

**Extending SSIS Capabilities by Creating Custom SSIS Components for SharePoint**

SQL Server Technical Article

**Writer:** Arshad Ali

**Technical Reviewer:** Yaswant Vishwakarma

**Content Publisher**: Sreedhar Pelluru

**Published:** January, 2013

**Applies to:** SQL Server 2008, SQL Server 2008 R2, SQL Server 2012

**Summary:** SQL Server Integration Services (SSIS) - a component of SQL Server - is an extensible platform for building high performance data integration (ETL - Extraction, Transformation and Loading) and workflow solutions. The extensibility part of SSIS comes from the fact that, the developers have ability to create custom components if the in-built components of it do not suffice the specific need. The platform makes it very simple to embed your own code into control flow or data flow pipeline.

This article demonstrates how to create custom SSIS components (connection manager, source component and destination component) that integrate data to/from SharePoint lists and work like any other native components of SSIS utilizing the SSIS pipeline for better performance.

Copyright

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2013 Microsoft. All rights reserved.

Contents

[Introduction 4](#_Toc346792602)

[Getting Started 7](#_Toc346792603)

[Creating Custom Connection Manager 8](#_Toc346792604)

[Creating Custom Source Component 18](#_Toc346792605)

[Design Time Functionalities 18](#_Toc346792606)

[Run Time Functionalities 26](#_Toc346792607)

[Creating Custom Destination Component 30](#_Toc346792608)

[Design Time Functionalities 30](#_Toc346792609)

[Run Time Functionalities 38](#_Toc346792610)

[Using and Debugging Custom Components 44](#_Toc346792611)

[Software Requirements 44](#_Toc346792612)

[SQL Server 2008 or SQL Server 2008 R2 44](#_Toc346792613)

[SQL Server 2012 44](#_Toc346792614)

[Using or Testing Components 44](#_Toc346792615)

[Conclusion 57](#_Toc346792616)

# Introduction

SQL Server Integration Services (SSIS) - a component of SQL Server - is an extensible platform for building high performance data integration (ETL - Extraction, Transformation and Loading) and workflow solutions.

The extensibility part of SSIS comes from the fact that, the developers have ability to create custom components if the in-built components of it do not suffice the specific need. The platform makes it very simple to embed your own code into control flow or data flow pipeline. These are different types of components that can be extended:

* **Control Flow Component** – The run-time engine of SSIS implements the control/work flow and package management infrastructure that lets developers control the flow of execution and set options for logging, event handlers, and variables. SSIS includes several in-built control flow components that are sufficient to implement most common data integration scenarios. You might need to create a custom control flow task when these inbuilt components do not suit your requirements. More information on creating custom control flow task can be found [here.](http://msdn.microsoft.com/en-us/library/ms135965.aspx)
* **Log Provider Component** – Log Provider is used as a target for creating and storing captured data related to events that occur during package execution. SSIS includes several inbuilt log providers that perform extensive logging in different formats like text, XML, database, Windows event log and SQL trace. In a scenario where these inbuilt log providers don’t suit your requirement, you might need to create a custom log provider. More information on creating custom log provider can be found [here.](http://msdn.microsoft.com/en-us/library/ms403346)
* **Connection Manager Component** – Connection Managers encapsulate the information needed to connect to an external data source. SSIS includes a variety of inbuilt connection managers that support connections to the most commonly used data sources, from enterprise databases to text files, SSAS cube to FTP sites and Excel worksheets etc. In a scenario where these inbuilt connection managers don’t suit your requirement, you might need to create a custom connection manager. More information on creating connection manager can be found [here.](http://msdn.microsoft.com/en-us/library/ms403359.aspx)
* **Data Flow Component** – The data flow engine of SSIS manages the data flow task, which is a specialized, high performance task dedicated to moving and transforming data from disparate sources to the destination. Unlike other control flow tasks, the data flow task contains additional objects called data flow components, which can be either sources, transformations, or destinations as discussed below:
  + Source components or source adapters pull data from specified source and feed into data flow pipeline of the engine.
  + Transformation components transform the data to the required format.
  + Destination components or destination adapters load or write data to the specified destination.

More information on creating data flow components can be found [here.](http://msdn.microsoft.com/en-us/library/ms136078.aspx)

With the growing market share of SharePoint and storage of critical business data in SharePoint, it becomes evident to integrate data from SharePoint with a data warehouse (DW) for decision making and business analytics. But, SSIS has no inbuilt support or task for integrating data from SharePoint, which makes it really difficult to implement data integration to/from SharePoint. The delay in implementing the solution will impact key business decisions.

This article demonstrates how to create custom SSIS components (connection manager, source component and destination component as shown in the images below), which integrate data to/from SharePoint and work like any other native components of SSIS utilizing the SSIS pipeline for better performance.

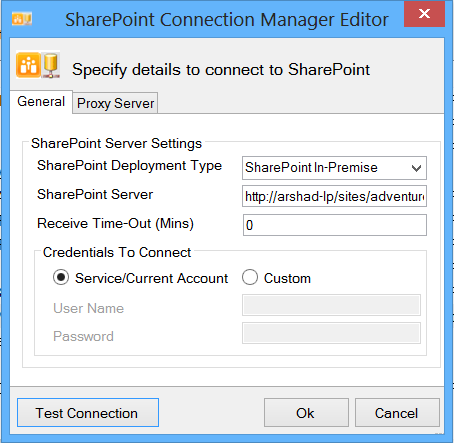


Figure 1 - SharePoint Connection Managers

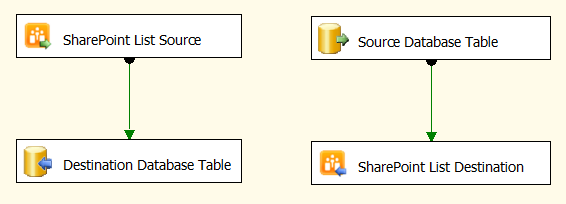


Figure 2 - SharePoint Adapters

Not only this, the step by step guide, idea and process, presented in this article, can be leveraged for creating similar types of SSIS components to connect to any other external sources (like Microsoft Dynamic CRM etc.) for which no inbuilt components are available for data integration.

# Getting Started

Create a console application project and add references of the following assemblies:

|  |  |
| --- | --- |
| Figure 3 - Assembly References |  |

Depending on the objects used by your component, your project will require references to some or all of the following assemblies. In addition, you will need to import required namespaces in our code:

|  |  |  |
| --- | --- | --- |
| Features | Assemblies to reference | Description |
| Data flow | Microsoft.SqlServer.PipelineHost | Contains managed classes that are used to develop managed data flow components |
| Data flow wrapper | Microsoft.SqlServer.DTSPipelineWrap | Contains the classes and interfaces used to create custom Data Flow components and to automate the Data Flow task |
| Runtime | Microsoft.SQLServer.ManagedDTS | Contains the classes and interfaces to create packages, custom tasks, and other package control flow elements |
| Runtime wrapper | Microsoft.SqlServer.DTSRuntimeWrap | Contains the classes and interfaces used to create Control Flow components in the runtime |
| User Interface | Microsoft.SqlServer.Dts.Design | Contains classes and interfaces to create user interface elements and connect them to the tasks so properties of the tasks can be set |

Table 1 - Assemblies needed for development

# Creating Custom Connection Manager

First of all, you need to create a custom connection manager, which encapsulates all the necessary information needed to connect to a SharePoint site. The user interface of the connection manager should look like this:

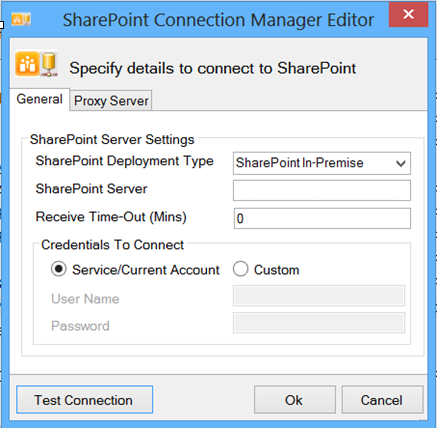


Figure 4 - SharePoint Connection Manager

Clicking on **Test Connection** button should validate or test the connection to the specified SharePoint site with the specified credentials or parameters and should show appropriate message to the users:

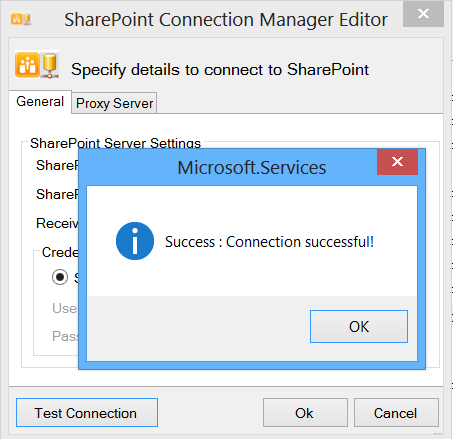


Figure 5 - SharePoint Connection Validation

Let's start with coding for the above connection manager. First of all you need to create a class by inheriting it from [ConnectionManagerBase](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.connectionmanagerbase.aspx) (abstract base class for connection manager) and [IDTSComponentPersist](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.idtscomponentpersist.aspx) (allows properties and settings for the components and tasks to be saved and loaded from the package XML). Then you need to apply the attribute that identifies the type of object to the class (in this case the attribute is [DtsConnectionAttribute](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.dtsconnectionattribute.aspx)) as shown below:

|  |
| --- |
| [DtsConnection(  ConnectionType = "SharePoint CM",  DisplayName = "SharePoint Connection",  Description = "Connection manager for SharePoint connections either based in In-Premise or Online",  UITypeName = "Microsoft.Sevices.GD.SharePointConnectionManager.SharePointConnectionManagerUI, SharePointConnectionManager, Version=1.0.0.0, Culture=neutral,PublicKeyToken=e6311be36a0230f0"  )]  public class SharePointConnectionManager : ConnectionManagerBase, IDTSComponentPersist  { |

In the above derived class, you need to override the methods and properties of the base class to implement our own custom functionality. But, let’s first add some string constants in the same class so that you can refer in multiple places:

|  |
| --- |
| private const string PERSIST\_XML\_ELEMENT = "SharePointConnectionManager";  private const string PERSIST\_XML\_SENSITIVE = "Sensitive";  private const string PERSIST\_XML\_INPREMISEDEPLOYMENT = "InPremiseDeployment";  private const string PERSIST\_XML\_SHAREPOINTSERVER = "SharePointServer";  private const string PERSIST\_XML\_TIMEOUT = "TimeOut";  private const string PERSIST\_XML\_SERVICEACCOUNTFORCONNECTION = "ServiceAccountForConnection";  private const string PERSIST\_XML\_USERNAME = "UserName";  private const string PERSIST\_XML\_PASSWORD = "Password";  private const string PERSIST\_XML\_PROXYSERVER = "ProxyServer";  private const string PERSIST\_XML\_PROXYPORT = "ProxyPort";  private const string PERSIST\_XML\_PROXYUSERNAME = "ProxyUserName";  private const string PERSIST\_XML\_PROXYPASSWORD = "ProxyPassword"; |

SSIS packages are persisted as XML documents that contain information about packages’ objects, properties, and settings. When developing a custom task, you must implement the [IDTSComponentPersist](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.idtscomponentpersist.aspx) interface to allow the custom task to save custom properties and settings in the package XML as well to load the properties and settings back again.

You need to do this only when the custom object has properties that use complex data types, or you want to perform custom processing on property values as they are loaded and saved. When you implement custom persistence, you must persist all the properties of the object, including both inherited properties and custom properties that you have added to the custom component. For the custom components that have properties which use only simple data types such as integer and string, the default implementation of persistence saves the metadata for the custom object along with the values of all its properties and hence you can skip this step.

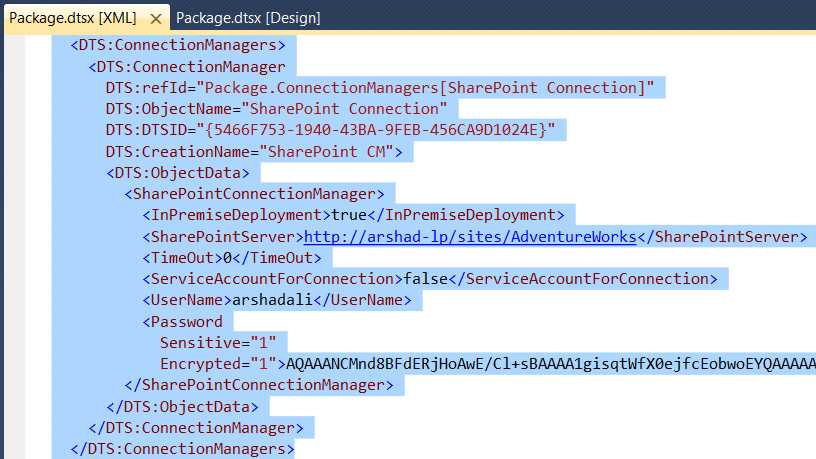
[IDTSComponentPersist.LoadFromXML](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.idtscomponentpersist.loadfromxml.aspx) method loads component information from saved XML. You need to override this method when you are creating custom objects and want saved information to be retrieved and assigned to component’s properties.

|  |
| --- |
| void IDTSComponentPersist.LoadFromXML(XmlElement rootNode, IDTSInfoEvents infoEvents)  {  // Create a root node for the data  if (rootNode.Name != PERSIST\_XML\_ELEMENT)  throw new ArgumentException("Unexpected element");  // Unpersist the elements  foreach (XmlNode childNode in rootNode.ChildNodes)  {  if (childNode.Name == PERSIST\_XML\_INPREMISEDEPLOYMENT)  if (childNode.InnerText == "true")  sharePointInPremiseDeployment = true;  else  sharePointInPremiseDeployment = false;  if (childNode.Name == PERSIST\_XML\_SHAREPOINTSERVER)  sharePointServerName = childNode.InnerText;  if (childNode.Name == PERSIST\_XML\_TIMEOUT)  sharePointTimeOut = Convert.ToInt32(childNode.InnerText);  if (childNode.Name == PERSIST\_XML\_SERVICEACCOUNTFORCONNECTION)  if (childNode.InnerText == "true")  serviceAccountForConnection = true;  else  serviceAccountForConnection = false;  if (childNode.Name == PERSIST\_XML\_USERNAME)  userName = childNode.InnerText;  if (childNode.Name == PERSIST\_XML\_PASSWORD)  password = childNode.InnerText; // The SSIS runtime will already have decrypted it for us and you don't need to do it again  //Proxy details  if (childNode.Name == PERSIST\_XML\_PROXYSERVER)  proxyServer = childNode.InnerText;  if (childNode.Name == PERSIST\_XML\_PROXYPORT)  proxyPort = childNode.InnerText;  if (childNode.Name == PERSIST\_XML\_PROXYUSERNAME)  proxyUserName = childNode.InnerText;  if (childNode.Name == PERSIST\_XML\_PROXYPASSWORD)  proxyPassword = childNode.InnerText; // The SSIS runtime will already have decrypted it for us and you don't need to do it again  }  } |

[IDTSComponentPersist.SaveToXML](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.idtscomponentpersist.savetoxml.aspx) method persists a component and its properties to the package XML file. Tasks and containers implement this method. You need to override this method when you are creating custom objects and want to save information to the package XML.

|  |
| --- |
| void IDTSComponentPersist.SaveToXML(XmlDocument doc, IDTSInfoEvents infoEvents)  {  // Create a root node for the data  XmlElement rootElement = doc.CreateElement(String.Empty, PERSIST\_XML\_ELEMENT, String.Empty);  doc.AppendChild(rootElement);  // Persist the SharePoint Deployment Type  if (sharePointInPremiseDeployment == true)  {  XmlNode node = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_INPREMISEDEPLOYMENT, String.Empty);  XmlElement DeploymentType = node as XmlElement;  DeploymentType.InnerText = "true";  rootElement.AppendChild(node);  }  else  {  XmlNode node = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_INPREMISEDEPLOYMENT, String.Empty);  XmlElement DeploymentTypeElement = node as XmlElement;  DeploymentTypeElement.InnerText = "false";  rootElement.AppendChild(node);  }  // Persist the SharePoint Server name and Time Out  if (!String.IsNullOrEmpty(sharePointServerName))  {  XmlNode node2 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_SHAREPOINTSERVER, String.Empty);  XmlElement ServerNameElement = node2 as XmlElement;  ServerNameElement.InnerText = sharePointServerName;  rootElement.AppendChild(node2);  XmlNode node3 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_TIMEOUT, String.Empty);  XmlElement TimeOut = node3 as XmlElement;  TimeOut.InnerText = sharePointTimeOut.ToString();  rootElement.AppendChild(node3);  }  // Persist the serviceAccountForConnection  if (serviceAccountForConnection == true)  {  XmlNode node = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_SERVICEACCOUNTFORCONNECTION, String.Empty);  XmlElement DeploymentType = node as XmlElement;  DeploymentType.InnerText = "true";  rootElement.AppendChild(node);  }  else  {  XmlNode node = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_SERVICEACCOUNTFORCONNECTION, String.Empty);  XmlElement DeploymentTypeElement = node as XmlElement;  DeploymentTypeElement.InnerText = "false";  rootElement.AppendChild(node);  // Persist the SharePoint User name  if (!String.IsNullOrEmpty(userName))  {  XmlNode node2 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_USERNAME, String.Empty);  XmlElement userNameElement = node2 as XmlElement;  userNameElement.InnerText = userName;  rootElement.AppendChild(node2);  }  // Persist the password separately  if (!String.IsNullOrEmpty(password))  {  XmlNode node3 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_PASSWORD, String.Empty);  XmlElement pswdElement = node3 as XmlElement;  rootElement.AppendChild(node3);  // Adding the sensitive attribute tells the SSIS runtime that this value should be protected according to the ProtectionLevel of the package  pswdElement.InnerText = password;  XmlAttribute pwAttr = doc.CreateAttribute(PERSIST\_XML\_SENSITIVE);  pwAttr.Value = "1";  pswdElement.Attributes.Append(pwAttr);  }  }  // Persist the Proxy Server Name and Port Number  if (!String.IsNullOrEmpty(proxyServer))  {  XmlNode node = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_PROXYSERVER, String.Empty);  XmlElement ServerNameElement = node as XmlElement;  ServerNameElement.InnerText = proxyServer;  rootElement.AppendChild(node);  XmlNode node3 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_PROXYPORT, String.Empty);  XmlElement TimeOut = node3 as XmlElement;  TimeOut.InnerText = proxyPort.ToString();  rootElement.AppendChild(node3);  }  // Persist the Proxy User name  if (!String.IsNullOrEmpty(proxyUserName))  {  XmlNode node2 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_PROXYUSERNAME, String.Empty);  XmlElement userName = node2 as XmlElement;  userName.InnerText = proxyUserName;  rootElement.AppendChild(node2);  }  // Persist the Proxy password separately  if (!String.IsNullOrEmpty(proxyPassword))  {  XmlNode node3 = doc.CreateNode(XmlNodeType.Element, PERSIST\_XML\_PROXYPASSWORD, String.Empty);  XmlElement pswdElement = node3 as XmlElement;  rootElement.AppendChild(node3);  // Adding the sensitive attribute tells the SSIS runtime that this value should be protected according to the ProtectionLevel of the package  pswdElement.InnerText = proxyPassword;  XmlAttribute pwAttr = doc.CreateAttribute(PERSIST\_XML\_SENSITIVE);  pwAttr.Value = "1";  pswdElement.Attributes.Append(pwAttr);  }    } |

Once component and its properties have been saved to XML file using the code above, you can verify it and you will see something like this in Visual Studio:



Now you need to add a form for user interface where users can provide information needed to connect to the SharePoint site:

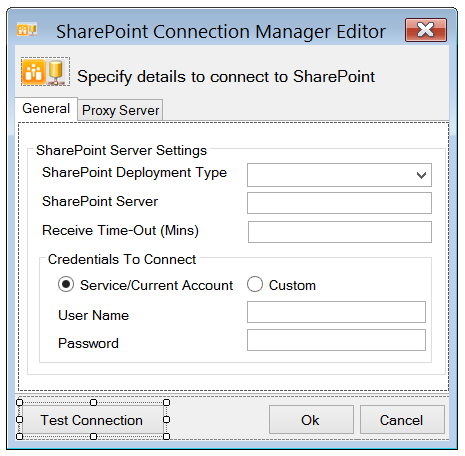


Figure 6 - SharePoint Connection Manager Form UI

Clicking on **Test Connection** button should test the connection to the specified SharePoint site with the additional information provided by the users.

|  |
| --- |
| private void btnTest\_Click(object sender, EventArgs e)  {  try  {  if (txtServer.Text.Trim().Length == 0)  {  MessageBox.Show("Invalid data : SharePoint server name should not be blank.", "Microsoft.Services", MessageBoxButtons.OK, MessageBoxIcon.Information);  txtServer.Focus();  return;  }  Uri \_sharepointUri = new Uri(txtServer.Text.Trim());  NetworkCredential \_credentials = null;  if (rdServiceAccount.Checked == true)  {  \_credentials = CredentialCache.DefaultNetworkCredentials;  }  else  {  if (txtUser.Text.Trim().Length > 0)  {  if (txtUser.Text.Trim().Contains(@"\"))  {  var userNameElements = txtUser.Text.Trim().Split('\\');  if (userNameElements.Length == 2)  {  \_credentials = new NetworkCredential(userNameElements[1], txtPassword.Text.Trim(), userNameElements[0]);  }  }  else  {  \_credentials = new NetworkCredential(txtUser.Text.Trim(), txtPassword.Text.Trim());  }  }  else  {  MessageBox.Show("Invalid data : When custom authentication mode is selected, you need to provide user name and password.", "Microsoft.Services", MessageBoxButtons.OK, MessageBoxIcon.Information);  txtUser.Focus();  return;  }  }  int timeOut = 0;  if (txtTimeOut.Text.Length == 0 || System.Convert.ToInt16(txtTimeOut.Text) == 0)  timeOut = 10;  else  timeOut = System.Convert.ToInt16(txtTimeOut.Text);  string statusMessage = SharePointListServiceCommons.SharePointConnect(\_sharepointUri, \_credentials, timeOut);  if (!string.IsNullOrEmpty(statusMessage) && statusMessage.Contains("Failure"))  MessageBox.Show(statusMessage, "Microsoft.Services", MessageBoxButtons.OK, MessageBoxIcon.Error);  else  MessageBox.Show(statusMessage, "Microsoft.Services", MessageBoxButtons.OK, MessageBoxIcon.Information);  }  catch (Exception ex)  {  MessageBox.Show(ex.Message + " " + ex.StackTrace, "Microsoft.Services", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  } |

Now that you have user interface for editing connection manager, you will need to derive the class from [IDtsConnectionManagerUI](http://technet.microsoft.com/en-us/library/microsoft.sqlserver.dts.runtime.design.idtsconnectionmanagerui.aspx) Interface which defines the interface for editing connection managers. It returns the connection manager user interface that is used for editing the connection manager. There are four methods to be defined from this interface:

* **Initialize** – This method is called when SSIS creates a connection manager. It initializes the connection manager user interface.
* **New** – This method is used for creating connections of a specific type that you need and called after a new connection manager is added to the package. It provides notification to tasks of newly created connection managers.
* **Edit** – This method is used for editing an existing connection manager.
* **Delete** – This method is called before an existing connection manager is removed from the package.

|  |
| --- |
| **void IDtsConnectionManagerUI.Delete**(IWin32Window parentWindow)  {  //do nothing  }  **bool IDtsConnectionManagerUI.Edit**(IWin32Window parentWindow, Connections, ConnectionManagerUIArgs connectionUIArg)  {  return EditSharePointCredentialConnection(parentWindow, connections);  }  **void IDtsConnectionManagerUI.Initialize**(ConnectionManager, IServiceProvider serviceProvider)  {  this.\_serviceProvider = serviceProvider;  this.\_connectionManager = connectionManager;  }  **bool IDtsConnectionManagerUI.New**(IWin32Window parentWindow, Connections, ConnectionManagerUIArgs connectionUIArg)  {  IDtsClipboardService clipboardService;  clipboardService = (IDtsClipboardService)\_serviceProvider.GetService(typeof(IDtsClipboardService));  if (clipboardService != null) // no action needed if connection manager has been copied and pasted  {  if (clipboardService.IsPasteActive)  return true;  }  return EditSharePointCredentialConnection(parentWindow, connections);  }  private bool EditSharePointCredentialConnection(IWin32Window parentWindow, Connections connections)  {  using (SharePointConnectionManagerForm form = new SharePointConnectionManagerForm())  {  form.Initialize(this.\_serviceProvider, this.ConnectionManager, this.ErrorCollectionService);  if (form.ShowDialog(parentWindow) == DialogResult.OK)  {  SharePointConnectionManager innerManager = (SharePointConnectionManager)ConnectionManager.InnerObject;  innerManager.SharePointInPremiseDeployment = form.SharePointInPremiseDeployment;  innerManager.SharePointServerName = form.SharePointServer;  innerManager.SharePointTimeOut = form.SharePointTimeOut;  innerManager.ServiceAccountForConnection = form.ServiceAccountForConnection;  innerManager.UserName = form.UserName;  innerManager.Password = form.Password;  innerManager.ProxyServer = form.ProxyServer;  innerManager.ProxyPort = form.ProxyPort;  innerManager.ProxyUserName = form.ProxyUserName;  innerManager.ProxyPassword = form.ProxyPassword;  return true;  }  }  return false;  } |

# Creating Custom Source Component

Custom source component can be created for connecting to data sources that cannot be accessed by using one of the existing SSIS source components. Source components have one or more outputs and zero inputs. At design time, source components are used to create and configure connections, read column metadata from the external data source, and configure the source's output columns based on the external data source. During execution they connect to the external data source and add rows to an output buffer. The data flow task then provides this buffer of data rows to downstream components.

To create a custom source component, you need to create a class by inheriting it from [PipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.aspx) (base class that is used when developing managed data flow components and which implements the required component interfaces, [IDTSDesigntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.aspx) and [IDTSRuntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsruntimecomponent100.aspx)). Then, you need to apply the attribute that controls how the SSIS Designer displays and interacts with the object (in this case the attribute is [DtsPipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.dtspipelinecomponentattribute.aspx)) as shown below:

|  |
| --- |
| [DtsPipelineComponent(DisplayName = "SharePoint List Source",  CurrentVersion = 1,  IconResource = "Microsoft.Sevices.GD.SharePointSSISComponents.Icons.SharePointSourceAdapter.ico",  Description = "SharePoint List Source Adapter to extract data from SharePoint (In Premise or Cloud) lists.",  ComponentType = ComponentType.SourceAdapter)]  public class SharePointListSourceAdapter : PipelineComponent  { |

## Design Time Functionalities

[IDTSDesigntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.aspx) interface defines the design-time interface for a component and contains methods that are called when the component properties and collections of the [ComponentMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsruntimecomponent100.componentmetadata.aspx) are modified during design time. You don’t need to explicitly implement this interface as the base class [PipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.aspx) already implements it. You still need to override the methods in this interface to provide your own implementation.

Implementing the design-time functionality of a source component involves specifying a connection to an external data source, adding and configuring output columns that reflect the data source, and validating that the component is ready to execute. By definition, a source component has zero inputs and one or more outputs.

Source components connect to external data sources by using **Connection Manager** objects defined in a package. They indicate their requirement for a connection manager by adding an element to the [RuntimeConnectionCollection](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtscomponentmetadata100.runtimeconnectioncollection.aspx) collection of the ComponentMetaData property as shown below.

|  |
| --- |
| //The ProvideComponentProperties() method provides initialization of the component when the component is first added to the data Flow designer  public override void ProvideComponentProperties()  {  // Reset the component.  ComponentMetaData.RuntimeConnectionCollection.RemoveAll();  ComponentMetaData.CustomPropertyCollection.RemoveAll();  RemoveAllInputsOutputsAndCustomProperties();  // Add custom properties  AddUserProperties();  IDTSRuntimeConnection100 conn = ComponentMetaData.RuntimeConnectionCollection.New();  conn.Name = "SharePointConnection";  }  private void AddUserProperties()  {  // Add Custom Properties  var sharepointListName = ComponentMetaData.CustomPropertyCollection.New();  sharepointListName.Name = C\_SHAREPOINTLISTNAME;  sharepointListName.Description = "Specify the name of the SharePoint list to load data from.";  sharepointListName.ExpressionType = DTSCustomPropertyExpressionType.CPET\_NOTIFY;  var sharepointListViewName = ComponentMetaData.CustomPropertyCollection.New();  sharepointListViewName.Name = C\_SHAREPOINTLISTVIEWNAME;  sharepointListViewName.Description = "Specify the name of the view within SharePoint list to load data from; if not specified default view will be used.";  sharepointListViewName.ExpressionType = DTSCustomPropertyExpressionType.CPET\_NOTIFY;  var sharepointCamlQuery = ComponentMetaData.CustomPropertyCollection.New();  sharepointCamlQuery.Name = C\_CAMLQUERY;  sharepointCamlQuery.Description = "Specify CAML Query to pull filetered rows from SharePoint list.";  sharepointCamlQuery.Value = "<Query />";  sharepointCamlQuery.ExpressionType = DTSCustomPropertyExpressionType.CPET\_NOTIFY;  var batchSize = ComponentMetaData.CustomPropertyCollection.New();  batchSize.Name = C\_BATCHSIZE;  batchSize.Value = (short)1000;  batchSize.Description = "Specify # of rows to pull from SharePoit list at a time. The more number of columns or wider the columns you should consider keeping it low.";  batchSize.TypeConverter = typeof(short).AssemblyQualifiedName;  var isRecursive = ComponentMetaData.CustomPropertyCollection.New();  isRecursive.Name = C\_ISRECURSIVE;  isRecursive.Value = Enums.TrueFalseValue.False;  isRecursive.Description = "Specify to include rows from the subfolders within the list.";  isRecursive.TypeConverter = typeof(Enums.TrueFalseValue).AssemblyQualifiedName;  var includeFolders = ComponentMetaData.CustomPropertyCollection.New();  includeFolders.Name = C\_INCLUDEFOLDERS;  includeFolders.Value = Enums.TrueFalseValue.False;  includeFolders.Description = "Specify to include folder with the list content.";  includeFolders.TypeConverter = typeof(Enums.TrueFalseValue).AssemblyQualifiedName;  ComponentMetaData.Name = "SharePoint List Source";  ComponentMetaData.Description = "SharePoint List Source Adapter to extract data from SharePoint (In Premise or Cloud) lists.";  ComponentMetaData.ContactInfo = "http://www.microsoft.com/microsoftservices/en/us/home.aspx";  } |

After a connection has been added to the [RuntimeConnectionCollection](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtscomponentmetadata100.runtimeconnectioncollection.aspx), you need to override the [AcquireConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.acquireconnections.aspx) method to establish a connection to the external data source. This method is called during both design and execution time. After the connection is established, it gets cached internally by the component in a local member variable and released when the [ReleaseConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.releaseconnections.aspx) method is called. The [ReleaseConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.releaseconnections.aspx) method is called at design and execution time, like the [AcquireConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.acquireconnections.aspx) method.

|  |
| --- |
| //Called at design time and runtime. Establishes a connection using a ConnectionManager embedded in the package.  public override void AcquireConnections(object transaction)  {  \_sharePointConnectionManager = null;  \_credentials = null;  if (ComponentMetaData.RuntimeConnectionCollection.Count > 0)  {  if (ComponentMetaData.RuntimeConnectionCollection[0].ConnectionManager != null)  {  var cm = Microsoft.SqlServer.Dts.Runtime.DtsConvert.GetWrapper(  ComponentMetaData.RuntimeConnectionCollection[0].ConnectionManager);  object o = cm.AcquireConnection(null);  if (o.GetType().Name == "SharePointConnectionManager")  {  \_sharePointConnectionManager = o as Microsoft.Sevices.GD.SharePointConnectionManager.SharePointConnectionManager;  if (\_sharePointConnectionManager.ServiceAccountForConnection == false)  {  if (\_sharePointConnectionManager.UserName.Length > 0)  {  if (\_sharePointConnectionManager.UserName.Contains(@"\"))  {  var userNameElements = \_sharePointConnectionManager.UserName.Split('\\');  if (userNameElements.Length == 2)  \_credentials = new NetworkCredential(userNameElements[1], \_sharePointConnectionManager.GetPassword(), userNameElements[0]);  }  else  \_credentials = new NetworkCredential(\_sharePointConnectionManager.UserName, \_sharePointConnectionManager.GetPassword());  }  }  else  \_credentials = CredentialCache.DefaultNetworkCredentials;  }  }  }  }  //Frees the connections established during AcquireConnections. Called at design time and run time.  public override void ReleaseConnections()  {  \_sharePointConnectionManager = null;  } |

You should now validate a source component and verify that the columns defined in its output column collections match the columns at the external data source. But, as you have not added any output columns so far, the validation will fail and [ReinitializeMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.reinitializemetadata.aspx) method will be called:

|  |
| --- |
| //The Validate() function is mostly called during the design-time phase of the component. Its main purpose is to perform validation of the contents of the component.  public override DTSValidationStatus Validate()  {  bool canCancel = false;  if ((ComponentMetaData.RuntimeConnectionCollection.Count == 0) ||  (\_sharePointConnectionManager == null))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "SharePoint Connection Manager is either not selected or is not of correct type; It must be 'SharePoint Connection Manager'...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value == null) ||  (((string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value).Length == 0))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify/choose the SharePoint list name...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_BATCHSIZE].Value == null) ||  ((short)ComponentMetaData.CustomPropertyCollection[C\_BATCHSIZE].Value) == 0)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify batch Size greater than 0...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_ISRECURSIVE].Value == null) ||  ((ComponentMetaData.CustomPropertyCollection[C\_ISRECURSIVE].Value.ToString()).Length == 0))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify to include rows from the subfolders within the list recursively...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_INCLUDEFOLDERS].Value == null) ||  ((ComponentMetaData.CustomPropertyCollection[C\_INCLUDEFOLDERS].Value.ToString()).Length == 0))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify to include folder with the list content...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_CAMLQUERY].Value == null) ||  ((ComponentMetaData.CustomPropertyCollection[C\_CAMLQUERY].Value.ToString()).Length != 0))  {  try  {  XElement.Parse((string)ComponentMetaData.CustomPropertyCollection[C\_CAMLQUERY].Value);  }  catch (System.Xml.XmlException)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify the CAML query in the valid XML format to filter rows...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  }  if ((ComponentMetaData.OutputCollection.Count == 0))  {  return DTSValidationStatus.VS\_NEEDSNEWMETADATA;  }  // Validate the columns defined against an actual SharePoint Site. Lookup the data dynamically against the SharePoint, and check if the columns marked for output exist and are up to date.  var isValid = ValidateSharePointColumns();  if (isValid != DTSValidationStatus.VS\_ISVALID)  return isValid;  return base.Validate();  } |

The [ReinitializeMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.reinitializemetadata.aspx) method is called in response to a component returning VS\_NEEDSNEWMETADATA from Validate method (validation process). In this method, components fix any errors that were identified in Validate method, if those errors can be fixed by the component that result in the component returning VS\_NEEDSNEWMETADATA.

|  |
| --- |
| using IDTSInputColumnCollection = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSInputColumnCollection100;  using IDTSExternalMetadataColumn = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSExternalMetadataColumn100;  using IDTSOutput = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSOutput100;  using IDTSOutputColumn = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSOutputColumn100;  using IDTSCustomProperty = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSCustomProperty100;  using IDTSVirtualInputColumn = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSVirtualInputColumn100;  using IDTSRuntimeConnection = Microsoft.SqlServer.Dts.Pipeline.Wrapper.IDTSRuntimeConnection100;  //The ReinitializeMetaData() method will be called when the Validate() function returns VS\_NEEDSNEWMETADATA. Its primary purpose is to repair the component's metadata to a consistent state.  public override void ReinitializeMetaData()  {  Validate();  IDTSOutput output;  var existingColumnData = new Dictionary<string, string>();  if (ComponentMetaData.OutputCollection.Count == 0)  {  output = ComponentMetaData.OutputCollection.New();  output.Name = "Public List Output";  output.ExternalMetadataColumnCollection.IsUsed = true;  }  else  {  output = ComponentMetaData.OutputCollection[0];  // Capture the existing column names and detail data  existingColumnData =  (from col in  output.OutputColumnCollection.Cast<IDTSOutputColumn>()  join metaCol in output.ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  on col.ExternalMetadataColumnID equals metaCol.ID  select new  {  ColumnName = (string)col.Name,  SpColName = (string)metaCol.CustomPropertyCollection["Id"].Value  }).ToDictionary(a => a.SpColName, a => a.ColumnName);  output.OutputColumnCollection.RemoveAll();  output.ExternalMetadataColumnCollection.RemoveAll();  }  // Reload in the output objects  LoadDataSourceInformation(output, existingColumnData);  base.ReinitializeMetaData();  }  //Loads the column data into the dts objects from the datasource for columns  private void LoadDataSourceInformation(IDTSOutput output, Dictionary<string, string> existingColumnData)  {  object sharepointUrl = \_sharePointConnectionManager.SharePointServerName;  object sharepointListName = ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value;  // Reset the values  if ((sharepointListName != null))  {  CreateExternalMetaDataColumns(output,  (string)sharepointUrl,  (string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value,  (string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTVIEWNAME].Value,  existingColumnData);  }  } |

The output columns of a source component reflect the columns from the external data source that the component adds to the data flow during execution. At design time, you need to add output columns after the component has been configured to connect to an external data source or during [ReinitializeMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.reinitializemetadata.aspx) method execution if the validation failed. After an output column has been created, you need to set its data type properties by calling the [SetDataTypeProperties](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsoutputcolumn100.setdatatypeproperties.aspx) method. This method call is necessary because the DataType, Length, Precision, and CodePage properties are read-only and each property is dependent on the settings of the other. This method enforces the need for these values to be set consistently, and the data flow task validates that they are set correctly.

|  |
| --- |
| //Connects to SharePoint and gets any columns on the target  private void CreateExternalMetaDataColumns(IDTSOutput output, string sharepointUrl, string listName, string viewName, Dictionary<string, string> existingColumnData)  {  // No need to load if the Url is bad.  if ((sharepointUrl == null) || (sharepointUrl.Length == 0))  return;  // Need a list to continue  if ((listName == null) || (listName.Length == 0))  return;  // If the list has changed, then you do not want any of the existing column data to  // influence it (provides a way to actually reset the names if needed)  if (output.Description != listName)  {  existingColumnData.Clear();  output.Description = listName;  }  try  {  List<ColumnData> accessibleColumns =  GetAccessibleSharePointColumns(sharepointUrl, listName, viewName);  foreach (var column in accessibleColumns)  {  // Setup the primary column details from the List  IDTSExternalMetadataColumn dtsColumnMeta = output.ExternalMetadataColumnCollection.New();  if (existingColumnData.ContainsKey(column.Name))  dtsColumnMeta.Name = existingColumnData[column.Name];  else  dtsColumnMeta.Name = column.Name;  dtsColumnMeta.Description = column.DisplayName;  dtsColumnMeta.Length = 0;  dtsColumnMeta.Precision = 0;  dtsColumnMeta.Scale = 0;  if ("Boolean|AllDayEvent|Attachments|CrossProjectLink|Recurrence".Contains(column.SharePointType))  {  dtsColumnMeta.DataType = DataType.DT\_BOOL;  }  else if (column.SharePointType == "DateTime")  {  dtsColumnMeta.DataType = DataType.DT\_DBTIMESTAMP;  }  else if ("Number|Currency".Contains(column.SharePointType))  {  // Max = 100,000,000,000.00000  dtsColumnMeta.DataType = DataType.DT\_R8;  }  else if ("Counter|Integer".Contains(column.SharePointType))  {  dtsColumnMeta.DataType = DataType.DT\_I4;  }  else if ("Guid".Contains(column.SharePointType))  {  dtsColumnMeta.DataType = DataType.DT\_GUID;  }  else  {  if (column.MaxLength == -1)  {  dtsColumnMeta.DataType = DataType.DT\_NTEXT;  dtsColumnMeta.Length = 0;  }  else  {  dtsColumnMeta.DataType = DataType.DT\_WSTR;  dtsColumnMeta.Length = column.MaxLength;  }  }  IDTSCustomProperty fieldNameMeta = dtsColumnMeta.CustomPropertyCollection.New();  fieldNameMeta.Name = "Id";  fieldNameMeta.Description = "SharePoint ID";  fieldNameMeta.Value = column.Name;  // Create default output columns for all of the fields returned and link to the original columns  IDTSOutputColumn dtsColumn = output.OutputColumnCollection.New();  dtsColumn.Name = dtsColumnMeta.Name;  dtsColumn.Description = dtsColumnMeta.Description;  dtsColumn.ExternalMetadataColumnID = dtsColumnMeta.ID;  IDTSCustomProperty fieldName = dtsColumn.CustomPropertyCollection.New();  fieldName.Name = fieldNameMeta.Name;  fieldName.Description = fieldNameMeta.Description;  fieldName.Value = fieldNameMeta.Value;  dtsColumn.SetDataTypeProperties(  dtsColumnMeta.DataType, dtsColumnMeta.Length, dtsColumnMeta.Precision, dtsColumnMeta.Scale, 0);  }  }  catch (ApplicationException)  {  // Exception happened, so clear the columns, which will invalidate this object.  output.ExternalMetadataColumnCollection.RemoveAll();  }  } |

## Run Time Functionalities

The [IDTSRuntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsruntimecomponent100.aspx) interface defines the methods and properties that are called during execution of a data flow component. You don’t need to explicitly implement this interface as the base class [PipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.aspx) already implements them. You need to override them with your own implementation though.

During execution, components add rows to output buffers that are created by the data flow task and provided to the component in the PrimeOutput method, which is called once for source components. This method receives an output buffer for each IDTSOutput100 of the component that is connected to a downstream component.

You should factor as much functionality into [PreExecute](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.preexecute.aspx) method as possible because it is called one time per component instead of one time per buffer for [ProcessInput](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.processinput.aspx).

|  |
| --- |
| public override void PreExecute()  {  base.PreExecute();  // Get the field names from the input collection  \_bufferLookup = (from col in  ComponentMetaData.OutputCollection[0].OutputColumnCollection.Cast<IDTSOutputColumn>()  join metaCol in ComponentMetaData.OutputCollection[0].ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  on col.ExternalMetadataColumnID equals metaCol.ID  select new  {  Name = (string)metaCol.CustomPropertyCollection["Id"].Value,  BufferColumn = BufferManager.FindColumnByLineageID(ComponentMetaData.OutputCollection[0].Buffer, col.LineageID)  }).ToDictionary(a => a.Name, a => a.BufferColumn);  // Get the field data types from the input collection  \_bufferLookupDataType = (from col in  ComponentMetaData.OutputCollection[0].OutputColumnCollection.Cast<IDTSOutputColumn>()  join metaCol in ComponentMetaData.OutputCollection[0].ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  on col.ExternalMetadataColumnID equals metaCol.ID  select new  {  Name = (string)metaCol.CustomPropertyCollection["Id"].Value,  DataType = metaCol.DataType  }).ToDictionary(a => a.Name, a => a.DataType);  } |

Rows are added to the output buffer by calling the AddRow method, which creates a new buffer row with empty values in its columns. You then need to assign values to the individual columns of the component. The output buffers provided to a component are created and monitored by the data flow task. As they become full, the rows in the buffer are moved to the next component. There is no way to determine when a batch of rows has been sent to the next component because the movement of rows by the data flow task is transparent to the component developer, and the RowCount property is always zero on output buffers. When a source component is finished adding rows to its output buffer, it notifies the data flow task by calling the SetEndOfRowset method of the PipelineBuffer, and the remaining rows in the buffer are passed to the next component. The following code example shows a component that adds rows to an output buffer in the [PrimeOutput](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.primeoutput.aspx) method (called at run time for source components and transformation components with asynchronous outputs to let these components add rows to the output buffers).

|  |
| --- |
| //PrimeOutput method is used to load the data in output buffer  public override void PrimeOutput(int outputs, int[] outputIDs, PipelineBuffer[] buffers)  {  string sharepointUrl = \_sharePointConnectionManager.SharePointServerName;  string sharepointList = (string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value;  string sharepointListView = (string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTVIEWNAME].Value;  XElement camlQuery = XElement.Parse((string)ComponentMetaData.CustomPropertyCollection[C\_CAMLQUERY].Value);  short batchSize = (short)ComponentMetaData.CustomPropertyCollection[C\_BATCHSIZE].Value;  Enums.TrueFalseValue isRecursive = (Enums.TrueFalseValue)ComponentMetaData.CustomPropertyCollection[C\_ISRECURSIVE].Value;  Enums.TrueFalseValue includeFolders = (Enums.TrueFalseValue)ComponentMetaData.CustomPropertyCollection[C\_INCLUDEFOLDERS].Value;  PipelineBuffer outputBuffer = buffers[0];  // Get the field names from the output collection  var fieldNames = (from col in  ComponentMetaData.OutputCollection[0].OutputColumnCollection.Cast<IDTSOutputColumn>()  select (string)col.CustomPropertyCollection[0].Value);  // Load the data from sharepoint  System.Diagnostics.Stopwatch timer = new System.Diagnostics.Stopwatch();  timer.Start();  var listData = SharePointListServiceCommons.GetListItemData(new Uri(sharepointUrl), \_credentials, sharepointList, sharepointListView, fieldNames, camlQuery,  isRecursive == Enums.TrueFalseValue.True ? true : false, batchSize, \_sharePointConnectionManager.SharePointTimeOut);  timer.Stop();  bool fireAgain = false;  int actualRowCount = 0;  foreach (var row in listData)  {  // Determine if you should continue based on if this is a folder item or not (filter can be pushed up to CAML if  // perf becomes an issue)  bool canContinue = true;  if ((row.ContainsKey("ContentType")) &&  (row["ContentType"] == "Folder") &&  (includeFolders == Enums.TrueFalseValue.False))  {  canContinue = false;  }  if (canContinue)  {  actualRowCount++;  outputBuffer.AddRow();  foreach (var fieldName in \_bufferLookup.Keys)  {  if (row.ContainsKey(fieldName))  {  switch (\_bufferLookupDataType[fieldName])  {  case DataType.DT\_NTEXT:  outputBuffer.AddBlobData(\_bufferLookup[fieldName],  Encoding.Unicode.GetBytes(row[fieldName].ToString()));  break;  case DataType.DT\_WSTR:  outputBuffer.SetString(\_bufferLookup[fieldName], row[fieldName]);  break;  case DataType.DT\_R8:  outputBuffer.SetDouble(\_bufferLookup[fieldName], double.Parse(row[fieldName]));  break;  case DataType.DT\_I4:  outputBuffer.SetInt32(\_bufferLookup[fieldName], int.Parse(row[fieldName]));  break;  case DataType.DT\_BOOL:  outputBuffer.SetBoolean(\_bufferLookup[fieldName], (int.Parse(row[fieldName]) == 1));  break;  case DataType.DT\_GUID:  outputBuffer.SetGuid(\_bufferLookup[fieldName], new Guid(row[fieldName]));  break;  case DataType.DT\_DBTIMESTAMP:  outputBuffer.SetDateTime(\_bufferLookup[fieldName], DateTime.Parse(row[fieldName]));  break;  }  }  else  {  switch (\_bufferLookupDataType[fieldName])  {  case DataType.DT\_NTEXT:  outputBuffer.AddBlobData(\_bufferLookup[fieldName],  Encoding.Unicode.GetBytes(String.Empty));  break;  case DataType.DT\_WSTR:  outputBuffer.SetString(\_bufferLookup[fieldName], String.Empty);  break;  case DataType.DT\_BOOL:  outputBuffer.SetBoolean(\_bufferLookup[fieldName], false);  break;  default:  outputBuffer.SetNull(\_bufferLookup[fieldName]);  break;  }  }  }  }  }  string infoMsg = string.Format(CultureInfo.InvariantCulture,  "SharePoint Source List - {0} records loaded from the list '{1}' at '{2}'. Elapsed time is {3}ms",  actualRowCount,  sharepointList,  sharepointUrl,  timer.ElapsedMilliseconds);  ComponentMetaData.FireInformation(0, ComponentMetaData.Name, infoMsg, "", 0, ref fireAgain);  ComponentMetaData.IncrementPipelinePerfCounter(  DTS\_PIPELINE\_CTR\_ROWSREAD, (uint)actualRowCount);  outputBuffer.SetEndOfRowset();  } |

# Creating Custom Destination Component

Custom destination components can be created to connect to and store data in any custom data destination. Custom destination components are useful when you need to connect to data destination that cannot be accessed by using one of the existing destination components included within SSIS. Destination components have one or more inputs and zero outputs. At design time, they create and configure connections and read column metadata from the external data source. During execution, they connect to their external data source and add/write rows that are received from the components upstream in the data flow to the external data source. If the external data source exists prior to execution of the component, the destination component must also ensure that the data types of the columns that the component receives match the data types of the columns at the external data source.

To create a custom destination component, you need to create a class by inheriting it from [PipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.aspx) (base class that is used when developing managed data flow components and which implements the required component interfaces, [IDTSDesigntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.aspx) and [IDTSRuntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsruntimecomponent100.aspx)). Then you need to apply the attribute that controls how the SSIS Designer displays and interacts with the object (in this case the attribute is [DtsPipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.dtspipelinecomponentattribute.aspx)) as shown below:

|  |
| --- |
| [DtsPipelineComponent(DisplayName = "SharePoint List Destination",  CurrentVersion = 1,  IconResource = "Microsoft.Sevices.GD.SharePointSSISComponents.Icons.SharePointDestinationAdapter.ico",  Description = "SharePoint List Destination Adapter to write (Add/Modify/Delete) data to SharePoint (In Premise or Cloud) lists.",  ComponentType = ComponentType.SourceAdapter)]  public class SharePointListDestinationAdapter : PipelineComponent  { |

## Design Time Functionalities

[IDTSDesigntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.aspx) interface defines the design-time interface for a component and contains methods that are called when the component properties and collections of the [ComponentMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsruntimecomponent100.componentmetadata.aspx) are modified. You don’t need to explicitly implement this interface as the base class [PipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.aspx) already implements them. You need to override them with our own implementation though.

Implementing the design-time functionality of a destination component involves specifying a connection to an external data source and validating that the component has been correctly configured. By definition, a destination component has one input and possibly one error output.

Destination components connect to external data sources by using Connection Manager objects defined in a package. They indicate their requirement for a connection manager by adding an element to the [RuntimeConnectionCollection](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtscomponentmetadata100.runtimeconnectioncollection.aspx) collection of the ComponentMetaData property as shown below.

|  |
| --- |
| //The ProvideComponentProperties() method provides initialization of the component when the component is first added to the data Flow designer  public override void ProvideComponentProperties()  {  ComponentMetaData.RuntimeConnectionCollection.RemoveAll();  ComponentMetaData.CustomPropertyCollection.RemoveAll();  RemoveAllInputsOutputsAndCustomProperties();  // Add the custom Properties  AddUserProperties();  // Add the connection manager.  var connection = ComponentMetaData.RuntimeConnectionCollection.New();  connection.Name = "SharePointConnection";  }  private void AddUserProperties()  {  // Add Custom Properties  var sharepointListName = ComponentMetaData.CustomPropertyCollection.New();  sharepointListName.Name = C\_SHAREPOINTLISTNAME;  sharepointListName.Description = "Specify the name of the SharePoint list to load data to.";  sharepointListName.ExpressionType = DTSCustomPropertyExpressionType.CPET\_NOTIFY;    var batchSize = ComponentMetaData.CustomPropertyCollection.New();  batchSize.Name = C\_BATCHSIZE;  batchSize.Value = (short)200;  batchSize.Description = "Specify # of rows to push to SharePoint list at a time. The more number of columns or wider the columns you should consider keeping it low.";  batchSize.TypeConverter = typeof(short).AssemblyQualifiedName;  var batchType = ComponentMetaData.CustomPropertyCollection.New();  batchType.Name = C\_BATCHTYPE;  batchType.Value = Enums.BatchType.Add;  batchType.Description = "Specify destination action type; for example Add to adding a row; Modify for updating a row; or Delete for deleting a row. Note - Modify and Delete require ID Column with the SharePoint ID.";  batchType.TypeConverter = typeof(Enums.BatchType).AssemblyQualifiedName;  var input = ComponentMetaData.InputCollection.New();  input.Name = "Component Input";  input.Description = "This is what you see from the upstream component";  input.HasSideEffects = true;  ComponentMetaData.Name = "SharePoint List Destination";  ComponentMetaData.Description = "SharePoint List Destination Adapter to write data to SharePoint (In Premise or Cloud) lists.";  ComponentMetaData.ContactInfo = "http://www.microsoft.com/microsoftservices/en/us/home.aspx";  } |

After a connection has been added to the [RuntimeConnectionCollection](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtscomponentmetadata100.runtimeconnectioncollection.aspx), you need to override the [AcquireConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.acquireconnections.aspx) method to establish a connection to the external data source. This method is called during both design time and execution time. After the connection is established, it gets cached internally by the component in a local member variable and released when the [ReleaseConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.releaseconnections.aspx) method is called. The [ReleaseConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.releaseconnections.aspx) method is called at design time and execution time, like the [AcquireConnections](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.acquireconnections.aspx) method.

|  |
| --- |
| //Called at design time and runtime. Establishes a connection using a ConnectionManager embedded in the package.  public override void AcquireConnections(object transaction)  {  \_sharePointConnectionManager = null;  \_credentials = null;  if (ComponentMetaData.RuntimeConnectionCollection.Count > 0)  {  if (ComponentMetaData.RuntimeConnectionCollection[0].ConnectionManager != null)  {  var cm = Microsoft.SqlServer.Dts.Runtime.DtsConvert.GetWrapper(  ComponentMetaData.RuntimeConnectionCollection[0].ConnectionManager);  object o = cm.AcquireConnection(null);  if (o.GetType().Name == "SharePointConnectionManager")  {  \_sharePointConnectionManager = o as Microsoft.Sevices.GD.SharePointConnectionManager.SharePointConnectionManager;  if (\_sharePointConnectionManager.ServiceAccountForConnection == false)  {  if (\_sharePointConnectionManager.UserName.Length > 0)  {  if (\_sharePointConnectionManager.UserName.Contains(@"\"))  {  var userNameElements = \_sharePointConnectionManager.UserName.Split('\\');  if (userNameElements.Length == 2)  \_credentials = new NetworkCredential(userNameElements[1], \_sharePointConnectionManager.GetPassword(), userNameElements[0]);  }  else  \_credentials = new NetworkCredential(\_sharePointConnectionManager.UserName, \_sharePointConnectionManager.GetPassword());  }  }  else  \_credentials = CredentialCache.DefaultNetworkCredentials;  }  }  }  }  //Frees the connections established during AcquireConnections. Called at design time and run time.  public override void ReleaseConnections()  {  \_sharePointConnectionManager = null;  } |

You should now validate a destination component and verify that the data type properties of the columns defined in the component's input column collection match the columns at the external data source. But, as you have not added any output columns, the validation will fail and [ReinitializeMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.reinitializemetadata.aspx) method will be called:

|  |
| --- |
| //The Validate() function is mostly called during the design-time phase of the component. Its main purpose is to perform validation of the contents of the component.  public override DTSValidationStatus Validate()  {  bool canCancel = false;  if (ComponentMetaData.OutputCollection.Count != 0)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Unexpected Output found. Destination components do not support outputs.",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISCORRUPT;  }  if (ComponentMetaData.InputCollection.Count != 1)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "There must be one input into this component, please specify one.",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISCORRUPT;  }  if (ComponentMetaData.AreInputColumnsValid == false)  {  return DTSValidationStatus.VS\_NEEDSNEWMETADATA;  }  if ((ComponentMetaData.RuntimeConnectionCollection.Count == 0) || (\_sharePointConnectionManager == null))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "SharePoint Connection Manager either not selected or not selected of correct type; It must be 'SharePoint Connection Manager'...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value == null) ||  (((string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value).Length == 0))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify/choose the SharePoint list name...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_BATCHSIZE].Value == null) ||  ((short)ComponentMetaData.CustomPropertyCollection[C\_BATCHSIZE].Value) == 0)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify batch Size greater than 0...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value == null) ||  ((ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value.ToString()).Length == 0))  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify the destination action type; it must be Modification or Deletion...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  if ((ComponentMetaData.InputCollection.Count == 0))  {  return DTSValidationStatus.VS\_NEEDSNEWMETADATA;  }  // Validate the columns defined against an actual SharePoint Site  var isValid = ValidateSharePointColumns();  if (isValid != DTSValidationStatus.VS\_ISVALID)  return isValid;  return base.Validate();  }  private DTSValidationStatus ValidateSharePointColumns()  {  bool canCancel;  // Check the input columns and see if they are the same as the # of columns in the selected list  string sharepointUrl = \_sharePointConnectionManager.SharePointServerName;  string listName = (string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value;  // Get the column information from SharePoint  List<ColumnData> accessibleColumns = null;  try  {  accessibleColumns = GetAccessibleSharePointColumns(sharepointUrl, listName,"");  }  catch (SharePointUnhandledException)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Unable to get list information from SharePoint - Site: " + sharepointUrl + ", List: " + listName,  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  // Get the field names of the columns  var fieldNames = (from col in ComponentMetaData.InputCollection[0].ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  select (string)col.CustomPropertyCollection["Id"].Value);  // Join them together and see if you get the full sharepoint column list.  if ((from spCol in accessibleColumns  join inputCol in fieldNames on spCol.Name equals inputCol  select spCol).Count() != fieldNames.Count())  {  // Column names do not match, request new data.  return DTSValidationStatus.VS\_NEEDSNEWMETADATA;  }  // Verify the field mappings by getting the column and the meta column after joining them together  var mappedFields =  from col in  ComponentMetaData.InputCollection[0].InputColumnCollection.Cast<IDTSInputColumn>()  join metaCol in ComponentMetaData.InputCollection[0].ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  on col.ExternalMetadataColumnID equals metaCol.ID  select new { col, metaCol };  // Make sure at least one field is mapped.  if (mappedFields.Count() == 0)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "There are no input fields mapped to the output columns, please specify mapping; Click Refresh to refresh mapping automatically!...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  // If modifying or deleting, make sure one of the columns is mapped to the SharePoint ID column, or  // else the adapter will not know what to delete.  if (((Enums.BatchType)ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value == Enums.BatchType.Delete) || ((Enums.BatchType)ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value == Enums.BatchType.Modify))  {  if ((from col in mappedFields  where (string)col.metaCol.CustomPropertyCollection["Id"].Value == "ID"  select col).FirstOrDefault() == null)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please specify a mapping of a column from the input for the ID output column if modifying or deleting data; Click Refresh to refresh mapping automatically!...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  }  // If adding then SharePoint ID column shouldn't be mapped  if ((Enums.BatchType)ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value == Enums.BatchType.Add)  {  if ((from col in mappedFields  where (string)col.metaCol.CustomPropertyCollection["Id"].Value == "ID"  select col).FirstOrDefault() != null)  {  ComponentMetaData.FireError(0, ComponentMetaData.Name,  "Please don't specify a mapping of a column from the input for the ID output column if adding data; specify <ignore> for ID destination...",  "", 0, out canCancel);  return DTSValidationStatus.VS\_ISBROKEN;  }  }  return DTSValidationStatus.VS\_ISVALID;  } |

The [ReinitializeMetaData](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsdesigntimecomponent100.reinitializemetadata.aspx) method is called in response to a component returning VS\_NEEDSNEWMETADATA from the Validate method. In this method, components fix any errors that were identified in the Validate method, if those errors can be fixed by the component that result in the component returning VS\_NEEDSNEWMETADATA.

|  |
| --- |
| public override void ReinitializeMetaData()  {  if (ComponentMetaData.InputCollection.Count > 0)  {  var input = ComponentMetaData.InputCollection[0];  // Capture the existing column names and detail data before data is re-initialized.  \_existingColumnData =  (from metaCol in  input.ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  select new  {  ColumnName = (string)metaCol.Name,  SpColName = (string)metaCol.CustomPropertyCollection["Id"].Value  }).ToDictionary(a => a.SpColName, a => a.ColumnName);  // Reset the input columns  ComponentMetaData.InputCollection[0].InputColumnCollection.RemoveAll();  // Reload the input path columns  OnInputPathAttached(ComponentMetaData.InputCollection[0].ID);  }  base.ReinitializeMetaData();  } |

The following code adds external metadata columns based on existing input columns.

|  |
| --- |
| //Connects to SharePoint and gets any columns on the target  private void CreateExternalMetaDataColumns(  IDTSInput input, string sharepointUrl, string listName, string viewName,  Dictionary<string, string> existingColumnData)  {  // No need to load if the Url is bad.  if ((sharepointUrl == null) || (sharepointUrl.Length == 0))  return;  // Need a list to continue  if ((listName == null) || (listName.Length == 0))  return;  input.ExternalMetadataColumnCollection.IsUsed = true;  // If the list has changed, then you do not want any of the existing column data to  // influence it (provides a way to actually reset the names if needed)  if (input.Description != listName)  {  existingColumnData.Clear();  input.Description = listName;  }  try  {  List<ColumnData> accessibleColumns = GetAccessibleSharePointColumns(sharepointUrl, listName, viewName);  foreach (var column in accessibleColumns)  {  // Setup the primary column details from the List  var dtsColumnMeta = input.ExternalMetadataColumnCollection.New();  if (existingColumnData.ContainsKey(column.Name))  dtsColumnMeta.Name = existingColumnData[column.Name];  else  dtsColumnMeta.Name = column.FriendlyName;  dtsColumnMeta.Description = column.DisplayName;  dtsColumnMeta.Length = 0;  dtsColumnMeta.Precision = 0;  dtsColumnMeta.Scale = 0;  if ("Boolean|AllDayEvent|Attachments|CrossProjectLink|Recurrence".Contains(column.SharePointType))  {  dtsColumnMeta.DataType = DataType.DT\_BOOL;  }  else if (column.SharePointType == "DateTime")  {  dtsColumnMeta.DataType = DataType.DT\_DBTIMESTAMP;  }  else if ("Number|Currency".Contains(column.SharePointType))  {  // Max = 100,000,000,000.00000  dtsColumnMeta.DataType = DataType.DT\_R8;  }  else if ("Counter|Integer".Contains(column.SharePointType))  {  dtsColumnMeta.DataType = DataType.DT\_I4;  }  else if ("Guid".Contains(column.SharePointType))  {  dtsColumnMeta.DataType = DataType.DT\_GUID;  }  else  {  if (column.MaxLength == -1)  {  dtsColumnMeta.DataType = DataType.DT\_NTEXT;  dtsColumnMeta.Length = 0;  }  else  {  dtsColumnMeta.DataType = DataType.DT\_WSTR;  dtsColumnMeta.Length = column.MaxLength;  }  }  var fieldNameMeta = dtsColumnMeta.CustomPropertyCollection.New();  fieldNameMeta.Name = "Id";  fieldNameMeta.Description = "SharePoint ID";  fieldNameMeta.Value = column.Name;  Enums.BatchType batchType = (Enums.BatchType)ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value;  if (!(batchType == Enums.BatchType.Add && dtsColumnMeta.Name == "ID"))  {  // Map any columns found with the same name in the input  var foundCol =  (from col in input.InputColumnCollection.Cast<IDTSInputColumn>()  where col.Name == dtsColumnMeta.Name  select col).SingleOrDefault();  if (foundCol != null)  {  foundCol.ExternalMetadataColumnID = dtsColumnMeta.ID;  }  }  }  }  catch (SharePointUnhandledException)  {  // Exception happened, so clear the columns, which will invalidate this object.  input.ExternalMetadataColumnCollection.RemoveAll();  throw;  }  } |

## Run Time Functionalities

The [IDTSRuntimeComponent100](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.wrapper.idtsruntimecomponent100.aspx) interface defines the methods and properties that are called during execution of a data flow component. You don’t need to explicitly implement this interface as the base class [PipelineComponent](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.aspx) already implements it. You need to override the methods in the interface with our own implementation though.

During execution, the destination component receives a call to the [ProcessInput](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.processinput.aspx) method each time a full PipelineBuffer is available from the upstream component. This method is called repeatedly until there are no more buffers available and the EndOfRowset property is true. During this method, destination components read columns and rows in the buffer, and add/write them to the external data source.

You should factor as much functionality into [PreExecute](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.preexecute.aspx) as possible because it is called one time per component instead of one time per buffer for [ProcessInput](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.processinput.aspx).

|  |
| --- |
| public override void PreExecute()  {  base.PreExecute();  // Get the field names from the input collection  \_bufferLookup = (from col in  ComponentMetaData.InputCollection[0].InputColumnCollection.Cast<IDTSInputColumn>()  join metaCol in ComponentMetaData.InputCollection[0].ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  on col.ExternalMetadataColumnID equals metaCol.ID  select new  {  Name = (string)metaCol.CustomPropertyCollection["Id"].Value,  BufferColumn = BufferManager.FindColumnByLineageID(ComponentMetaData.InputCollection[0].Buffer, col.LineageID)  }).ToDictionary(a => a.Name, a => a.BufferColumn);  // Get the field data types from the input collection  \_bufferLookupDataType = (from col in  ComponentMetaData.InputCollection[0].InputColumnCollection.Cast<IDTSInputColumn>()  join metaCol in ComponentMetaData.InputCollection[0].ExternalMetadataColumnCollection.Cast<IDTSExternalMetadataColumn>()  on col.ExternalMetadataColumnID equals metaCol.ID  select new  {  Name = (string)metaCol.CustomPropertyCollection["Id"].Value,  DataType = col.DataType  }).ToDictionary(a => a.Name, a => a.DataType);  } |

Once the input columns have been located in the buffer, they can be read and written to the external data source. The following example shows a component that reads rows from the buffer provided in [ProcessInput](http://msdn.microsoft.com/en-us/library/microsoft.sqlserver.dts.pipeline.pipelinecomponent.processinput.aspx).

|  |
| --- |
| //PrimeOutput method is used to load the data in output buffer  public override void ProcessInput(int inputID, PipelineBuffer buffer)  {  string sharepointUrl = \_sharePointConnectionManager.SharePointServerName;  string sharepointList = (string)ComponentMetaData.CustomPropertyCollection[C\_SHAREPOINTLISTNAME].Value;  short batchSize = (short)ComponentMetaData.CustomPropertyCollection[C\_BATCHSIZE].Value;  Enums.BatchType batchType = (Enums.BatchType)ComponentMetaData.CustomPropertyCollection[C\_BATCHTYPE].Value;  if (!buffer.EndOfRowset)  {  // Queue the data up for batching by the sharepoint accessor object  var dataQueue = new List<Dictionary<string, string>>();  while (buffer.NextRow())  {  var rowData = new Dictionary<string, string>();  foreach (var fieldName in \_bufferLookup.Keys)  {  switch (\_bufferLookupDataType[fieldName])  {  case DataType.DT\_STR:  case DataType.DT\_WSTR:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetString(\_bufferLookup[fieldName]));  break;  case DataType.DT\_NTEXT:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  {  int colDataLength = (int)buffer.GetBlobLength(\_bufferLookup[fieldName]);  byte[] stringData = buffer.GetBlobData(\_bufferLookup[fieldName], 0, colDataLength);  rowData.Add(fieldName, Encoding.Unicode.GetString(stringData));  }  break;  case DataType.DT\_R4:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetSingle(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_R8:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetDouble(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_UI1:  case DataType.DT\_I1:  case DataType.DT\_BOOL:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetBoolean(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_UI2:  case DataType.DT\_I2:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetInt16(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_UI4:  case DataType.DT\_I4:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetInt32(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_UI8:  case DataType.DT\_I8:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, string.Empty);  else  rowData.Add(fieldName, buffer.GetInt64(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_GUID:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, String.Empty);  else  rowData.Add(fieldName, buffer.GetGuid(\_bufferLookup[fieldName]).ToString());  break;  case DataType.DT\_DBTIMESTAMP:  if (buffer.IsNull(\_bufferLookup[fieldName]))  rowData.Add(fieldName, String.Empty);  else  rowData.Add(fieldName, buffer.GetDateTime(\_bufferLookup[fieldName]).ToString("u").Replace(" ", "T"));  break;  }  }  dataQueue.Add(rowData);  }  bool fireAgain = false;  if (dataQueue.Count() > 0)  {  System.Diagnostics.Stopwatch timer = new System.Diagnostics.Stopwatch();  timer.Start();  System.Xml.Linq.XElement resultData;  if (batchType == Enums.BatchType.Add)  {  // Perform the Add  resultData = SharePointListServiceCommons.AddListItems(  new Uri(sharepointUrl), \_credentials, sharepointList, "", dataQueue, batchSize, \_sharePointConnectionManager.SharePointTimeOut);  }  else if (batchType == Enums.BatchType.Modify)  {  // Perform the update  resultData = SharePointListServiceCommons.UpdateListItems(  new Uri(sharepointUrl), \_credentials, sharepointList, "", dataQueue, batchSize, \_sharePointConnectionManager.SharePointTimeOut);  }  else  {  // Get the IDs read from the buffer  var idList = from data in dataQueue  where data["ID"].Trim().Length > 0  select data["ID"];  // Delete the list items with IDs  resultData = SharePointListServiceCommons.DeleteListItems(  new Uri(sharepointUrl), \_credentials, sharepointList, "", idList, \_sharePointConnectionManager.SharePointTimeOut);  }  timer.Stop();  var errorRows = from result in resultData.Descendants("errorCode")  select result.Parent;  int successRowsWritten = resultData.Elements().Count() - errorRows.Count();  string infoMsg = string.Empty;  if (batchType == Enums.BatchType.Add)  {  infoMsg = string.Format(CultureInfo.InvariantCulture,  "Added {0} records in list '{1}' at '{2}'. Elapsed time is {3}ms", successRowsWritten, sharepointList, sharepointUrl, timer.ElapsedMilliseconds);  }  else if (batchType == Enums.BatchType.Modify)  {  infoMsg = string.Format(CultureInfo.InvariantCulture,  "Modified {0} records in list '{1}' at '{2}'. Elapsed time is {3}ms", successRowsWritten, sharepointList, sharepointUrl, timer.ElapsedMilliseconds);  }  else  {  infoMsg = string.Format(CultureInfo.InvariantCulture,  "Deleted {0} records in list '{1}' at '{2}'. Elapsed time is {3}ms", successRowsWritten, sharepointList, sharepointUrl, timer.ElapsedMilliseconds);  }  ComponentMetaData.FireInformation(0, ComponentMetaData.Name, infoMsg, "", 0, ref fireAgain);  ComponentMetaData.IncrementPipelinePerfCounter(DTS\_PIPELINE\_CTR\_ROWSWRITTEN, (uint)successRowsWritten);  // Shovel any error rows to the error flow  bool cancel;  int errorIter = 0;  foreach (var row in errorRows)  {  // Do not flood the error log.  errorIter++;  if (errorIter > 10)  {  ComponentMetaData.FireError(0,  ComponentMetaData.Name,  "Total of " + errorRows.Count().ToString() + ", only showing first 10.", "", 0, out cancel);  // Need to throw an exception, or else this step's box is green (should be red), even though the flow  // is marked as failure regardless.  throw new PipelineProcessException("SharePoint List Destination Failed - see Progress/Execution tab error logs.");  //return;  }  string idString = "";  XAttribute attrib = row.Element("row").Attribute("ID");  if (attrib != null)  idString = "(SP ID=" + attrib.Value + ")";  string errorString = string.Format(CultureInfo.InvariantCulture,  "Exception {1} - Row {0} : {2} - {3}",  row.Attribute("ID"),  row.Element("errorCode").Value,  row.Element("errorDescription").Value,  idString);  ComponentMetaData.FireError(0, ComponentMetaData.Name, errorString, "", 0, out cancel);  if (errorIter >= errorRows.Count())  {  // Need to throw an exception, or else this step's box is green (should be red), even though the flow  // is marked as failure regardless.  throw new PipelineProcessException("SharePoint List Destination Failed - see Progress/Execution tab error logs.");  }  }  }  else  {  ComponentMetaData.FireInformation(0, ComponentMetaData.Name,  "No rows found to update in destination.", "", 0, ref fireAgain);  }  }  } |

# Using and Debugging Custom Components

After you are done with the coding for a custom object for SSIS, you must build the assembly, deploy it, and integrate it into SSIS Designer to make it available for use in packages, and test and debug it. To learn more about steps for building, deploying, using/testing and debugging it [click here.](http://msdn.microsoft.com/en-US/library/ms403356(v=sql.110).aspx)

## Software Requirements

### SQL Server 2008 or SQL Server 2008 R2

* Microsoft .NET Framework version 3.5.
* Microsoft SQL Server Integration Services (SSIS) 2008 or 2008 R2.
* Microsoft Visual Studio 2008 (or SQL Server Business Intelligence Development Studio).
* Microsoft SharePoint Server 2007 or 2012 *– for the above SharePoint components only*

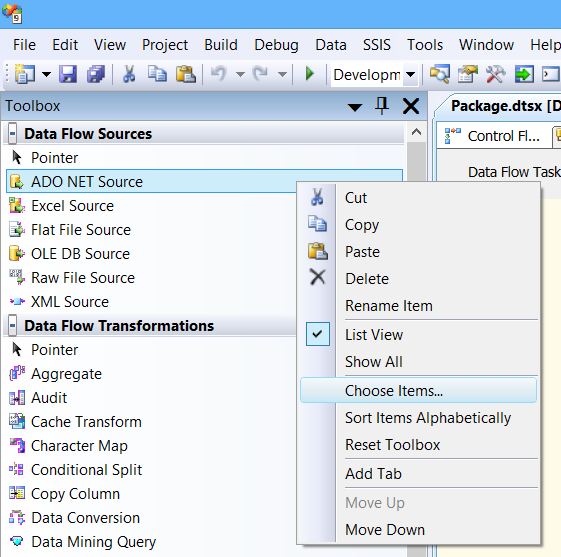
### SQL Server 2012

* Microsoft .NET Framework version 4.0.
* Microsoft SQL Server Integration Services (SSIS) 2012.
* Microsoft Visual Studio 2010 (or SQL Server Data Tools).
* Microsoft SharePoint Server 2007 or 2012 *– for the above SharePoint components only*

## Using or Testing Components

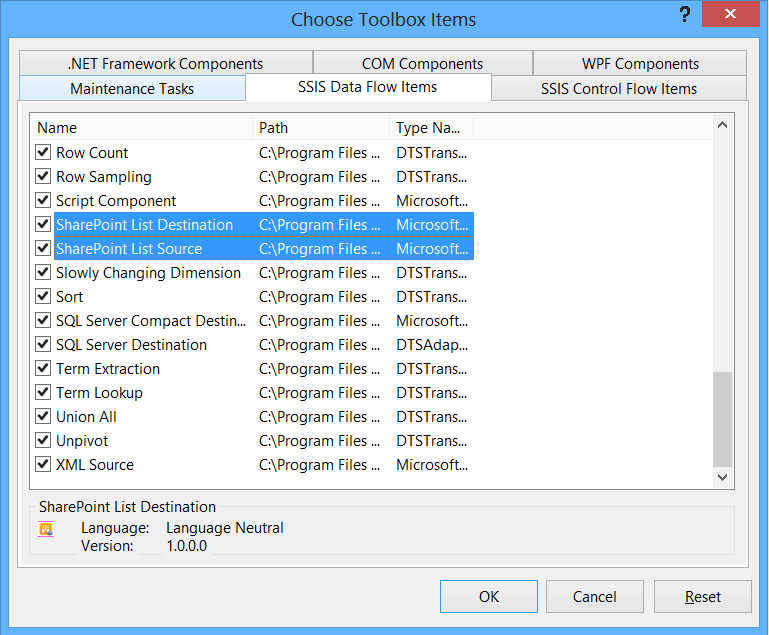
You can use the installed custom components like any other in-built components of SSIS.

1. Open **Microsoft Visual Studio 2008** (or **SQL Server Business Intelligence Development Studio**) or **Microsoft Visual Studio 2010** (or **SQL Server Data Tools**).
2. Open any **SQL Server Integration Services** (SSIS) project and open any package.
3. Open the **Toolbox** window (Ctrl+Alt+X).
4. Drag a Data Flow task on the designer and double click it.
5. If the components don’t appear in the Toolbox, right-click **Data Flow Sources** section and click **Choose Items**.



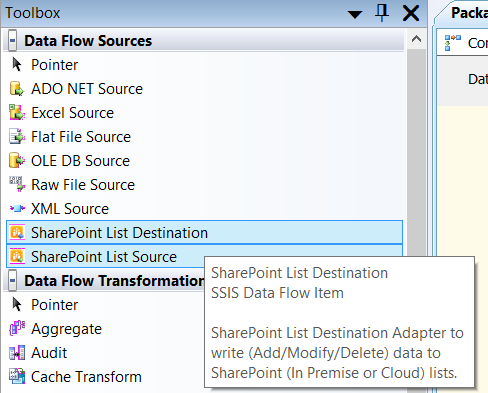
**Figure 7 - Adding components to BIDS - 1**

1. In the **Choose Toolbox Items** window, click **SSIS Data Flow Items** tab, select **SharePoint List Destination** and **SharePoint List Source** tasks (or the name of your components) and click **OK**.



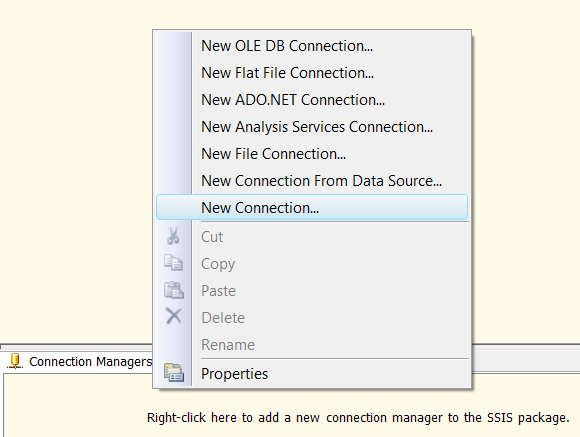
**Figure 8 - Adding components to BIDS - 2**

1. The **SharePoint List Destination** and **SharePoint List Source** tasks (or your components) will start appearing in the Toolbar as shown below.



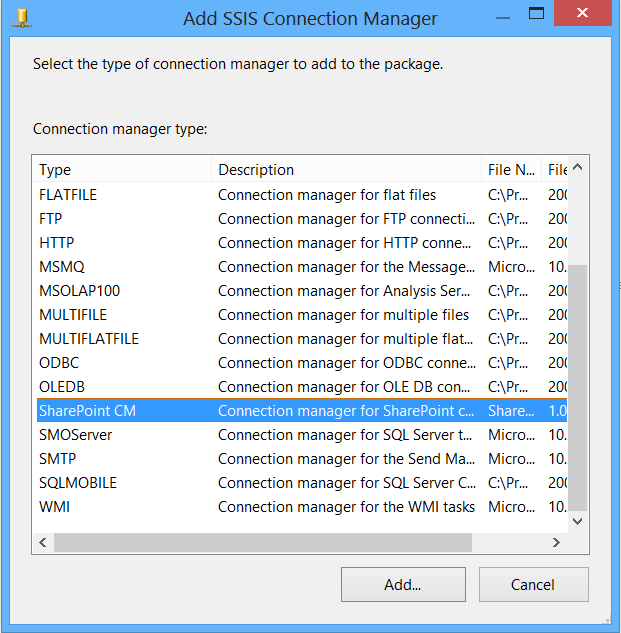
**Figure 9 - Adding components to BIDS - 3**

1. To add a **SharePoint Connection Manager** to the package right click the **Connection Managers** pane and then click **New Connection** as shown below:



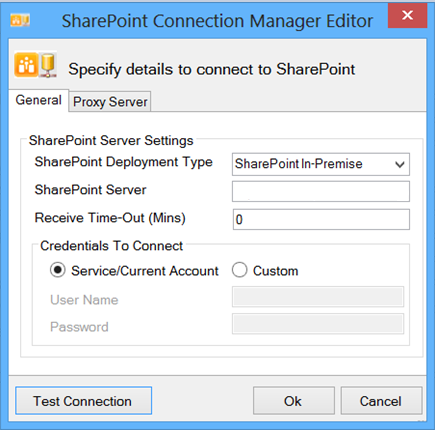
**Figure 10 - Adding a connection manager to the package - 1**

1. In the **Add SSIS Connection Manager** window, select **SharePoint CM** as connection manager type and then click **Add** button as shown below:



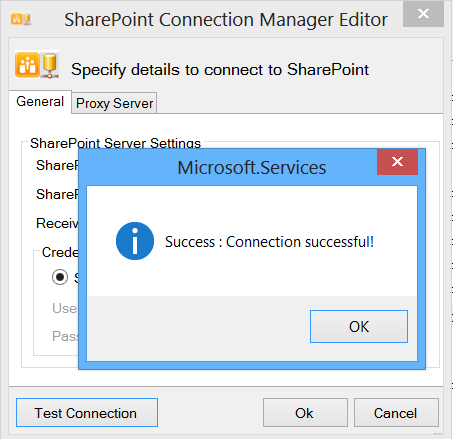
**Figure 11- Adding a connection manager to the package - 2**

1. Clicking on the **Add** button will add a SharePoint connection manager; next you can specify the SharePoint deployment type, SharePoint server name, authentication type, user name and password if custom authentication is chosen as shown below:



**Figure 12 - Connecting to SharePoint - 1**

1. You can click **Ok** button to save the changes for the connection manager or click **Cancel** button to cancel it. You can click **Test Connection** to test the connection to the SharePoint server the with the given configuration details as shown below:



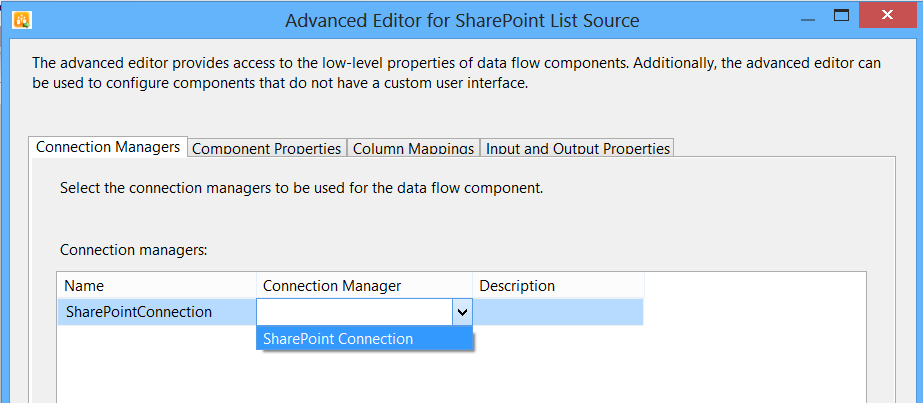
**Figure 13 - Connecting to SharePoint - 2**

1. Now based on your requirement, you can use **SharePoint List Source** to read data from SharePoint list or **SharePoint List Destination** to write data to SharePoint list. The following example shows a **SharePoint List Source** component dragged to the data flow task to read data from SharePoint list source.



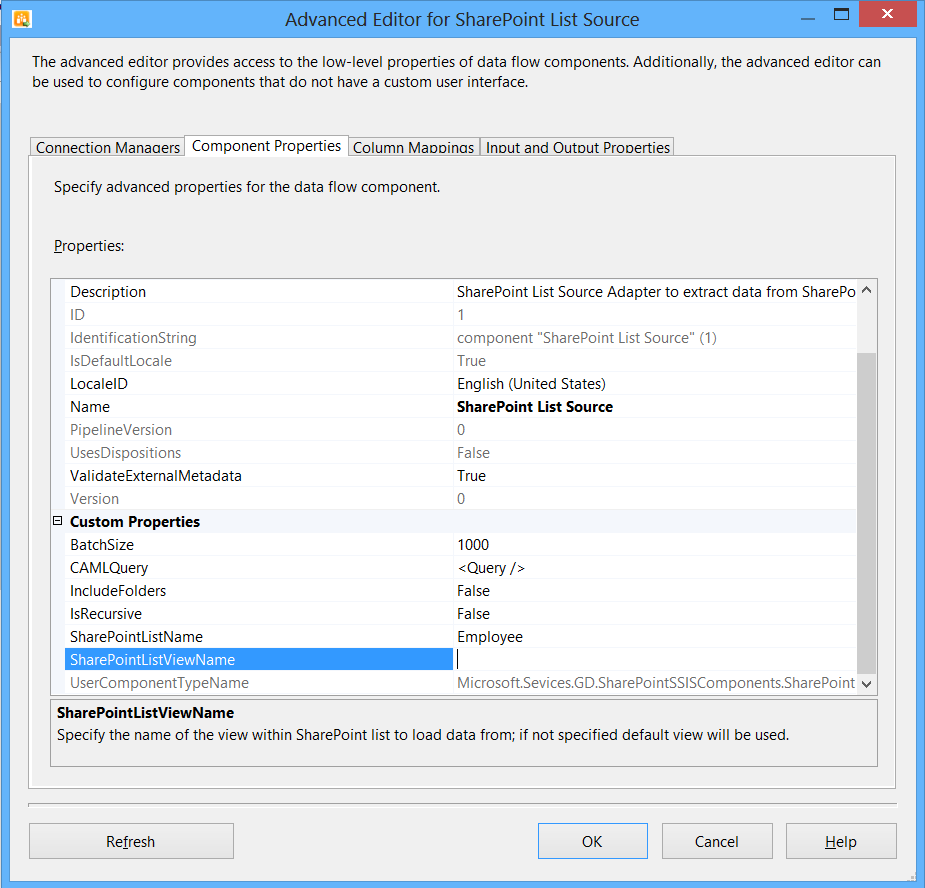
**Figure 14 - SharePoint Source Adapter**

1. Once a **SharePoint List Source** or **SharePoint List Destination** component is added to the package, you need to configure it next. Double click the component and an editor will open up to specify different parameters, for example you can see below the editor for SharePoint List Source component; on the **Connection Managers** tab, select the connection you created earlier as shown below:



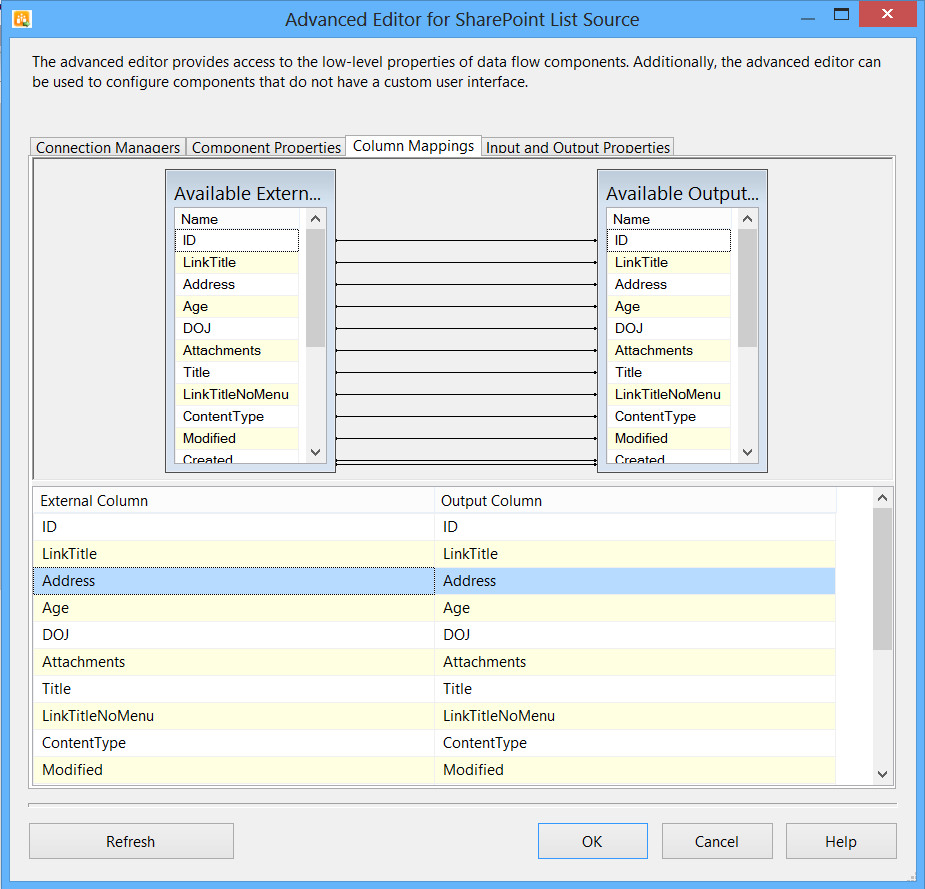
**Figure 15 - Source Adapter Advance Properties - 1**

1. On the **Component Properties** tab, you need to specify these properties:
   * **BatchSize** – This component is capable of pulling data from source in batches. By default, it pulls 1000 items from the list in each batch though you should configure it as per your requirement. If you have few columns or columns with smaller width you should configure it higher for better performance. Making it too large will make your component to fail as data from SharePoint to component moves into XML format and because of this bulky XML data coming from SharePoint, the component might fail.
   * **CAMLQuery** – By default, all the items from the list are pulled by the component. If you want to restrict it, you can use CAML query to restrict rows and pull only needed items. This property might be of your choice if want to pull incremental data from the SharePoint list.
   * **IncludeFolders** – By default, folders that are found in a list are not returned. You can change this setting to **True** if you want them to be returned.
   * **IsRecursive** – By default, only the list items at the top level are returned. If the list contains folders that contain other items, you can change this setting to **True** to pull data from both top-level folder as well as all children folders.
   * **SharePointListName** – The name of the SharePoint list whose items you want to pull.
   * **SharePointListViewName** – This property is not required and if left blank the default view of the specified list is used to pull the data from SharePoint list or else you can specify the view name based on the specified list using which data will be pulled.



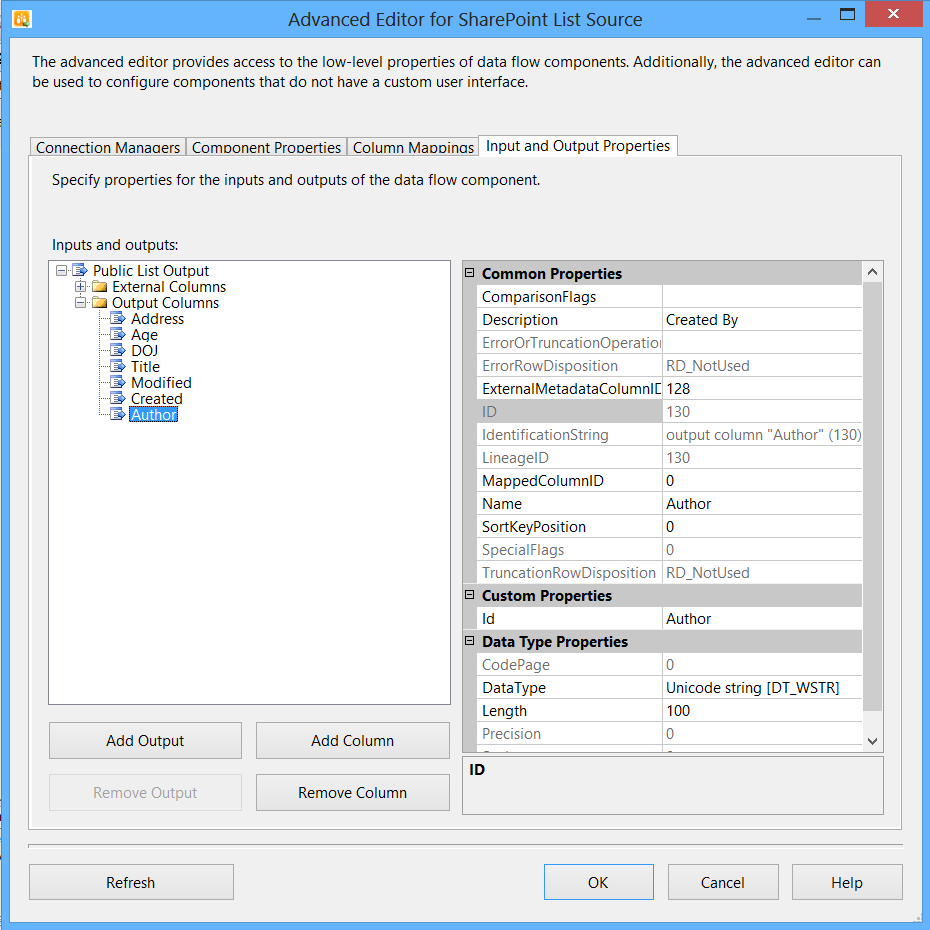
**Figure 16 - Source Adapter Advance Properties - 2**

1. On the **Column Mappings** tab, you can see the columns available at SharePoint list vs. column being output by the component; if you don’t see them or see them incorrectly click **Refresh** button to refresh it:



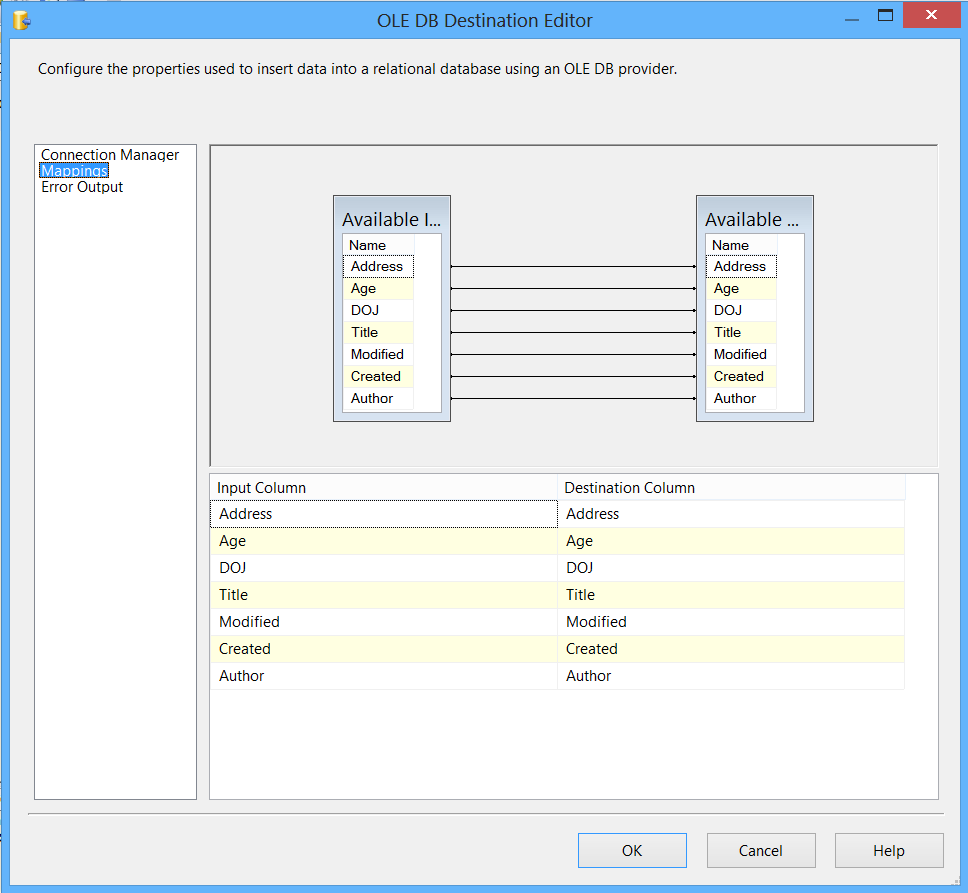
**Figure 17 - Source Adapter Advance Properties - 3**

1. On the **Input and Output Properties** tab, you can see the columns being input to the component as well as columns being output by the components; if you don’t want some of the source columns to enter into SSIS data flow pipeline and you can remove them from here:



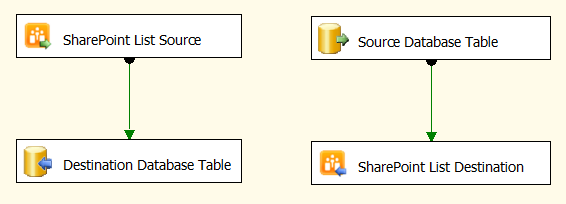
**Figure 18 - Source Adapter Advance Properties - 4**

1. In this example, data from SharePoint list is loaded in to a database table and hence an OLEDB destination is used. The example also shows column mappings from source to destination; it’s very much like what you do with other types of source components:



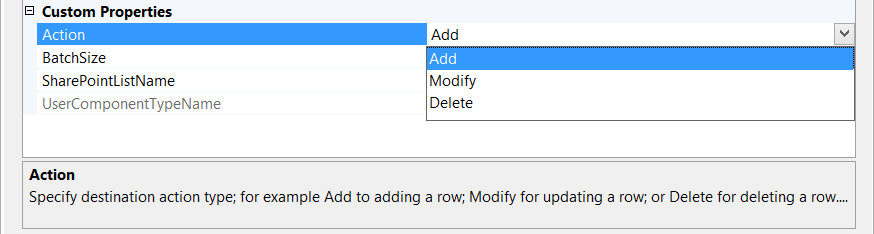
**Figure 19 - Destination adapter mapping**

1. In this example below, data from SharePoint list is loaded in to a database table and an OLEDB destination is used. Likewise data from database table is loaded to SharePoint list and hence SharePoint List Destination component is used.



**Figure 20 - Source and Destination components example**

1. When you are using destination adaptor or component, you need to specify the action type which will be performed by the component on the destination; for example **Add** to adding a row; **Modify** for updating a row; or **Delete** for deleting a row. *Note - Modify and Delete require ID Column with the SharePoint ID.*



**Figure 21 - Destination Adapter - Action to take when writing to destination**

# Conclusion

SQL Server Integration Services (SSIS) - a component of SQL Server - is an extensible platform for building high performance data integration (ETL - Extraction, Transformation and Loading) and workflow solutions. The extensibility part of SSIS comes from the fact that, the developers have ability to create custom components if the in-built components of it do not suffice the specific need. The platform makes it very simple to embed your own code into control flow or data flow pipeline.

This article demonstrated how to create custom SSIS components (connection manager, source component and destination component) which will integrate data to/from SharePoint and work like any other native components of SSIS utilizing the SSIS pipeline for better performance. Not only this, the step by step guide, idea and process, presented in this article, can be leveraged for creating these similar types of SSIS components to connect to any other external sources (like Microsoft Dynamic CRM etc.) for data integration for which no inbuilt components are available.

**For more information:**

* [Developing Custom Objects for Integration Services](http://msdn.microsoft.com/en-us/library/ms403361.aspx) (<http://msdn.microsoft.com/en-us/library/ms403361.aspx>)
* [Extending Packages with Custom Objects](http://msdn.microsoft.com/en-us/library/ms345161.aspx) (<http://msdn.microsoft.com/en-us/library/ms345161.aspx>)
* [Developing a Custom Data Flow Component](http://msdn.microsoft.com/en-us/library/ms136078.aspx) (<http://msdn.microsoft.com/en-us/library/ms136078.aspx>)
* [Creating a Custom Data Flow Component](http://msdn.microsoft.com/en-us/library/ms135993.aspx) (<http://msdn.microsoft.com/en-us/library/ms135993.aspx>)

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

* Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

[Send feedback](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20[Extending%20SSIS%20Capabilities%20by%20Creating%20Custom%20SSIS%20Components%20for%20SharePoint])