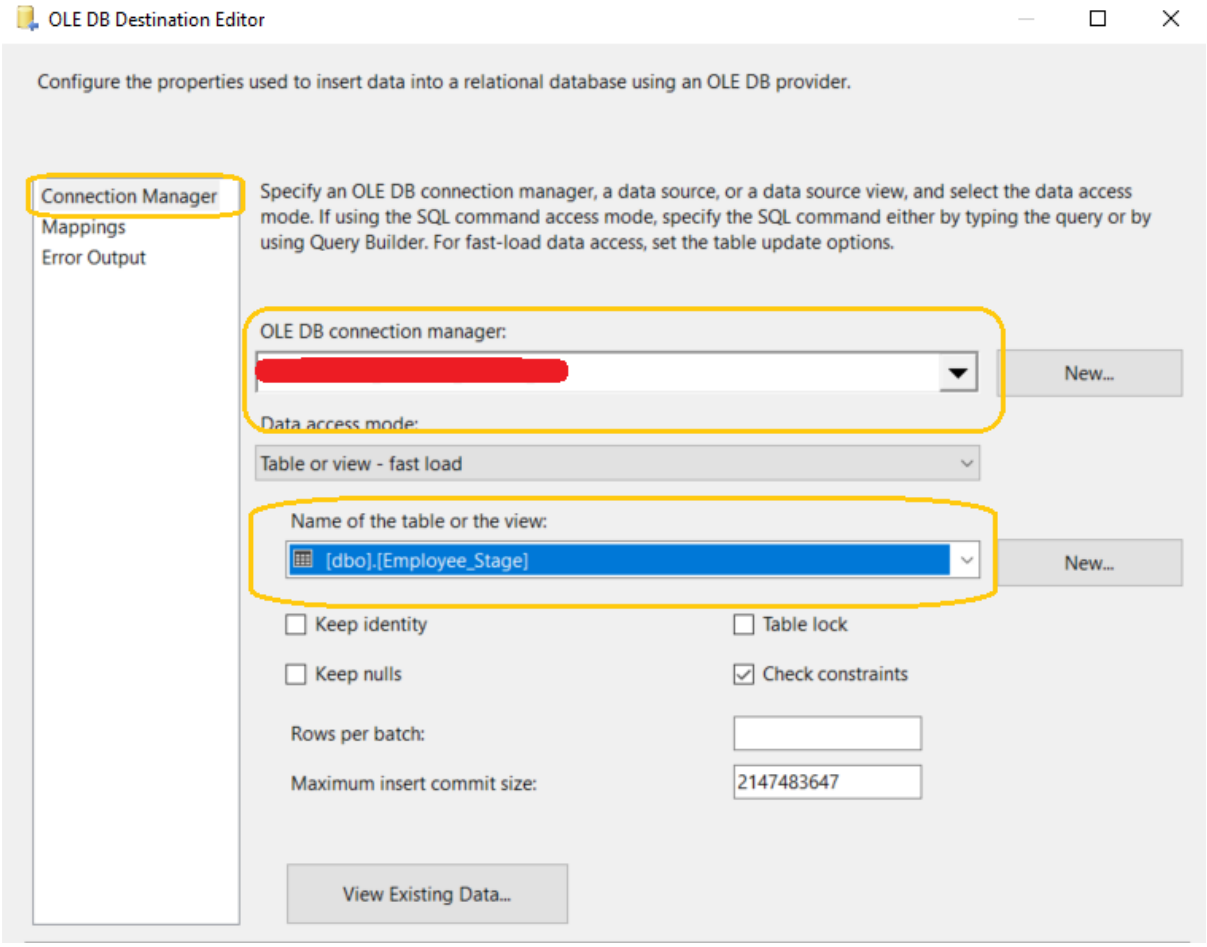
Save the changes by clicking on Ok button in script section of script component.

OLE DB Destination

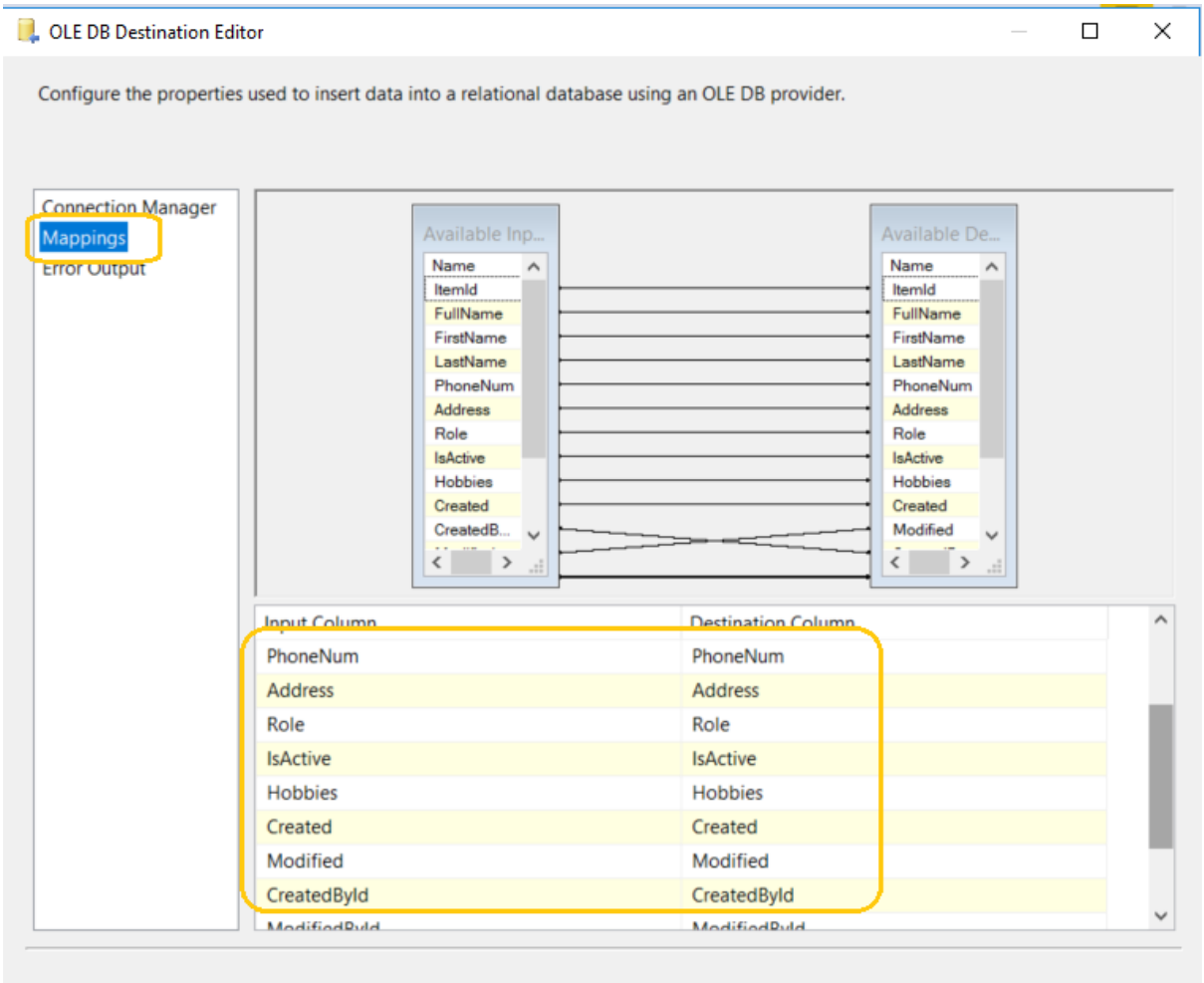
Configure the OLE DB Destination for Employee parts i.e *OLE DB Destination*

Each OLE DB Destination has three parts i.e. Connection Managers, Mappings and Error Output.

Connection Managers



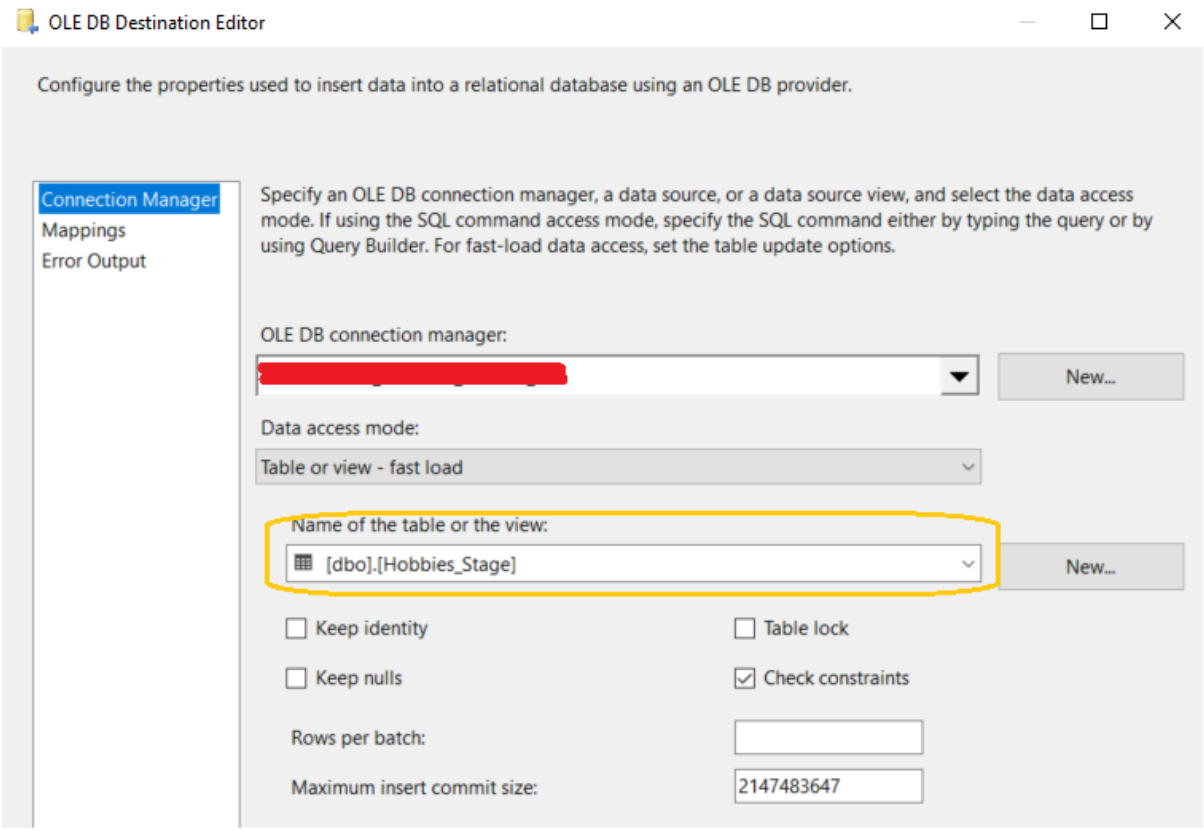
Mappings



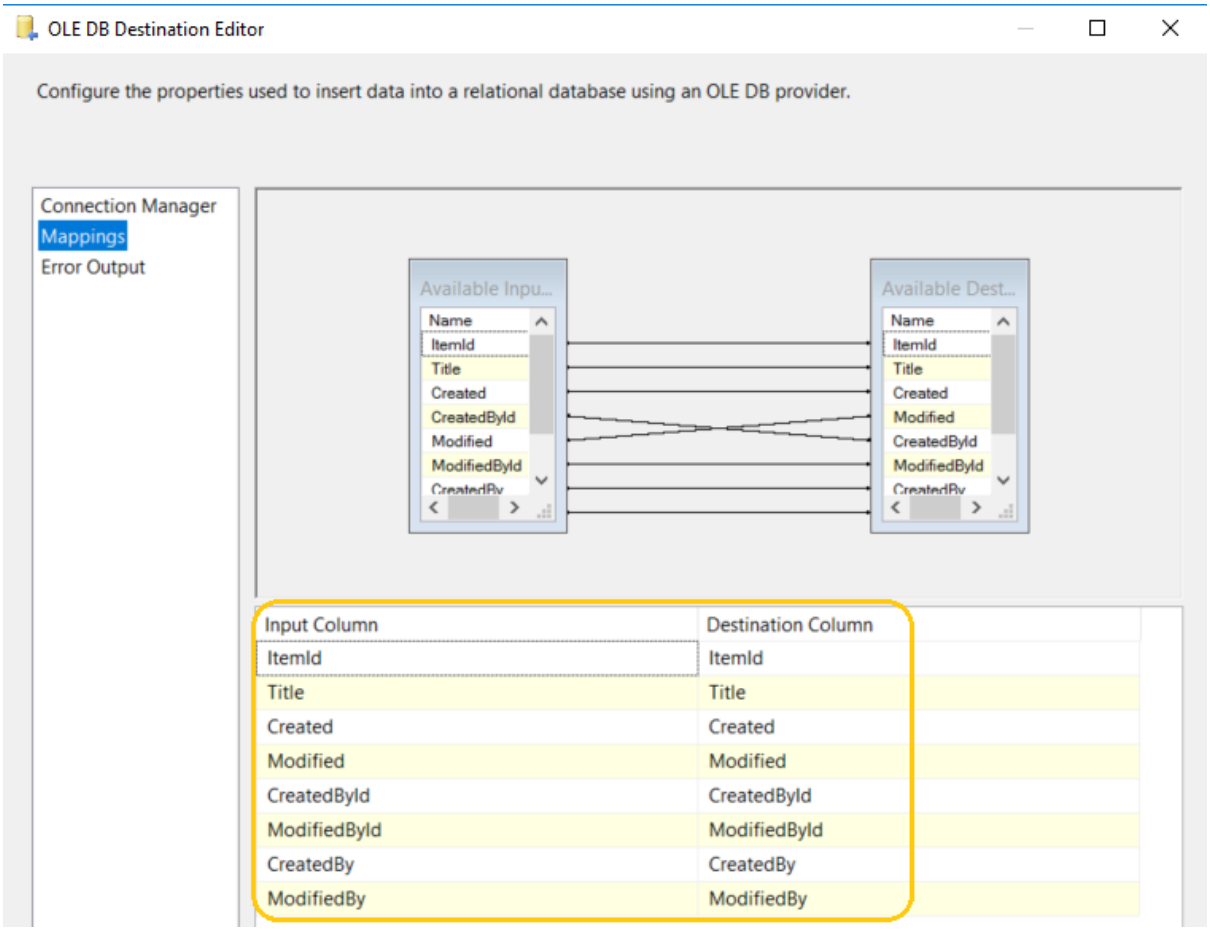
Configure the OLE DB Destination for Hobbies parts i.e *OLE DB Destination 1.*

Each OLE DB Destination has three parts i.e. Connection Managers, Mappings and Error Output.

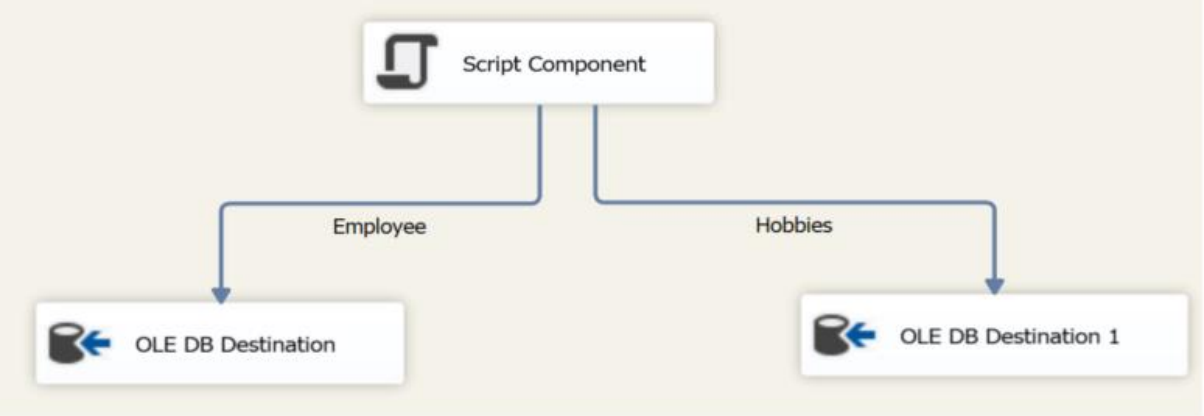
Connection Managers


OLE DB Destination Editor
Configure the properties used to insert data into a relational database using an OLE DB provider.
Connection Manager
Mappings
Error Output
Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.
OLE DB connection manager:
New...
Data access mode:
Table or view - fast load
Name of the table or the view:
[dbo].[Hobbies_Stage]
New...
Keep identity
Table lock
Keep nulls
Check constraints
Rows per batch:
Maximum insert commit size:
2147483647

Mappings



If data flow task is configured properly for full package it will look like

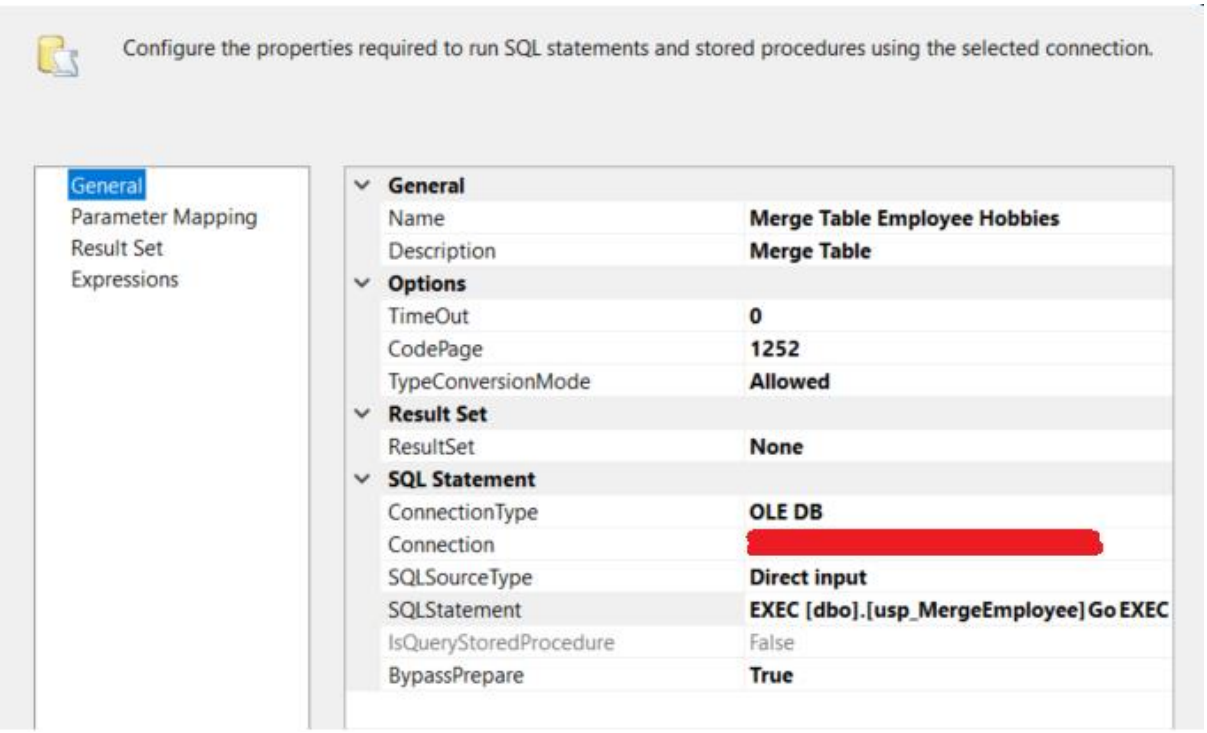


Now we are going to configure the last component of control flow task for EmployeeFullPackage.dtsx

Execute SQL Task1

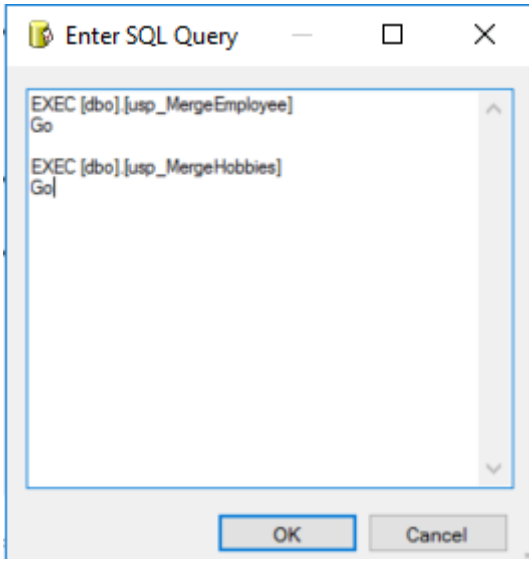
In this task we insert records in main table (i.e Employee and Hobbies) from stage table (i.e. Employee_Stage and Hobbies_Stage) by calling stored procedures(usp_MergeEmployee & usp_MergeHobbies).

Open the task, rename the task as *Merge Table Employee Hobbies* and configure in this way.



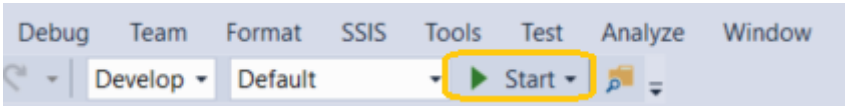
Open SQL Statment and paste this command.

```
EXEC [dbo].[usp_MergeEmployee]
Go

EXEC [dbo].[usp_MergeHobbies]
Go
```

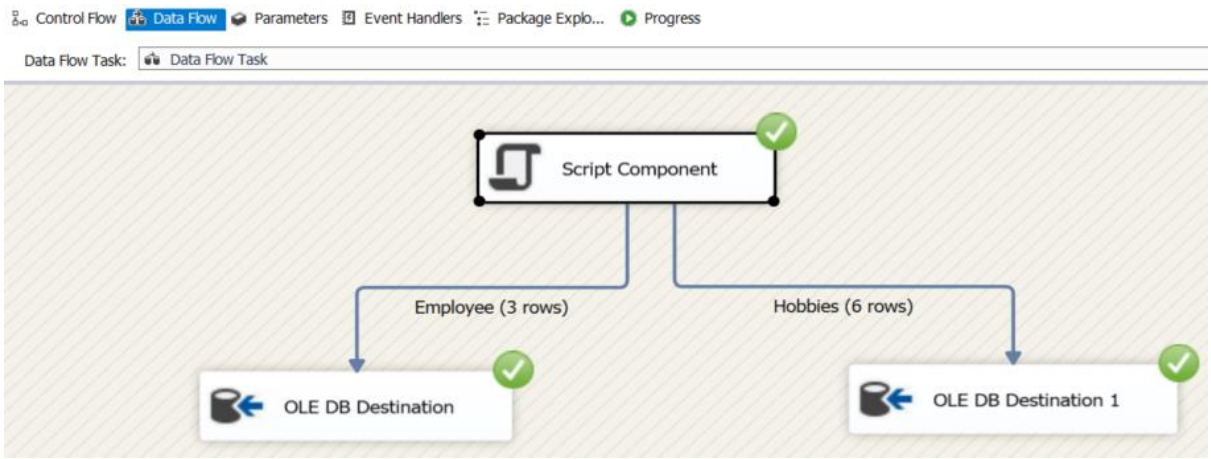


Run and validate the EmployeeFullPackage

Simply click on start icon present in visual studio menu.



If package runs without any errors then it will look like



Now check the Employee_Stage and Employee table in database SP_POC both have identical data. In the same way Hobbies_Stage and Hobbies table have identical data.

Records in Employee_Stage table

Results Messages

		FirstName	LastName	PhoneNum	Address	Role	IsActive	Hobbies	Created	N
1	a	Pravin	Kushwaha	459800	Bhopal	Tester	Yes	Cricket,Football,Luddo	2020-04-30 12:08:32.000	2
2	a	Arvind	Kushwaha	90834093	Mumbai Maharashtra	Developer	Yes	Cricket,Reading,Watching Movies	2020-05-01 17:39:16.000	2
3		Randeep	Singh	3990348	Delhi	Developer	Yes	Carrom	2020-05-06 11:52:31.000	2

Records in Employee table

Results Messages

		FirstName	LastName	PhoneNum	Address	Role	IsActive	Hobbies	Created	N
1	a	Pravin	Kushwaha	459800	Bhopal	Tester	Yes	Cricket,Football,Luddo	2020-04-30 12:08:32.000	2
2	a	Arvind	Kushwaha	90834093	Mumbai Maharashtra	Developer	Yes	Cricket,Reading,Watching Movies	2020-05-01 17:39:16.000	2
3		Randeep	Singh	3990348	Delhi	Developer	Yes	Carrom	2020-05-06 11:52:31.000	2

Records in Hobbies_Stage table

100 %

Results Messages

	ItemId	Title	Created	Modified	CreatedById	ModifiedById	CreatedBy	ModifiedBy
1	1	Cricket	2020-05-06 11:45:27.000	2020-05-06 11:45:27.000	13	13	Arvind Kushwaha	Arvind Kushwaha
2	2	Football	2020-05-06 11:45:32.000	2020-05-06 11:45:32.000	13	13	Arvind Kushwaha	Arvind Kushwaha
3	3	Carrom	2020-05-06 11:45:35.000	2020-05-06 11:45:35.000	13	13	Arvind Kushwaha	Arvind Kushwaha
4	4	Luddo	2020-05-06 11:45:45.000	2020-05-06 11:45:45.000	13	13	Arvind Kushwaha	Arvind Kushwaha
5	5	Watching Movies	2020-05-06 11:46:08.000	2020-05-06 11:46:08.000	13	13	Arvind Kushwaha	Arvind Kushwaha
6	6	Reading	2020-05-06 11:46:19.000	2020-05-06 11:46:19.000	13	13	Arvind Kushwaha	Arvind Kushwaha

Records in Hobbies table

100 %

Results Messages

	ItemId	Title	Created	Modified	CreatedById	ModifiedById	CreatedBy	ModifiedBy
1	1	Cricket	2020-05-06 11:45:27.000	2020-05-06 11:45:27.000	13	13	Arvind Kushwaha	Arvind Kushwaha
2	2	Football	2020-05-06 11:45:32.000	2020-05-06 11:45:32.000	13	13	Arvind Kushwaha	Arvind Kushwaha
3	3	Carrom	2020-05-06 11:45:35.000	2020-05-06 11:45:35.000	13	13	Arvind Kushwaha	Arvind Kushwaha
4	4	Luddo	2020-05-06 11:45:45.000	2020-05-06 11:45:45.000	13	13	Arvind Kushwaha	Arvind Kushwaha
5	5	Watching Movies	2020-05-06 11:46:08.000	2020-05-06 11:46:08.000	13	13	Arvind Kushwaha	Arvind Kushwaha
6	6	Reading	2020-05-06 11:46:19.000	2020-05-06 11:46:19.000	13	13	Arvind Kushwaha	Arvind Kushwaha

Debugging the package

- Open the script component -> click on Edit Script.
- Add the debbuger in the code and click on ok button which exists in previous window.
- Click on Start icon present in visual studion menu.

EmployeeIncrementalPackage

This package will run on daily basis after every 15 mins (although you can configure it diffently).

Functionality

- In the first step, it will truncate only stage table; i.e Employee_Stage and Hobbies_Stage.
- In the second step, it will copy items which are modified or added within 15 min in Employee and Hobbies list and stored into Employee and Hobbies buffer.
- In the third step, it will add all items from Employee buffer to Employee_Stage table and from Hobbies buffer to Hobbies_Stage table.
- In the last step, it will call the stored procedures *usp_MergeEmployee* and *usp_MergeHobbies* (created in previous article) to copy records from stage to main table.

Right click on *SSIS package -> New SSIS package* and name it as EmployeIncrementalPackage.dtsx

Control Flow

Open the EmployeeIncrementalPackage.dstx and drag the relevent component from SSIS Toolbox to control flow.

The control flow task consists of three main tasks; i.e Execute SQL Task, Data Flow Task and Execute SQL Task 1.

Create the same control flow for incremental package.

Execute SQL Task

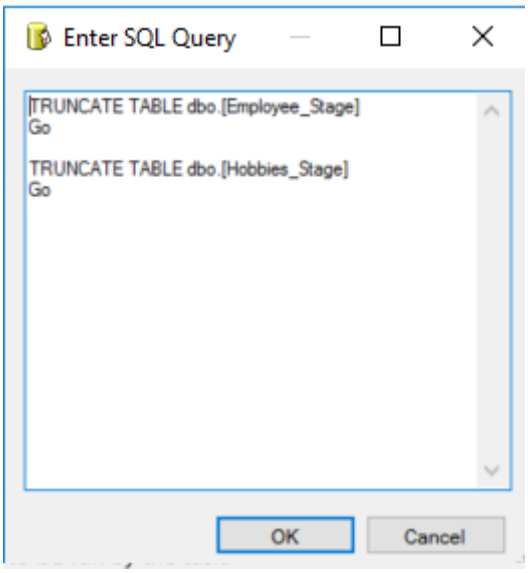
It is used to execute the SQL statement.

In this task, we will truncate our stage tables i.e. *Employee_Stage and Hobbies_Stage.*

Open the task, rename the task as *Truncate Table Employee Hobbies* and configure similar to employeefullpackage except the SQL Statement.

Open the SQL Statement and write the below command.

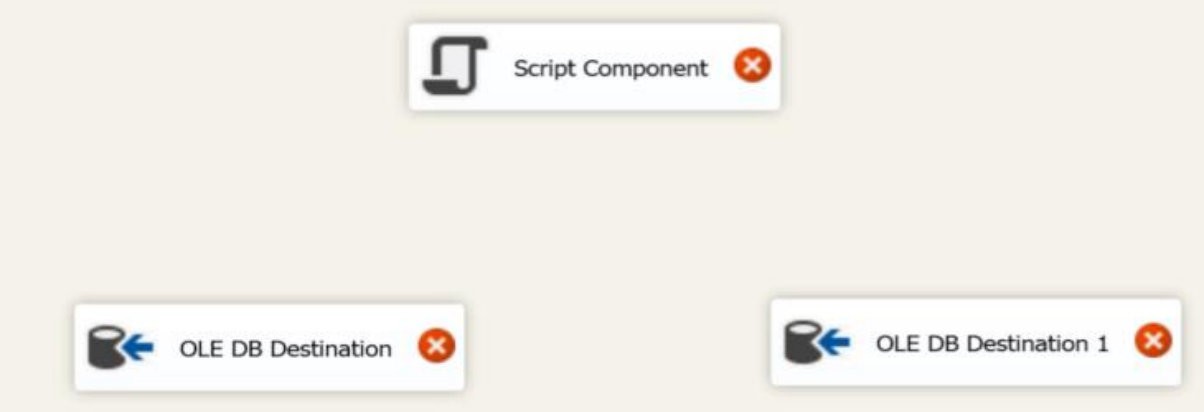
```
1. TRUNCATE TABLE dbo.[Employee_Stage]
2. Go
3.
4. TRUNCATE TABLE dbo.[Hobbies_Stage]
5. Go
```



Data Flow Task

In this task, we will fetch those records which are created or updated within 15 mins in Employee and Hobbies list and add into Employee_Stage and Hobbies_Stage table.

Click on Data flow task and select the relevant component from SSIS toolbox by dragging and dropping to **data flow** similar to employeefullpackage.



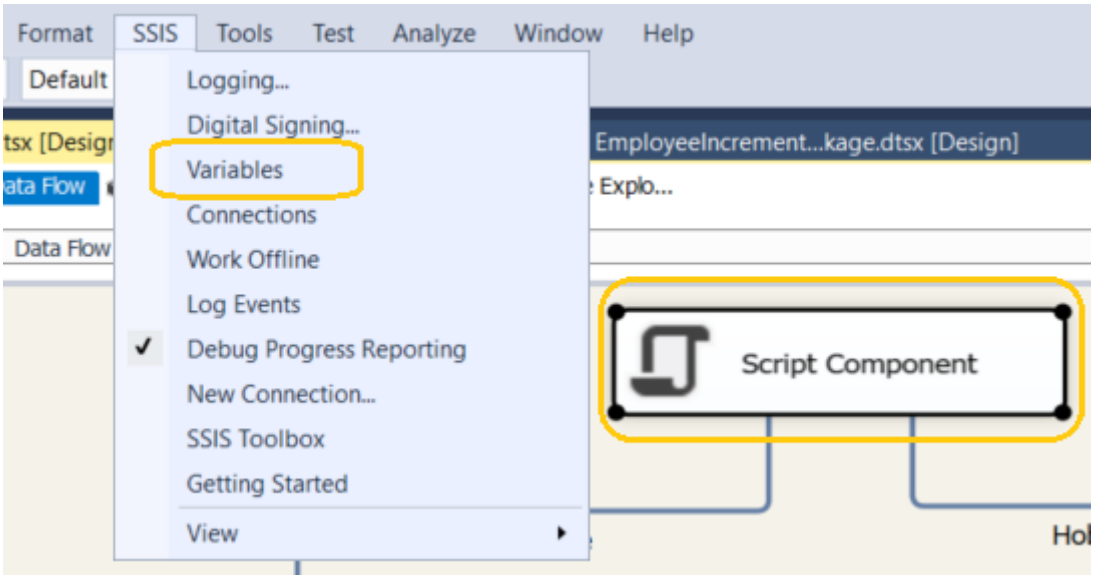
Script Component

Basically, the script component consist of three sub tasks; i.e Script, Inputs and Outputs, Connection Managers.

Script

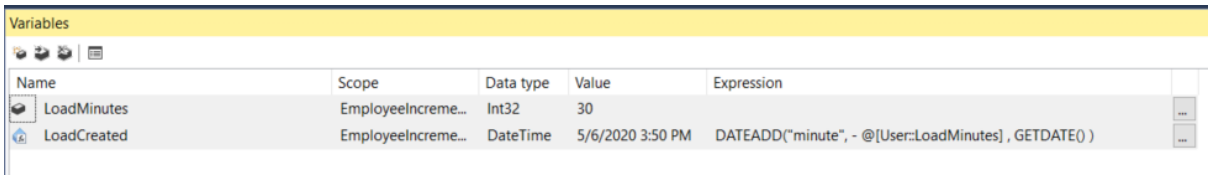
For incremental package we will create two more variables for passing the date and time.

Click on Script Component, from SSIS menu select variables and create the following variables.

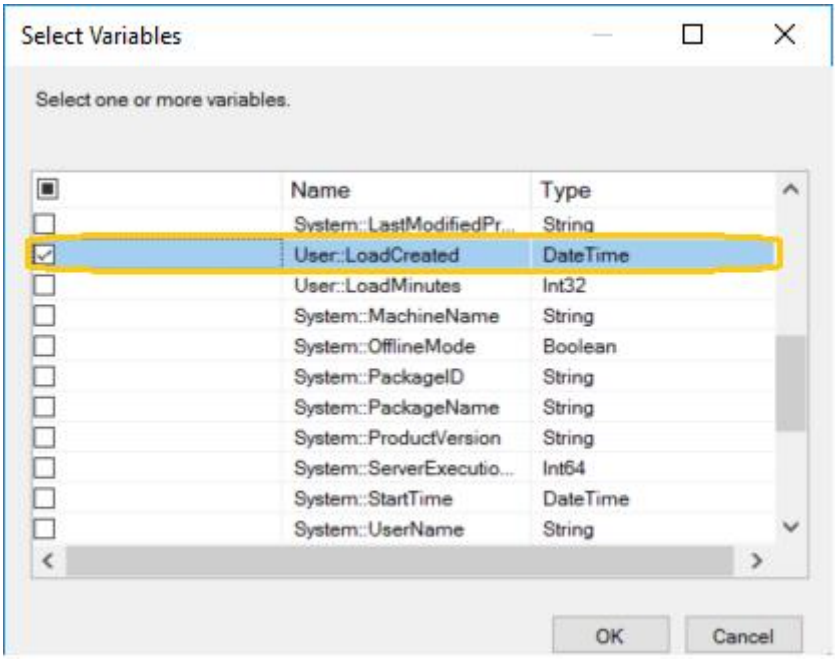


Paste this expression for Load Created variables

DATEADD("minute", - @[User::LoadMinutes] , GETDATE())



In this task, we will add the project variables as well as newly created variable (LoadCreated) under *Custom Properties - > ReadOnlyVariables*.



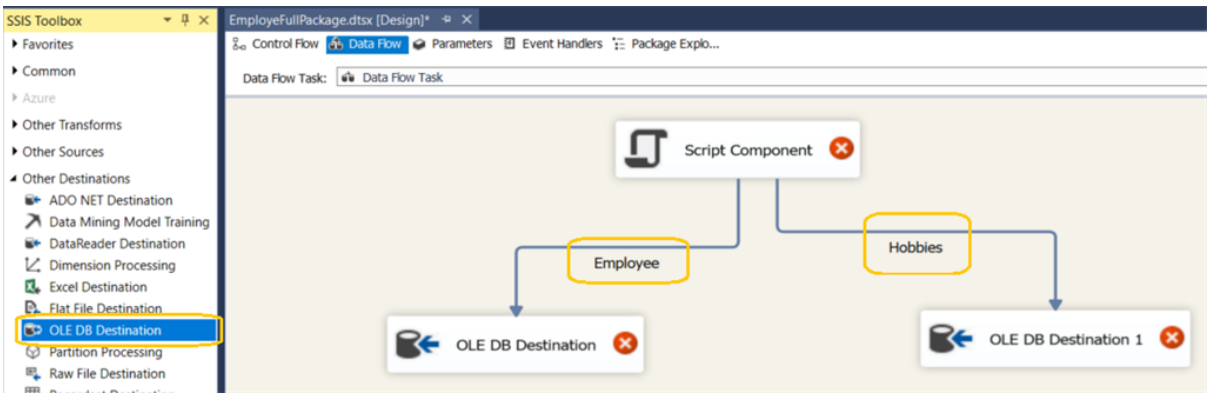
Inputs and Outputs

Create similar ouputs (Employee and Hobbies) and thier columns like EmployeeFullPackage.

Connection Managers

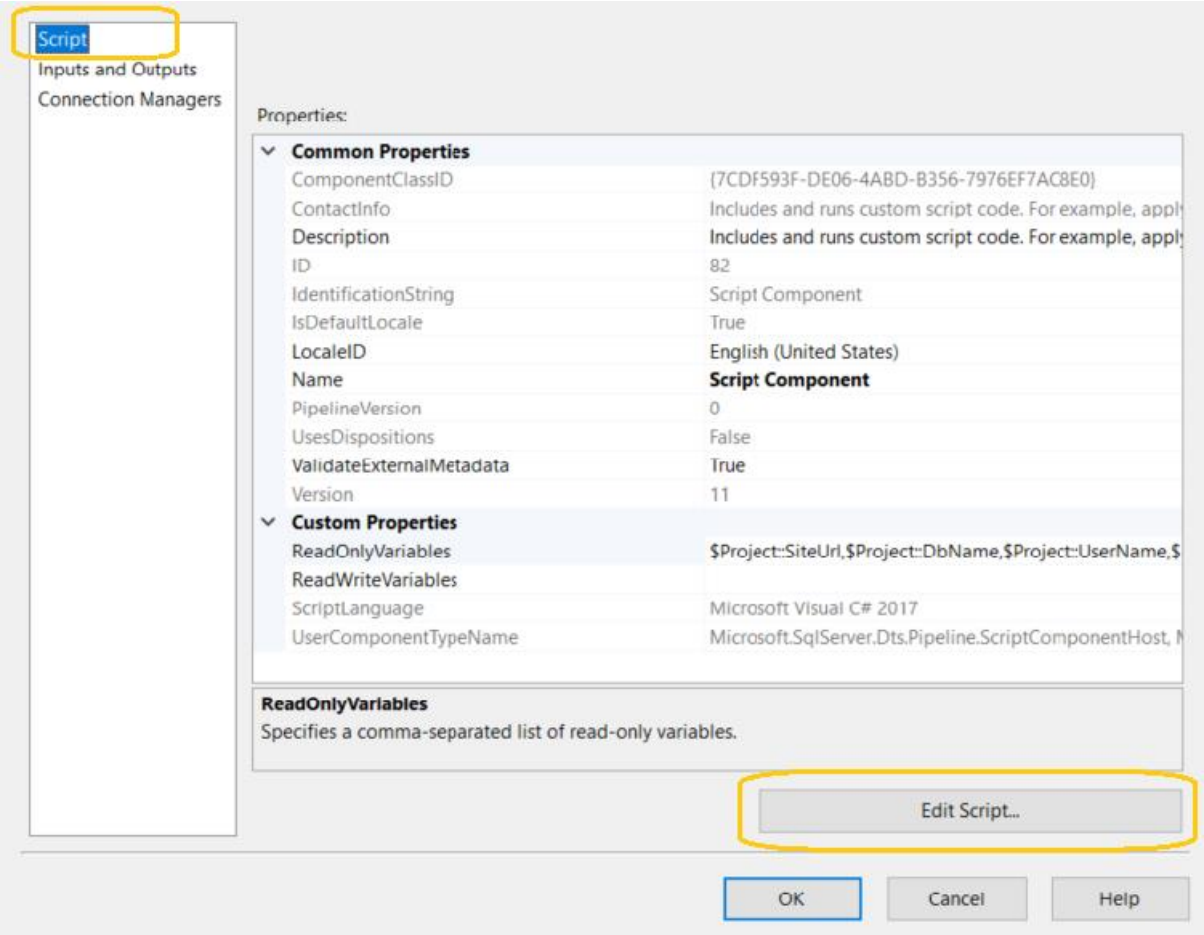
Create similar connections like EmployeeFullPackage.

Add the link from Script component to OLE DB Destination and OLE DB Destination1 for incremental package.



Now, we will write code to fetch the item from list and insert into table.

Go back to the script section and click on Edit Script button.



Add the same .dll files in references, as we have done for employeefullpackage

Create the similar helper class as we have created for employeefullpackage.

In incremental package script code, there is a change in only one file; i.e. main.cs.

main.cs

```
1. #region Help:  Introduction to the Script Component
2. /* The Script Component allows you to perform virtually any operatio
   n that can be accomplished in
```

```
 * a .Net application within the context of an Integration Services data flow.
 *
 * Expand the other regions which have "Help" prefixes for examples of specific ways to use
 * Integration Services features within this script component. */
#endregion

#region Namespaces
    using System;
    using System.Collections.Generic;
    using System.Data;
    using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
    using Microsoft.SqlServer.Dts.Runtime.Wrapper;
    using Newtonsoft.Json.Linq;
    using SC_f0b9a68df8ce499a8004dd3218dae9c0;
    #endregion

    /// <summary>
    /// This is the class to which to add your code.  Do not change the name, attributes, or parent
    /// of this class.
    /// </summary>
    [Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]
    public class ScriptMain : UserComponent
    {
        #region Help:  Using Integration Services variables and parameters
        /* To use a variable in this script, first ensure that the variable has been added to
         * either the list contained in the ReadOnlyVariables property or the list contained in
         * the ReadWriteVariables property of this script component, according to whether or not your
         * code needs to write into the variable.  To do so, save this script, close this instance of
         * Visual Studio, and update the ReadOnlyVariables and ReadWriteVariables properties in the
         * Script Transformation Editor window.
         * To use a parameter in this script, follow the same steps. Parameters are always read-only.
         *
         * Example of reading from a variable or parameter:
         *  DateTime startTime = Variables.MyStartTime;
         *
         * Example of writing to a variable:
         *  Variables.myStringVariable = "new value";
         */
        #endregion
```

```
42.
43.         #region Help:  Using Integration Services Connnection Managers
44.         /* Some types of connection managers can be used in this script component.  See the help topic
45.          * "Working with Connection Managers Programatically" for details.
46.          *
47.          * To use a connection manager in this script, first ensure that the connection manager has
48.          * been added to either the list of connection managers on the Connection Managers page of the
49.          * script component editor.  To add the connection manager, save this script, close this instance of
50.          * Visual Studio, and add the Connection Manager to the list.
51.          *
52.          * If the component needs to hold a connection open while processing rows, override the
53.          * AcquireConnections and ReleaseConnections methods.
54.          *
55.          * Example of using an ADO.Net connection manager to acquire a SqlConnection:
56.          *  object rawConnection = Connections.SalesDB.AcquireConnection(transaction);
57.          *  SqlConnection salesDBConn = (SqlConnection)rawConnection;
58.          *
59.          * Example of using a File connection manager to acquire a file path:
60.          *  object rawConnection = Connections.Prices_zip.AcquireConnection(transaction);
61.          *  string filePath = (string)rawConnection;
62.          *
63.          * Example of releasing a connection manager:
64.          *  Connections.SalesDB.ReleaseConnection(rawConnection);
65.          */
66.         #endregion
67.
68.         #region Help:  Firing Integration Services Events
69.         /* This script component can fire events.
70.          *
71.          * Example of firing an error event:
72.          *  ComponentMetaData.FireError(10, "Process Values", "Bad value", "", 0, out cancel);
73.          *
74.          * Example of firing an information event:
75.          *  ComponentMetaData.FireInformation(10, "Process Values", "Processing has started", "", 0, fireAgain);
```

```
76.           *
77.           * Example of firing a warning event:
78.           *  ComponentMetaData.FireWarning(10, "Process Values", "N
   o rows were received", "", 0);
79.           */
80.          #endregion
81.
82.          /// <summary>
83.          /// This method is called once, before rows begin to
   be processed in the data flow.
84.          ///
85.          /// You can remove this method if you don't need to d
   o anything here.
86.          /// </summary>
87.          public override void PreExecute()
88.          {
89.              base.PreExecute();
90.              /*
91.               * Add your code here
92.               */
93.          }
94.
95.          /// <summary>
96.          /// This method is called after all the rows have passed t
   hrough this component.
97.          ///
98.          /// You can delete this method if you don't need to do any
   thing here.
99.          /// </summary>
100.         public override void PostExecute()
101.         {
102.             base.PostExecute();
103.              /*
104.              * Add your code here
105.               */
106.         }
107.
108.         public override void CreateNewOutputRows()
109.         {
110.             /*
111.                Add rows by calling the AddRow method on the me
   mber variable named "<Output Name>Buffer".
112.               For example, call MyOutputBuffer.AddRow() if your ou
   tput was named "MyOutput".
113.              */
114.
115.              string siteURL = Variables.SiteUrl;
116.             string userName = Variables.UserName;
117.              string password = Variables.Pwd;
118.             DateTime date = Variables.LoadCreated;
```

```
119.             string lastModified = date.ToUniversalTime().ToSt
   ring("yyyy-MM-ddTHH:mm:ssZ");
120.
121.             CallDataEmployee(siteURL, userName, password, "ID
   ", lastModified);
122.             //CallDataEmployee(archiveURL, userName, password, "So
   urceID", obj, true, lastModified);
123.
124.             CallDataHobbies(siteURL, userName, password, "ID", las
   tModified);
125.              //CallDataHobbies(archiveURL, userName, password,
    "SourceID", lastModified);
126.         }
127.         /// <summary>
128.         /// This method is used to get data from employee list for
    last 15 min.
129.         /// </summary>
130.         /// <param name="siteURL"></param>
131.         /// <param name="userName"></param>
132.         /// <param name="password"></param>
133.         /// <param name="IDCol"></param>
134.         /// <param name="lastModified"></param>
135.         public void CallDataEmployee(string siteURL, string u
   serName, string password, string IDCol, string lastModified)
136.         {
137.             var webUri = new Uri(siteURL);
138.             using (var client = new SPHttpClient(webUri, userName,
    password))
139.             {
140.                 try
141.                 {
142.                     var listTitle = "Employee";
143.                      string colNames = IDCol + ",Title,FirstNa
   me,LastName,PhoneNo,Address,Role,IsActive,Hobbies/ID,Hobbies/T
   itle,Created,Modified,AuthorId,EditorId,Author/Title,Editor/Ti
   tle";
144.                     string filter = "&$top=4000&$filter=Modified g
   e datetime'" + lastModified + "'&$expand=Hobbies,Author,Editor";
145.
146.                     var endpointUrl = string.Format("{0}/_api/web/
   lists/getbytitle('{1}')/items?$select=" + colNames + filter, webUri,
    listTitle);
147.                      var returndata = client.ExecuteJson(endpo
   intUrl);
148.                     try
149.                      {
150.                         CallDataEmployee(returndata, IDCol);
151.                      }
152.                     catch (Exception e)
153.                      {
```

```
154.
155.                     }
156.                 }
157.                  catch (Exception e)
158.                 {
159.
160.                 }
161.             }
162.         }
163.         /// <summary>
164.         /// This method is used to iterate the result and store in
   to employeebuffer.
165.         /// </summary>
166.         /// <param name="data"></param>
167.         /// <param name="IDCol"></param>
168.         public void CallDataEmployee(JToken data, string IDCol)
169.         {
170.             foreach (var item in data["d"]["results"])
171.             {
172.                 try
173.                 {
174.                     EmployeeBuffer.AddRow();
175.                      EmployeeBuffer.ItemId = GetDecimal(item,
   IDCol, -1);
176.                     EmployeeBuffer.FullName = GetString(item, "Tit
   le");
177.                      EmployeeBuffer.FirstName = GetString(item
   , "FirstName");
178.                     EmployeeBuffer.LastName = GetString(item, "Las
   tName");
179.                      EmployeeBuffer.PhoneNum = GetInt(item, "P
   honeNo", -1);
180.                     EmployeeBuffer.Address = GetString(item, "Addr
   ess");
181.                      EmployeeBuffer.Role = GetString(item, "Ro
   le");
182.                     EmployeeBuffer.IsActive = GetString(item, "IsA
   ctive");
183.                      EmployeeBuffer.Hobbies = GetMultipleCompl
   ex(item, "Hobbies", "Title", ",");
184.                     EmployeeBuffer.Modified = GetDateTime(item, "M
   odified");
185.                      EmployeeBuffer.Created = GetDateTime(item
   , "Created");
186.                     EmployeeBuffer.CreatedById = GetDecimal(item,
   "AuthorId", -1);
187.                      EmployeeBuffer.ModifiedById = GetDecimal(
   item, "EditorId", -1);
188.                     EmployeeBuffer.CreatedBy = GetComplex(item, "A
   uthor", "Title");
```

```
189.                     EmployeeBuffer.ModifiedBy = GetComplex(it
   em, "Editor", "Title");
190.                 }
191.                  catch (Exception e1)
192.                 {
193.
194.                 }
195.              }
196.
197.         }
198.         /// <summary>
199.         /// This method is used to get data from Hobbies list
    for last 15 min.
200.         /// </summary>
201.         /// <param name="siteURL"></param>
202.         /// <param name="userName"></param>
203.         /// <param name="password"></param>
204.         /// <param name="IDCol"></param>
205.         /// <param name="lastModified"></param>
206.         public void CallDataHobbies(string siteURL, string userNam
   e, string password, string IDCol, string lastModified)
207.         {
208.             var webUri = new Uri(siteURL);
209.              using (var client = new SPHttpClient(webUri, user
   Name, password))
210.             {
211.                  try
212.                 {
213.                     var listTitle = "Hobbies";
214.                     string colNames = IDCol + ",Title,Created,Modi
   fied,AuthorId,EditorId,Author/Title,Editor/Title";
215.                     string filter = "&$top=4000&$filter=Modif
   ied ge datetime'" + lastModified + "'&$expand=Author,Editor";
216.
217.                     var endpointUrl = string.Format("{0}/_api
   /web/lists/getbytitle('{1}')/items?$select=" + colNames + filt
   er, webUri, listTitle);
218.                     var returndata = client.ExecuteJson(endpointUr
   l);
219.                     try
220.                     {
221.                         CallDataHobbies(returndata, IDCol);
222.                     }
223.                     catch (Exception e)
224.                     {
225.
226.                     }
227.                 }
228.                 catch (Exception e)
```

```
229.                 {
230.
231.                 }
232.             }
233.         }
234.
235.         /// <summary>
236.         /// This method is used to iterate the result and store in
    to Hobbies buffer.
237.         /// </summary>
238.         /// <param name="data"></param>
239.         /// <param name="IDCol"></param>
240.         public void CallDataHobbies(JToken data, string IDCol)
241.         {
242.             foreach (var item in data["d"]["results"])
243.              {
244.                 try
245.                  {
246.                     HobbiesBuffer.AddRow();
247.                      HobbiesBuffer.ItemId = GetDecimal(item, I
    DCol, -1);
248.                      HobbiesBuffer.Title =
    GetString(item, "Title");
249.                     HobbiesBuffer.Modified = GetDateTime(item, "Mo
    dified");
250.                      HobbiesBuffer.Created = GetDateTime(item,
     "Created");
251.                     HobbiesBuffer.CreatedById = GetDecimal(item, "
    AuthorId", -1);
252.                      HobbiesBuffer.ModifiedById = GetDecimal(i
    tem, "EditorId", -1);
253.                     HobbiesBuffer.CreatedBy = GetComplex(item, "Au
    thor", "Title");
254.                      HobbiesBuffer.ModifiedBy = GetComplex(ite
    m, "Editor", "Title");
255.                 }
256.                  catch (Exception e1)
257.                 {
258.
259.                 }
260.              }
261.
262.         }
263.         public string GetString(JToken token, string key)
264.         {
265.             string value = string.Empty;
266.
267.             try
268.              {
269.                 value = Convert.ToString(token[key]);
```

```
270.             }
271.             catch (Exception e)
272.             {
273.
274.             }
275.
276.             return value;
277.         }
278.
279.         public string GetComplex(JToken token, string key, string
    var)
280.         {
281.             string value = string.Empty;
282.
283.             try
284.             {
285.                 JToken tok = token[key];
286.                  value = GetString(tok, var);
287.             }
288.             catch (Exception e)
289.             {
290.
291.             }
292.
293.             return value;
294.         }
295.
296.         public Decimal GetComplexID(JToken token, string key,
    string var)
297.         {
298.             Decimal dec = -1;
299.
300.             try
301.             {
302.                 JToken tok = token[key];
303.                dec = GetDecimal(tok, var, -1);
304.             }
305.             catch (Exception e)
306.             {
307.
308.             }
309.
310.             return dec;
311.         }
312.
313.         public Decimal GetDecimal(JToken token, string key, Decima
    l defaultVal)
314.         {
315.             Decimal dec = -1;
316.
```

```
317.             bool complete = Decimal.TryParse(GetString(token, key)
   , out dec);
318.
319.             if (complete)
320.              {
321.                 return dec;
322.              }
323.             else
324.              {
325.                 return defaultVal;
326.              }
327.         }
328.
329.         public Int32 GetInt(JToken token, string key, Int32 defaul
   tVal)
330.         {
331.             Int32 dec = -1;
332.
333.             bool complete = Int32.TryParse(GetString(token, key),
   out dec);
334.
335.             if (complete)
336.              {
337.                 return dec;
338.              }
339.             else
340.              {
341.                 return defaultVal;
342.              }
343.         }
344.
345.         public DateTime GetDateTime(JToken token, string key)
346.         {
347.             DateTime dec;
348.              bool complete = DateTime.TryParse(GetString(token
   , key), out dec);
349.             return dec.ToLocalTime();
350.         }
351.
352.         public string GetMultipleComplex(JToken token, string
    key, string objectKey, string seperater)
353.         {
354.              string tempString = string.Empty;
355.             List<string> strList = new List<string>();
356.              try
357.             {
358.                  JToken stringTokens = token[key]["results"];

359.
360.                  foreach (var user in stringTokens)
```

```
361.                 {
362.                     string tempStr = GetString(user, objectKe
   y);
363.                     strList.Add(tempStr);
364.                 }
365.                 if (strList.Count > 0)
366.                 {
367.                     tempString = string.Join(seperater, strList);

368.                 }
369.             }
370.             catch (Exception ex)
371.             {
372.
373.             }
374.
375.             return tempString;
376.         }
377.
378.     }
```

No changes are required in BufferWrapper.cs and ComponentWrapper.cs classes for incremental package.

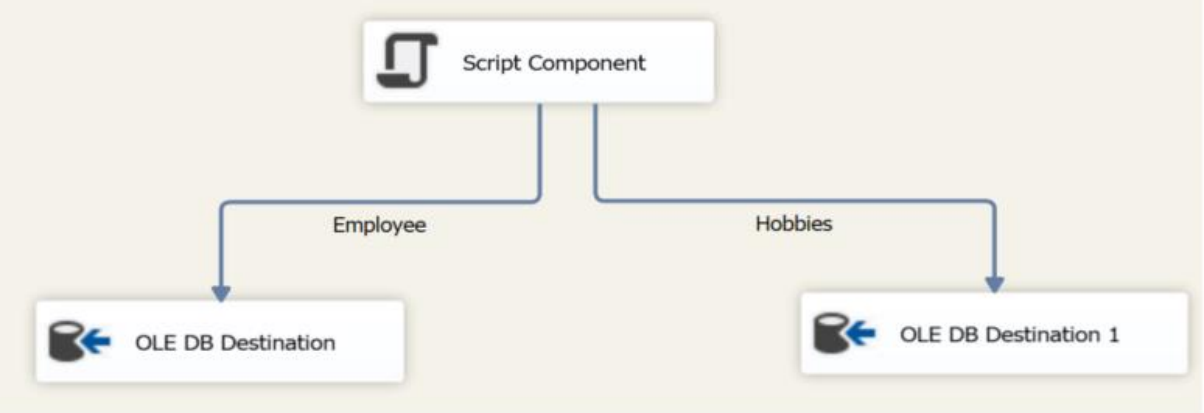
Save the changes by clicking on Ok button present in previous visual studio window.

OLE DB Destination

As we know, each OLE DB Destination has three parts; i.e. Connection Managers, Mappings and Error Output.

Configure the *Connection Managers* and *Mappings* for employee and hobbies in incremental package, similar to employeefullpackage.

If data flow task is configured properly for incremental package it will look like:



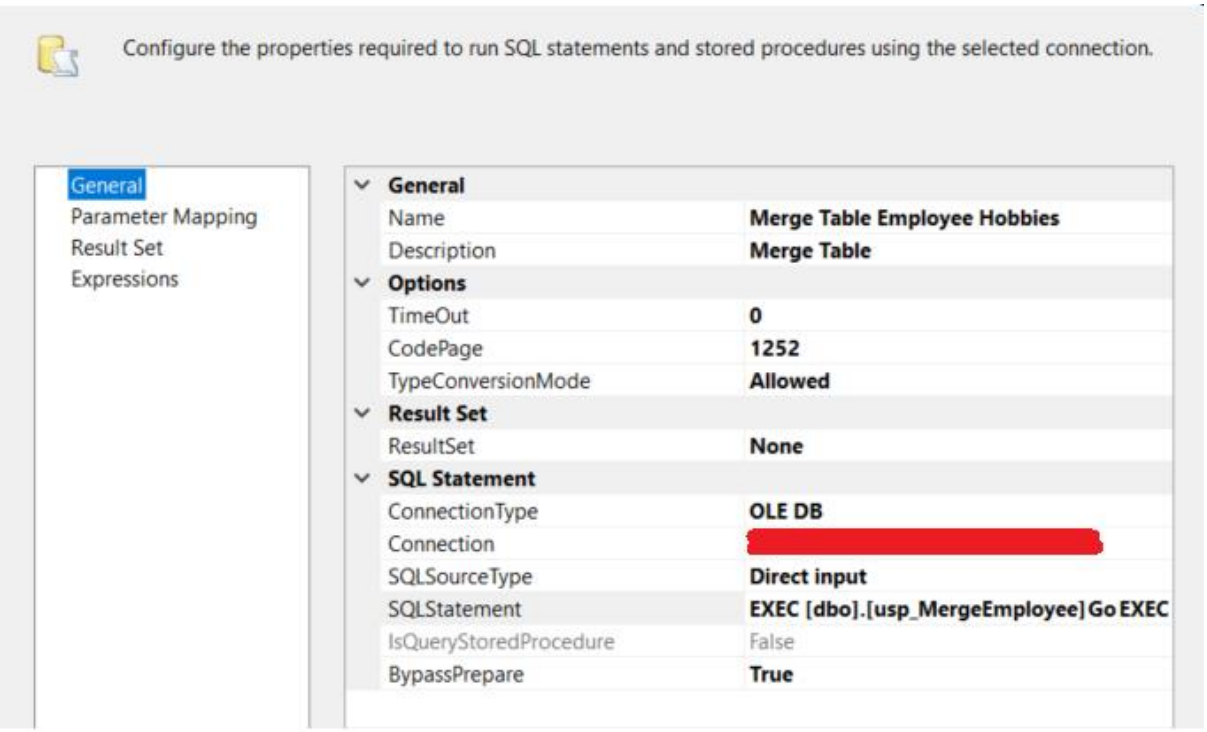
Now we are going to configure the last component of control flow task for EmployeeIncrementalPackage.dtsx

Execute SQL Task1

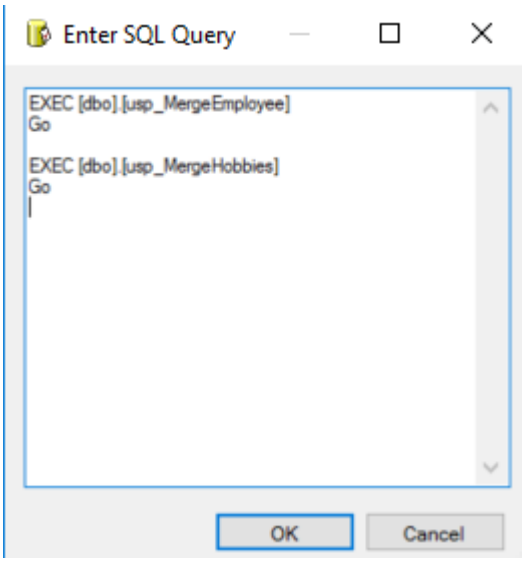
In this task we insert or update records in main table (i.e Employee and Hobbies) from stage table (i.e. Employee_Stage and Hobbies_Stage) by calling stored procedures (usp_MergeEmployee & usp_MergeHobbies).

If item already exists in main table then store procedure will update the item, else insert the item in main table.

Open the task, rename the task as *Merge Table Employee Hobbies* and configure in this way.

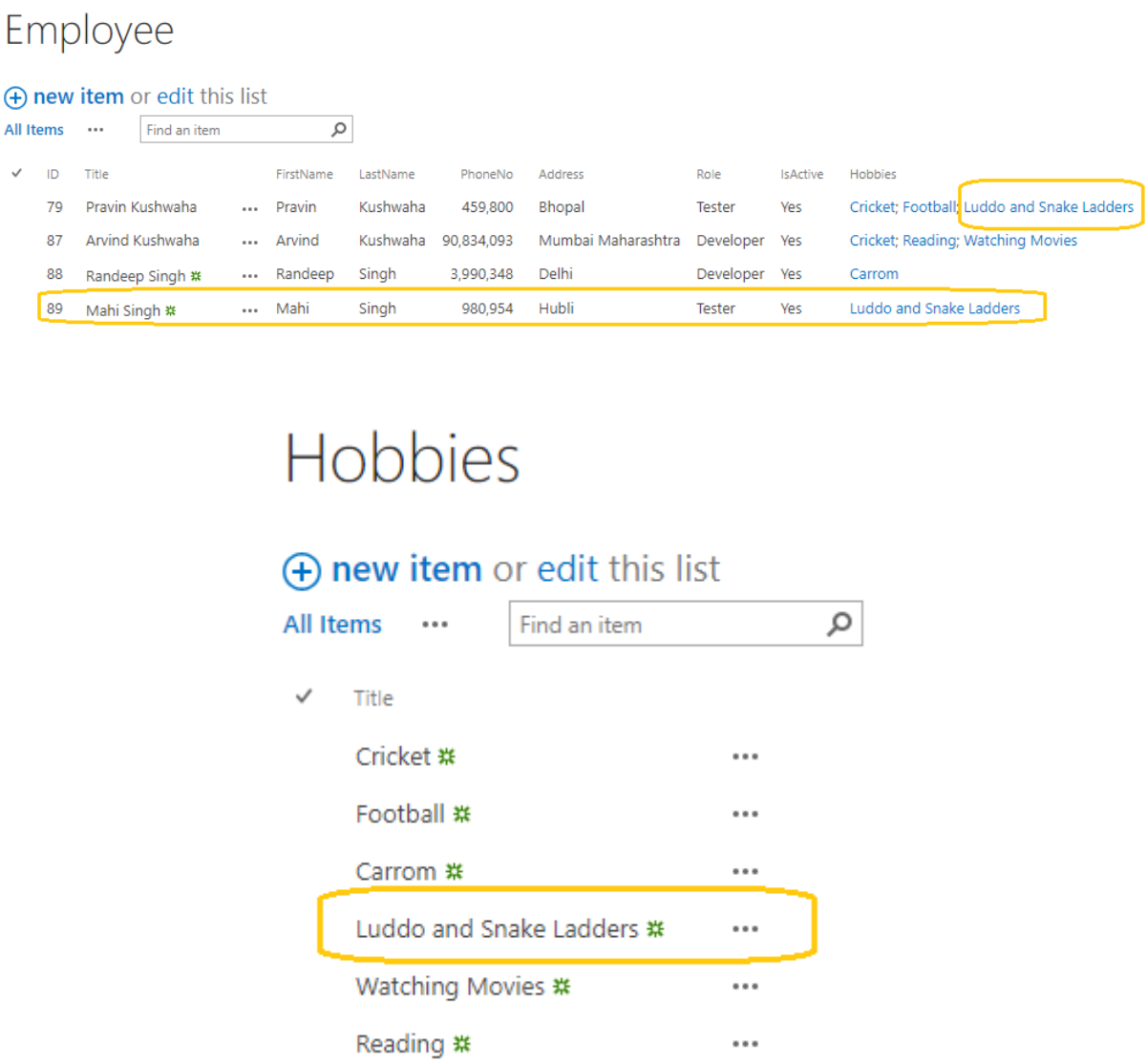


Open SQL Statement and paste the below command in it.

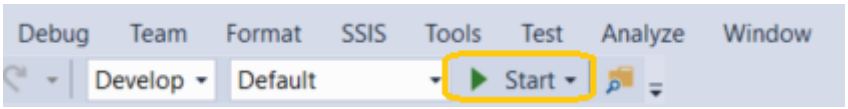


Run and validate the EmployeeIncrementalPackage

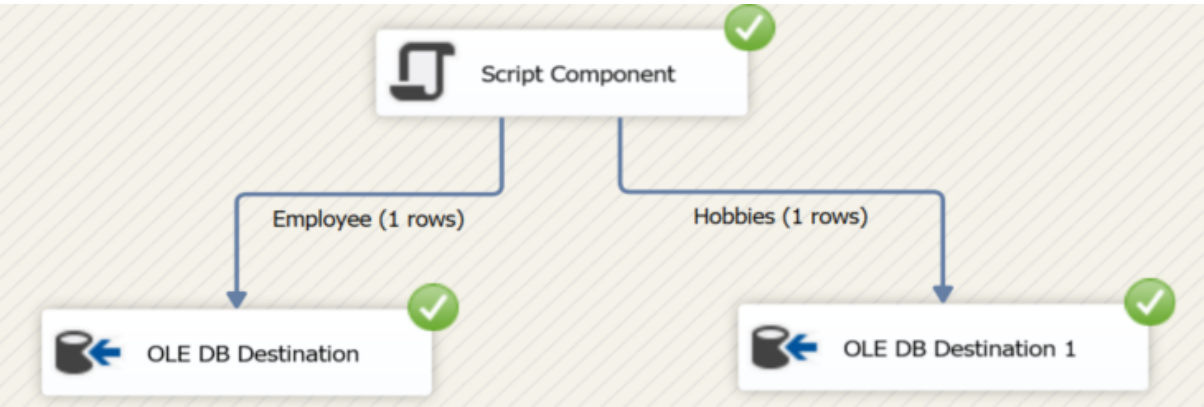
To test and verify please add one item in employee list and update Hobbies list like this.



Now, simply click on start icon present in Visual Studio menu.



If package runs without any errors then it will look like:



Now check the Employee_Stage and Employee table in database SP_POC where Employee_Stage has only new item and Employee has all the items.

In the same way Hobbies_Stage has updated item and Hobbies has all the items.

Records in Employee_Stage table

100 %

Results | Messages

	ItemId	FullName	FirstName	LastName	PhoneNum	Address	Role	IsActive	Hobbies	Created	Modified
1	89	Mahi Singh	Mahi	Singh	980954	Hubli	Tester	Yes	Luddo and Snake Ladders	2020-05-06 17:15:06.000	2020-05-07 19:16:14.00

Records in Employee table

100 %

Results | Messages

	ItemId	FullName	FirstName	LastName	PhoneNum	Address	Role	IsActive	Hobbies	Created
1	79	Pravin Kushwaha	Pravin	Kushwaha	459800	Bhopal	Tester	Yes	Cricket,Football,Luddo and Snake Ladders	2020-04-30 1
2	87	Arvind Kushwaha	Arvind	Kushwaha	90834093	Mumbai Maharashtra	Developer	Yes	Cricket,Reading,Watching Movies	2020-05-01 1
3	88	Randeep Singh	Randeep	Singh	3990348	Delhi	Developer	Yes	Carrom	2020-05-06 1
4	89	Mahi Singh	Mahi	Singh	980954	Hubli	Tester	Yes	Luddo and Snake Ladders	2020-05-06 1

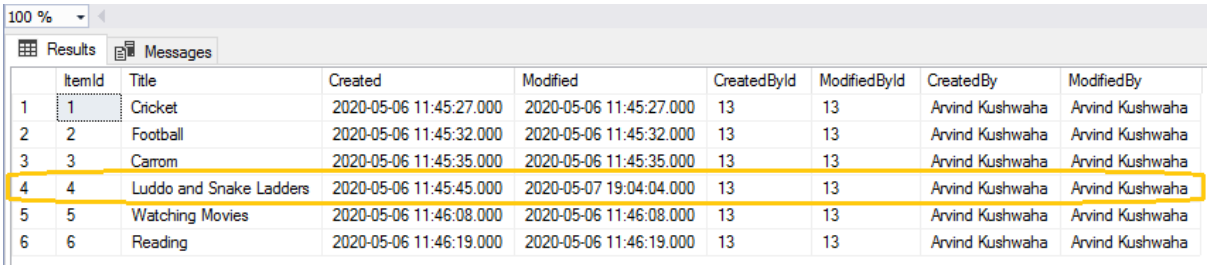
Records in Hobbies_Stage table

100 %

Results | Messages

	ItemId	Title	Created	Modified	CreatedById	ModifiedById	CreatedBy	ModifiedBy
1	4	Luddo and Snake Ladders	2020-05-06 11:45:45.000	2020-05-07 19:04:04.000	13	13	Arvind Kushwaha	Arvind Kushwaha

Records in Hobbies table



100 %

Results | Messages

	ItemId	Title	Created	Modified	CreatedById	ModifiedById	CreatedBy	ModifiedBy
1	1	Cricket	2020-05-06 11:45:27.000	2020-05-06 11:45:27.000	13	13	Arvind Kushwaha	Arvind Kushwaha
2	2	Football	2020-05-06 11:45:32.000	2020-05-06 11:45:32.000	13	13	Arvind Kushwaha	Arvind Kushwaha
3	3	Carrom	2020-05-06 11:45:35.000	2020-05-06 11:45:35.000	13	13	Arvind Kushwaha	Arvind Kushwaha
4	4	Luddo and Snake Ladders	2020-05-06 11:45:45.000	2020-05-07 19:04:04.000	13	13	Arvind Kushwaha	Arvind Kushwaha
5	5	Watching Movies	2020-05-06 11:46:08.000	2020-05-06 11:46:08.000	13	13	Arvind Kushwaha	Arvind Kushwaha
6	6	Reading	2020-05-06 11:46:19.000	2020-05-06 11:46:19.000	13	13	Arvind Kushwaha	Arvind Kushwaha

Conclusion

We have seen how to create, debug, run and see the result in database table for full and incremental package.

Introduction

Hi guys, this is the third article on how to Load data to a SQL Table from a SharePoint list using SSIS. If you have not checked my previous articles, then please go through them before starting this one.

- Created List and Store procedures - Load Data to an SQL Table from a SharePoint List Using SSIS - Part One
- Created Full and Incremental Package using SSIS - Load data to an SQL table from a sharepoint list Using SSIS - Part Two

In my previous articles, we have completed the first three major steps, now we will target the remaining ones.

- Build and Deploy the solution
- Schedule the SSIS packages.

According to an article on MSDN, integration services supports two deployment models:

- Project deployment model
- Legacy package deployment model

The project deployment model enables you to deploy your projects to the Integration Services server.

Note

The project deployment model was introduced in SQL Server 2012 Integration Services (SSIS). With this deployment model, you were not able to deploy one or more packages without deploying the whole project. SQL Server 2016 Integration Services (SSIS) introduced the Incremental Package Deployment feature, which lets you deploy one or more packages without deploying the whole project.

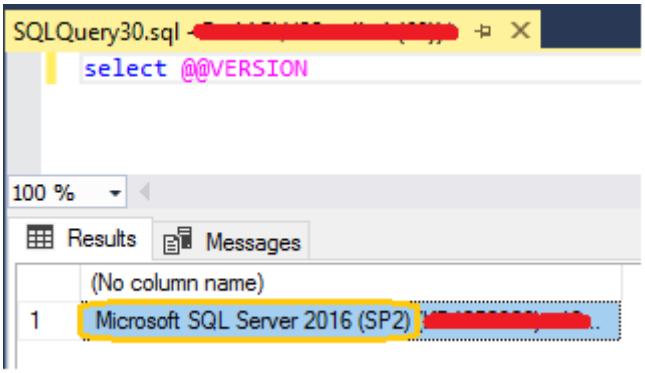
In this article, we will deploy our project using the project deployment model. For another method, you can check the MSDN article

Build and Deploy the solution

Before beginning with build and deployment, check the MS SQL Server version and targeted build version in Visual Studio.

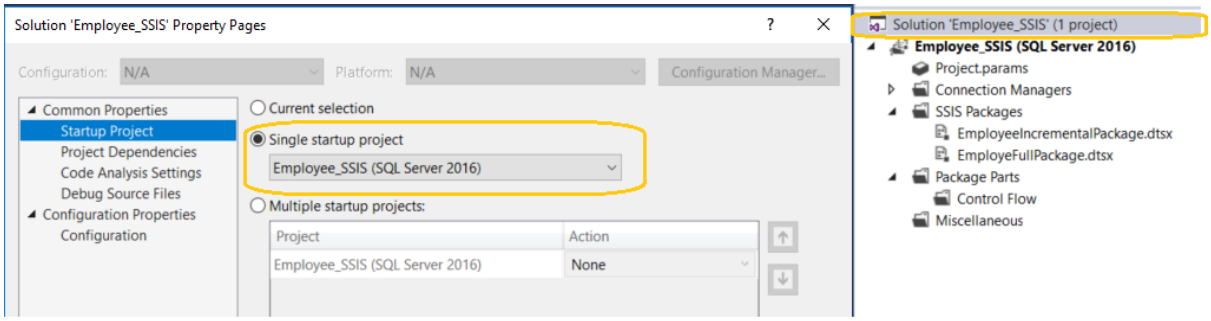
To check MS SQL Server version, open the Microsoft SQL Server Management Studio with proper credentials, open the *New Query* window and type below command.

select @@VERSION



Check the targeted version in Visual Studio

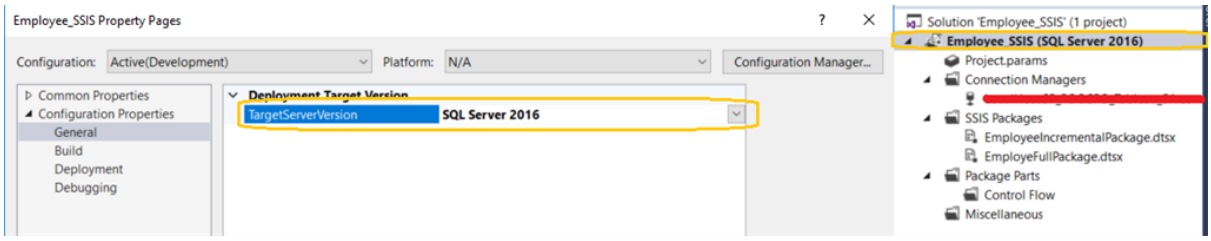
Right-click on the solution - > Properties, check the *Single startup project* field value. It should match with your SQL Server version.



If the *Single startup project* field has a different SQL Server version, then change by,

Right-click on the project -> Properties, it will open the Configuration window then expand the Configuration Properties - >General, change the value of the TargetServerVersion field to your SQL server version.

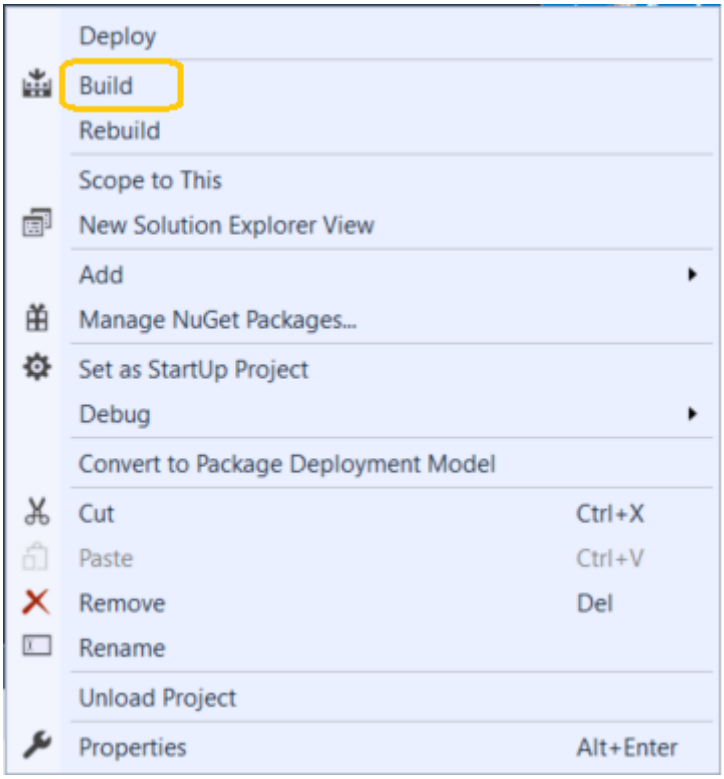
Once the above step has completed, the *Single startup projec*t field should contain the new value.

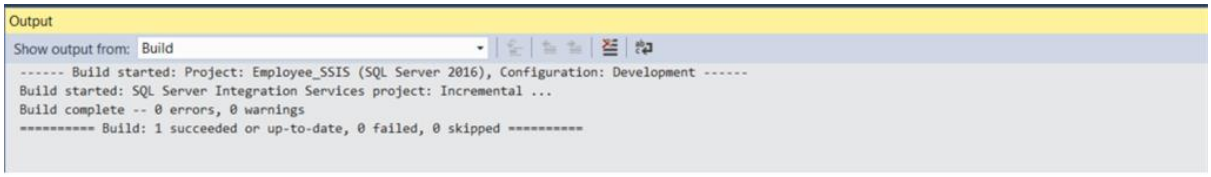


Since I have a similar SQL server version and target server version, let's build the project and then deploy it.

To Build

Right-click on the project and select build options.



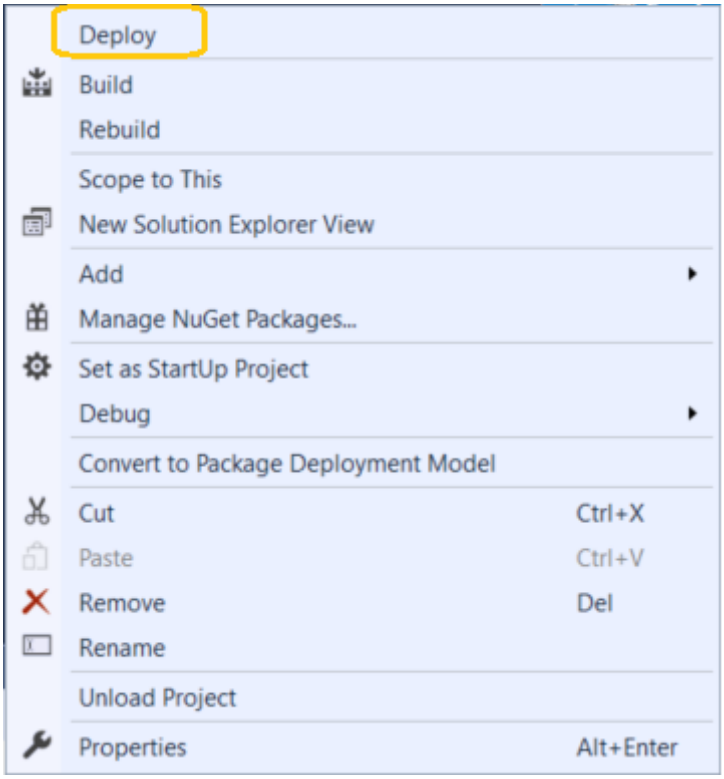


To Deploy

We can deploy our project in two ways.

Method 1

Right-click on the project and select deploy options.



This will open the SSIS deployment wizard. Remember, it will deploy the entire project, with all packages included.

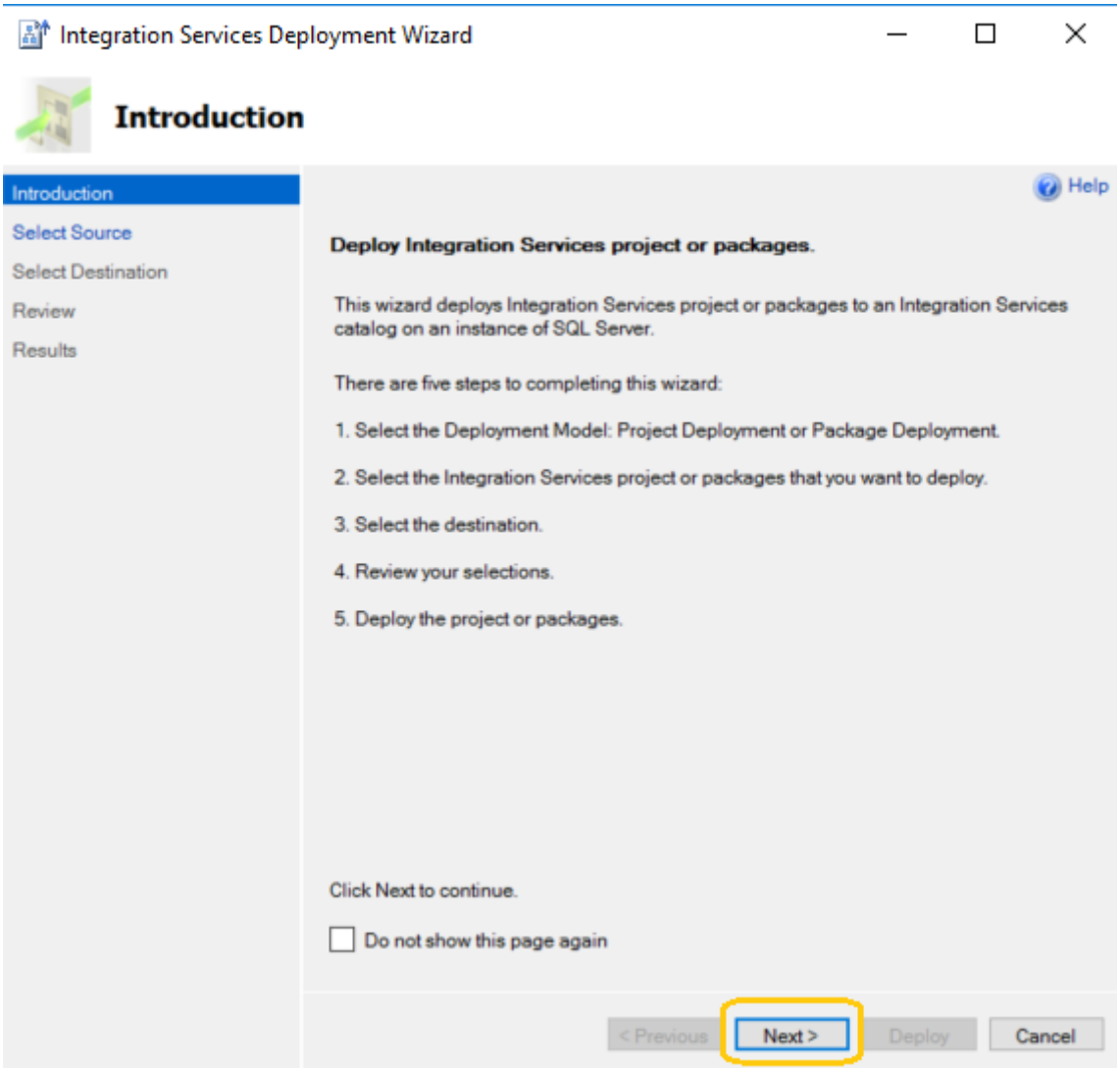
If you want to deploy an individual package, simply right-click on the particular package itself and choose to *Deploy* it (This has been possible since SSIS 2016).

The SSIS deployment Wizard consists of five steps, i.e...

- Introduction
- Select Source
- Select Destination
- Review
- Results

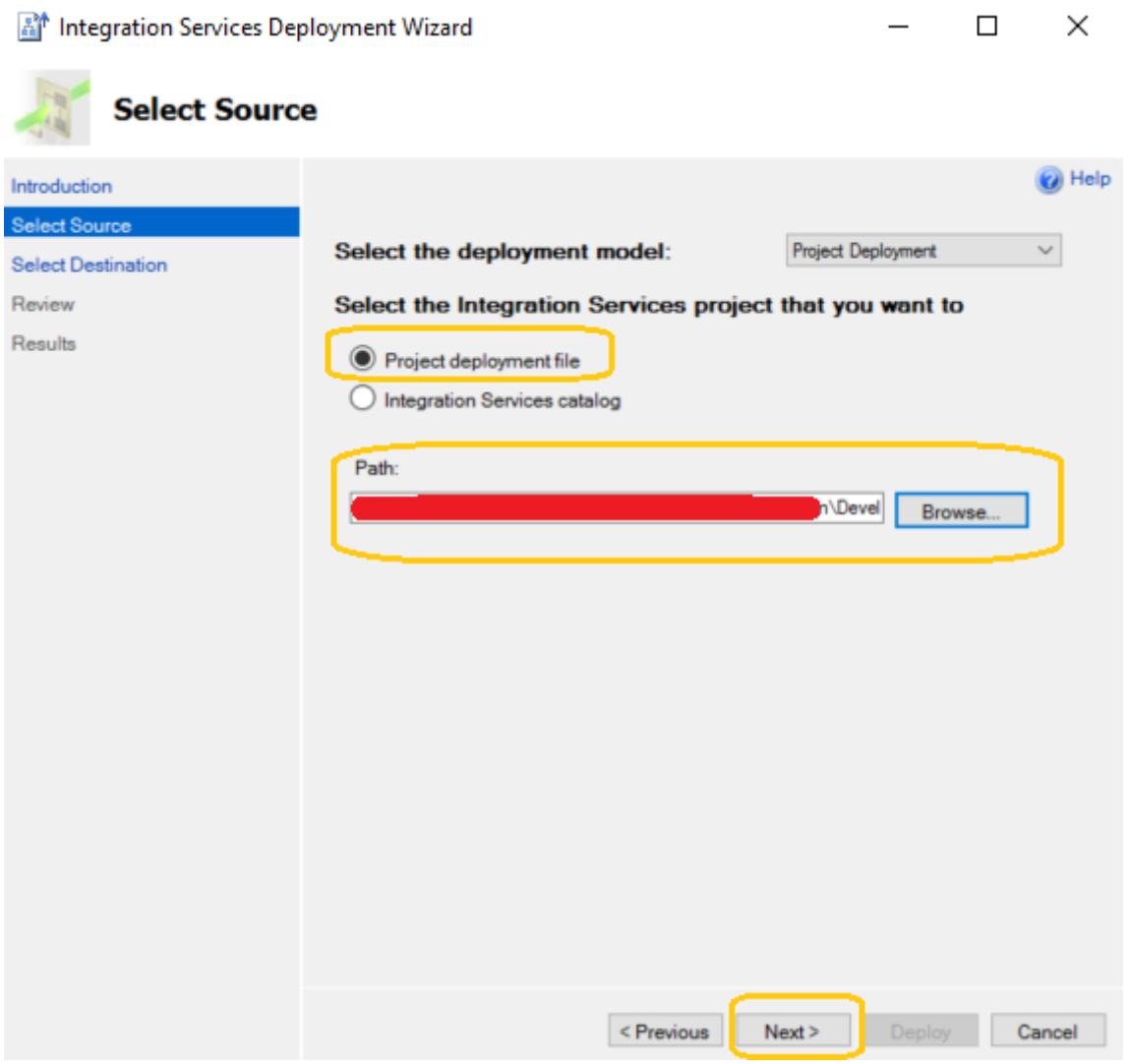
Introduction

Click on the Next button



Select Source

Select the path of our .ispac file in source wizard i.e. Project Directory\Project\Employee_SSIS\bin\Development\Employee_SSIS.ispac

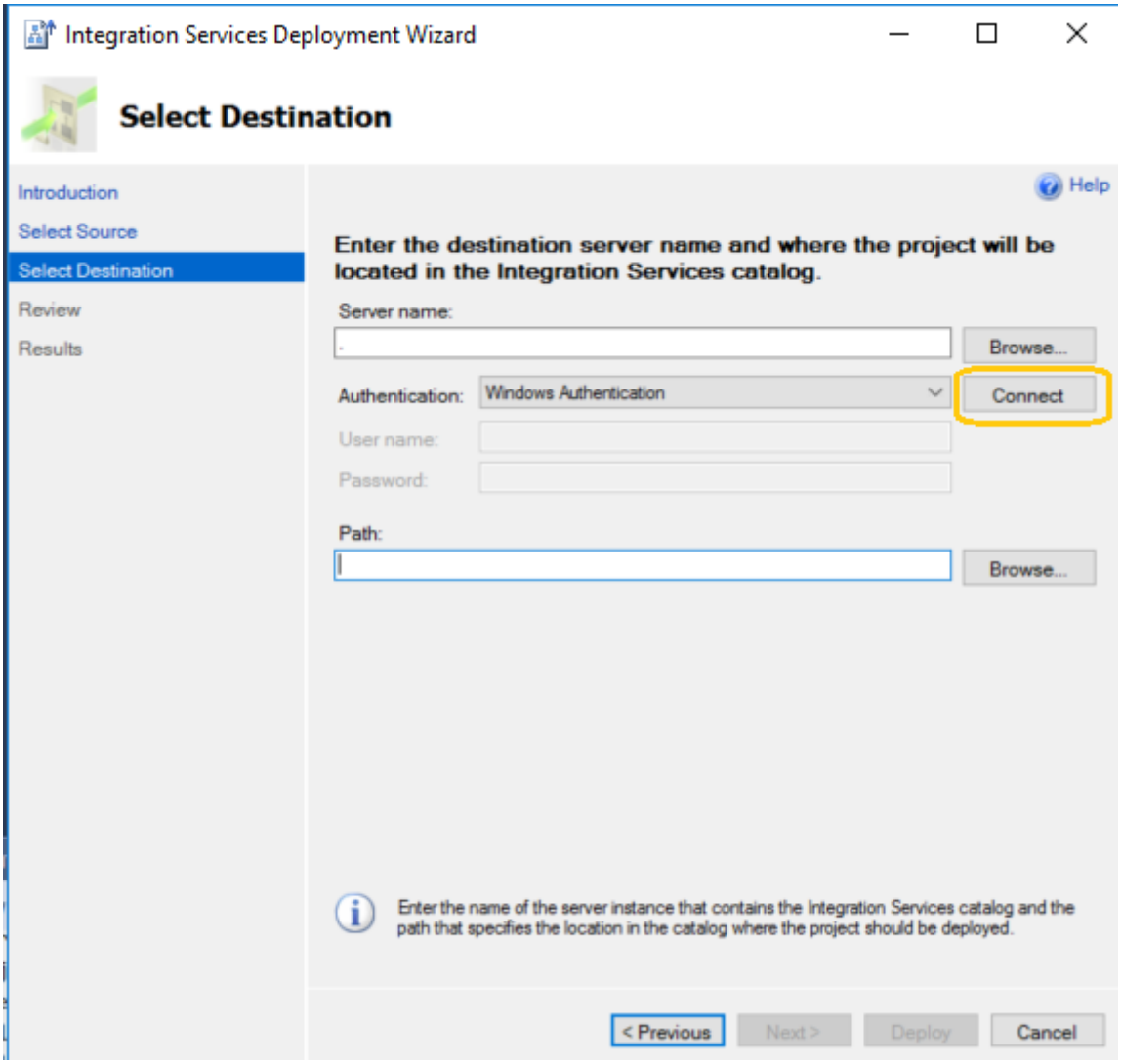


Select Destination

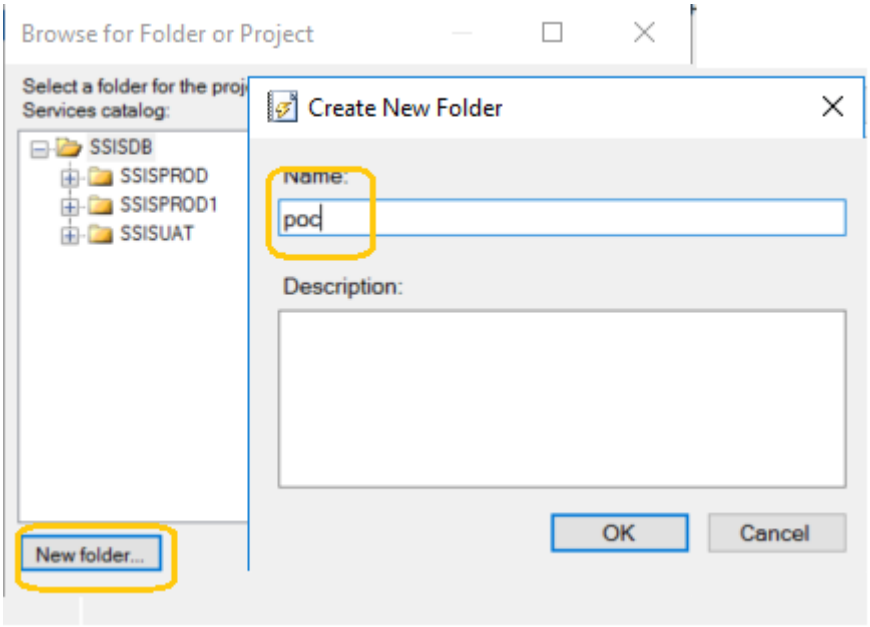
Here, we have to choose our destination.

ServerName - Enter .(dot) or actual server name

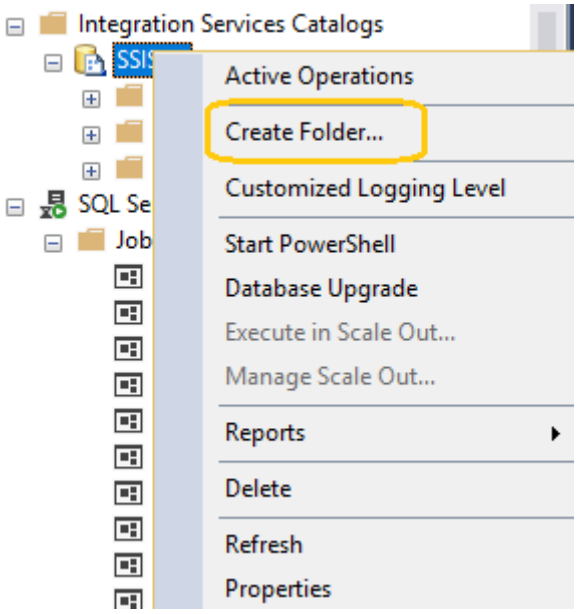
After selecting the proper authentication, click on the connect button to enable path field.



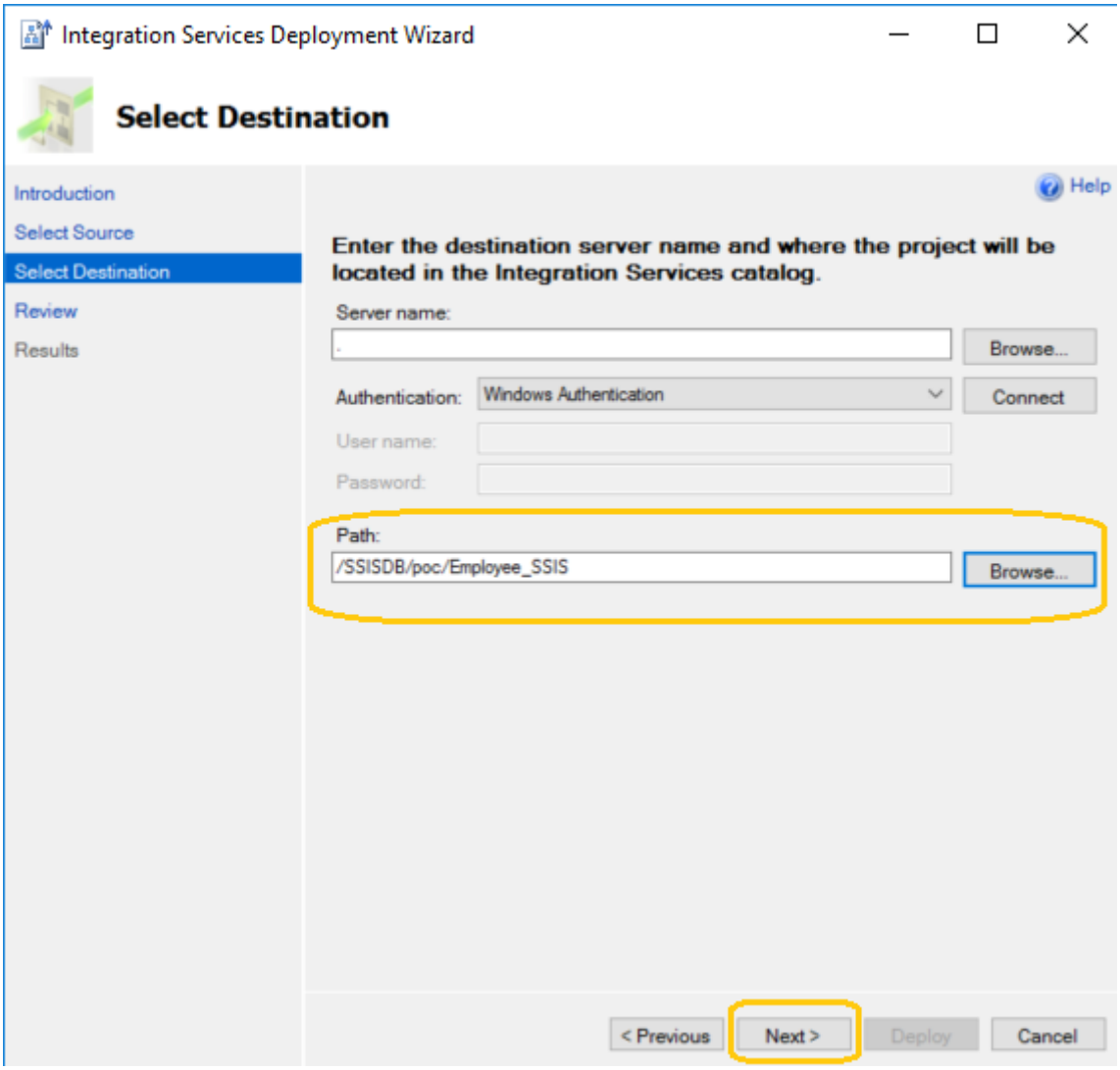
If the SSIS catalog already exists, then please choose it, otherwise, you can also create a new folder to store the project in.



You can also create a folder from Microsoft SQL Server Management Studio under Integration Services Catalogs -> SSISDB and select this folder in the destination path.

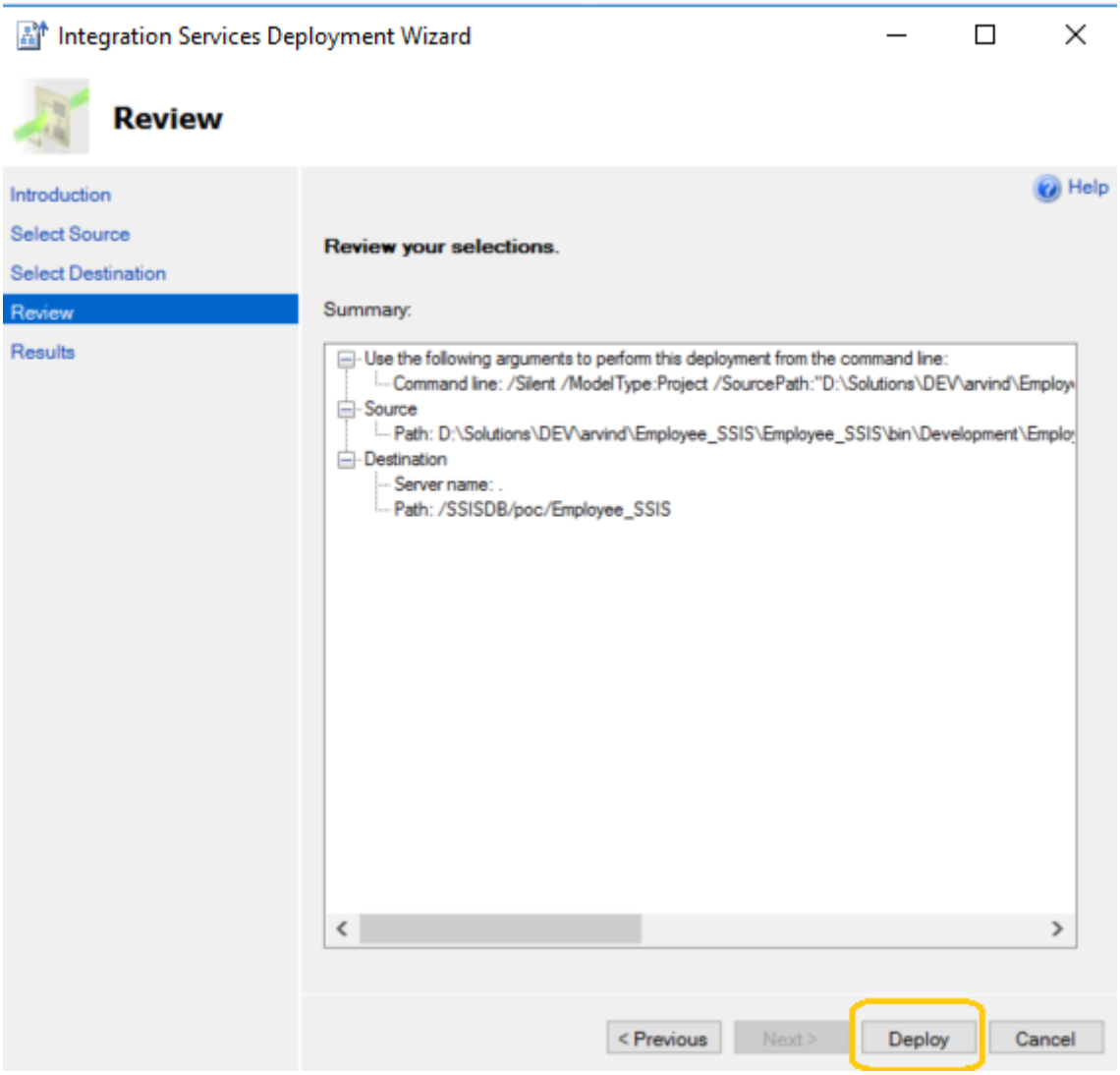


Once the path of destination is selected, click on the Next button.



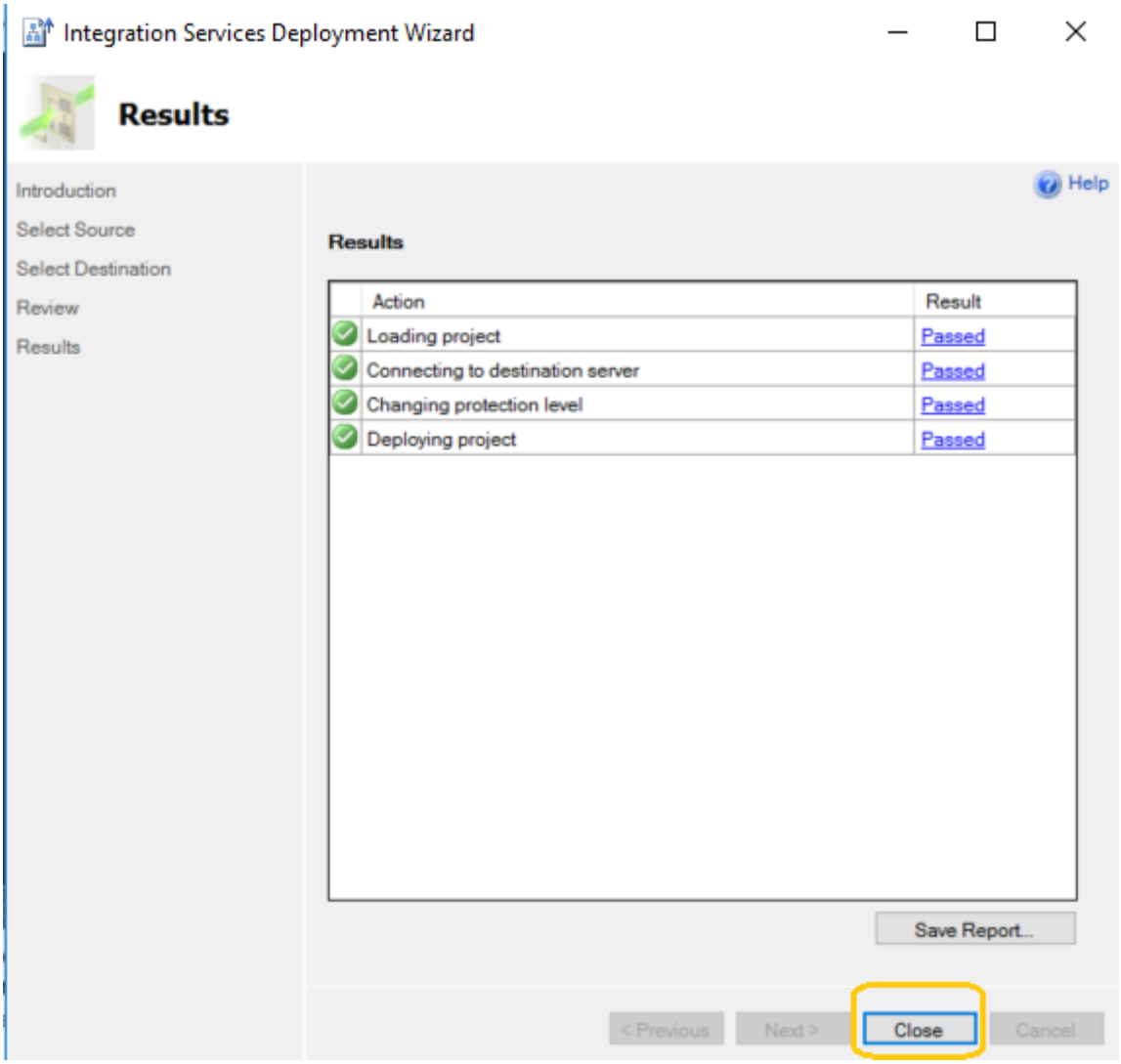
Review

In this step, you will see an overview of the actions the wizard will take. Hit the Deploy button to start the deployment.



Results

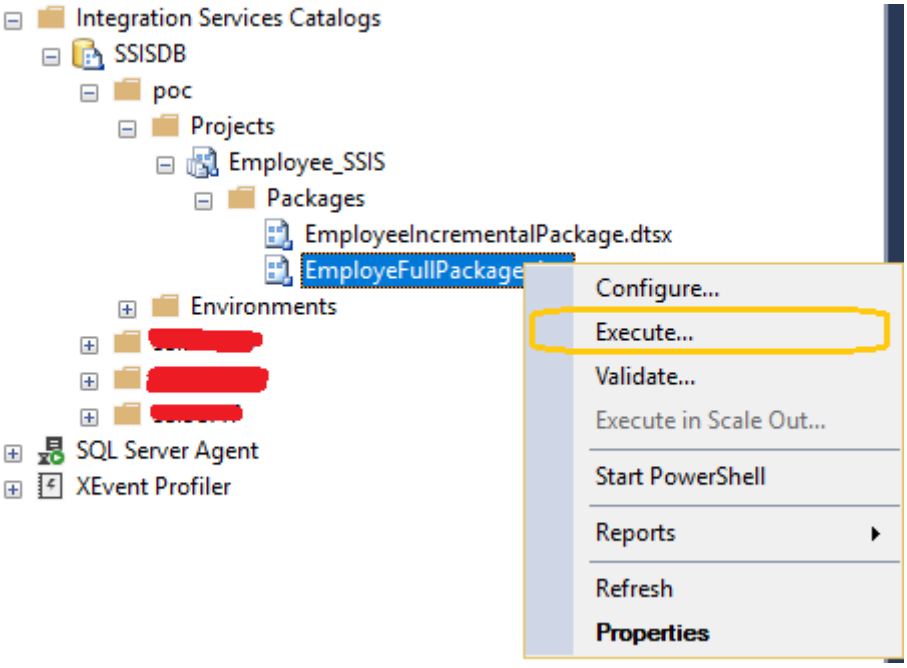
The deployment will go through a couple of steps.



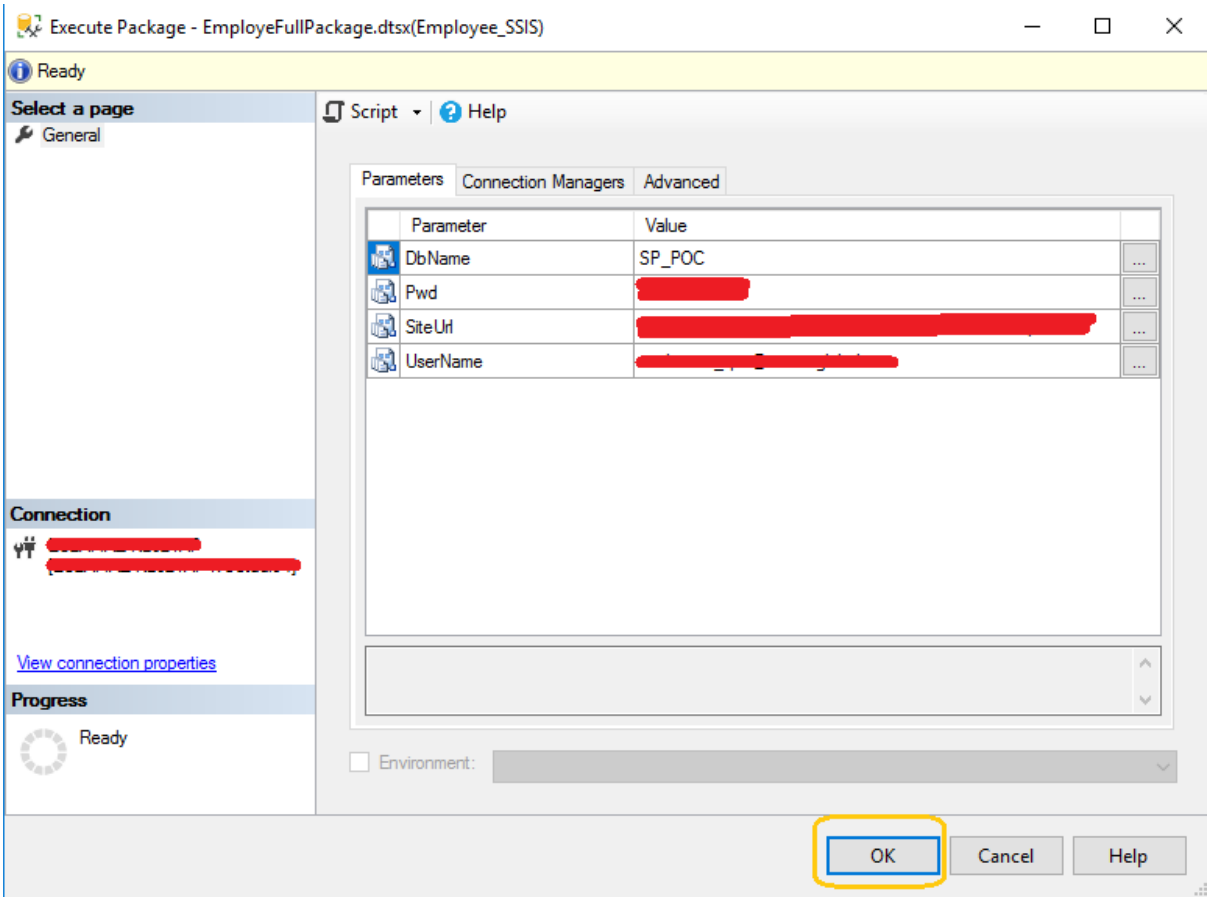
Now, the project has been successfully deployed to the server. You can find the project in the catalog of the SQL server.

Execute the package on SSIS server

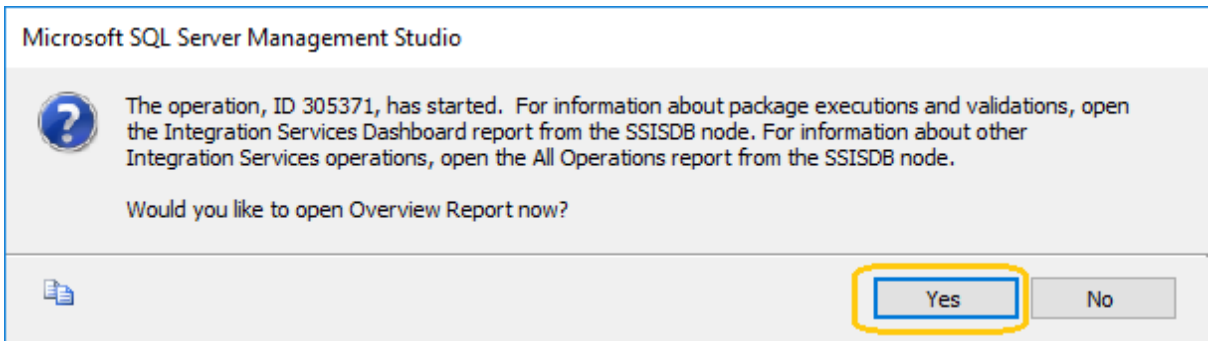
Right-click on the package and hit execute to execute the package on the SSIS server.



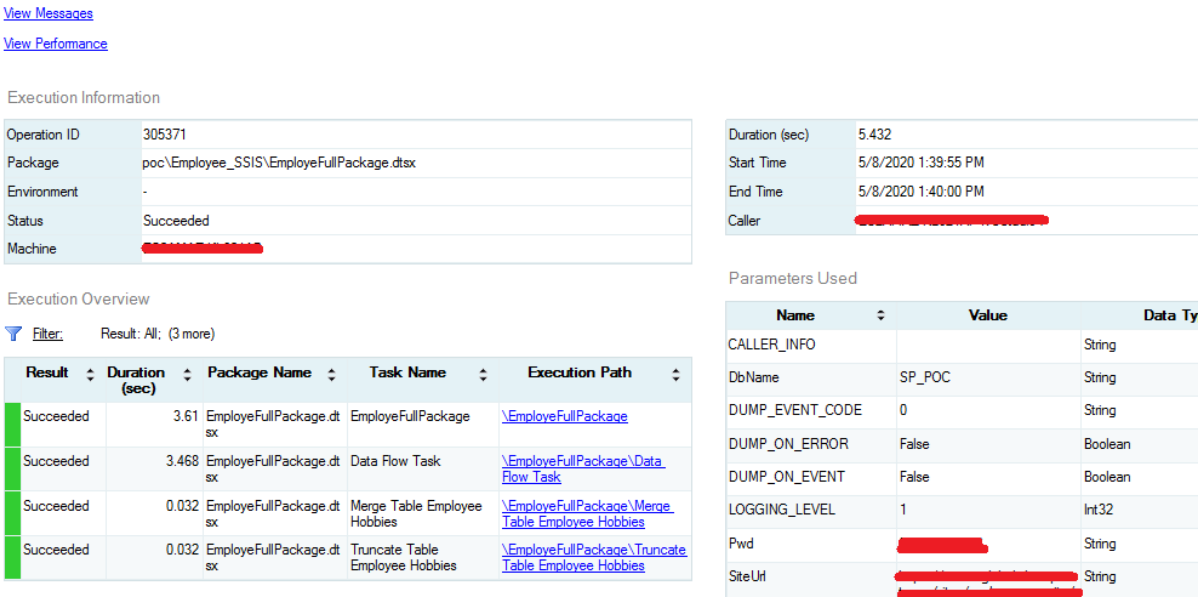
You will be taken to a dialog where you can edit certain properties, such as the connection managers, parameters, and so on.



Confirm the pop-up to open the overview reports.

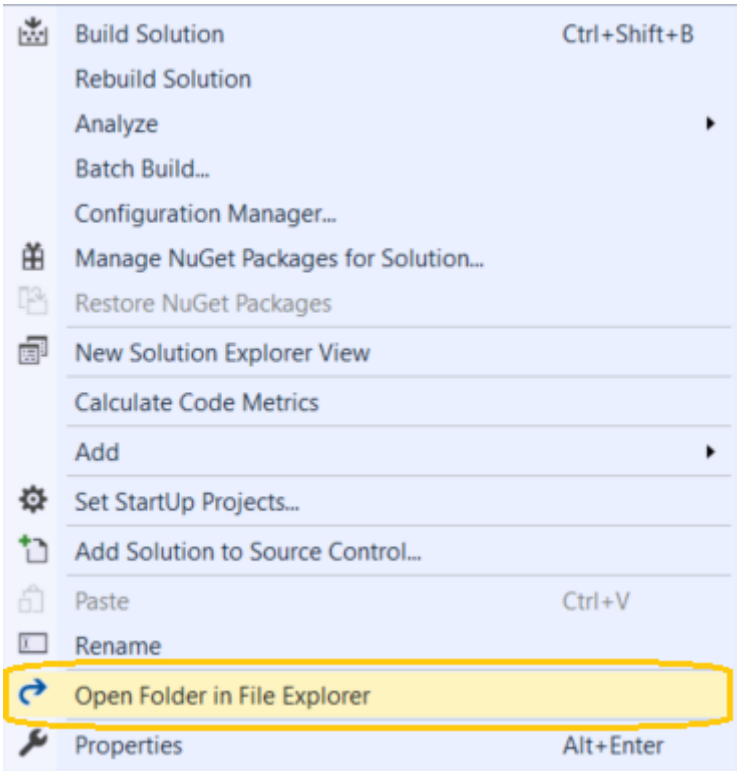


The results of the reports will look like the following:

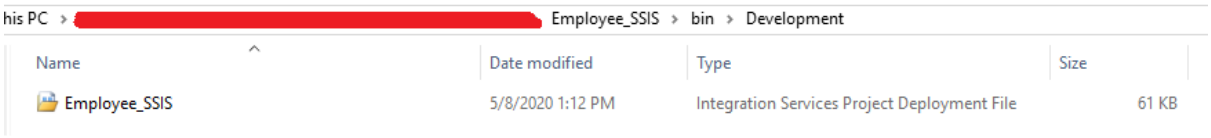


Method 2

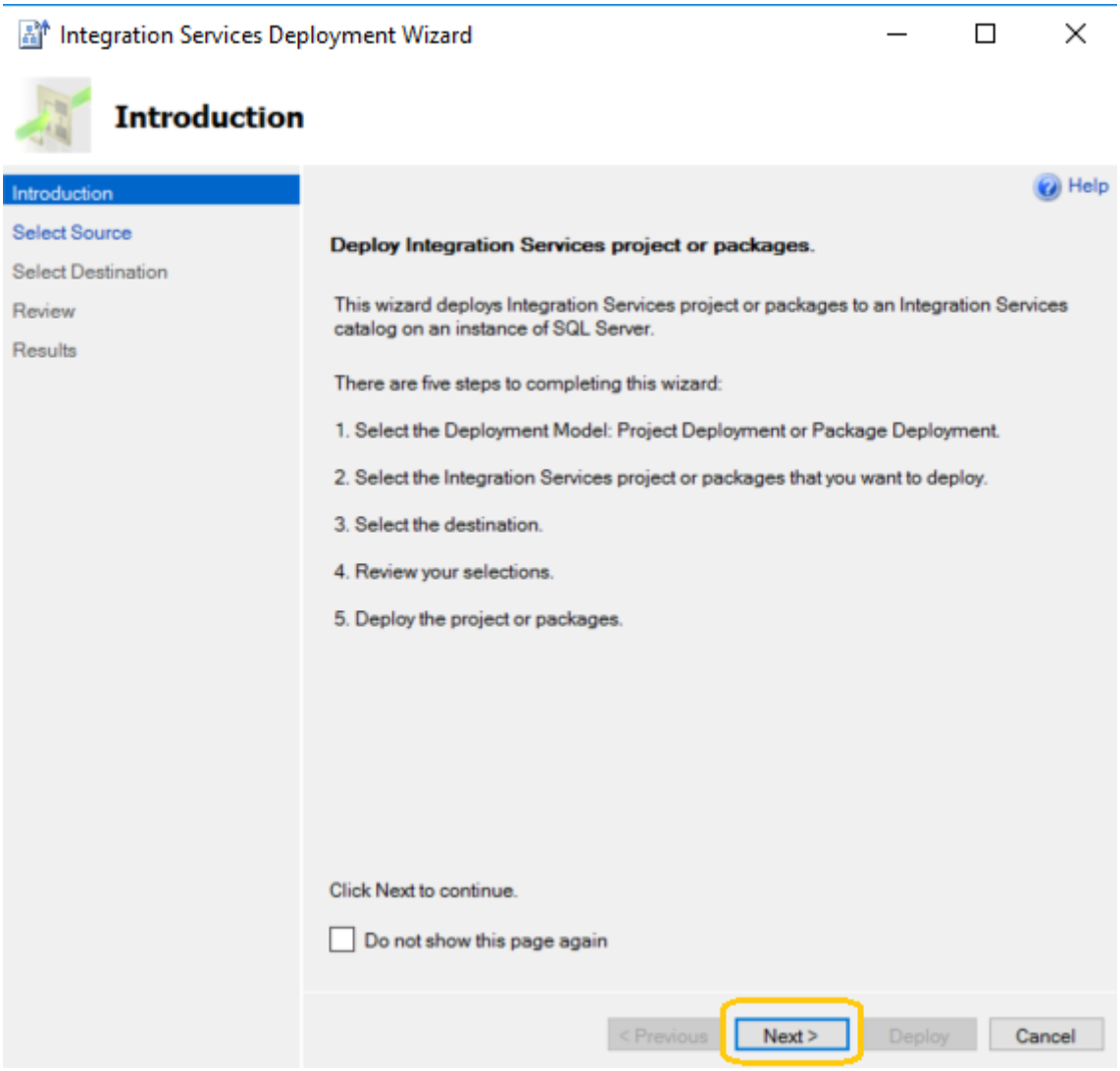
Right-click on the solution and select *Open Folder in File Explorer.*



Navigate to \bin\Development i.e my case \Employee_SSIS\bin\Development, double click on Employee_SSIS file to start the deployment.



Now, it will open a deployment wizard similar to Method 1.



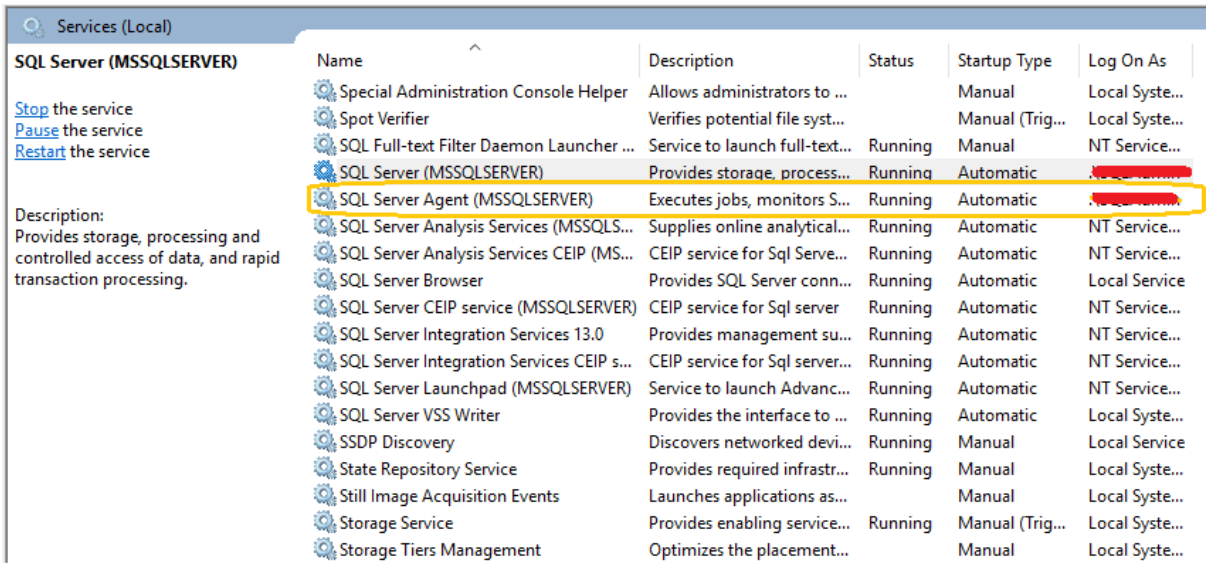
Follow the same steps of Method 1 to deploy the project in the SSIS server.

Schedule the SSIS packages

We are scheduling the deployed SSIS package using SQL Server Agent to avoid manually running these packages.

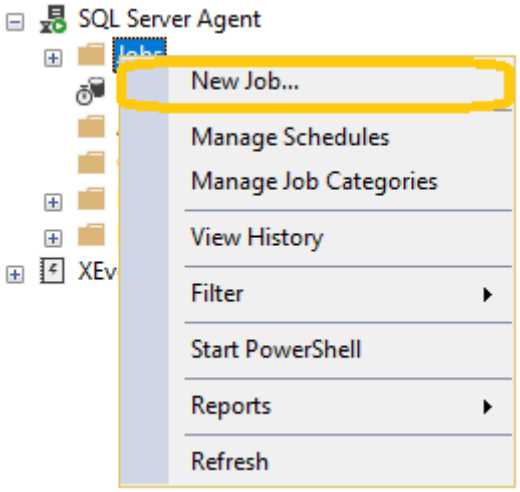
Please make sure that the *SQL Server Agent* service is running on our machine.

Open the services by typing *services.msc* run window. If highlighted service is not running then right-click and select the *Start* option to start the service.



Employee Full Package Schedule

Open the Microsoft SQL Server Management Studio with proper credentials and navigate to the *Jobs* node.



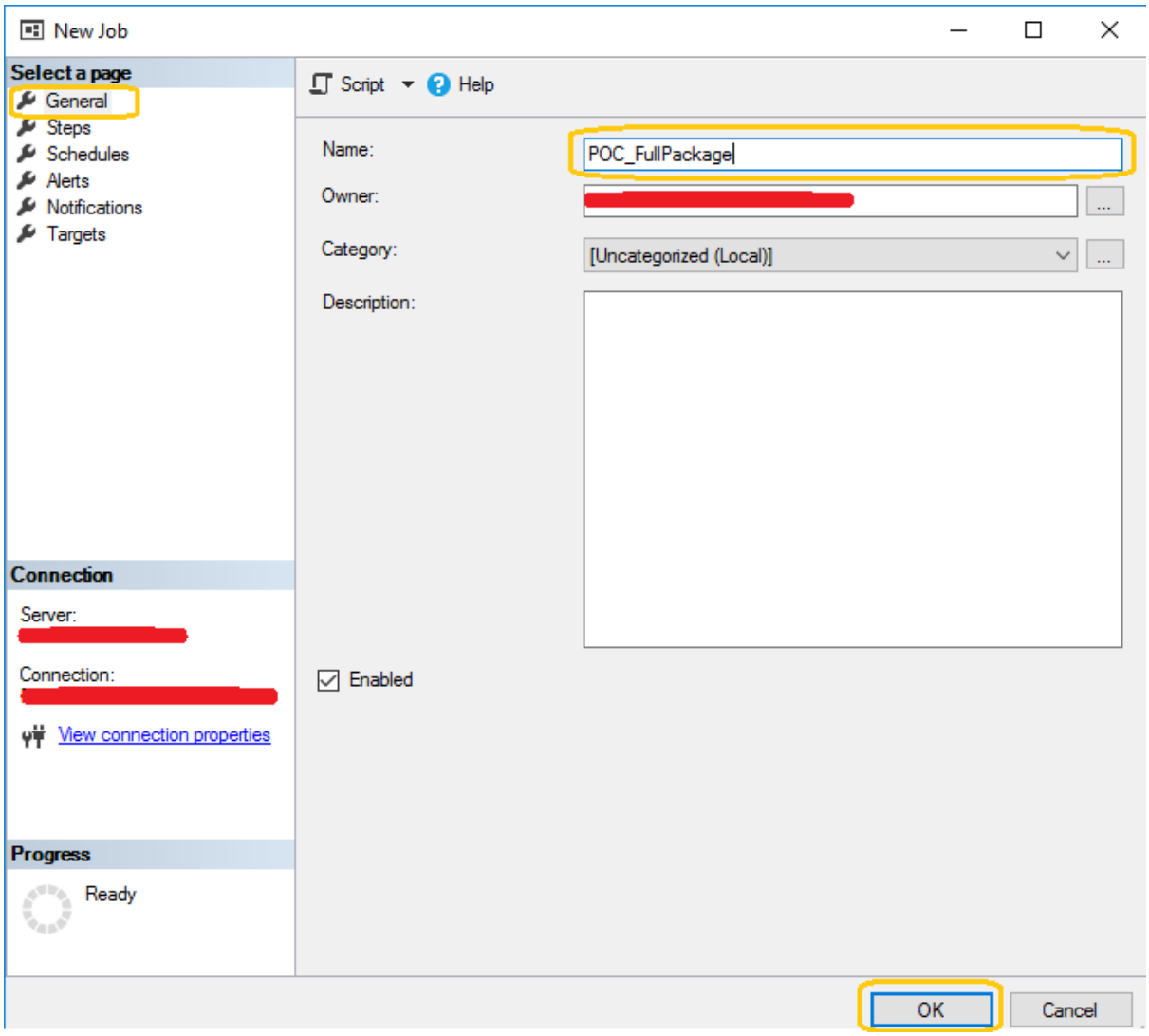
The new job wizard consists of six steps, i.e...

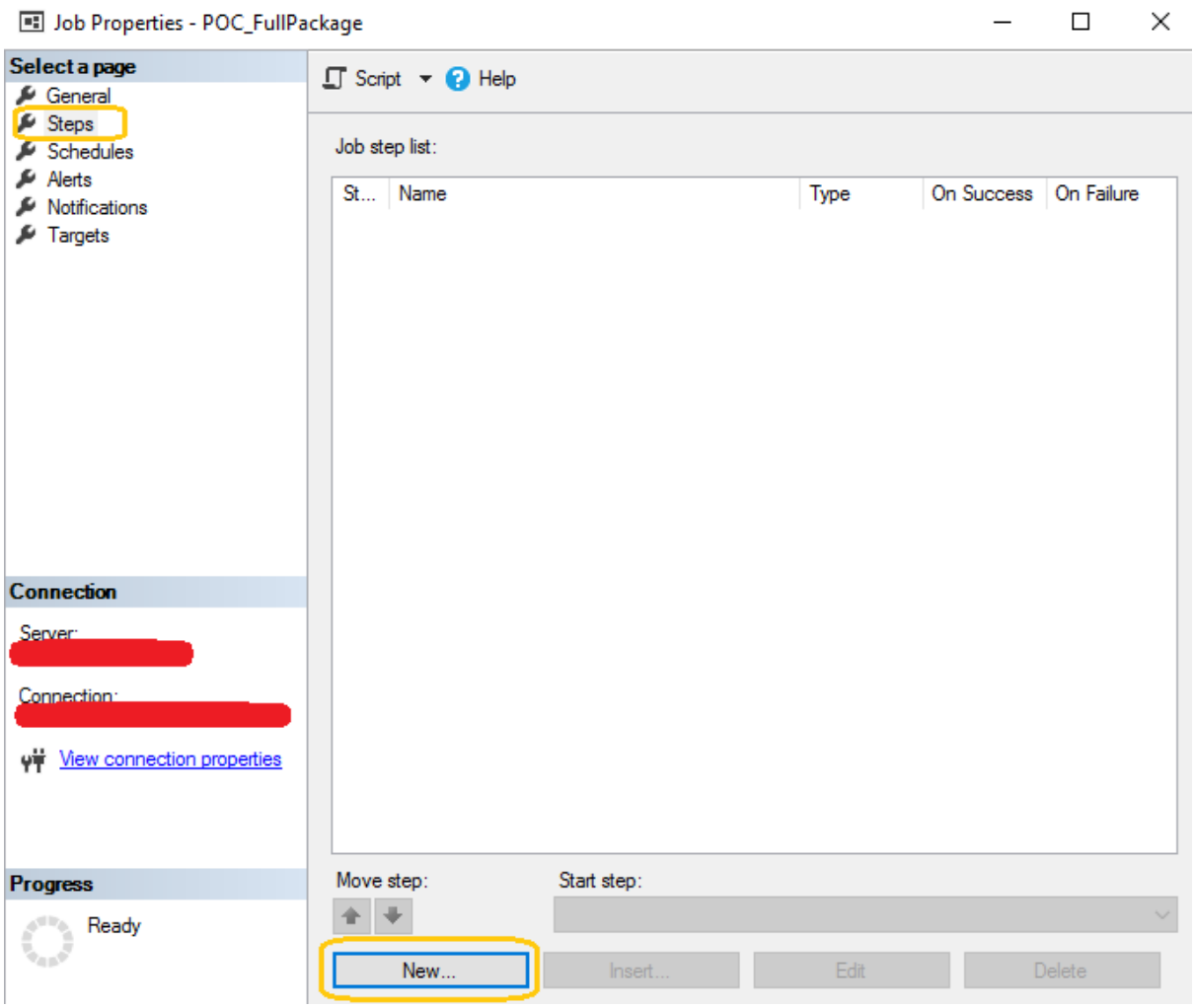
- General
- Steps
- Schedules
- Alerts
- Notifications
- Targets.

General

Provide the name of the job and choose the owner i.e. POC_FullPackage.

A description is optional.



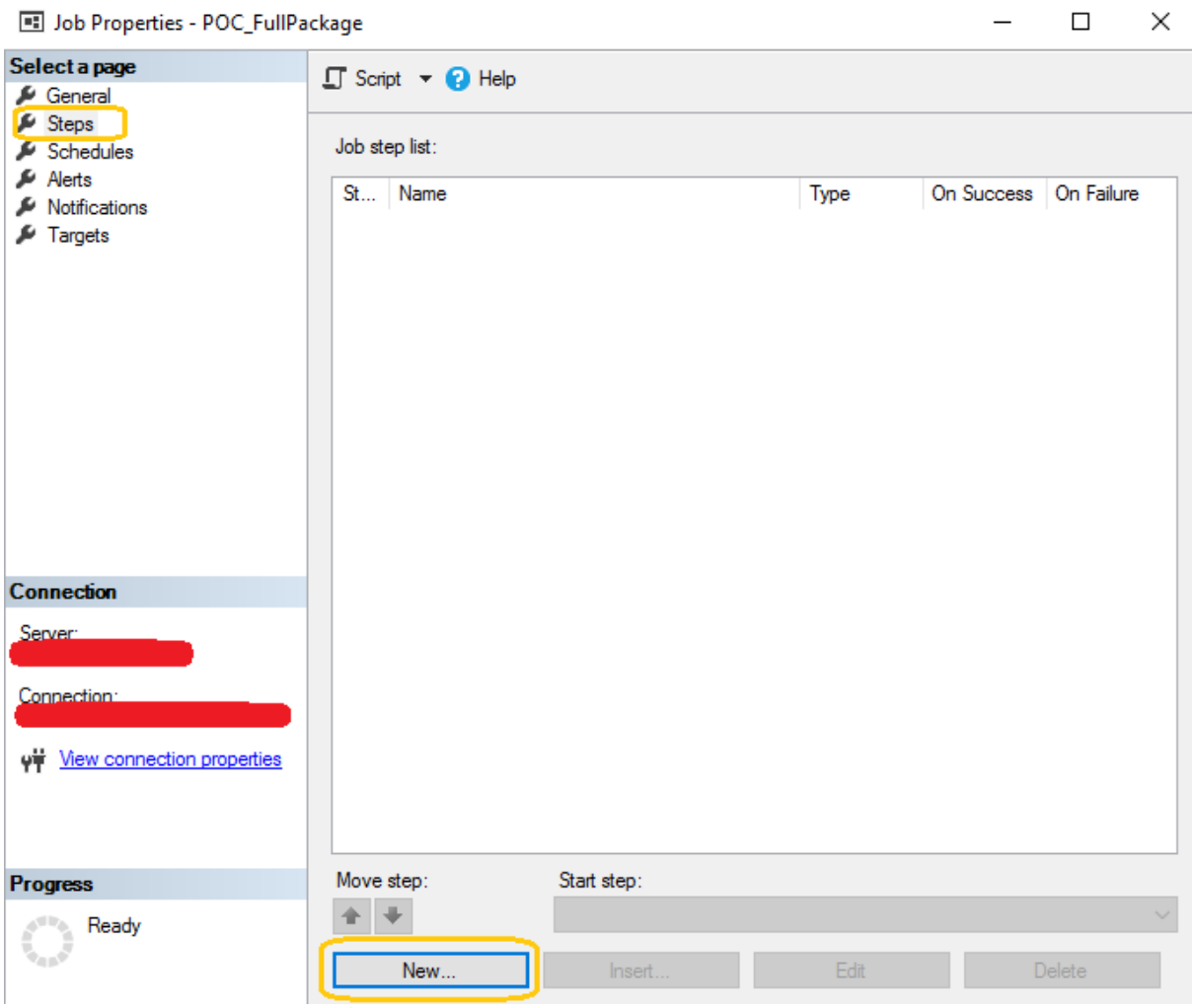


Steps

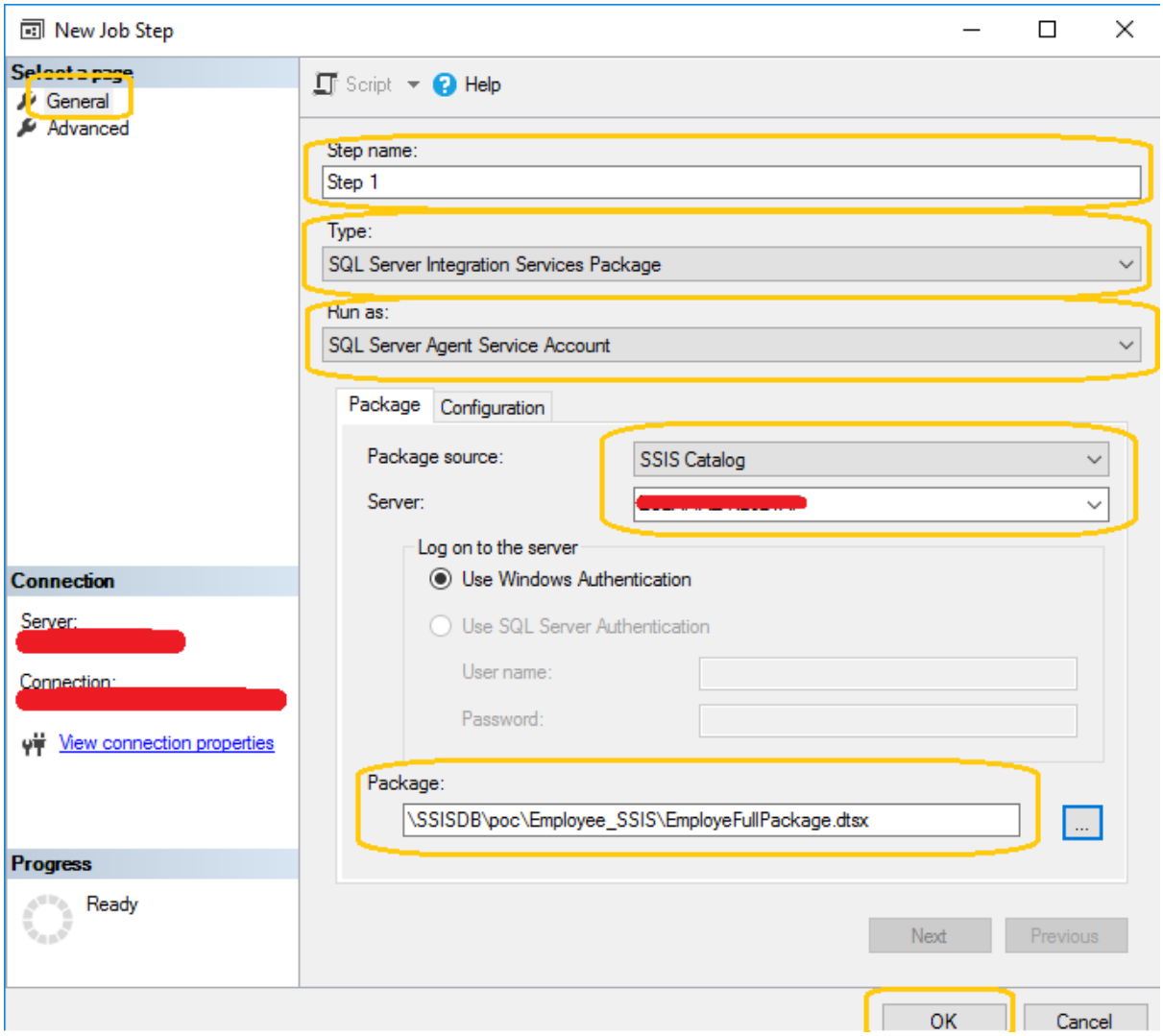
In the steps tab, create the new step by clicking on *New* button

Note

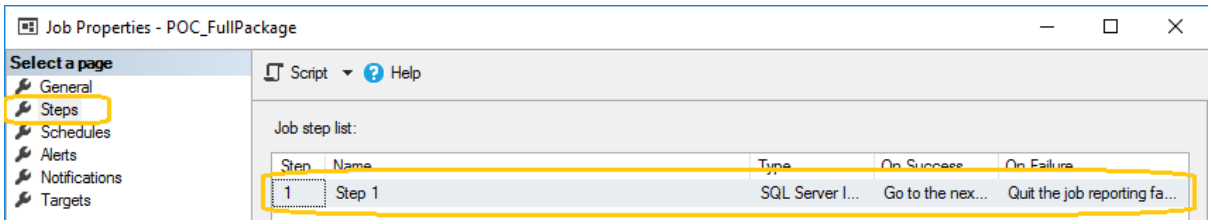
Here, we can add multiple steps and also arrange its execution sequence one-by-one.



In step configuration, enter the name of the step, select the type as *SQL Server Integration Services Package*, select the Server and select the package (i.e.EmployeeFullPackage.dtsx).



After creating the step configuration, it will look like the following:



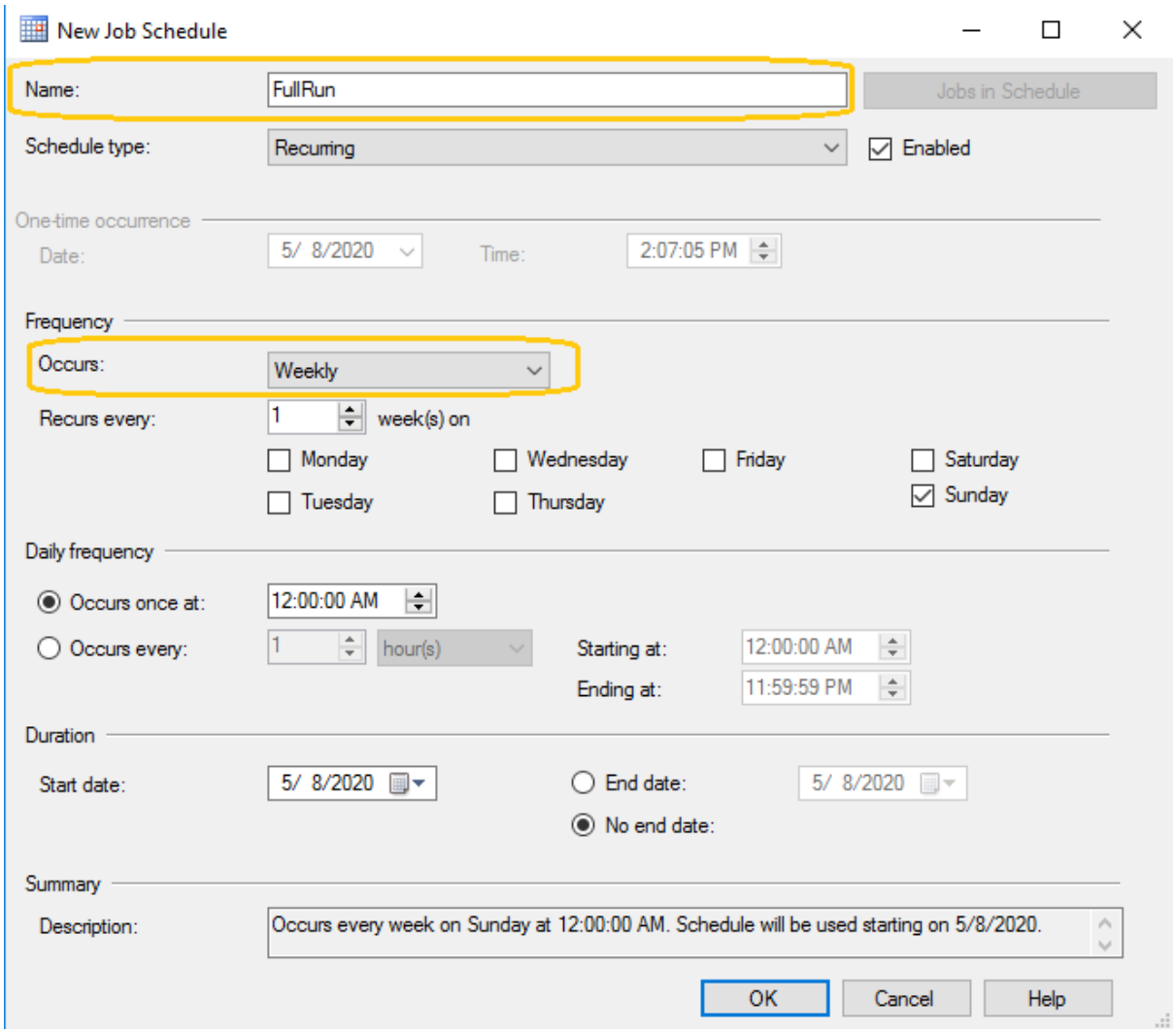
Schedules

In the Schedules tab, we can define one or more schedule to execute the package at a predefined time.

Click on the *New* button to create a new schedule.

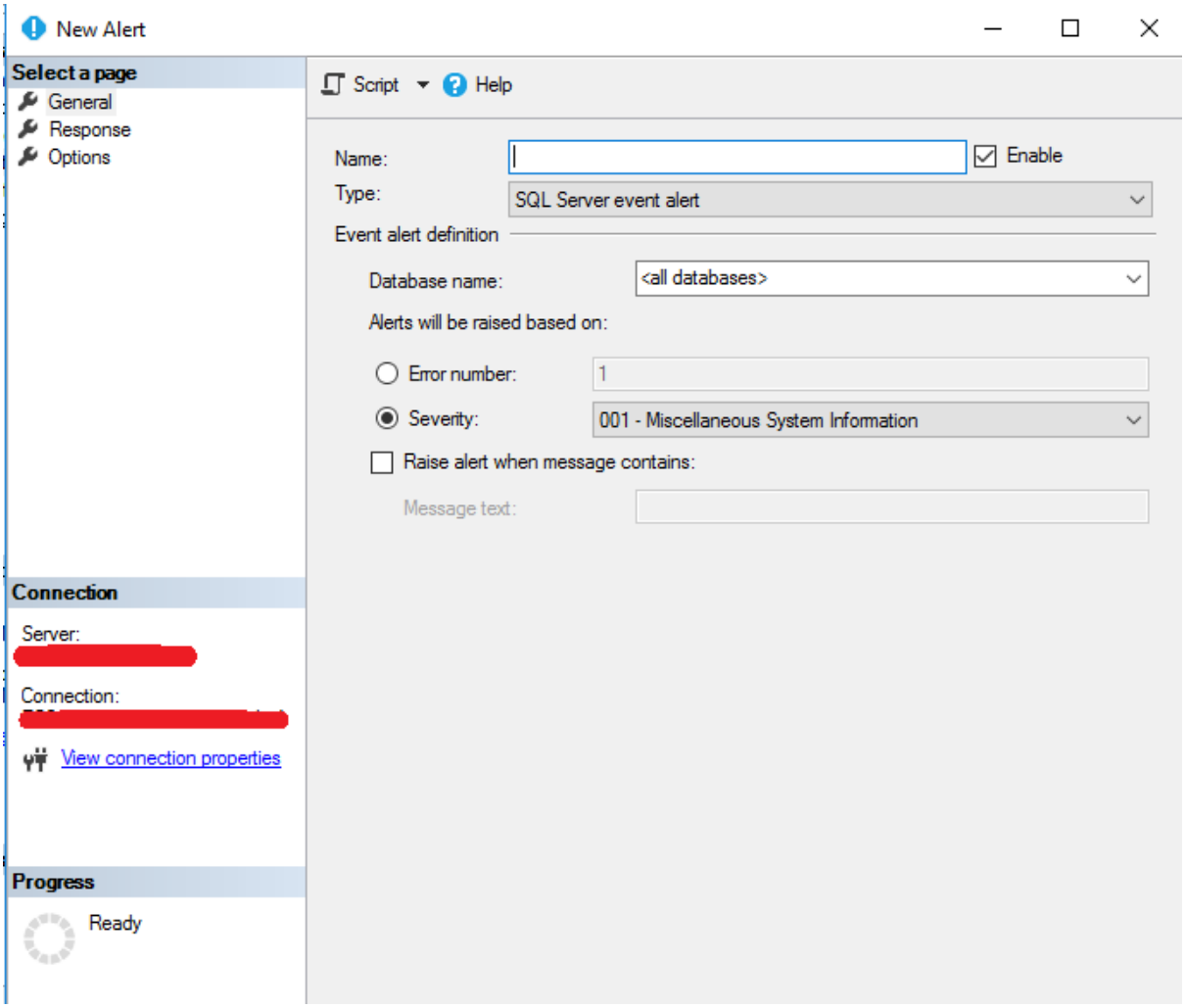
Note

Since I want my EmployeeFullPackage to run only on a weekly basis, I have selected the frequency to occurs on a weekly basis. You can choose different properties depending upon the requirement.



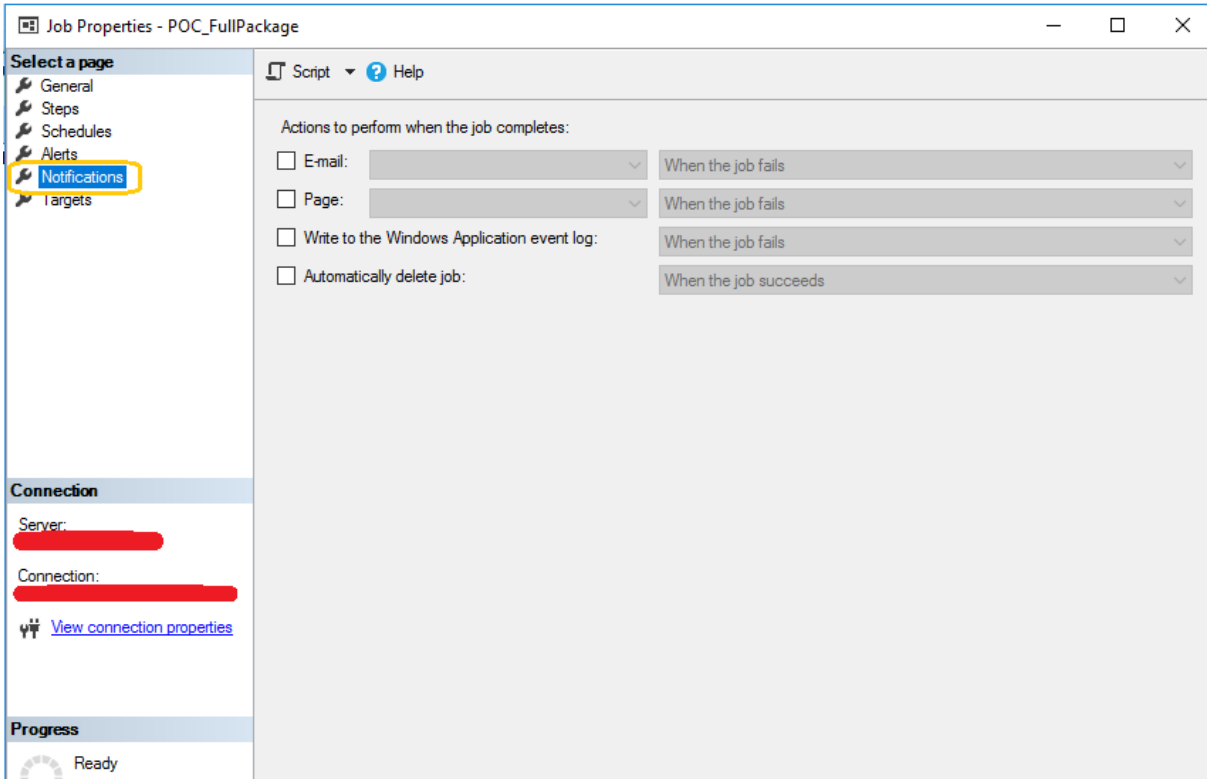
Alerts

Configure the alerts tab to display a message to the user(I have not set any alert messages).



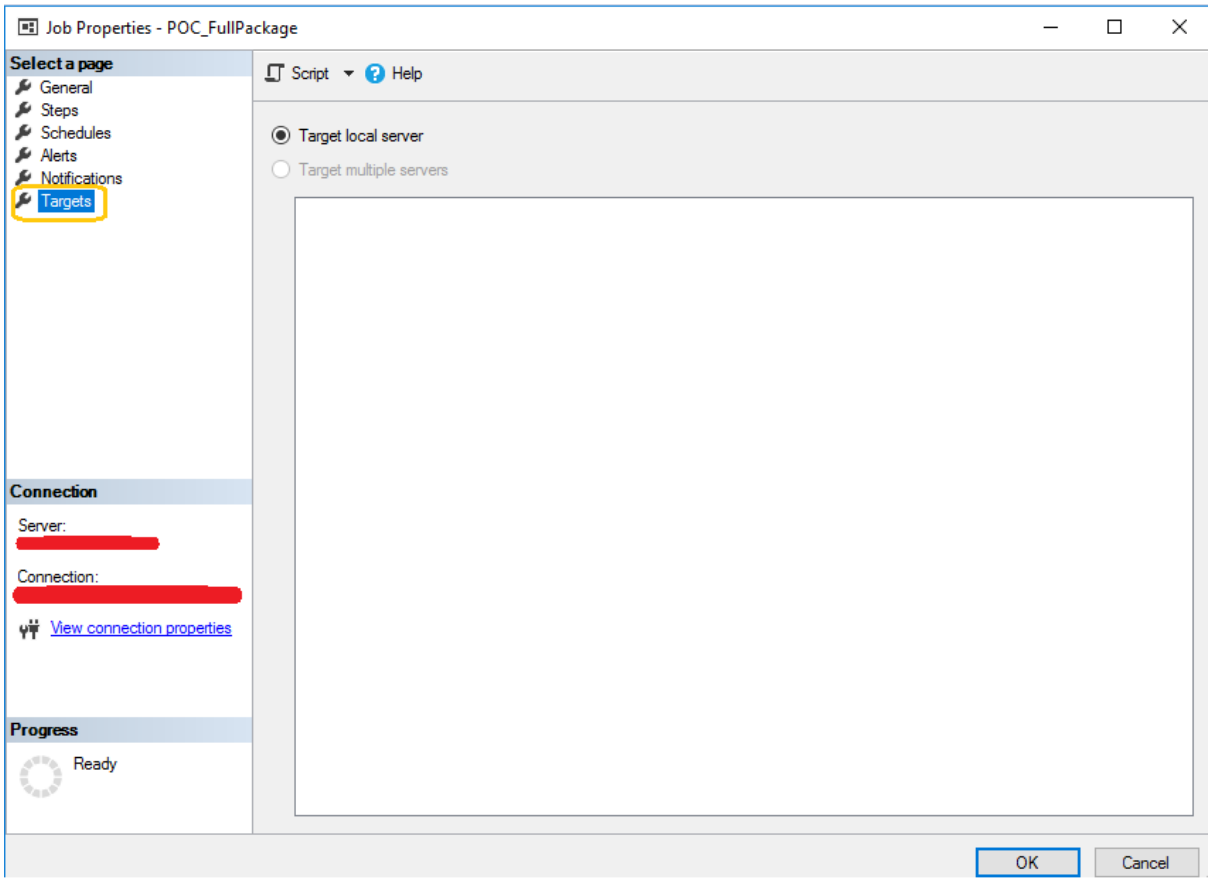
Notifications

We can configure this tab when we want to get the notification by email or create a log file for every job that fails or automatically delete the job when the job succeeds. (I have not set any notifications)



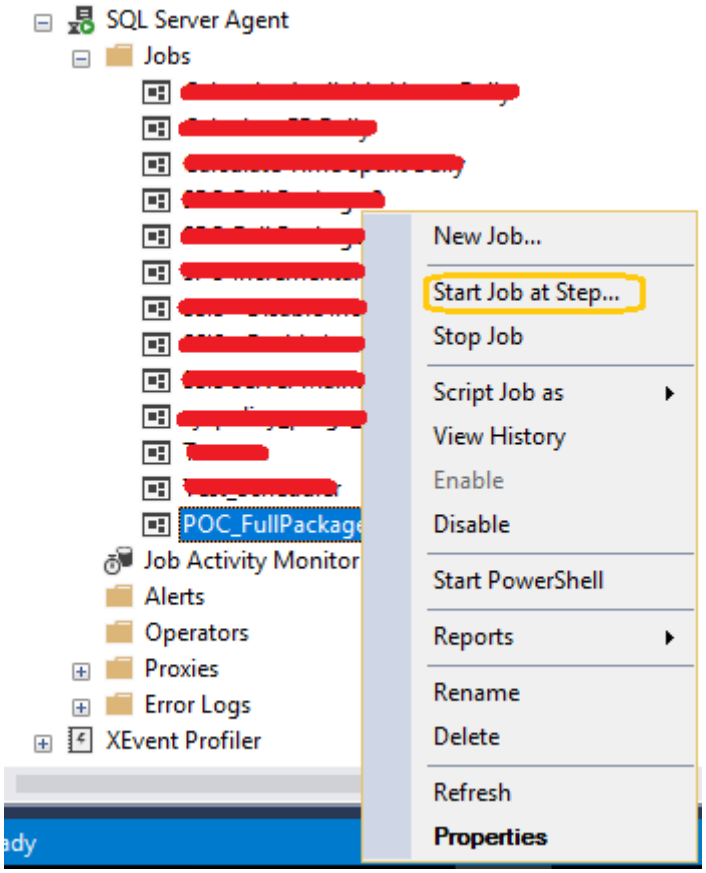
Targets

In this tab, select the server in which you want to execute the job (I have selected any target)

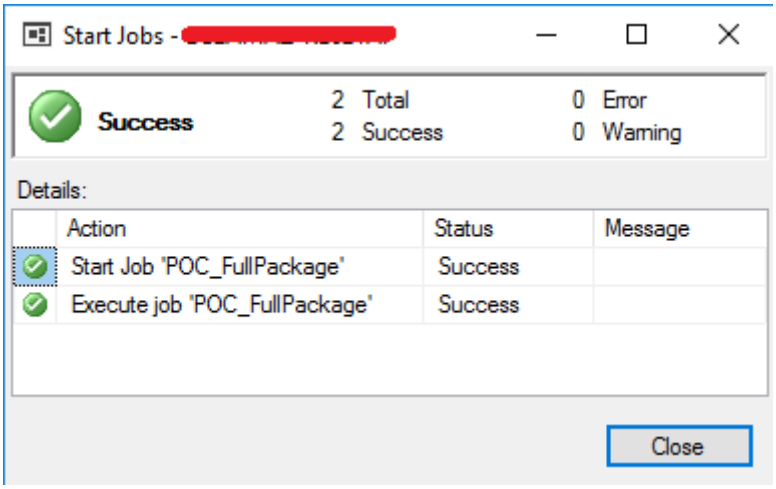


Now the job is ready in the SQL server agent.

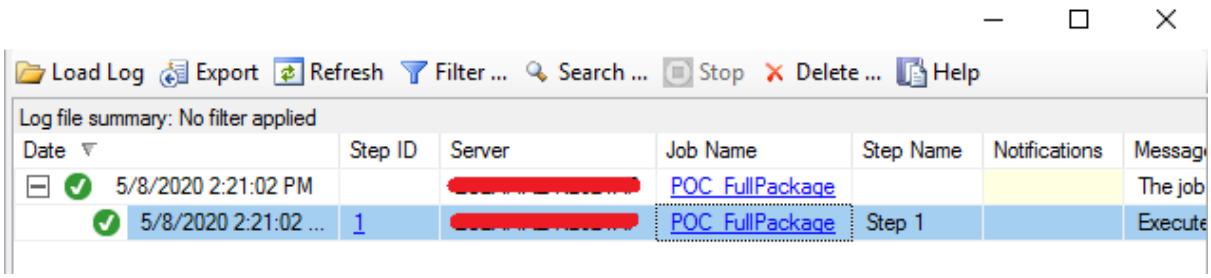
To start the job, right-click on the job (POC_FullPackage) and select *Start Job at Step.*



It will start the job and the screen will look like the following:



We can view the history of our jobs, to do so, right-click on POC_FullPackage job select *View History.*



Employee Incremental Package Schedule

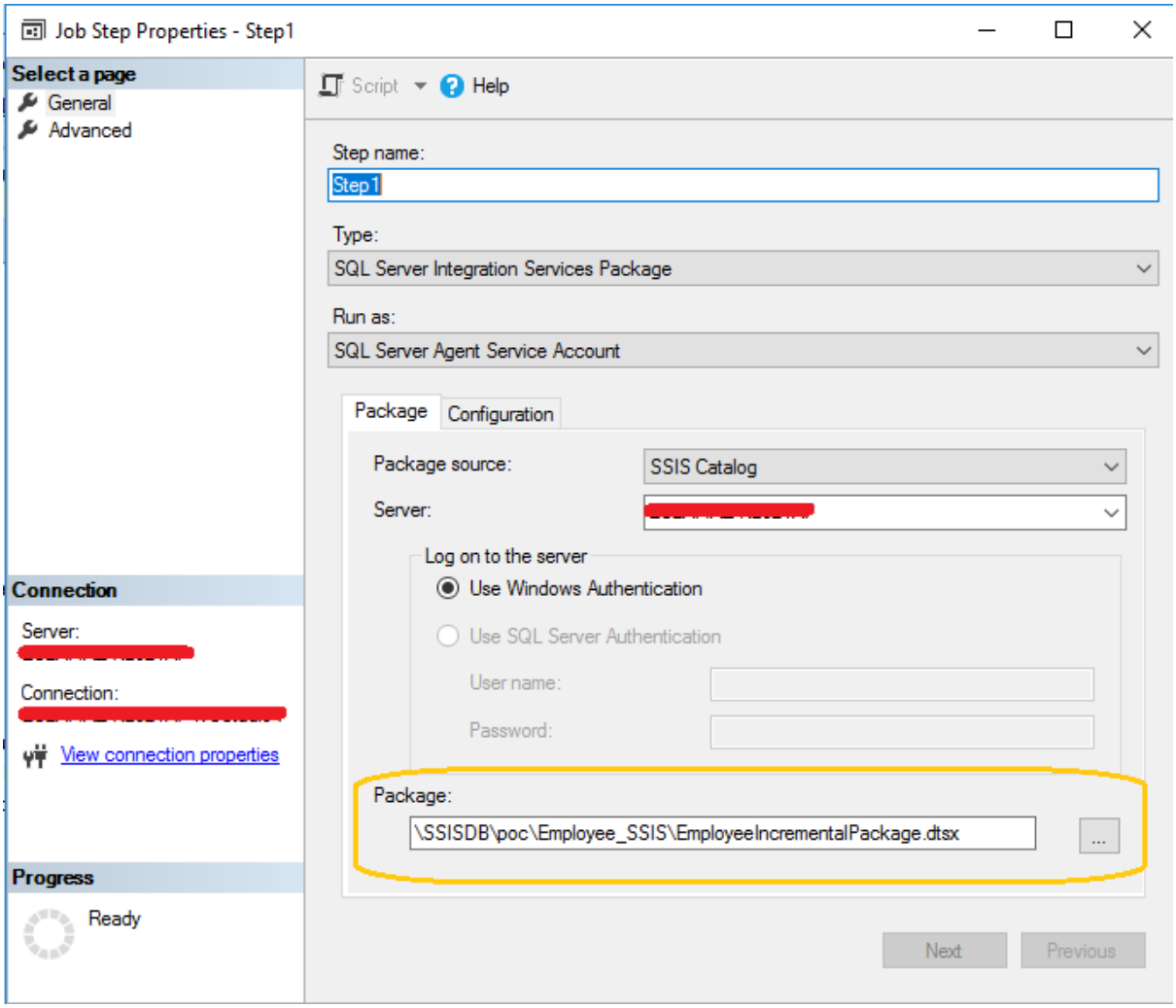
General

Name the job POC_Incremental.

Steps

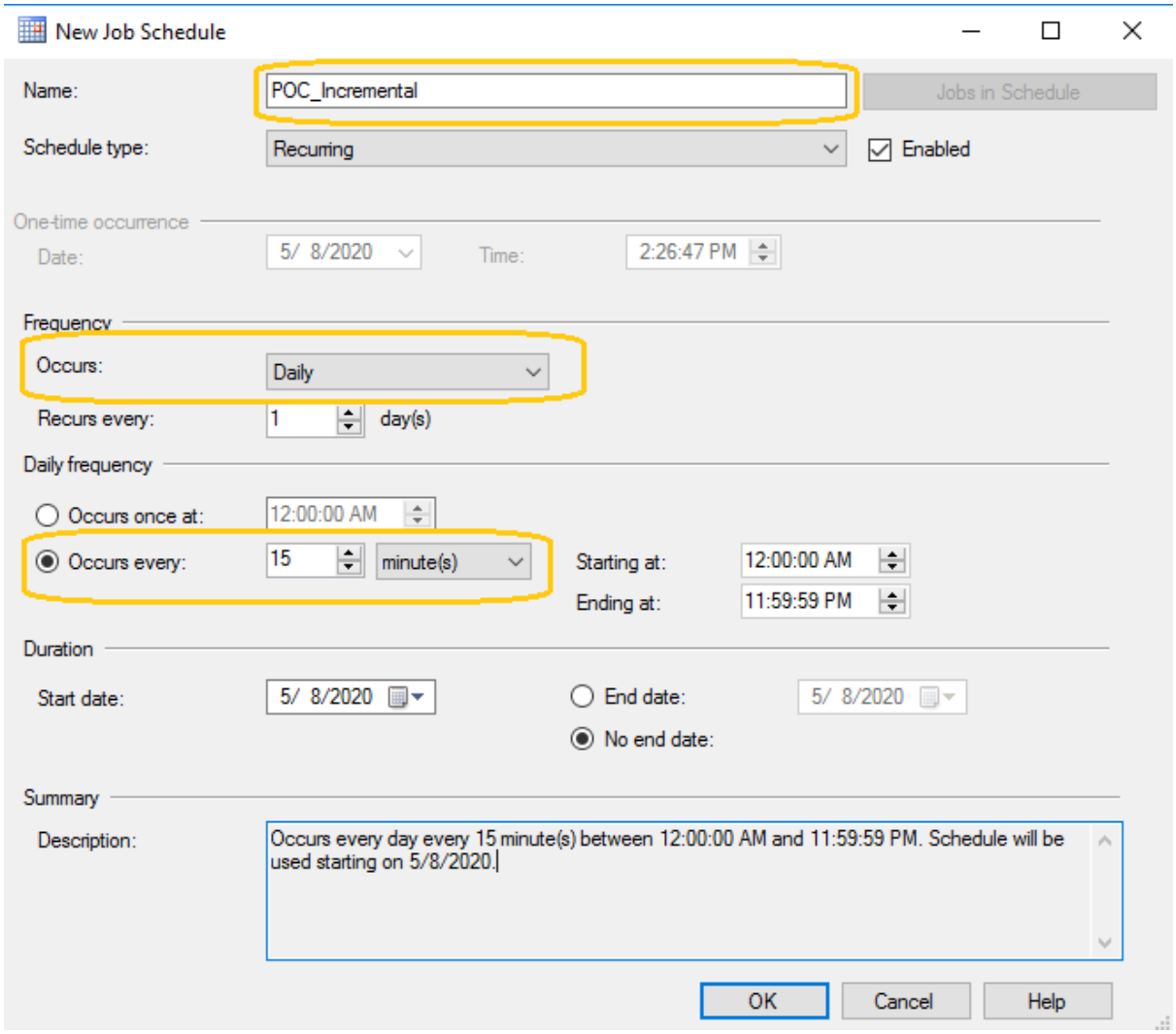
Configure the steps similar to the *Employee Full Package Schedule* except for the package field.

In the package field, select the incremental package path instead of the full package. (i.e EmployeeIncrementalPackage.dtsx)



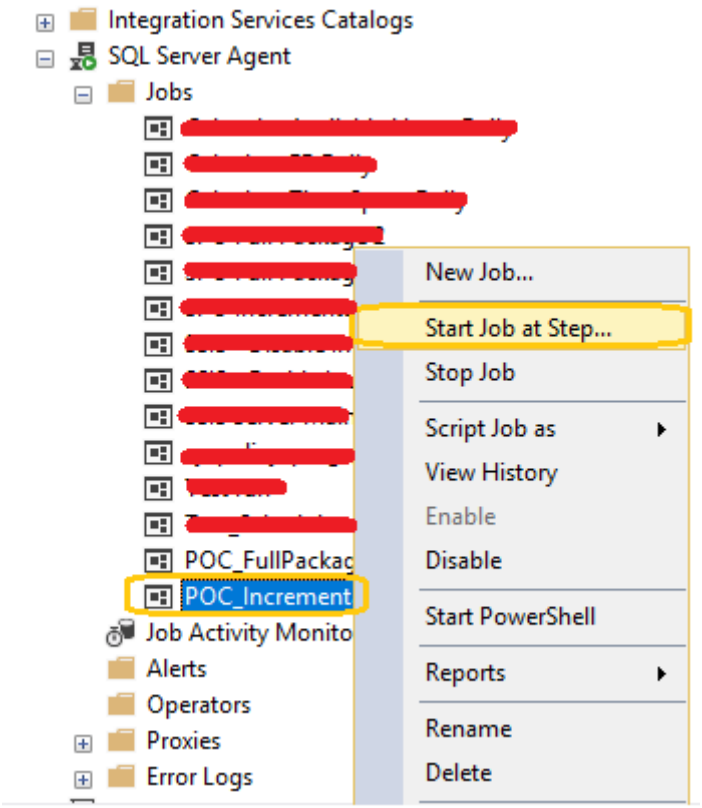
schedules

Since my requirement is to run the incremental package daily after every 15 min, I have configured the schedules tab like this:

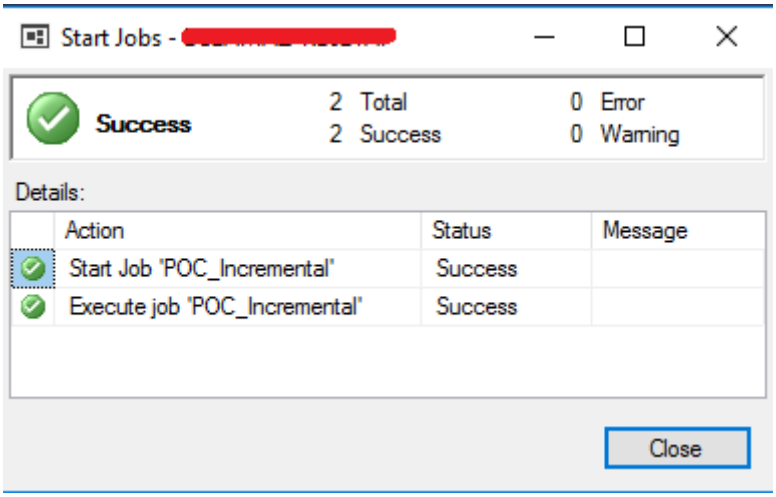


I have skipped the next three-tab i.e. Alerts, Notifications and Targets.

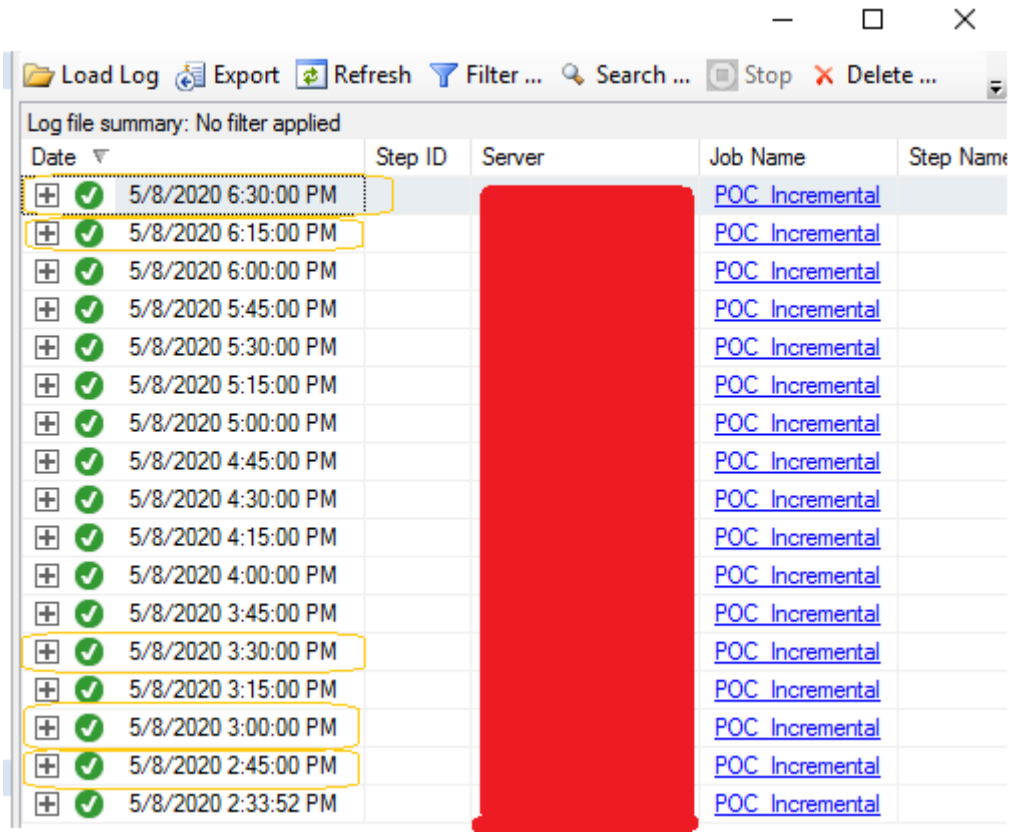
To start an incremental job, right-click on the POC_Incremental job and select *Start Job at Step.*



It will start the job, and the screen will look like the following:



Now we will open the history of our POC_Incremental job *(Right-click on POC_Incremental and select View History)* to verify that it is running after every 15 mins.



Conclusion

Now we can load records from SharePoint List to SQL Server using SSIS service, deploy packages to SSIS server and schedule the jobs in SQL server agent.

I hope you have enjoyed this series of articles