Paper SAS063-2014

How to Create a SAS® Enterprise Guide® Custom Task to Get Data from a SharePoint List into a SAS® Data Set

William Reid, SAS Institute Inc.

ABSTRACT

Do you have data in SharePoint that you would like to run analysis on with SAS? This workshop teaches you how to create a custom task in SAS Enterprise Guide in order to find, retrieve, and format that data into a SAS data set for use in your SAS programs.

INTRODUCTION

Do you have data in a SharePoint list that you need to do analysis on? Is that data updated periodically? Would you like a point-and-click way to get that data from the list into a SAS data set? If so, keep reading. This paper will explain how to accomplish this task as well as how to get a sample SAS Enterprise Guide Custom Task with full source code.

INSTALL AND SET UP THE REQUIRED SOFTWARE

Install Microsoft Visual Studio Express 2013 for Windows Desktop

Any licensed copy of Microsoft Visual Studio will work. If you are willing to use the sample task "as is," you can skip this step. To get a free copy of Visual Studio Express 2013, the version used for this paper, go to http://www.visualstudio.com/downloads/download-visual-studio-vs.

Install SAS Enterprise Guide

Version 6.1 (64-bit) was used for this paper. The sample task discussed in this paper is compatible with versions as far back as 4.3.

SET UP THE DEVELOPMENT ENVIRONMENT

COPY THE SHAREPOINT CLIENT-SIDE LIBRARIES

The DLLs are located on a Windows server machine that has SharePoint 2010 installed in the following directory: \Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI. SharePoint 2013 is similar except that it is in \15\ISAPI.

1. Microsoft.SharePoint.Client.Runtime.dll
2. Microsoft.SharePoint.Clent.dll

I recommend copying these files directly to your Custom directory and referencing them from there. The Custom directory is where your task will be copied to and where SAS Enterprise Guide looks for all custom tasks. It can be found at C:\Users\<userid>\AppData\Roaming\SAS\EnterpriseGuide\6.1\Custom, where <userid> is the userID that you used to log on to the computer. You will need to supply these two DLLs with your custom task to any users who will be using your task. If the Custom directory does not exist, you will have to create it via a command prompt or other method.

CREATE A NEW SOLUTION

Open Visual Studio and create a new Visual C# Class Library.

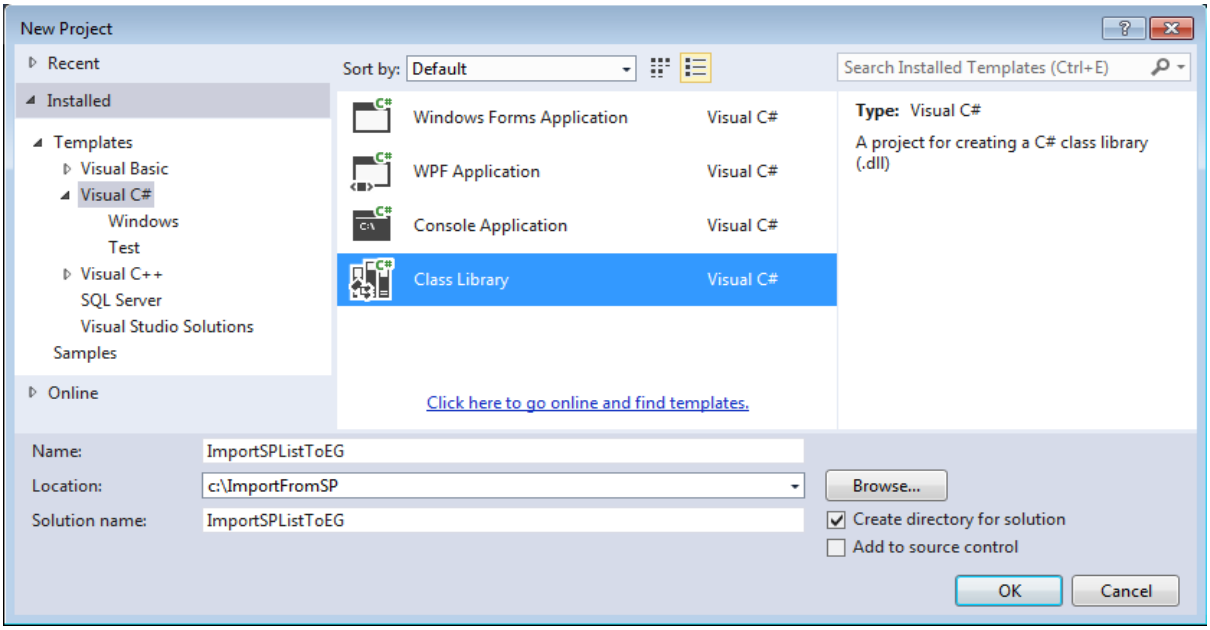

Figure 1. New Project Window

Add the SAS Enterprise Guide Folder to the Visual Studio Reference Paths

The folder is located in \SASHome\SASEnterpriseGuide\6.1.

Bring up the properties for the new project.

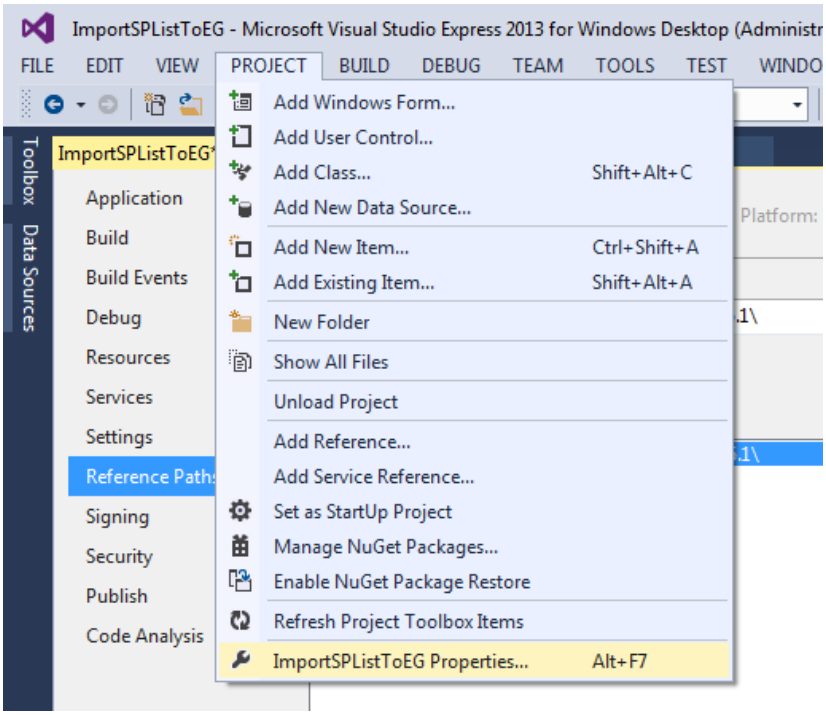

Figure 2. Project Properties Menu Item

Select **Reference Paths**, navigate to the SAS Enterprise Guide Folder, and then add the folder.

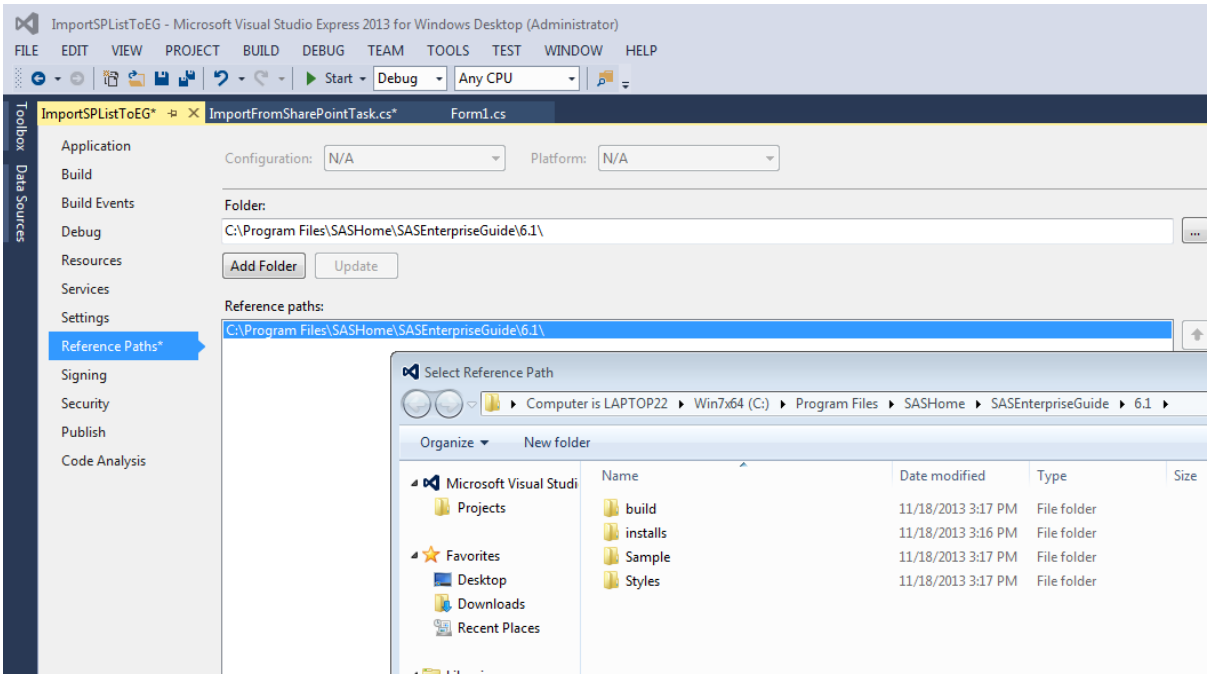

Figure 3. Select Reference Path Window

Add References to the Required DLLs

Open the Solution Explorer and expand the **References** node.

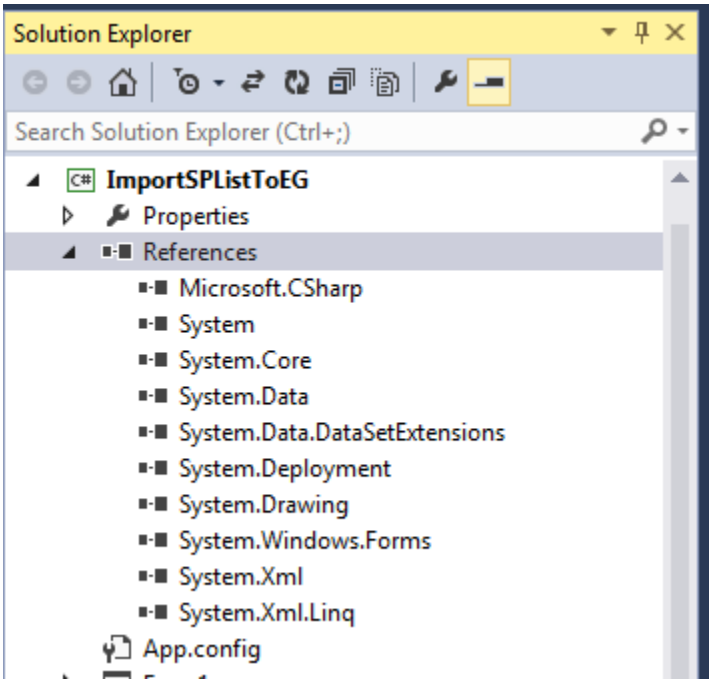

Figure 4. Solution Explorer Task Pane

Right-click the **References** node and select **Add Reference**.

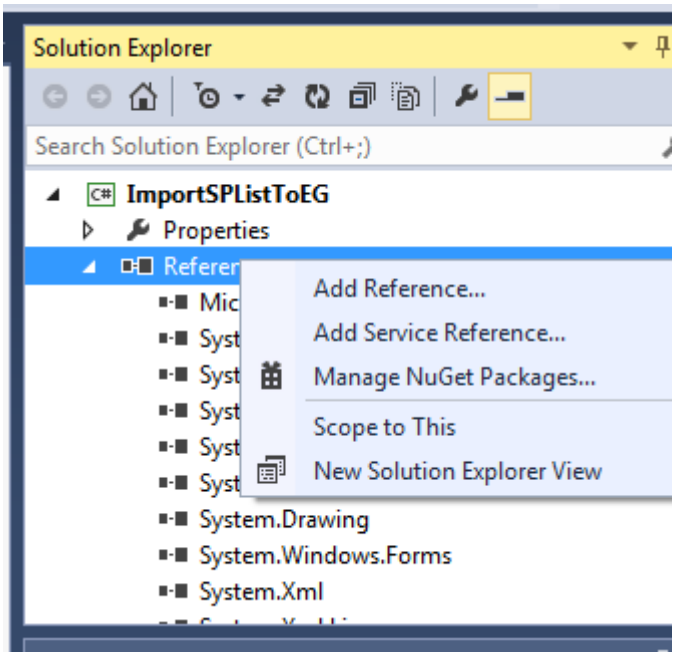

Figure 5. Add Reference Menu Item

Navigate to the Custom directory where you copied the two SharePoint DLLs and select them.

Navigate to the SAS Enterprise Guide folder location \SASHome\SASEnterpriseGuide\6.1. Then select **SAS.Shared.AddIns.dll** and **SAS.Tasks.Toolkit.dll**.

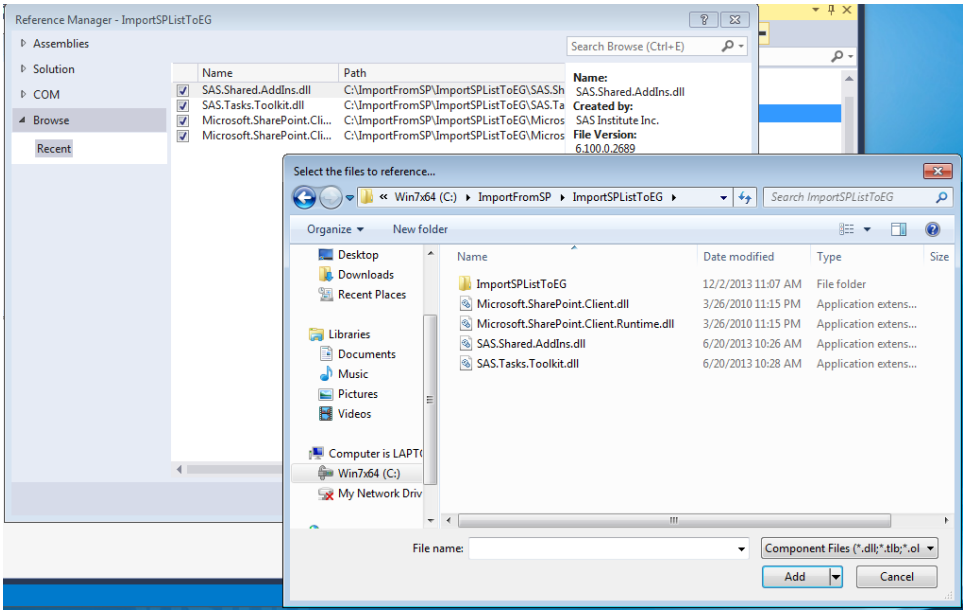

Figure 6. Select Files Window

The four DLLs should now show up in your list of references.

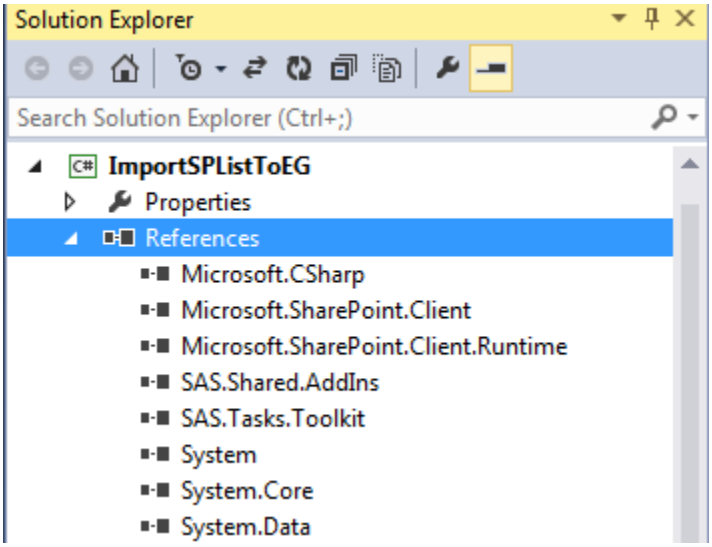

Figure 7. Project References

In the same manner, add any other DLLs that you might need now or as you develop.

Set the Copy Local Setting to False

Right-click the **SAS.Shared.AddIns** module in the Solution Explorer under References. Then click **Properties**. Locate the Copy Local setting and set to **False**. Repeat for the other three DLLs.

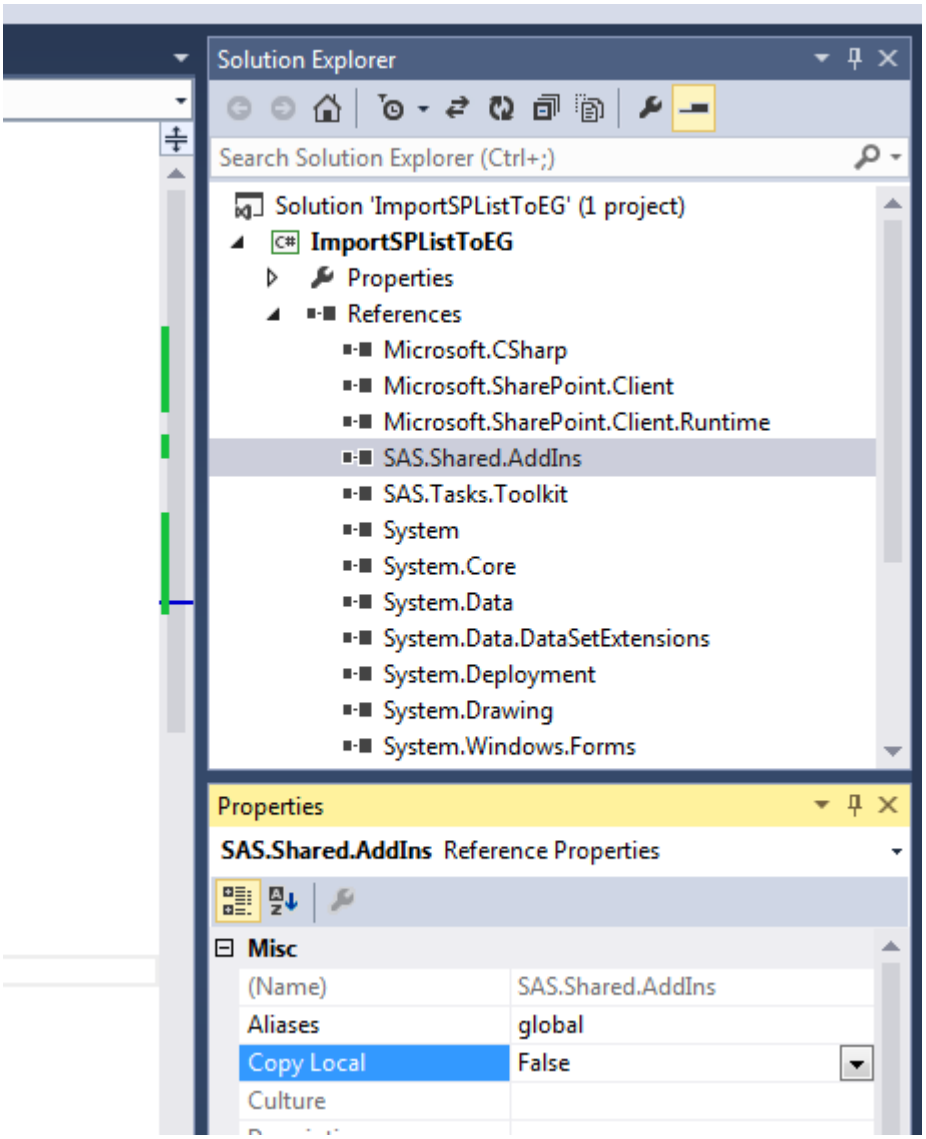

Figure 8. Reference Properties Task Pane

A Note about the .NET Framework Run-time Version

Because your DLL will be running as part of SAS Enterprise Guide, you will need to target your DLL to the framework version that SAS Enterprise Guide will be loading or to an earlier version. SAS Enterprise Guide 6.1 and 6.1M1 both target .NET framework 4.0. But SAS Enterprise Guide 4.3 targets .NET framework 3.5. So if you plan to support older versions of SAS Enterprise Guide, it would be wise to use the above-mentioned DLLs from the earliest version you need to support and the .NET framework version that the SAS Enterprise Guide version runs on. The sample application that will be provided was built with SAS Enterprise Guide 4.3 DLLs. It targets .NET framework 3.5.

To modify this setting, select **Project ▶ ImportSPListToEG Properties** or your project name Properties and select the **Application** setting.

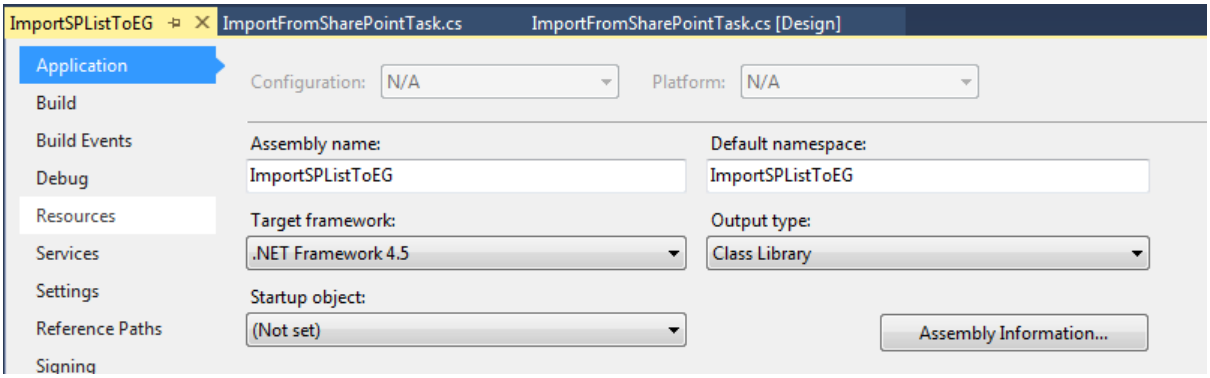

Figure 9. Application Properties

SETTING UP FOR DEBUGGING

If you are lucky enough to have a licensed copy of Visual Studio, you can set up SAS Enterprise Guide to be the start-up application. If not, you can get around that by setting a post-build event command line option to automatically start SAS Enterprise Guide.

SET A POST-BUILD EVENT COMMAND

After your library builds successfully, you will want to test it out in a debug session via Visual Studio. You will need to copy your output DLL along with any referenced DLLs that are not already in your computer's LIBPATH or in the same directory as SAS Enterprise Guide. To do this, open the project properties and select the Build Events section. Add the command shown below.

In the following commands, the <custom> directory is %USERPROFILE%\AppData\Roaming\SAS\EnterpriseGuide\6.1\Custom\ImportSPListToEG.dll.

ImportSPListToEG.dll is the name of the output DLL from your project. Note: You might need to create the custom directory if this will be your first custom task on this computer.

```
Del "<custom>\ImportSPListToEG.dll"
Copy "c:\pathtoyourproject\bin\debug\ImportSPListToEG.dll" "<custom>"
```

To start SAS Enterprise Guide, add this command:

```
Call "SASHome\SASEnterpriseGuide\6.1\SEGuide.exe"
```

NOTE: You might not want to start SAS Enterprise Guide automatically this way until you get a more finished product because each time you build, SAS Enterprise Guide will start.

ATTACH TO THE DEBUGGER

If you are using Visual Studio Express or if you like to manually start SAS Enterprise Guide, you will need to attach to the process before you can debug. Select **DEBUG ▶ Attach to Process**.

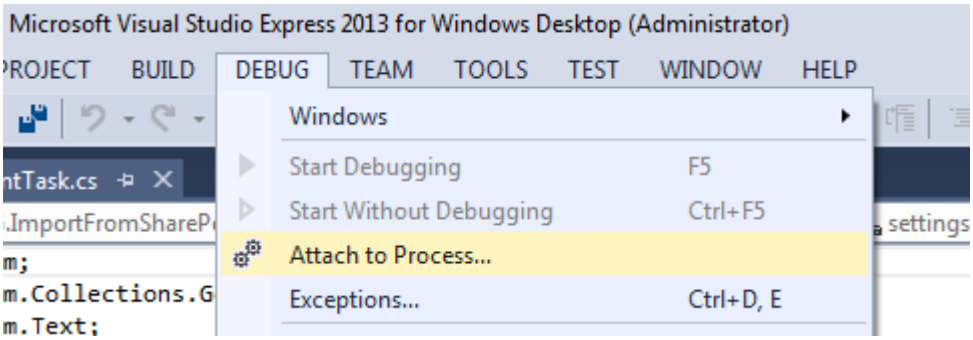

Figure 10. Attach to Process Debug Menu Item

Select **SEGuide.exe** from the list and click **Attach**.

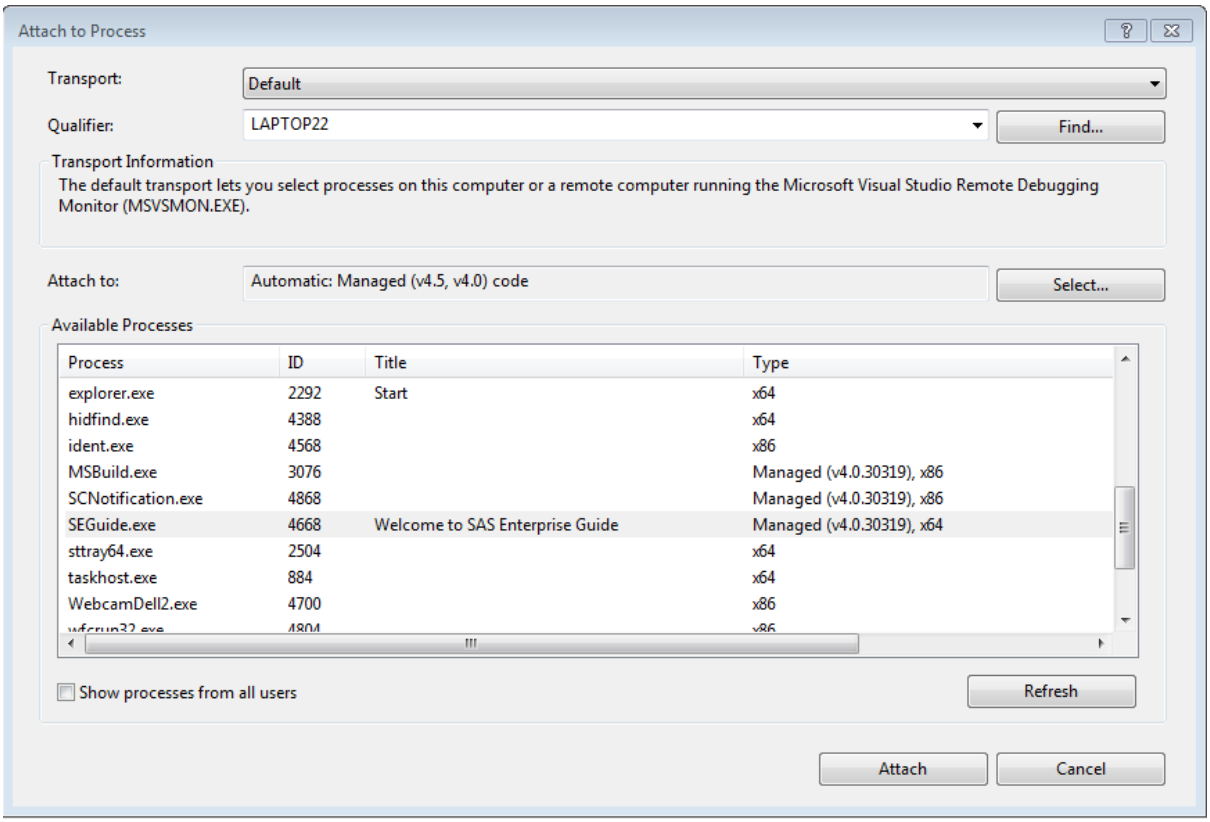

Figure 11. Attach to Process Window

NOTE: It is important to close SAS Enterprise Guide after your debug session or your post-build event commands will fail. Now you can begin to test and debug your custom task.

CODE THE UI AND SAS CODE GENERATION

The Task UI and code generation code is left as an exercise for the reader. A complete sample SAS Enterprise Guide custom task with source code is available at http://support.sas.com/documentation/onlinedoc/guide/customtasks/samples/ImportSharepointListTask.zip.

For more help writing an SAS Enterprise Guide Custom task, see the sections “Creating Custom Add-In Tasks for SAS Enterprise Guide” and “Introducing: Custom Tasks for SAS Enterprise Guide Using Microsoft .NET” by Chris Hemedinger at http://go.sas.com/customtasksapi.

You can also consult Chris’ blog (http://blogs.sas.com/content/sasdummy). He frequently shares examples and tips.

GENERATING SAS DATA STEP CODE

There are many ways to use your SharePoint data within SAS Enterprise Guide. I find that creating a data set on your server is the most efficient means. SharePoint 2010 allows 30,000,000 items in a list, but limits views to a threshold of 5,000. If you plan to create a data set from a list with more than 10,000 items, I would suggest using a different method to create the data set. As you guide yourself or your users through your task to gather information from SharePoint to create a data set, you will need to collect and store a certain minimum amount of information.

OUTPUT DATA

Where the data set that you create will be stored is one of the more import pieces of information. SAS Enterprise Guide will communicate with your task and ask it how many output data sets it is going to create and ask for the location information. In your SASTask derived class, you override the OutputDataCount and the OutputDataDescriptorList properties to supply that information. SAS Enterprise Guide will take care of prepending the

code to drop that data set if it already exists.

INPUT DATA

Your task will be using a SharePoint list as input data. SAS Enterprise Guide does not know how to show a SharePoint list visually. So your task will be reporting that it does not have input data and will have to keep track of that internally. To connect to SharePoint and get the required information, it will need to store the URL to the SharePoint server and the Unique ID for the List. If you plan to take the list with all the columns "as is," then that is all you need. But if you want to select only certain columns or use a LINQ query to restrict the amount of data returned, you will need to save information that will enable you to re-create your data set.

CONNECTING TO SHAREPOINT

Use the Microsoft.SharePoint.Client.ClientContext class to return information about a SharePoint site. You create the ClientContext with the URL for the site that you want the information about. Note that the following code has been edited for brevity and is not meant to be in one method or property:

```
using ( ClientContext SPContext = new
ClientContext(http://site.company.com/Analysis)
{
```

We use a "using" block to guarantee that the ClientContext will be disposed of when we exit the loop. After you have the ClientContext, you can get information from the site, such as the list of lists, which you can then show to the user.

```
Web webSite = SPContext.Web;
SPContext.Load(webSite);
SPContext.ExecuteQuery();
```

The above is standard procedure for retrieving the information. First, you load the object into the context, and then you execute the query to get it.

```
Microsoft.SharePoint.Client.ClientListCollection lists = website.Lists;
SPContext.Load(lists);
SPContext.ExecuteQuery();
```

After you find the list that you want to work with, the next time you can get the information for that list by using its ID.

```
Microsoft.SharePoint.Client.List list = webSite.Lists.GetById("list_guid");
SPContext.Load(list);
SPContext.Load(list.Fields);
SPContext.ExecuteQuery();
```

That will also load the FieldCollection object for that list at the same time.

Each Microsoft.SharePoint.Client.Field will have a FieldType associated with it. You examine the FieldTypeKind property to determine whether you want to include this in your output data set.

```
foreach(Field fld in list.Fields)
{
  SPContext.Load(fld);
  SPContext.ExecuteQuery();
  Switch(fld.FieldTypeKind)
  {
    Case FieldType.Integer:
       //do stuff
      Break;
     Case FieldType.Text:
       //do text stuff
       Break;
       …
    }
}
```

After the user has selected all the fields that they want to include, you can get the values for those fields by using the Microsoft.SharePoint.Client.CamlQuery class.

```
StringBuilder sbQuery = new StringBuilder();
int cols = columns.Count;
if (cols > 0)
{
  sbQuery.AppendFormat("<View>{0}", Environment.NewLine);
  sbQuery.AppendFormat("    <Query>{0}", Environment.NewLine);
  sbQuery.AppendFormat("      <OrderBy>{0}", Environment.NewLine);
  sbQuery.AppendFormat("
       <FieldRefName='{0}' /> {1}", columns[0].SPName, Environment.NewLine);
  sbQuery.AppendFormat("      </OrderBy>{0}", Environment.NewLine);
  sbQuery.AppendFormat("    </Query>{0}", Environment.NewLine);
  sbQuery.AppendFormat("    <ViewFields>{0}", Environment.NewLine);
  foreach (SPColumn curcol in columns)
  {
    sbQuery.AppendFormat("      <FieldRef Name='{0}'/>{1}", curcol.SPName,
Environment.NewLine);
  }
  sbQuery.AppendFormat("    </ViewFields>{0}", Environment.NewLine);
  sbQuery.AppendFormat("</View>{0}", Environment.NewLine);
  Microsoft.SharePoint.Client.CamlQuery query = new CamlQuery();
  query.ViewXml = sbQuery.ToString();
  SPContext.Load(list);
  SPContext.ExecuteQuery();
  Microsoft.SharePoint.Client.ListItemCollection items = list.GetItems(query);
  SPContext.Load(items);
  SPContext.ExecuteQuery();
  int rows = items.Count;
  rc = new string[rows, cols];
  for (int i = 0; i < rows; i++)
  {
    Microsoft.SharePoint.Client.ListItem li = items[i];
    for (int j = 0; j < cols; j++)
    {
      string spValue = (".");
      if (li[ColName] != null)
          spValue = GetStringValueFromField(li[curcol.SPName]);
      rc[i, j] = spValue;
      }
     }
   }
```

Because the ListItem[“colname”] returns an object to you, you must get the type from the object and then get the string value based on the type. The sample task has all of the code to accomplish this.

TRANSLATING YOUR SHAREPOINT DATA INTO A SAS DATA STEP

Once again, I will refer you to the sample task that is available at http://support.sas.com/documentation/onlinedoc/guide/customtasks/samples/ImportSharepointListTask.zip for full functionality. First, we start building our DATA step code with the basics:

```
StringBuilder sbCode = new StringBuilder();
```

```
//start data statement
sbCode.AppendFormat("DATA {0}.{1};{2}", outputLibrary, outputDataName,
NewLine);
//SPColumn is a wrapper class that contains the SharePoint column information
IList<SPColumn> codeCols = GetColumnsForCode();
//get format statement returns a FORMAT block to specify any requested formats
string fmtStatment = GetFormatStatement(codeCols);
if (!string.IsNullOrEmpty(fmtStatment))
  sbCode.Append(fmtStatment);
// now we add the actual data
sbCode.AppendFormat("CARDS4;{0}", Environment.NewLine);
string[,] data = GetCardsData();
int upperCol = data.GetUpperBound(1);
for (int row = 0; row <= data.GetUpperBound(0); row++)
{
  for (int col = 0; col < upperCol; col++)
  {
   sbCode.AppendFormat("{0}, ", data[row, col]);
  }
  sbCode.AppendFormat("{0}{1}", data[row, upperCol], Environment.NewLine);
}
//append end of everything line
sbCode.AppendFormat("{0}{1}", ";;;;", Environment.NewLine);
```

The GetCardsData() is a very important method. Its job is to take the SharePoint data and convert it into a string representation that can be put into the DATA step code and then populated into a data set correctly.

THINGS TO LOOK OUT FOR

Time and DateTime Values

One of the gotchas involved with translating SharePoint list data into SAS data is date values. SAS uses 01/Jan/1960 for a reference date, but .net uses 01/Jan/0001. To make it even more interesting, SAS has a Date and a separate DateTime, but .net uses only a System.DateTime class. The following constants and methods will ease the conversion.

```
// Number of days between SAS reference (01JAN1960) and DotNet reference
(01JAN0001)
private const int DaysBetweenSASandDotNetReference = 715509;
// Number of seconds per day
private const int SecondsPerDay = 86400;
public static double DotNetToSASDate(DateTime dotNetDateTime)
{
  // Since SAS doesn't consider the years 4000 and 8000 leap years, but .NET
  // does, we have to take these extra leap days into account if the date is
  // greater
  // than or equal to 2/29/4000 or 2/29/8000. A rare case, but it does happen.
  int daysBetweenSASandDotNetReference = DaysBetweenSASandDotNetReference;
  if (dotNetDateTime >= new DateTime(8000, 2, 29))
    daysBetweenSASandDotNetReference += 2;
  else if (dotNetDateTime >= new DateTime(4000, 2, 29))
    daysBetweenSASandDotNetReference++;

  // result is number of days since 01JAN0001, according to .NET
  double days = (double)dotNetDateTime.Ticks / TimeSpan.TicksPerDay;
  // result is number of days since 01JAN1960, according to SAS
  days -= daysBetweenSASandDotNetReference;
  return days;
}

public static double DotNetToSASDatetime(DateTime dotNetDateTime)
{
  double sasDatetime = DotNetToSASDate(dotNetDateTime.Date) * SecondsPerDay;
```

```
            double partDayInSeconds = DotNetToSASTime(dotNetDateTime.TimeOfDay);
            return sasDatetime + partDayInSeconds;
        }

        public static double DotNetToSASTime(TimeSpan dotNetTime)
        {
            return (double)dotNetTime.Ticks / TimeSpan.TicksPerSecond;
}
```

If you have a SharePoint FieldType.DateTime value and are using these methods, you will want to set a SAS format for it to be DATETIME16.

Field Types

SAS really has only two types of data, numeric and character, but SharePoint defines many. Most of the SharePoint FiledType values should be treated as character values except for the following:

1. FieldType.Integer
2. FieldType.Number
3. FieldType.Counter
4. FieldType.Currency
5. FieldType.DateTime

ADDING YOUR TASK TO A SAS ENTERPRISE GUIDE PROJECT

You will find your task under the **Tools ▶ Add-In** menu item.

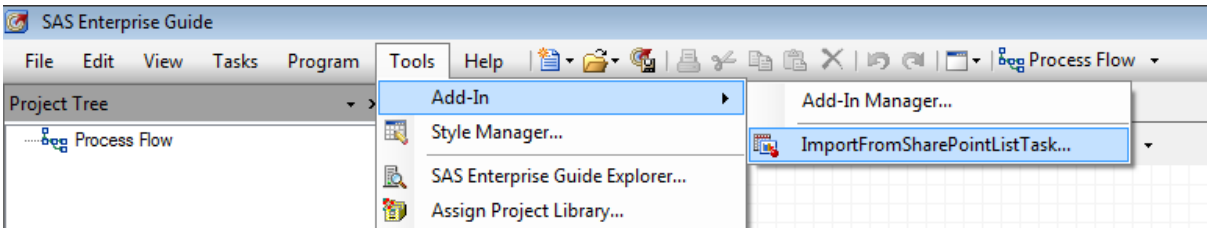

Figure 12. Add-In Tasks Menu

NOTE: The name ImportFromSharePointListTask is set in your code as the TaskName property in the class you derive from SAS.Tasks.Toolkit.SasTask.

When the task is running and you connect to the SharePoint site that contains the list that you are looking for, you select the list and columns that you want to include in your output.

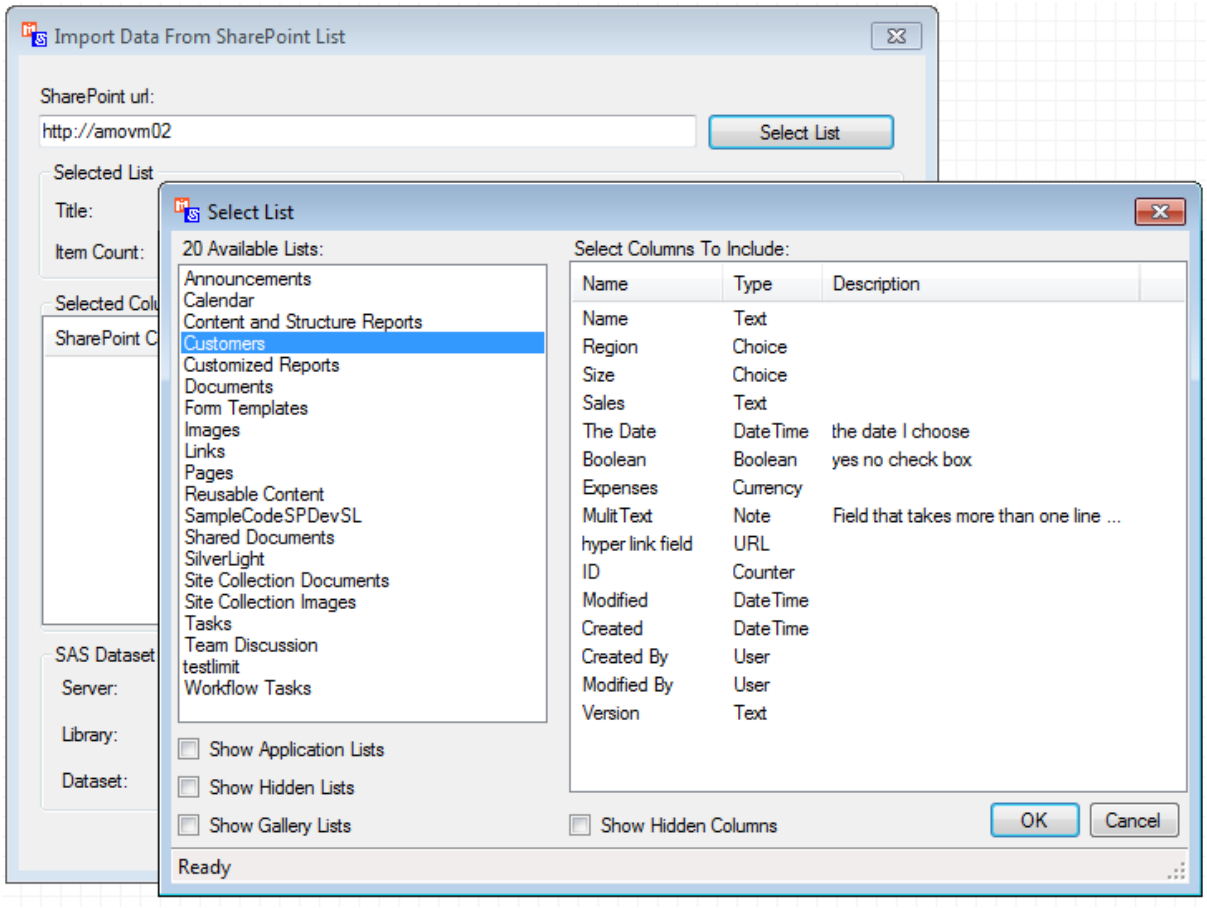

Figure 13. Select List Dialog Box

As you can see by the options in this dialog box, you can select from standard, Application, Hidden, and Gallery Lists. For columns, you can also ask to include Hidden columns. All these options are available for you to customize through the SharePoint Client Side Object Model. The list of lists will be security trimmed based on the ID that you used to log on to Windows. This is important to note because the person who runs this project or the dedicated ID that you set up to run this project will need to have appropriate access to the SharePoint list and data within the list. After you have selected the input list and columns, you can begin to set up your output data.

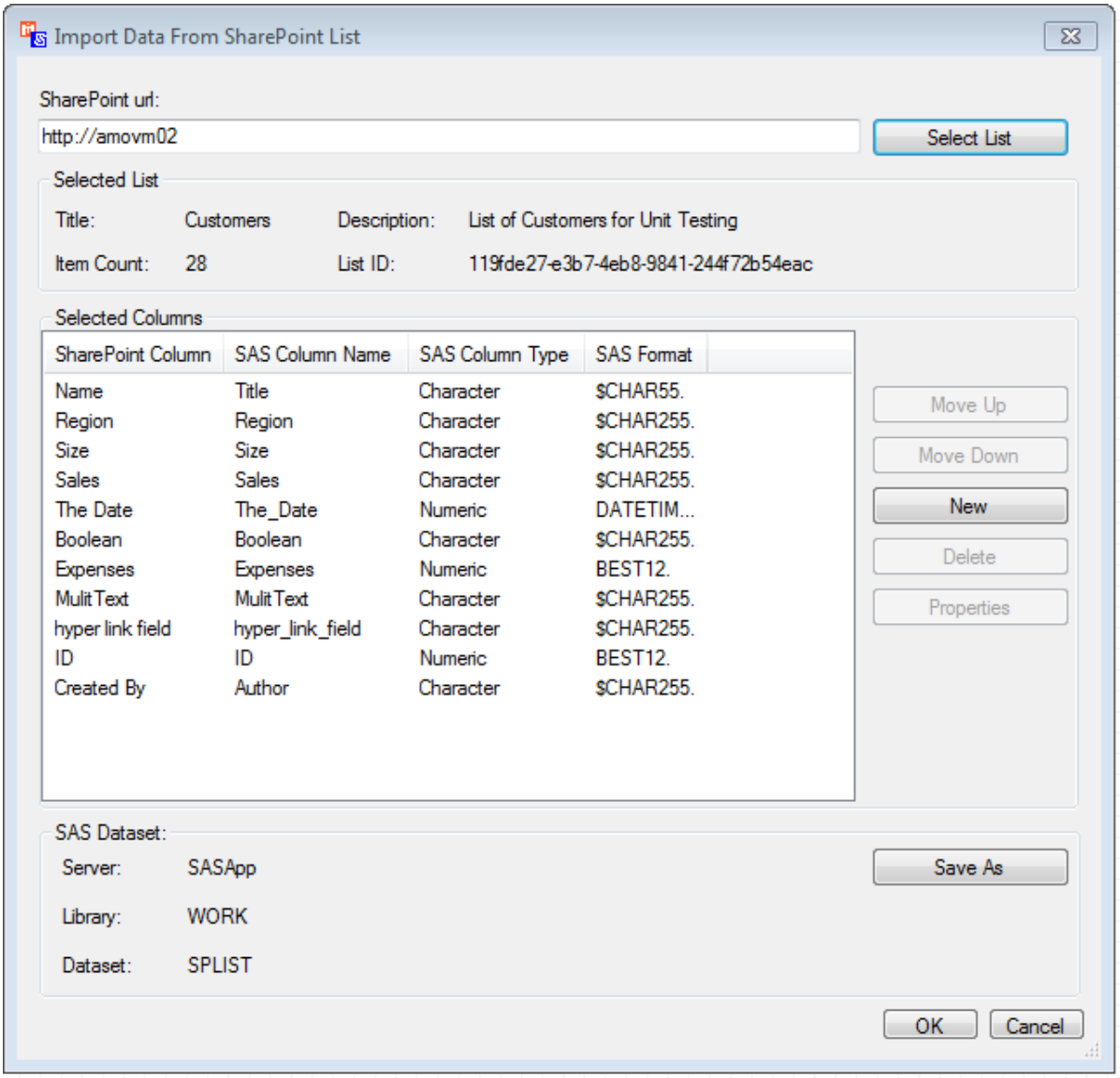

Figure 14. Sample Custom Task

If you don't agree with the default settings or order, you can change the order or the properties.

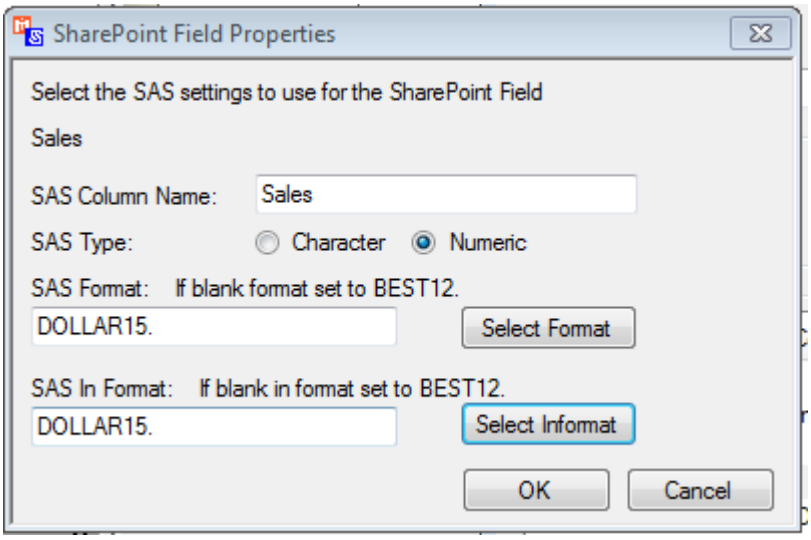

Figure 15. SharePoint Field Properties Dialog Box

Run your task and continue to refine the output by modifying the properties of your columns and rerunning until your output data is in the form that you want. After your data set is created, you can rerun the task to re-create your data set with the most up-to-date entries. You can then use all the power of SAS Enterprise Guide to further refine or run analysis on the data.

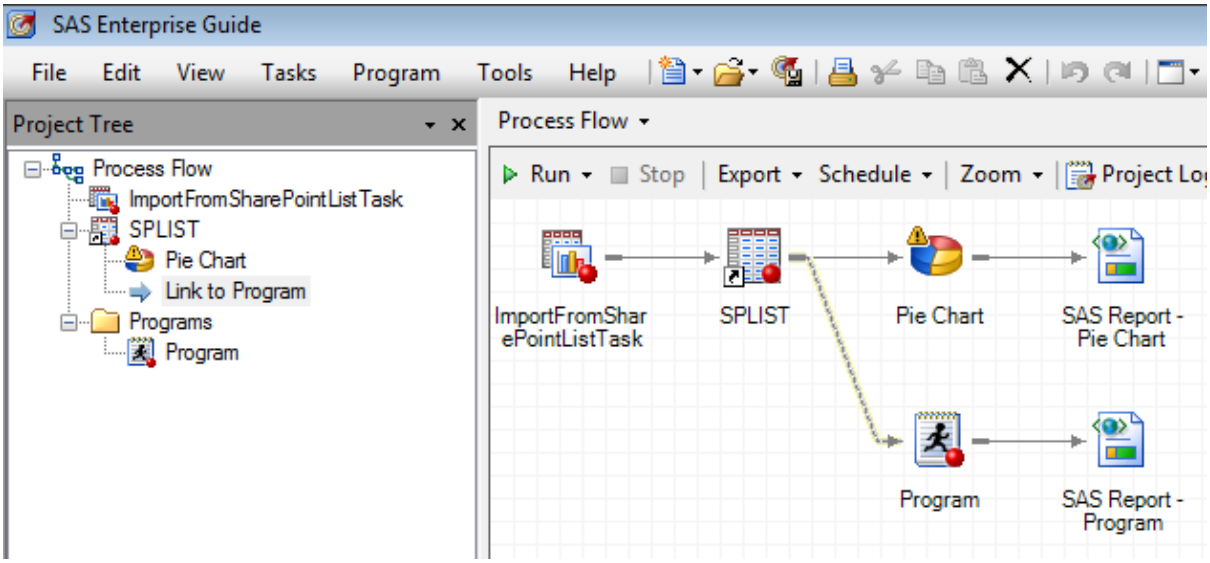

Figure 16.SAS Enterprise Guide Process Flow

At this point, you can run your process flow. The Pie Chart and Program will wait for the ImportFromSharePointListTask to run and retrieve the latest data from your SharePoint list and then run based on that data.

SCHEDULING YOUR PROJECT TO RUN AT A REPEATED INTERVAL

SAS Enterprise Guide has an integrated feature that enables you to schedule your project or process flow to run with the Windows Task Scheduler. Right-click on the process flow and click **Schedule Process Flow**.

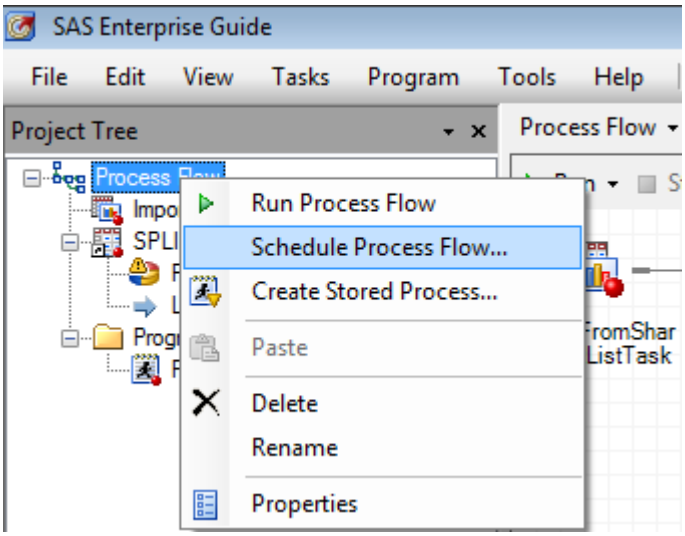

Figure 17. Schedule Process Flow Menu Item

This will create a VBScript file that can be used by the Task Scheduler to run your SAS Enterprise Guide project at a specified interval. You can refer to your operating system help for assistance in navigating the Task Scheduler. There are many options to choose from. So keep in mind that the computer that is going to run this will have to be available at the time you choose to run it.

The project will run under a specified ID, which will connect to SharePoint. The password will need to be stored to allow the process to connect as that userID. If you choose to set up a dedicated ID or use an existing one, you will need to remember that if the password of that ID changes, you will need to change the settings on the scheduled project as well. The task will run a VBScript file that is created and stored on the hard drive of the computer. That VBScript file has a line that loads your project. This means that if you move your project to a new location, you must update the script or reschedule the process flow. By default, the VBScript file is saved in the same directory as your project, but it will reference your project by a complete path. If you save you project as c:\SPListImport\importfromsp.egp, the VBScript file includes the following:

```
prjName = "C:\SPListImport\importfromsp.epg"
```

Complete the task scheduler setup and return to your project. You will now have a Schedules project folder that contains your Schedule and VBScript file.

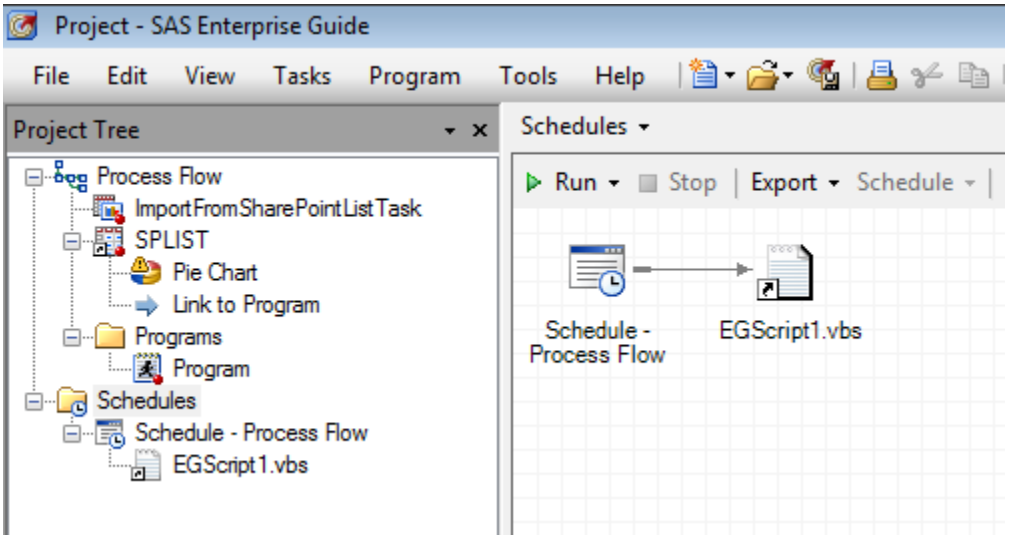

Figure 18. Schedules Folder

CONCLUSION

There is much more that you can do to integrate your SharePoint content into SAS Enterprise Guide and vice versa. This paper and sample task have only scratched the surface. Writing data back to SharePoint and publishing the results of processing, or whichever task (or tasks) you choose to run on the data to a SharePoint site or other website are just some examples. If you have a very large list in SharePoint, using this method might not be viable. But luckily SharePoint has Business Connectivity Services (BCS) that can be used for that case to export your list into a SAS readable database or text file, which can then be imported or used directly by SAS.

RECOMMENDED READING

- Hemedinger, Chris. 2012. *Custom Tasks for SAS® Enterprise Guide® Using Microsoft .NET.* SAS Institute Inc., Cary, NC. This book provides a guide for creating your own custom applications to host within SAS Enterprise Guide, which provides a number of APIs and services that make the job easier.
- Henderson, David R., and Sean Alexandre. *SAS Global Forum Paper 390-2009 Integrating SAS® Business Intelligence with Microsoft SharePoint.* This paper describes alternate methods for exporting SharePoint data for use with SAS.
- SharePoint forums. “Plan for Business Connectivity Services” at http://technet.microsoft.com/en-us/library/ee681491(v=office.14). This article is a good place to get started learning about BCS.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

William Reid
SAS Institute Inc.
100 SAS Campus Drive
Cary, NC, 27513
919-677-8000
Bill.Reid@sas.com
www.sas.com

TRADEMARK INFORMATION