Internalizing Assemblies with ILMerge

by Richard Carr, published at http://www.blackwasp.co.uk/ILMergeInternalize.aspx

ILMerge is a useful tool provided by Microsoft that allows a number of assemblies to be combined into a single dynamic linked library or executable. By default, public types from source assemblies remain public. In some cases they should be internalized.

Public Types in Merged Assemblies

ILMerge allows you to combine a number of .NET assemblies to create a single file containing all of the namespaces and types from the original files. If you create a solution that contains multiple projects, this allows you to merge all of the assemblies into one file, which can simplify deployment.

A common use of ILMerge is to combine assemblies that form a code library. One scenario is that the primary assembly contains all of the types that the users of the library should access. The secondary assemblies may only include code that is used by the primary assembly and that should not be visible to the end user. However, it is likely that the secondary assemblies include types and members that have public scope.

In the above situation, you may merge a primary assembly and two secondary assemblies using the following ILMerge command. The problem here is that projects that reference the Merged.dll library will be able to see the public classes from the secondary assemblies.

```
ilmerge /out:Merged.dll Primary.dll SecondaryOne.dll SecondaryTwo.dll
```

Internalizing Types

You can remove the problem of public items in secondary classes by *internalizing* the secondary assemblies. When you provide the internalize switch to the ILMerge command, any public types within the secondary assemblies are modified to provide internal visibility only. As the assemblies have been combined into one, internal visibility is enough to allow the types to remain visible to the merged files. However, the internalized types are no longer visible to projects that reference the merged assembly.

The following command shows the use of the internalize switch:

```
ilmerge /out:Merged.dll Primary.dll SecondaryOne.dll SecondaryTwo.dll /internalize
```

Exclude Files

In some cases you will wish to internalize some classes from secondary assemblies but allow several types to still be publicly accessible. To do so, you can use an *exclude file*. An exclude file

.NET Framework
Algorithms and Data Structures
Audio
C# Programming
Configuration
Debugging
Design Patterns
Documentation
Graphics
Input / Output
LINQ
Network and Internet
Parallel and Asynchronous
Performance
Programming Concepts
Refactoring
Reference Sheets
Reflection
Regular Expressions
Security
SQL Server
System Information
Testing
Tools
Visual Studio
Windows Programming
Windows Presentation Foundation
XML

is a simple text file that contains a list of types that will not be internalized. Each line of the text file represents one or more types that will remain public. The line can name a single type or be a regular expression that excludes all matching types.

For example, you could add the following text in an exclude file. This specifies that all types in the SecondaryOne namespace and the TestClass type in the SecondaryTwo namespace will remain public.

```
SecondaryOne.*
SecondaryTwo.TestClass
```

To use the exclude file, add a colon (:) and the name of the file to the internalize switch:

```
ilmerge /out:Merged.dll Primary.dll SecondaryOne.dll SecondaryTwo.dll
    /internalize:exclude.txt
```

3 July 2011