

Be Your Own Task Master - Adding Custom Tasks to EG

Peter Eberhardt, Fernwood Consulting Group Inc. Toronto, ON

ABSTRACT

In Enterprise Guide, SAS® provides a ton of tasks to tickle travels into the world of data exploration and presentation. However, there are times when our fancy is not tickled just as it should. This paper will show you how to create custom EG tasks using C#.NET so you can have the interface your application deserves. This paper assumes you may have little knowledge about EG, but you should have at least a rudimentary knowledge of object oriented languages, preferably Microsoft's C#.

INTRODUCTION

In Enterprise Guide (EG), SAS has addressed the refrain 'But it is too hard to learn SAS'. Instead of the sometimes-daunting blank programme editor screen we have project based palette with drag and drop capability. Analysts can now focus on the problem at hand without having to know the myriad of PROCs and not always intuitive syntax. Instead of having to type line after line of SAS code then debug the inevitable errors, analysts can now address their problem by selecting and linking tasks from a long list of tasks provided by SAS (Figure 1). And once a task is selected, a wizard steps the analyst through the steps needed to successfully complete it (Figure 2)]

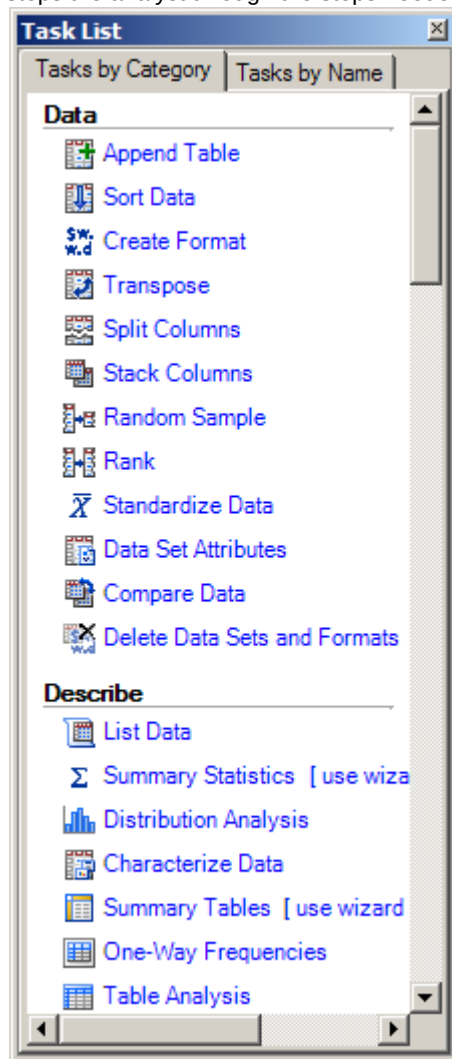


FIGURE 1 – THE SAS TASK LIST

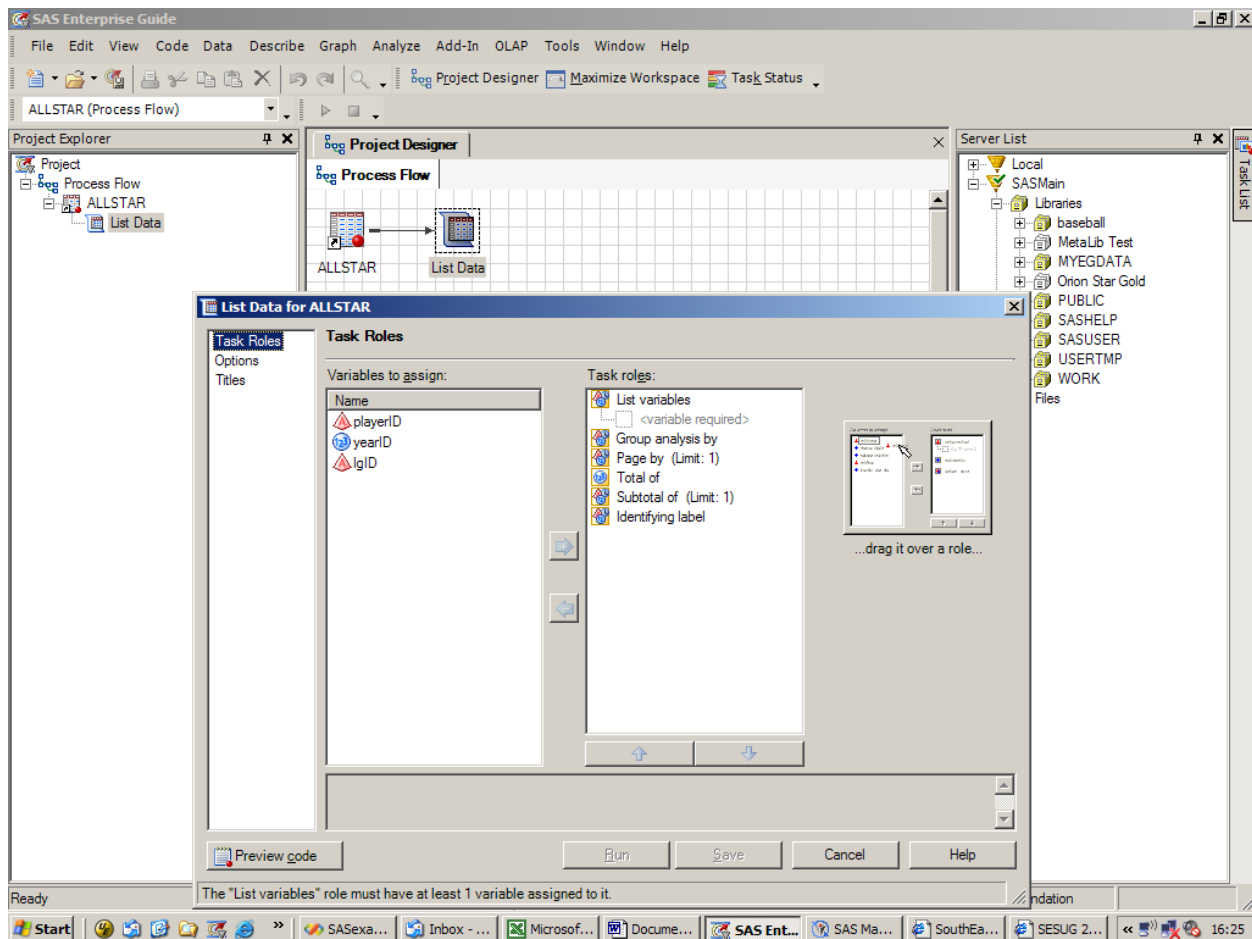


FIGURE 2 – A TASK WIZARD

All of this works very well – as long as the problem can be solved with one or more of the generic tasks that are provided by SAS. What do you do if you already have parameterized macro code written to help solve the problem. With EG you have a few options. First, you could make yourself very unpopular and tell the analysts they have to create a code node and invoke the macro by typing in all the parameters; not a very user friendly scenario. Next, you could create a SAS Stored Process to run the code; a SAS Stored Process is a centrally managed SAS programme that generates a flexible but limited interface to collect parameters. If your programme has simple input requirements, say enter a few scalar values with some constraints and/or select from a static list, then the Stored Process can be the answer. However, if you have a more complex set of requirements for your parameters, then creating your own custom task is the answer. This paper will outline the steps needed to create a custom task using Microsoft Visual Studio 2003 and C#.NET. It is assumed you are already familiar with C#.

SEPARATING THE INTERFACE FROM THE ENGINE

In order to separate the visual interface from the analytic engine you need three main things. First, you need to have a working engine – a thoroughly tested SAS programme; you will find it difficult to build an interface if you don't know what the engine needs. The code does not have to be fixed in stone; that is, your interface may generate a logically and syntactically correct programme rather than invoke parameterized code. In this case your tested code is really a sound and tested template that you use to generate code. Second, you will need a copy of Microsoft Visual Studio and C#.NET along with the .NET 1.1 framework. Visual Studio 2003 installs and used the version 1.1 framework. Later versions of Visual Studio use version 2.0 of the framework as well, but the SAS objects you will be accessing are version 1.1. Finally, you will need a copy of EG. The rest of the paper will focus on the 'housekeeping' chores needed to make an Addin work. It will not go into much detail about building and using C# forms, although they are indeed a critical part of the interface.

GETTING STARTED

Your first order of business should be to download the help files, the Addin template, and the sample project SAS has provided. The link to do this is:

<http://support.sas.com/documentation/onlinedoc/guide/customtasks/index.htm>

All of the downloads are in ZIP files; unzip them to an appropriate location on your computer.

INTERFACES REVISITED

In order to build your own custom Addin you need to follow some rules SAS has laid out; that is, there are certain things you must implement in your code. First, in order to access the internals of the Addin you need to add a reference to the SAS Addin classes. As the online docs point out, it is important you use the most current release of the classes. With the classes referenced, you have to implement a number of *interfaces*. An interface separates the name of a method – a name SAS expects, from the implementation of the method. The following three interfaces have to be implemented in order for you so successfully create an Addin

ISASTaskAddIn

Provides the basic mechanism for the host application to connect to the task and interact with it as part of the product.

ISASTaskDescription

Provides a set of metadata about the task to help the application surface it in menus, task lists, and decide under what conditions the task can operate.

ISASTaskDescription2 is an optional extension of this interface with a few additional properties for advanced features.

ISASTask

Provides information about the particular instance of the task so that the application can retrieve SAS code from it and gather information about expected results and output data.

ISASTask2 is an optional extension of this interface with a few additional properties and methods for advanced features.

In addition, there are two optional interfaces that can be implemented if you want to perform more sophisticated tasks:

ISASTaskSubmitSink

Notifies the Addin that SAS code submitted asynchronously has completed

ISASTaskDataSink

Notifies the Addin about changes to the input data

On the other side of the agreement, the following interfaces are implemented by SAS:

ISASTaskConsumer

Provides information about what the application supports and access to other supported interfaces. Also provides access to options specific to the particular instance of the Addin.

ISASTaskConsumer2 is an extension of this interface with a few additional methods for advanced features.

ISASTaskData, ISASTaskDataAccessor, ISASTaskDataColumn

Provides access to data services.

Luckily, once you download and unzip the documentation and templates from the SAS website, these interfaces are automatically made available when you create a SAS Addin project.

CREATING THE ADDIN PROJECT

With the Addin template downloaded from the SAS support site, you will see a new project type, SAS Enterprise Guide 4 Addin, on the Visual Studio New Project window; select this, fill in the appropriate project name and location and you are on your way. Figure 3 shows the screen creating a new SAS Enterprise Guide Addin project.

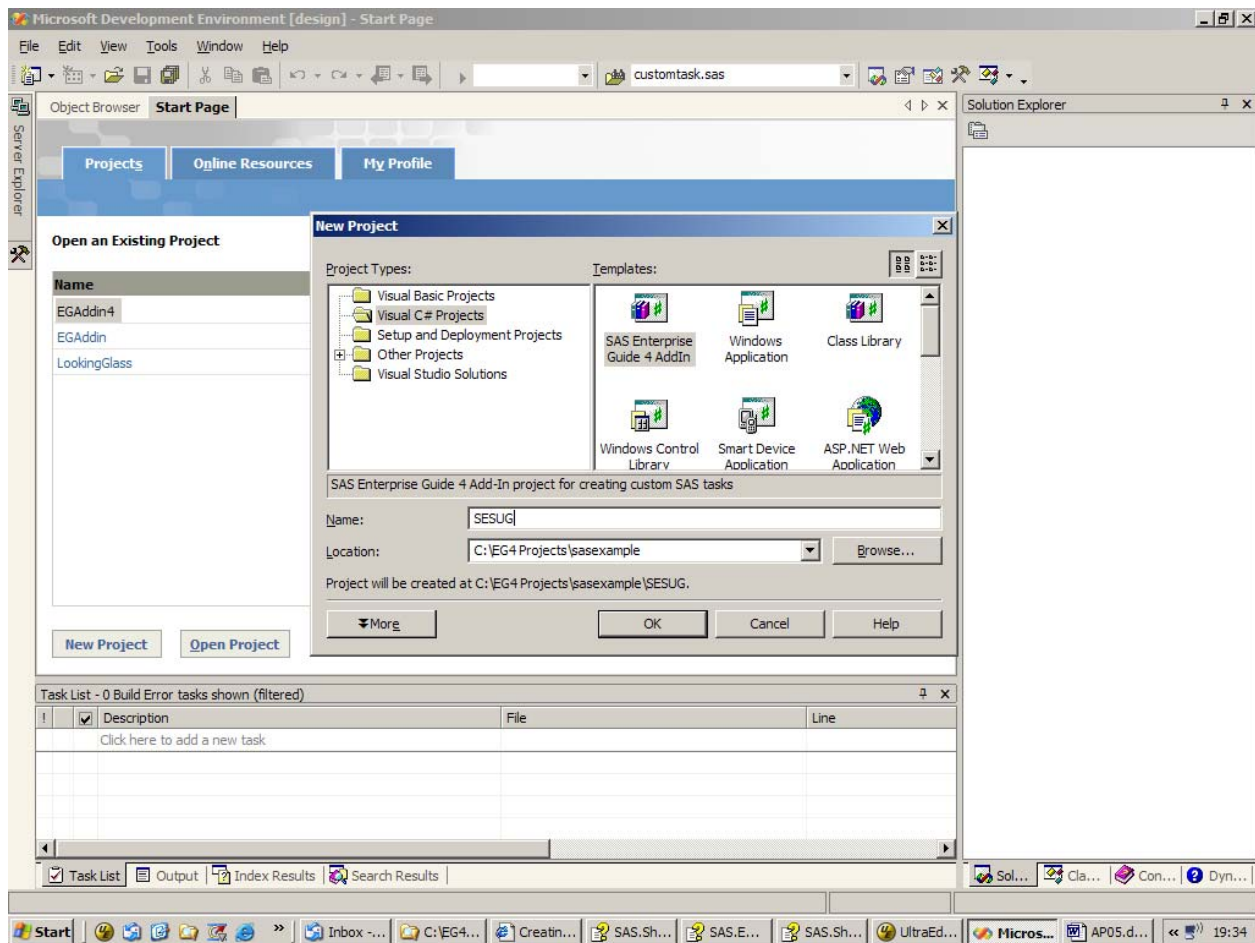


FIGURE 3 – CREATING A NEW SAS ENTERPRISE GUIDE ADDIN PROJECT

The template creates the bare bones of what you need to create an Addin, however there are a number of things you need to do. Before we start changing things, let's look at what the template has provided us with. Opening the Solution Explorer in Visual Studio you will see everything needed to create a custom Addin is there. Of course, the template has provided generic names for the form (CustomTaskForm1.cs) and the C# code (CustomeTask.cs), so our first tasks will be to rename these to something meaningful for the new Addin. This is a two staged process. First, in the Solution Explorer window rename the file. Then in an editor window do a global find and replace to change all instances of the generic name to your appropriate name. You have to do this for both the form file and the code file. Another important change you have to make is to one of the properties of the SAS.Shared.Addin reference; you should change the *Copy Local* property from *true* to *false*. This will avoid making a copy of the Addin functionality in your project; instead you will be using the 'global' copy. This will facilitate the distribution of your Addin, as well as making it easier to update projects should SAS.Shared.Addin be changed. To change the property, simply right click on SAS.Shared.Addin, select properties, and change Copy Local from true to false.

Finally, there are a number of descriptive text fields you have to change. These descriptions are used as task labels, descriptions etc. When first developing Addins I found it instructive to place the string name (e.g. sTaskDescription) as part of the string value until I had a clear idea of where the string values are surfaced. To change the placeholder strings search for the string 'PLACEHOLDER' in the project; this string appears multiple times so you will have to repeat the search until all are found. All of these code changes are addressing methods in the ISASTaskDescription interface.

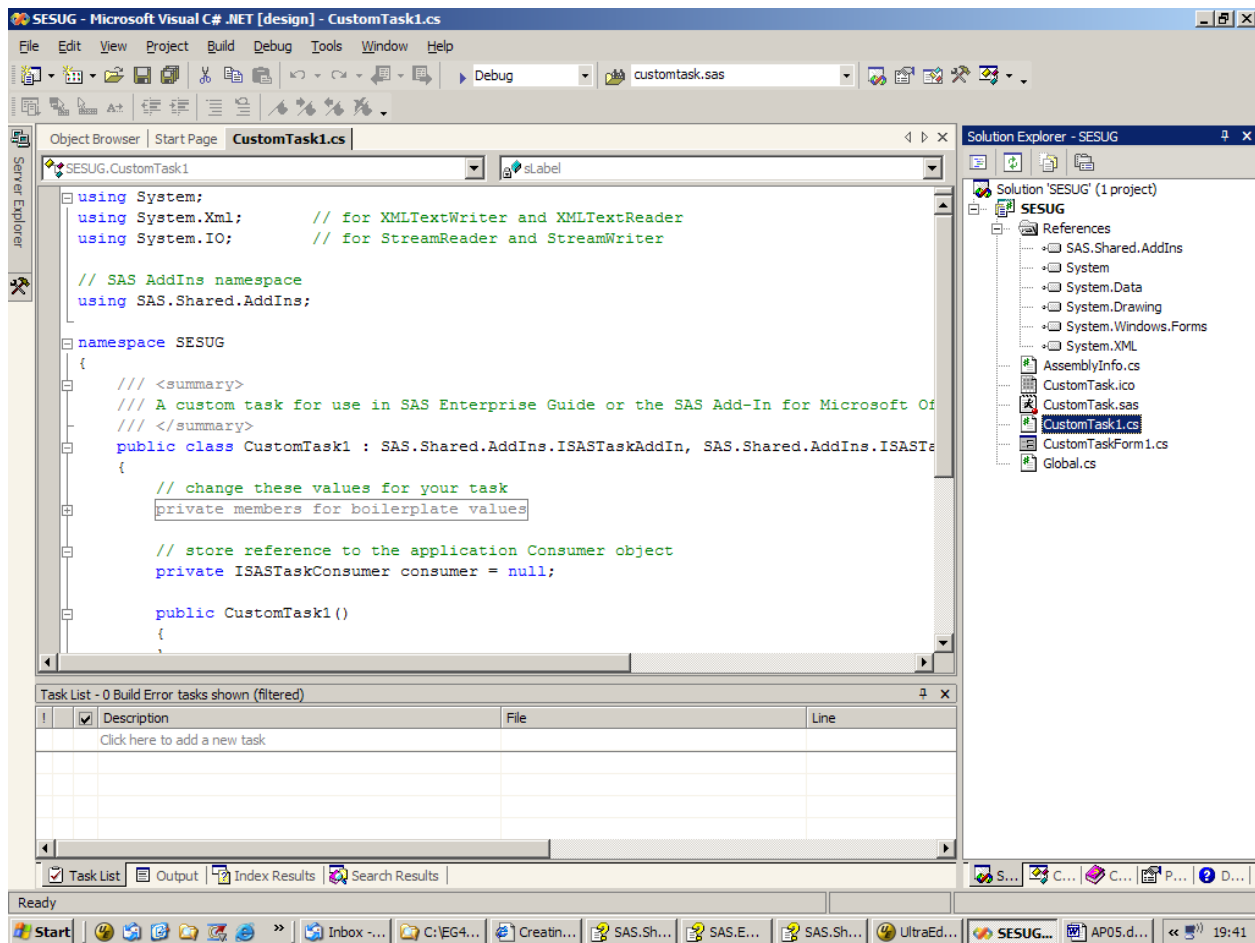


FIGURE 4 – THE CREATED PROJECT FILES

THE VISUAL INTERFACE

With all of the necessary housekeeping duties out of the way you can now work on the form that will be used to display the visual interface of the Addin. At this point you have access to a full suite of Windows form controls or any other 3rd party control packages you may have. One set of 3rd party controls you will have access to is the SAS controls package. By using the SAS controls you can ensure your custom Addins have the same sort of 'look and feel' as the SAS supplied tasks. In order to access the SAS controls you must first add then to your Visual Studio toolbox. To add them you select the *Add/Remove Toolbox Items* from the Visual Studio *Tools* menu. If the SAS controls are not on the list that is displayed, see Figure 5, click browse and navigate to *SAS.EG.Controls.dll*; by default it will be found in *C:\Program Files\SAS\Shared Files\BIClientTasks\4*.

Probably the most important piece of C# code you will have to write is the code behind the *SasCode* property.; *SasCode* is a public string property that returns valid code – remember the first of the three requirements noted above. All of the bits and pieces you need to build the code can come from your userform and/or other resources within the C# project.

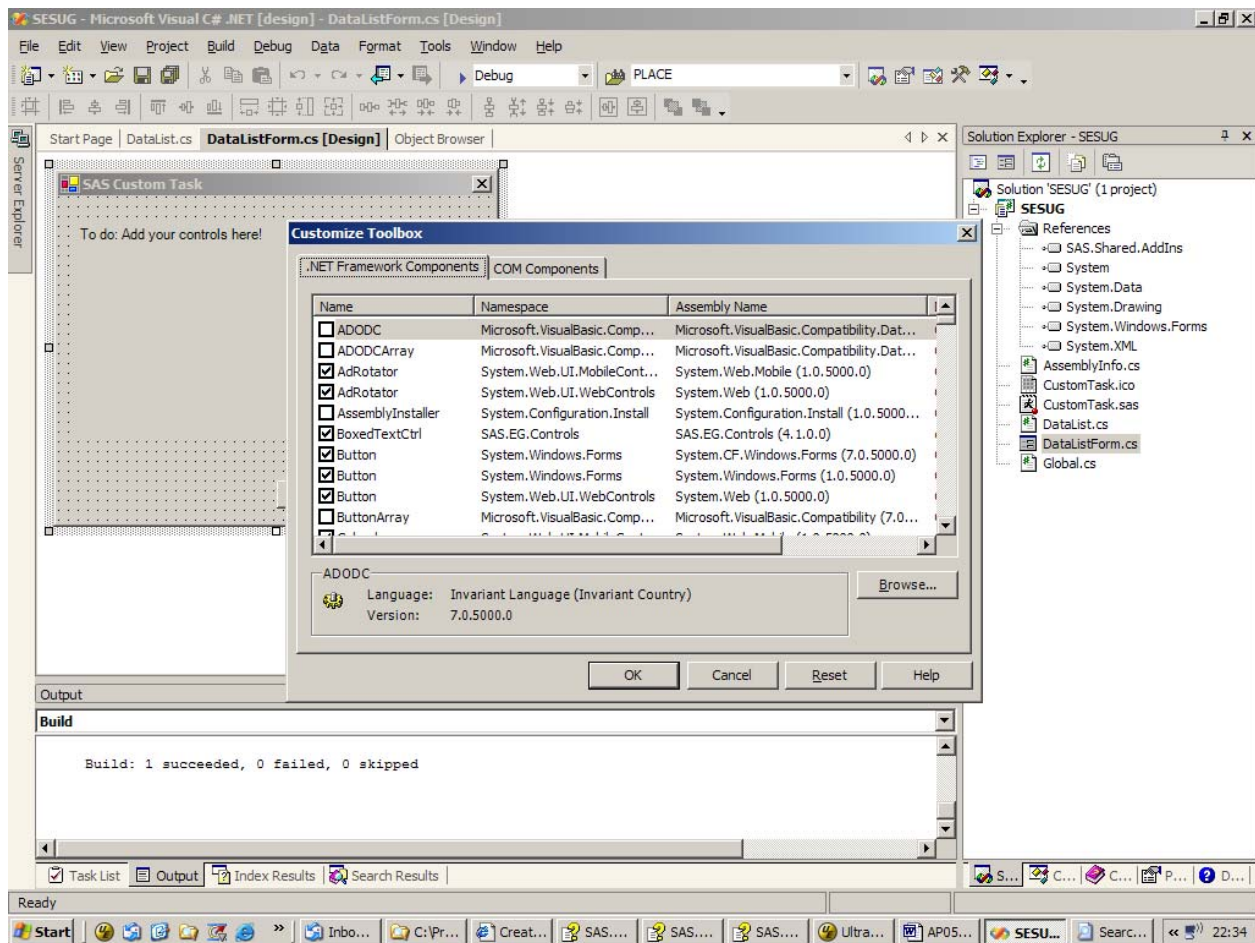


FIGURE 5 – ADDING THE SAS CONTROLS TO THE TOOLBOX

DEBUGGING AND TESTING

Once you have all of your C# code written and you get a clean debug build it is time to test the Addin in Enterprise Guide using the Visual Studio debugger. Although this is not a difficult task, it is time consuming because of the need to start and stop Enterprise Guide before each debugging session. First, the Addin is not a stand-alone application; it is designed to be run from within Enterprise Guide hence Enterprise Guide needs to be running in order to run you adding. Second, once the Addin is loaded by Enterprise Guide you can run it but you cannot alter your source code; Enterprise Guide essentially locks the project hence Enterprise Guide needs to be shut down so you can once again work on the project.

The first step to debugging is to inform Visual Studio that you have to start Enterprise Guide before you start debugging. This is done in the project properties window. From the *Configuration Properties* folder select *Debugging*, change the *Debug Mode* to *Program* and in the *Start Application* box navigate to your Enterprise Guide executable. See Figure 6 for an example. Once you have set the *Start Application* you can set your breakpoints and start the Visual Studio debugger.

Starting the debugger will cause Enterprise Guide to start. On you first debugging run you will have to tell Enterprise Guide about the new Addin task before you can access it. This is done from the *Add-in Manager* item on the Enterprise Guide *Add-In* menu. Figure 8 shows the Add-In Manager. From the Add-In Manager click the Browse button and navigate to the folder with you project; you will want to get the .DLL from the *Bin\Debug* subfolder.

After you include your custom task it will appear on the Task List, usually at the bottom. Like any other Enterprise Guide task, add it to your project and start it. Any breakpoints in your code will cause the execution to stop so you can examine the current status of your Visual Studio project.

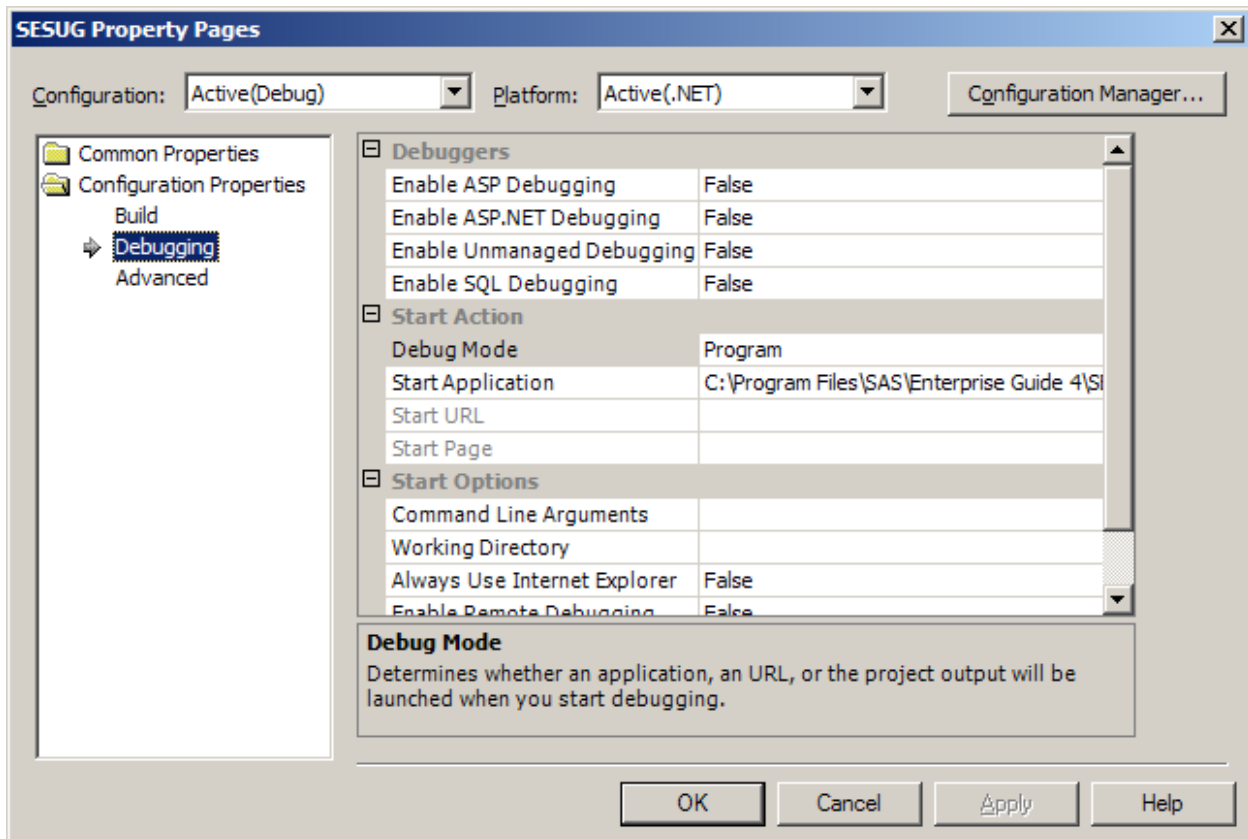


FIGURE 6 – SETTING THE DEBUGGING OPTIONS

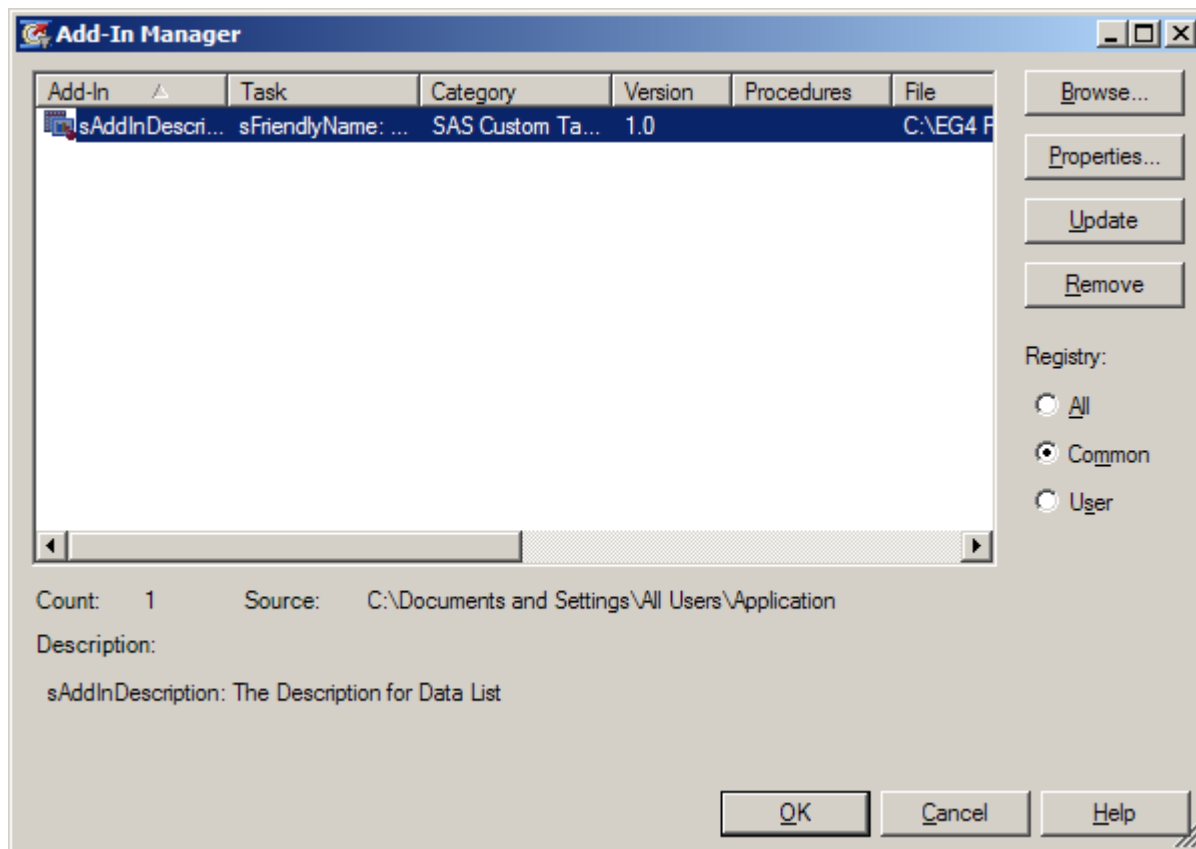


FIGURE 7 – THE ENTERPRISE GUIDE ADD-IN MANAGER

DEPLOYING THE ADDIN

After you have tested the Addin and are happy that not only does it present a clean and error free interface but also that it generates clean and error free SAS code that returns the expected results, you are ready to deploy the Addin. Your first step is to recreate the project in *Release* mode. Basically this removes all of the .NET debug code and generates smaller and normally more efficient code. Of course, after you create the release version you will test it once again to ensure that changing the compile options did not change the resultant generated code. The resultant code is in the form of a redistributable DLL.

One of the major features of using the .NET framework is the release from Windows 'DLL Hell'; you now longer have to register your DLL with Windows. This means that multiple versions of the DLL should peacefully co-exist on a computer. The simplest way to deploy the DLL then is to copy it into the location that EG expects to find it. By default this location is

C:\Program Files\SAS\Shared Files\BIClientTasks\Custom

Again, if you have installed SAS in a non-default location you will have to locate the equivalent folder. If the **Custom** folder does not exist on the computer it will need to be created.

Do **NOT** put the DLL in a folder on a shared network drive. Once any users accesses the DLL it will be locked and other users will not be able to access it.

If you choose to copy the DLL to some other location on the computer then the Addin will need to be registered in EG using the Add-in Manager in the same way you used the Add-In Manager to register your copy of the debug DLL.

CONCLUSION

SAS has addressed the 'Its too hard to learn' refrain by creating a project and task based interface analyst can easily use. To facilitate its use, SAS has created numerous tasks to hide the complexities and subtle nuances of SAS code. However, no matter how many tasks SAS provides, there can always be a need for a task interface that is very specific to your company or industry. When this happens, Enterprise Guide can be extended with custom tasks that analysts access just like any of the built in tasks. This paper stepped through the main steps needed to ensure that all of the basic housekeeping duties of creating a custom Addin are covered.

For further information or examples of custom Addin code contact me through the email address below.

REFERENCES

The help documents and samples are available at

<http://support.sas.com/documentation/onlinedoc/guide/customtasks/index.htm>

Sharp, John and Jon Jagger (2003) Visual C#.NET Step by Step

Microsoft Press, Redmond

Whitechapel, Andrew (2003) .NET Development for Microsoft Office

Microsoft Press, Redmond

CONTACT INFORMATION

Peter Eberhardt is SAS Certified Professional V8, SAS Certified Professional V6, and SAS Certified Professional - Data Management V6. In addition his company, Fernwood Consulting Group Inc. is a SAS Alliance Partner. Peter is a regular speaker at SUGI, SESUG and NESUG. If you have any questions or comments you can contact Peter at:

Fernwood Consulting Group Inc.,

288 Laird Dr.,

Toronto ON M4G 3X5

Canada

Voice: (416)429-5705

e-mail: peter@fernwood.ca

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.