Avoiding .NET Assembly Conflicts with Custom Modules

Application Note

Date	August 24, 2011
Applies To	Kofax Capture 8.0, 9.0, and 10.0
Summary	This application note provides a technique for avoiding .NET assembly conflicts when creating Custom Modules in Visual Basic.NET or C#
Revision	1.1

The Problem with .NET Custom Modules

It is a best practice with Kofax Capture that Custom Modules are always distributed to the Kofax\Capture\BIN folder, and that it not distribute any of the Kofax Capture Interop Libraries that it references.

This is essential for .NET Custom Modules, since the Interop Libraries and underlying COM objects can be updated by service packs and patches, not to mention an upgrade installation to a later version of Kofax Capture.

If a Managed Custom Module is distributed to a different folder, it means that the Primary Interop Assemblies need to be distributed, and a service pack or upgrade installation could make these libraries incompatible with the new COM objects installed by the service pack or patch.

However, one issue that may present itself, due to the best practice, is the fact that different Custom Modules could have several .NET assemblies it depends on, and it is possible that different versions of these assemblies could be required by different Custom Modules.

This could mean that one custom module would function, but another one using a different version of the same DLLs would be unable to function.

This is exacerbated by Interop libraries generated when adding COM references to a .NET project, since two different assemblies with the same file name may be created for the same object.

It is likely that in some situations, registering these assemblies in the GAC might not be an option, especially if the dependent assemblies are not strongly signed.

The Solution

In order to resolve this, it is necessary to isolate the assemblies in a different location, while making them accessible to the Custom Module's Executable.

.NET has a mechanism that answers this issue: probing.

Using some settings in the Custom Module's .config file, it is possible to specify a set of subfolders within which to search for dependent assemblies.

Instead of distributing the DLL files to the Capture\BIN folder with the .EXE, simply create a subfolder under Capture\BIN, place the DLL files there, and create a .exe.config file to tell .NET to look there for these assemblies, like so:

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="MySubFolder" />
    </assemblyBinding>
  </runtime>
</configuration>
```

Using this technique, when the Custom Module is run, .NET will search in the MySubFolder path for the DLLs the Custom Module requires, thus isolating the DLLs to the Custom Module without risking conflict with other Custom Modules.

It should be noted that the sub-folder should be given a unique name to the Custom Module, and that more than one subfolder can be created.

If more than one subfolder is created, the subfolder names in the privatePath attribute should be delimited by a semicolon, like so:

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="MySubFolder;MySubFolder2" />
    </assemblyBinding>
  </runtime>
</configuration>
```

For further details of the <probing> element, please see the MSDN Library entry located at the URL below. This URL is valid as of the date of this application note.

http://msdn.microsoft.com/en-us/library/823z9h8w.aspx

Alternative Solution

Another possible solution, if installing a Managed Custom Module in the BIN folder is undesirable, may be to install it in its own folder, and utilize the AppDomain's AssemblyResolve event.

For example:

```
Public Class MainClass
    Private Shared Function MyResolveEventHandler(ByVal sender As Object, ByVal
args As ResolveEventArgs) As System.Reflection.[Assembly]
        Dim sAssemblyLocation As String = args.Name.Split(",")(0) & ".dll"
        Using rk As Microsoft.Win32.RegistryKey =
Microsoft.Win32.Registry.LocalMachine.OpenSubKey("SOFTWARE\Kofax Image
Products\Ascent Capture\3.0", False)
            sAssemblyLocation = IO.Path.Combine("" & rk.GetValue("EXEPath"),
sAssemblyLocation)
        End Using
        Return System.Reflection.Assembly.LoadFrom(sAssemblyLocation)
    End Function

    Shared Sub Main()
        AddHandler AppDomain.CurrentDomain.AssemblyResolve, AddressOf
MyResolveEventHandler
        Application.EnableVisualStyles()
        Application.DoEvents()
        Application.Run(New Form1)
    End Sub
End Class
```

This way, when the .NET Framework attempts to load an assembly, and cannot locate the required assembly via the standard Probing mechanism, the AssemblyResolve event is fired, allowing the application to load and return the required assembly.