# Exposing a VisualAPL Class Library as COM to APL+Win

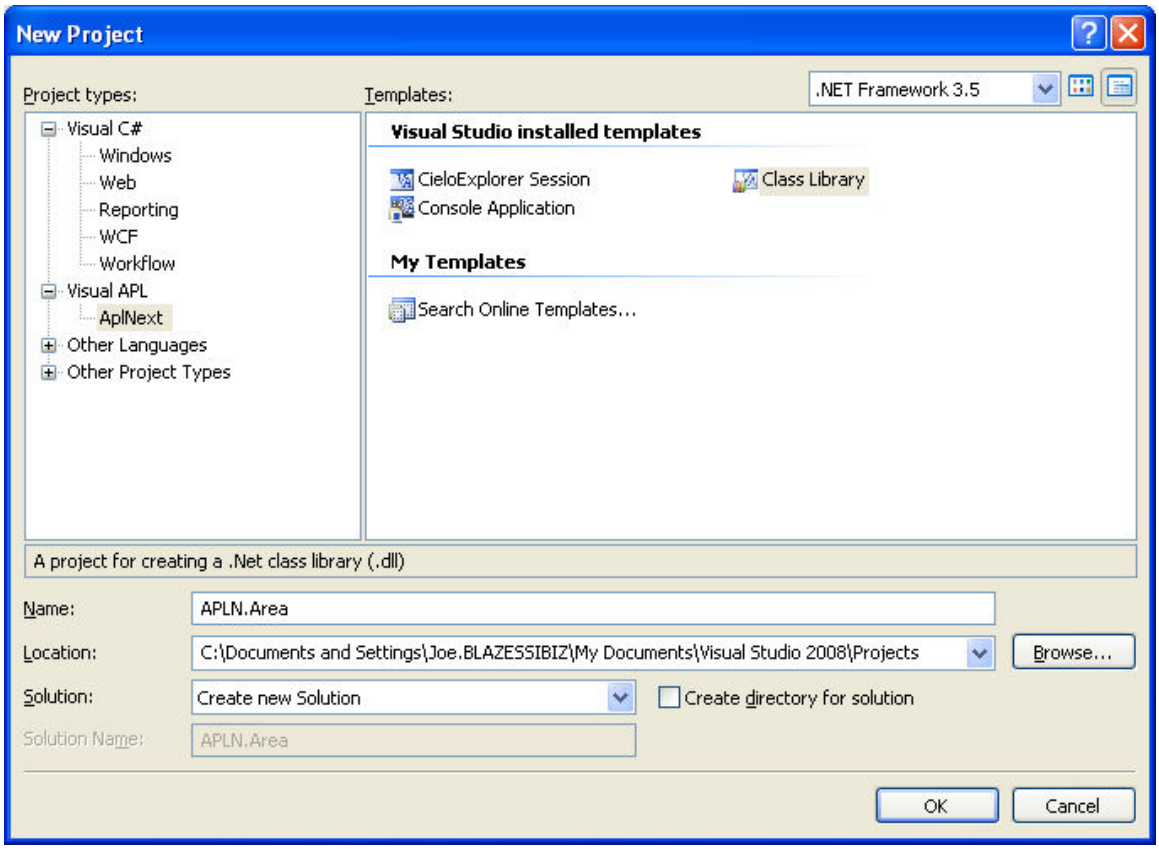
From the Visual Studio 2008 main form, use “File > New > Project > VisualAPL > AplNext > Class Library” dialog to create a new VisualAPL class library project (.Net assembly).

The “Name:” field on this dialog will be used as the project namespace name, so enter it in the format “BusinessEntityMnemonic.ProjectMnemonic”.

The “Location:” field on this dialog initially pointes to the Visual Studio 2008 default location for a project. You may select any other location, as long as you can find it later!

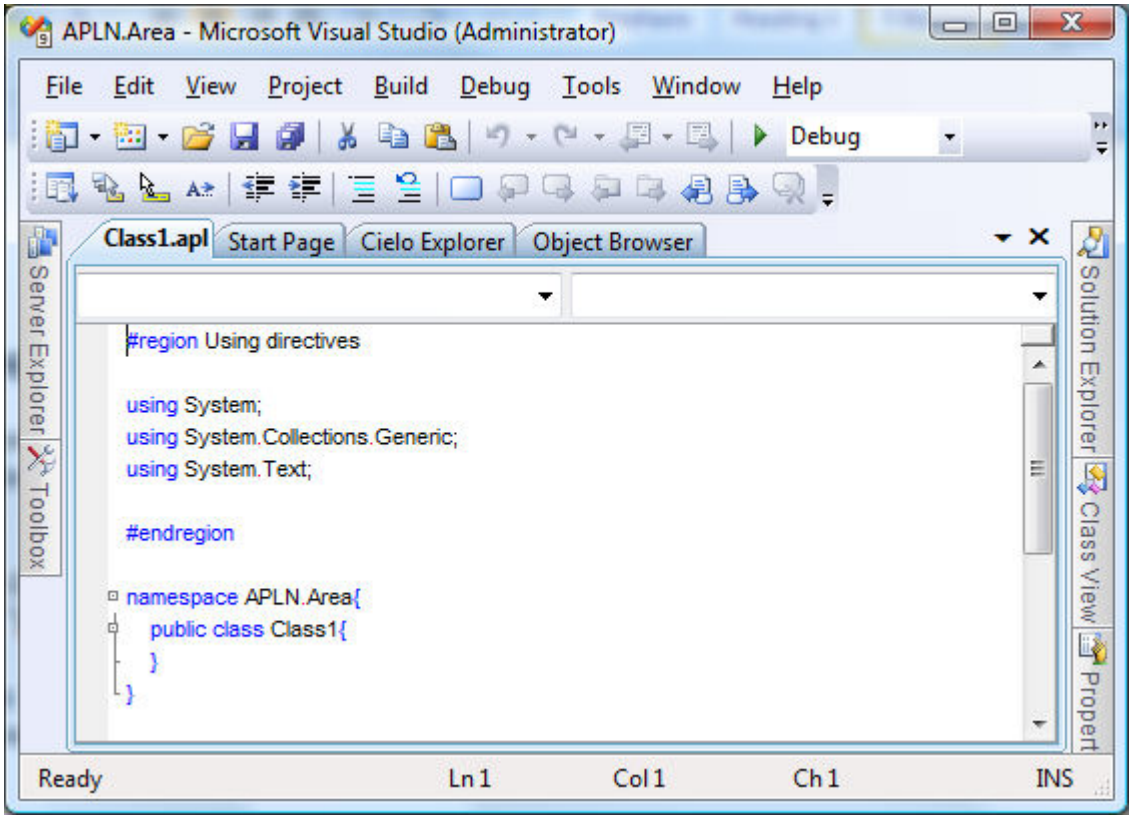
If you don’t see this dialog, go to http://forum.apl2000.com/viewtopic.php?t=443.

Click the “OK” button to continue creating the project.



The VisualAPL “Class Library” template is used to create default “Class1”.

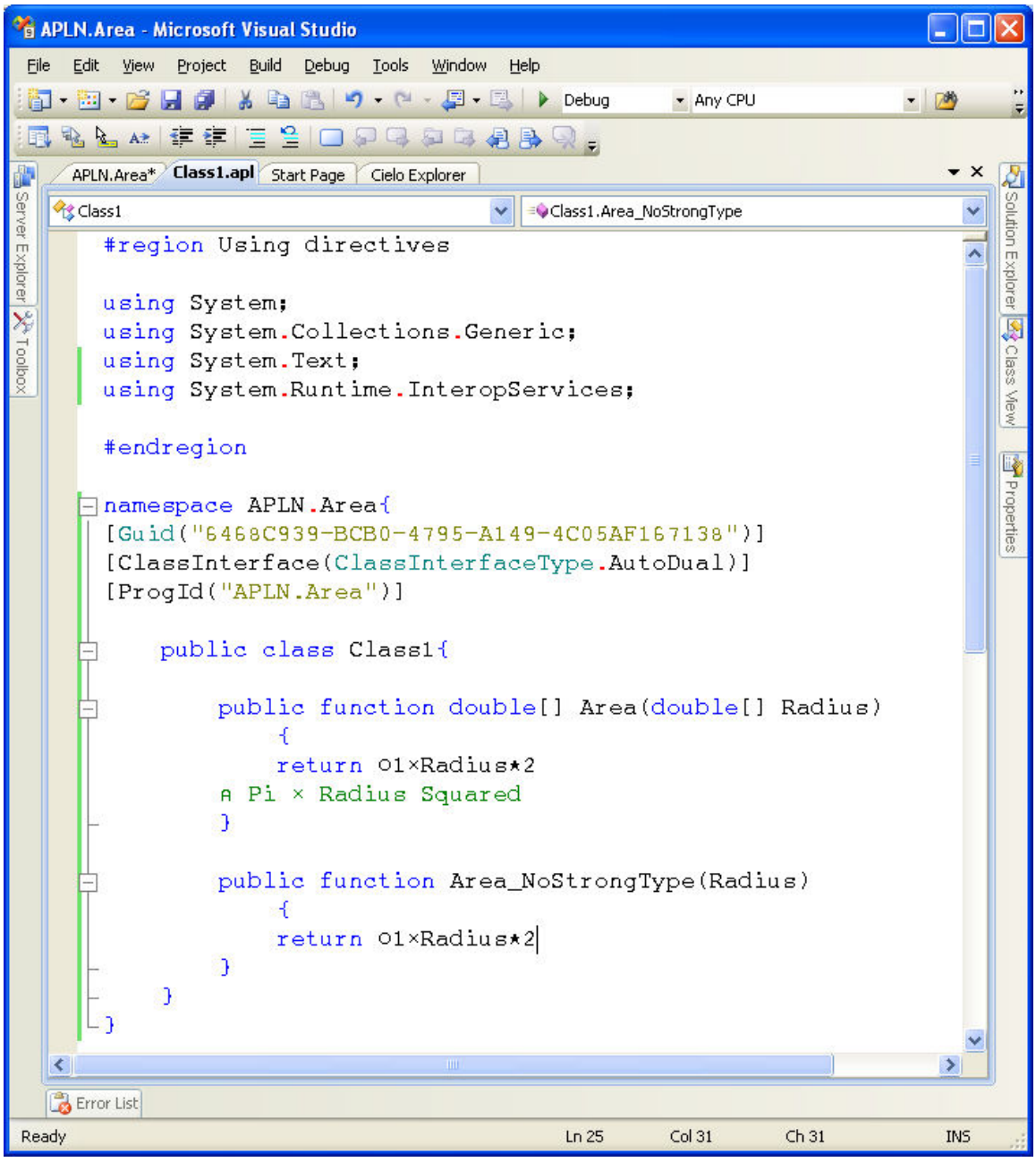
The VisualAPL method definitions for the application system are typed or copy/pasted by the programmer inside the ‘Class1” code block “public class Class1{…}”



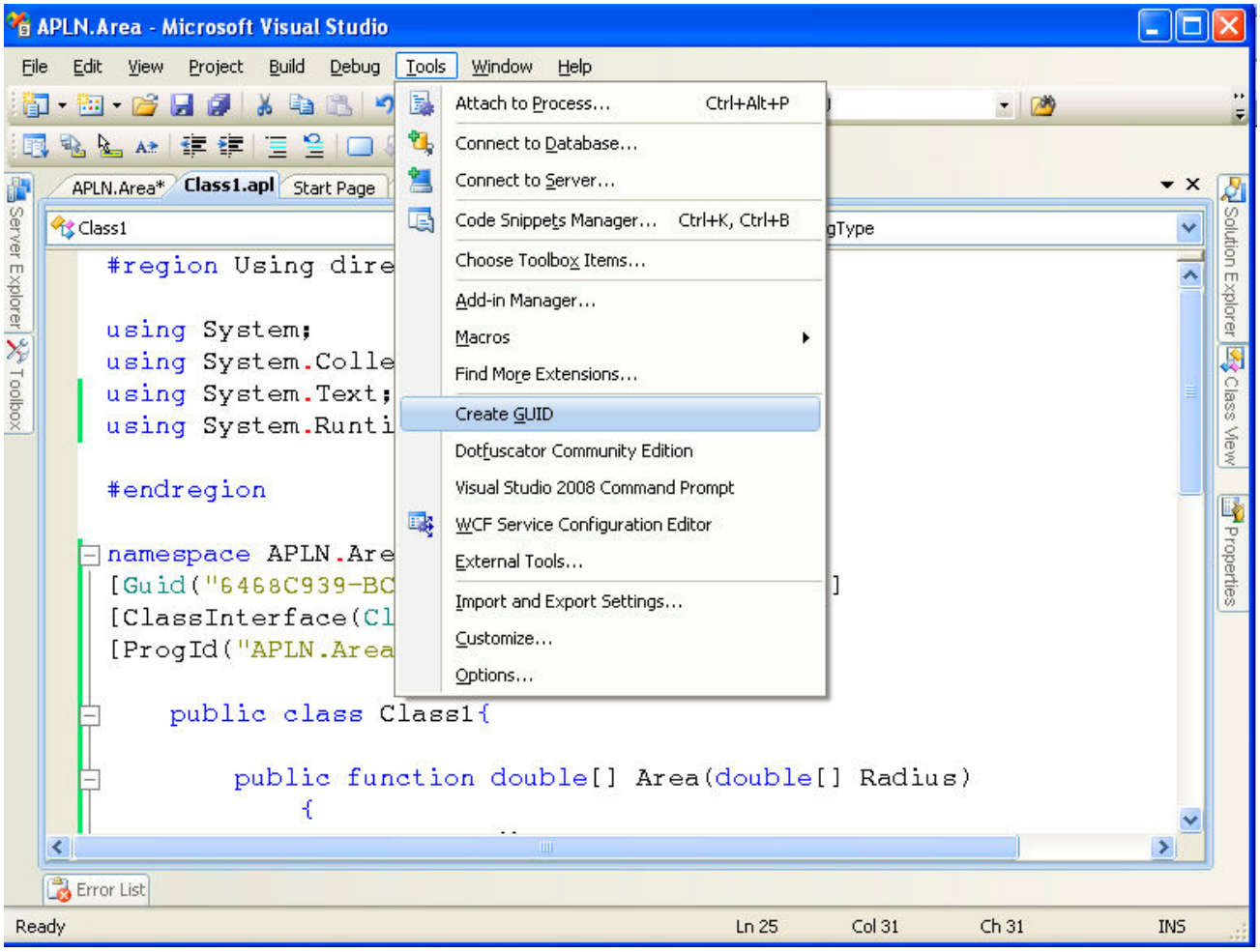
To define Class1 type or copy/paste the illustrated code statements in the VisualAPL Class1.apl code window.

In the “Using directives” region the “using System.Runtime.InteropServices;” namespace is required to expose the public methods in the class as COM.

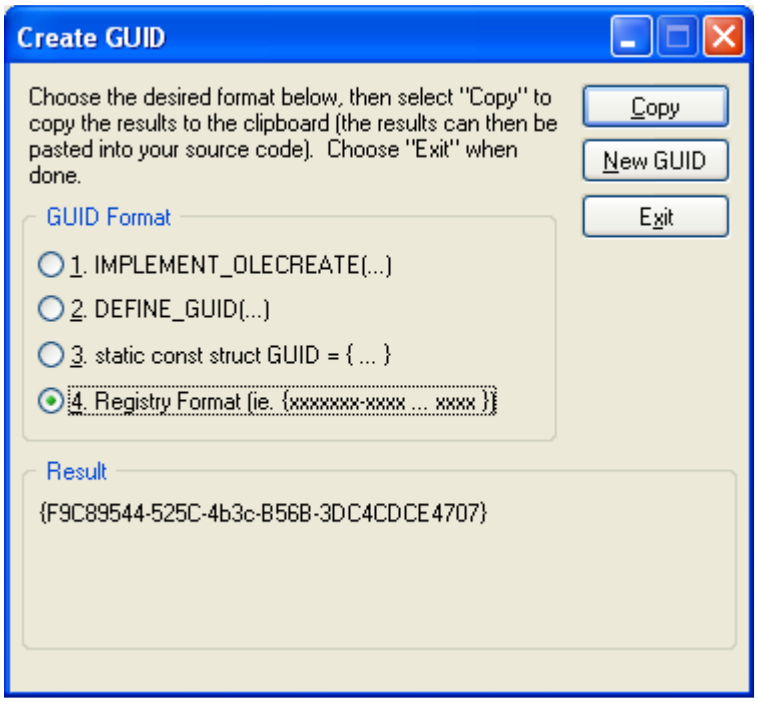
Depending on the project there may be other .Net namespaces which need to be referenced in the VisualAPL project. Microsoft .Net is well-documented, there are many web-based examples available and the VisualAPL Cielo Explorer makes learning about the .Net framework easy.



The "[Guid(…]" attribute is also required to uniquely identify the COM object when it is registered. Do not use the Guid (globally unique identifier) illustrated, instead use the Visual Studio Guid Tool to obtain a new Guid as follows:



When the following dialog is presented, click the “Copy” button to put the new Guid into the Windows Clipboard and then paste it into the Class1.apl “[Guid(…]” attribute code statement. It will be necessary to remove the “{}” delimiting the Guid after it is pasted into the VisualAPL Class1 code block as the “{}” are not used in the “Guid(…)]” attribute.



In the Class1.apl, the "[ClassInterface(…)]" is also an attribute required to expose the public methods in the class as COM. The "AutoDual" option means that a separate COM interface class in addition to the Class1 class will not be necessary.

The "[ProgId("APLN.Area")]" value "APLN.Area" is selected by the programmer as the name of the COM object which will be exposed. It does not have to match the namespace name, but it should be selected carefully to avoid conflicts with other COM object names.

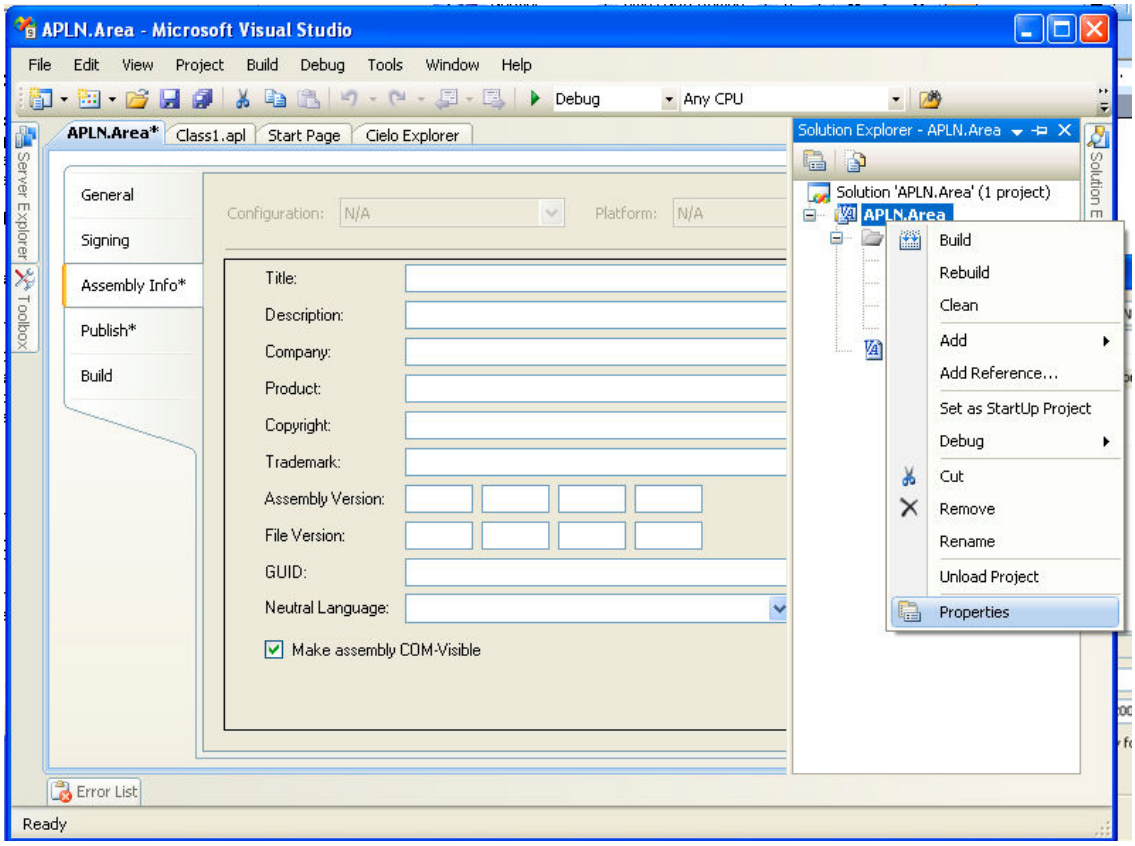
The public function "Area" has been defined with strong (explicit) data types (vectors of doubles) for the result and the argument. This is necessary so that the program which calls this function as a COM object will know the type of objects which must be supplied and will be returned.

The public function "Area_NoStrongType" does not provide strong data types for the result and the argument. When this function is called as a COM object, the calling environment will see the argument and result data types as the default "object" types. In most cases the calling environment will not be able to properly handle the "object" data type, as it is too general, and it will only have a pointer to the object available.

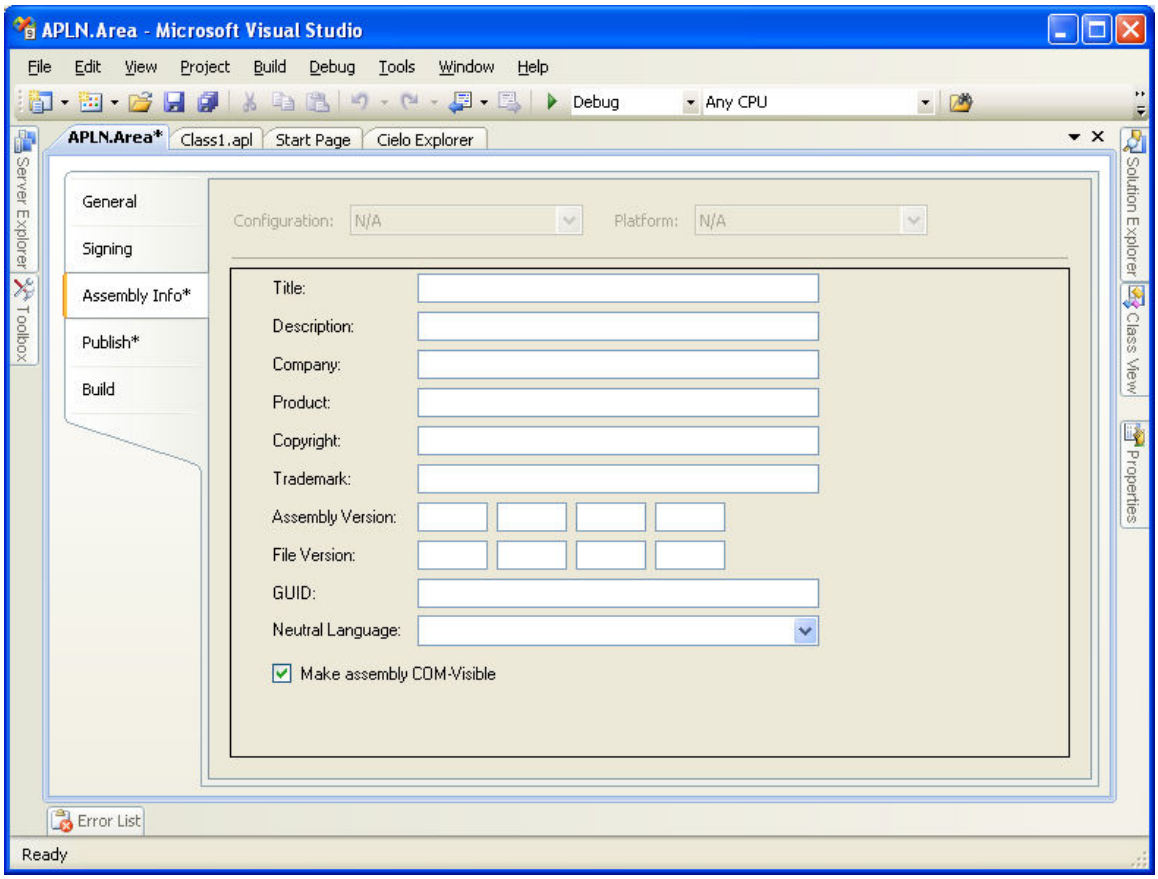
In some cases it is beneficial to have an underlying VisualAPL function which is not strongly typed. It can be called by cover functions which are strongly typed, but have different signatures for varying data type arguments or results. For example the "Area" function is strongly typed for vectors of doubles and could call the "Area_NoStrongType" function and another function "Area" could be strongly typed for a double and also call the "Area"NoStrongType". The two "Area" function signatures differ by data type and are considered "overloads" of each other. The non-strongly-typed "Area_NoStrongType" function utilized the "dynamic data typing" of VisualAPL to support both overloaded functions.

In the case of all of these functions, the formula for the area of a circle (in this example) is expressed in standard APL code statements, including the "circular" function for Pi.

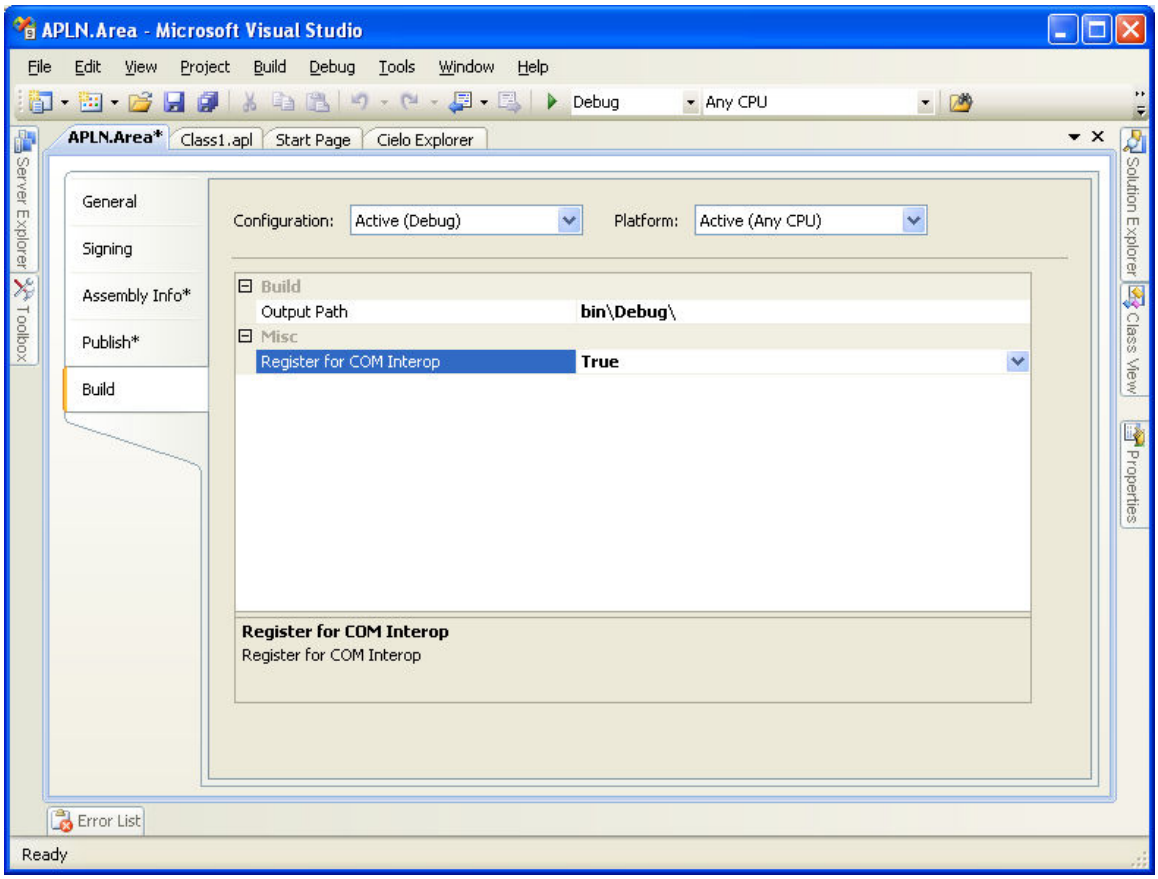
Another step is required to expose Class1 as COM. Using the Visual Studio Solution Explorer again, right click on the “APLN.Area” project and select the “Properties” pop-up menu item:



In the “Properties” window, click the “Assembly Info*” tab and check the “Make assembly COM-Visible” option:



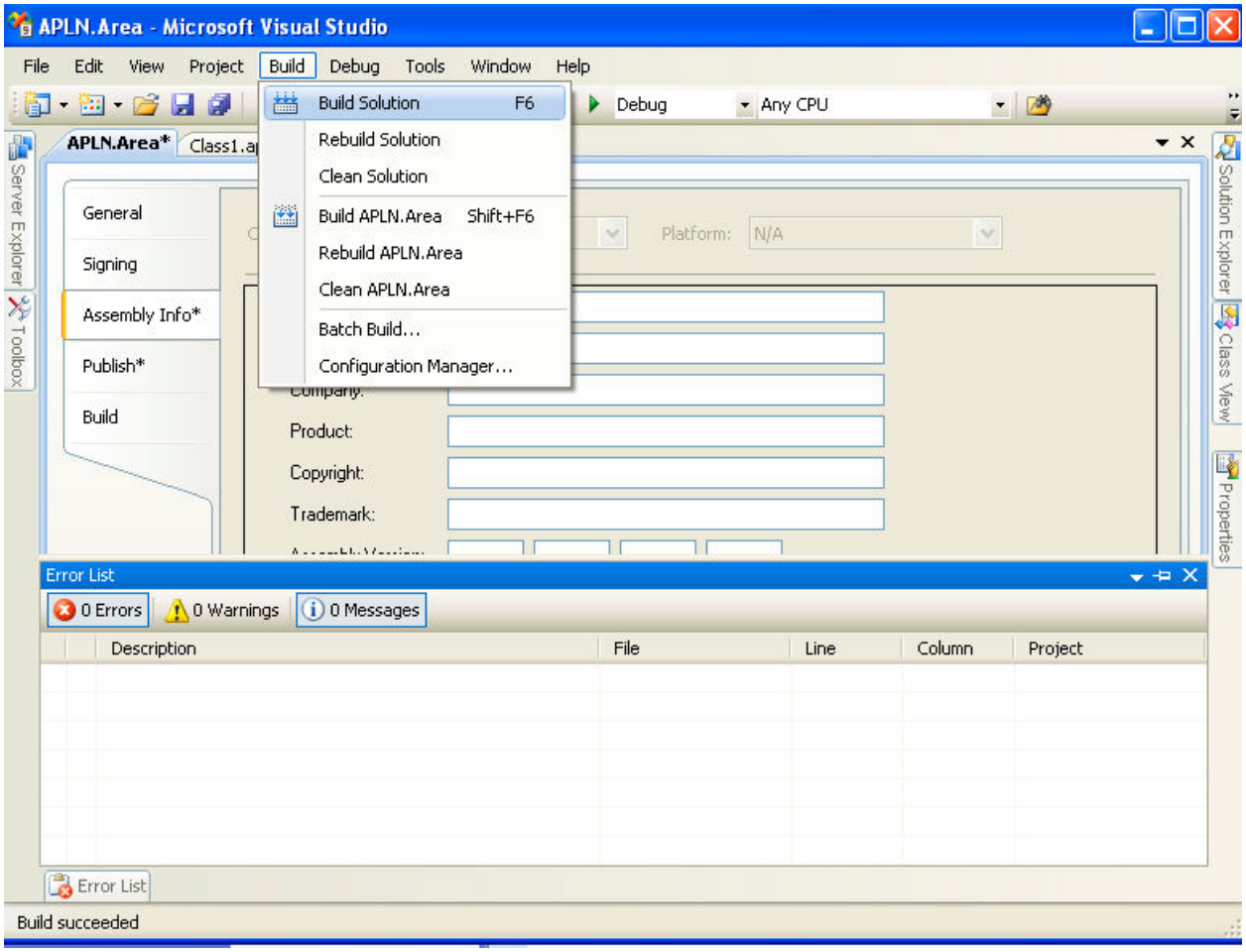
In the “Properties” window, click the “Build” tab and use the “Register for COM Interop” list box to select the “True” option:



Now that the above steps have been carried out, click the Visual Studio “Build” menu item and click the “Build Solution” sub-menu item.

Look for the “Build succeeded” message in the Visual Studio status bar. If there are programmer coding errors, they will be listed in the “Error List” window of Visual Studio. The programmer will correct the coding errors and try the “Build” option again.

If VisualAPL has not been activated yet or if the VisualAPL license is close to expiration, a reminder message may be presented when the “Build Solution” sub-menu item is clicked.



Now that the “Build” has been completed for the VisualAPL class library project, the Class1 class has been exposed as COM and registered using the Guid and the programmer-selected “ProgId” of “APLN.Area”.

Open a session of a Win32 application such as Visual Basic 6 or, in this case, APL+Win to access the methods in Class1. In APL+Win the COM methods are prefixed by “X”.

APL+Win displays the strongly-typed function signature of the “XArea” function using the “A []wi “??XArea” syntax and the default ‘object’ data types in the function signature of the “XArea_NoStrongType” function.

When the “XArea” function is called with the argument 100, APL+Win automatically coerces the singleton 100 to a one-element vector to satisfy the argument data type of the “XArea” function. Calling the “XArea” function with the argument (1 10 100 1000) illustrates the vector of doubles argument and result data types of the “XArea” function.