INNOVATE

Microsoft Dynamics® GP

Using a .NET Assembly from a Dexterity®-based Application

White Paper

This paper describes how to create a .NET assembly and use it in a Dexterity-based application.

Date: July, 2008

Microsoft Dynamics™

Table of Contents

Introduction

This document describes the steps to write a C# class and compile it into a .NET assembly that can be called via COM from a Dexterity-based application such as Microsoft Dynamics GP. After the .NET assembly is created, a Dexterity script is created that calls and tests the code in the assembly.

The C# class created for this example is a simple. However, more complex projects can be created and used successfully. This document assumes that the reader has some experience with Microsoft Dynamics GP, Microsoft Dexterity, and Microsoft Visual C#®.

This example was created and tested using the following:

- Microsoft Dynamics GP 10.0 SP2
- Microsoft Dexterity 10.0.313
- Microsoft Visual Studio® 2005 SP1

Getting Started

The first step is to use C# to create the .NET assembly that will be used from within the Dexterity-based application.

Create a Project

Launch Microsoft Visual Studio. After the development environment is started, choose to create a new project. Select Visual C# as the project type, and use the Class Library template. For this example, set the name of the project to **SimpleProperties**. Click OK to create the project.

Add Code for the Class

In Solution Explorer, rename the default file **Class1.cs** to **SimpleProperties.cs**. Enter the following C# code for the SimpleProperties.cs file to replace the default generated code. When you are creating your own .NET assembly, you will use a tool like GUIDGen to create your own GUID attribute value that uniquely identifies the COM interface.

Script: SimpleProperties.cs

```
using System;
using System.Runtime.InteropServices;

namespace SimpleProperties
{
    public class SimpleProperties
    {
         [GuidAttribute("AD4760A9-6F5C-4435-8844-D0BA7C66AC50"),
          ClassInterface(ClassInterfaceType.AutoDual)
          ComVisible(true)]
        public class ConvertDollars
        {
            private short cnts;
            private int dlrs;
            private int amnt;
```

```
public ConvertDollars()
{ // Arbitrarily initialize this object instance to $10.25
    dlrs = 10;
    cnts = 25;
}
public ConvertDollars(int x)
{
    // Second constructor,  not available to COM.
    // Set a different field with this constructor.
    amount = x;
}

 // Getter and setter methods for our properties.
public int amount
{
    get {return amnt;}
    set {amnt = value;}
}
public short cents
{
    get {return cnts;}
    set {cnts = value;}
}
public int dollars
{
    get {return dlrs;}
    set {dlrs = value;}
}

 // This one is read-only for illustration purposes.
public string sAmount
{
    get {return "$" + dollars.ToString() + "." + cents.ToString();}
}

// This is the only method of this class. It takes a
// string "in" as well as an "inout" reference param string array.
public string SetArray(string s,ref string[] myarray)
{
    // The example is passing a 5-element array.
    // C# uses a zero-based array
    // and Dexterity uses a one-based array.
    // Note that Dexterity will crash if this array is resized
    // because Dexterity cannot dynamically resize an array.
    try
    {
        myarray[0] = "one";
        myarray[1] = "two";
        myarray[2] = "three";
        myarray[3] = "four";
        myarray[4] = "five";
```

```
                return s;   // Pass back in the string sent
            }
            catch(Exception e)
            {
                return "Exception caught:" + e.Message;
            }
            finally
            {
            }
        }
    }
}
```

Edit the Assembly Information

In Solution Explorer, expand the Properties node and open the **AssemblyInfo.cs** file. This file specifies how the .NET assembly will be created. Locate the ComVisible attribute and set its value to true. This allows the .NET assembly to be accessed through COM. Save the changes and close the file.

```
// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(true)]
```

Another way to set this property is to use the Assembly Information window. In Solution Explorer, right-click the project and choose Properties. Display the Application properties. Click the Assembly Information button. In the Assembly Information window, mark the Make assembly COM-Visible check box, and then click OK.

Building and Registering the .NET Assembly

After the code is complete, you will build and register the .NET assembly.

Build the .NET Assembly

From the File menu, choose Save All to save your work. From the Build menu, choose to build the project, which will generate the .NET assembly. If the build operation is successful, Visual Studio will have generated a debug version of the SimpleProperties.dll assembly. This assembly is created in the \bin\debug folder for your Visual Studio project. At this point, copy the SimpleProperties.dll from that folder into the folder where you have Dexterity installed.

Register the .NET Assembly

The RegAsm.exe utility included with the Microsoft .NET Framework 2.0 is used to register the .NET assembly. It is also used to create a registry file and a type library file that describe the assembly. This utility is typically found in this location:

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\regasm.exe

The following examples assume that the path to this location is set in the PATH environment variable. If it is not, then the location to the RegAsm.exe must be fully-qualified when you use it.

The RegAsm.exe is called from a command prompt. *The utility must be run with administrative privileges.* On Windows® XP or Server 2003 you must be part of the Administrators group. On Windows Vista and Server 2008, you must start the command prompt with elevated privileges.

Open the command prompt. Set the current directory to the location where you copied the SimpleProperties.dll assembly. Use RegAsm.exe to register all of the public classes in the assembly. For example, the following command registers the SimpleProperties assembly.

regasm SimpleProperties.dll

Create the COM Type Library

The COM type library must be created that describes the COM objects that are available in the assembly. Using the RegAsm.exe application, create .tlb file that contains the definitions of the public types in the assembly. The following command creates the .tlb for the SimpleProperties assembly.

regasm SimpleProperties.dll /tlb:SimpleProperties.tlb

This is the type library you will reference from within Dexterity to access the capabilities made available through COM. When your application accesses the assembly you created with C#, the assembly and type library should be located in the same directory as Dexterity (Dex.exe) or the Runtime engine (Dynamics.exe).

Create and Merge the Registry File

A registry file must be created to allow access to the assembly through COM. Using the RegAsm.exe application, create a registry file that describes the assembly. This file must be merged into the registry on any systems that use the .NET assembly. The following command creates the .reg file for the SimpleProperties assembly.

regasm SimpleProperties.dll /regfile:SimpleProperties.reg

To merge the information from the SimpleProperties.reg file into the registry, type the following in the command line:

SimpleProperties.reg

Or, double-click the SimpleProperties.reg file to include it into the registry. Click Yes in the dialog that is displayed to merge the registry information.

Close the command prompt.

Using the .NET Assembly

After the .NET assembly has been created and registered, it can be used from the Dexterity-based application.

Create a Dexterity-based Application

To test the .NET assembly, you will build a stand-alone Dexterity application. Launch Dexterity. When prompted, choose the option to create a new dictionary with a Main Menu form. Specify the name and location of the new dictionary. For this document, the path and file name used is c:\dexterity\app.dic. Click OK to create the dictionary.

Add Resources to the Application Dictionary

Use the following procedure to add resources to the text dictionary.

1. After the Resource Explorer window opens, expand the Base node and select Libraries. Click the New button. The Library Definition window will appear.
2. In this window, specify COM Type Library as the type of library being created. Click the Name lookup button and choose the SimpleProperties COM library file that was registered in the previous section. Click OK button to save this library definition.

3. In the Resource Explorer, select the Forms node from the Resources list. In the right-hand pane of the Resource Explorer, double click the Main Menu item to open the Main Menu form. Click New on the Form Definition window to create a new window on the Main Menu form. The Layout window will open.
4. In the Layout window, create a new local push button by selecting the Push Button tool from the Toolbox and clicking on the window layout. Double-click the new button to open the Script Editor window.
5. In the Script Editor, paste the following Dexterity code. Note that curly brackets are beginning and ending comment markers for the Dexterity scripting language and will be ignored by the compiler.

Script: Button Change Script

```
local SimpleProperties.ConvertDollars obj;
local text tText;
local string ReturnValue;
local integer x;
local string myarray[5]; {5 element string array}

obj = new SimpleProperties.ConvertDollars();
{obj initializes with dollars = 10 and cents = 25}
tText = tText + "Object initialized to: " + str(obj.dollars)
   + "." + str(obj.cents) + char(13);

{Switch the dollars to 15}
obj.dollars = 15;

{Switch cents to 40}
obj.cents = 40;
tText = tText + "Dollars and cents switched to: " + str(obj.dollars)
   + "."+ str(obj.cents) + char(13);

{Return a string property}
tText = tText + "Getting a string property returned: "
   + str(obj.sAmount) + char(13);

{Pass in a string and string array}
ReturnValue = obj.SetArray("Test Array", myarray);

{Append to the output field the contents of the string array}
for x = 1 to 5 do
   tText = tText + "Array Element " + str(x) + ": "
    + myarray[x] + char(13);
end for;

{Destroy the object}
clear obj;

{Display the results}
warning tText;
```

6. Click Compile to compile the Dexterity code. Close the Script Editor. Close the Layout window, and click Yes to save the changes. Finally, click OK to close the Form Definition window.

Run the Test Application

Use the following procedure to run the test application.

1. Choose Test Mode from the Debug menu, or press Control-T. Dexterity will switch to Test Mode and automatically open the Main Menu form.

2. Click the button that was added in the previous section. The button script will be executed that invokes the ConvertDollars class created in the SimpleProperties assembly and will display a dialog box with the following results.

   Object initialized to: 10.25
   Dollars and cents switched to: 15.40
   Getting a string property returned: $15.40
   Array Element 1: one
   Array Element 2: two
   Array Element 3: three
   Array Element 4: four
   Array Element 5: five

Conclusion

By exposing managed .NET code to COM, a Dexterity-based application can be extended to take advantage of the functionality and benefits of the Microsoft .NET Framework.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics