ILMerge

Michael Barnett

Programming Languages and Methods

Microsoft Research

1 Introduction

This document describes the ILMerge utility which merges multiple .NET assemblies into a single assembly. However, some .NET assemblies may not be able to be merged because they may contain features such as unmanaged code. I would highly recommend using peverify (the .NET Framework SDK tool) on the output of ILMerge to guarantee that the output is verifiable and will load in the .NET runtime.

ILMerge is packaged as a console application. But all of its functionality is also accessible programmatically. Note that Visual Studio **does** allow one to add an executable as a reference, so you can write a client that uses ILMerge as a library.

ILMerge takes a set of *input assemblies* and merges them into one *target assembly*. The first assembly in the list of input assemblies is the *primary assembly*. When the primary assembly is an executable, then the target assembly is created as an executable with the same entry point as the primary assembly. Also, if the primary assembly has a strong name, and a .snk file is provided, then the target assembly is re-signed with the specified key so that it also has a strong name.

Note that anything that depended upon any of the names of the input assemblies, e.g., configuration files, must be updated to refer instead to the name of the target assembly.

Any Win32 Resources in the primary assembly are copied over into the target assembly.

There are many options that control the behavior of ILMerge. These are described in the next section.

2 Public Interface

The public interface for ILMerge is defined in the ILMerging namespace as the class ILMerge.

```
namespace ILMerging;

public class ILMerge { }
```

To use this class programmatically, just create an instance using the default (nullary) constructor. There are several properties and methods which are used to affect the behavior of the merging. The method Merge() is called to actually perform the merging.

2.1 AllowDuplicateType

The normal behavior of ILMerge is to not allow there to be more than one public type with the same name. If such a duplicate is found, then an exception is thrown. However, ILMerge can just rename the type so that it no longer causes a conflict. For private types, this is not a problem since no outside client can see it anyway, so ILMerge just does the renaming by default. For public types, it is not often a useful feature to have it renamed. However, there are situations where it is needed. In particular, for obfuscated assemblies, it seems that the obfuscator defines an attribute and creates an assembly-level attribute for the obfuscated assembly using that attribute. This would mean that obfuscated assemblies cannot be merged.

So this option allows the user to either allow all public types to be renamed when they are duplicates, or to specify it for arbitrary type names. On the command line, there can be as many options as desired, if followed by a colon and a type name, otherwise just specify it alone with no colon (and type name) to allow all duplicates.

When used through the API, an argument of "null" means to allow all public types to be renamed.

Default: no duplicates of public types allowed

Command line option: `[/allowDup[:typeName]]*`

2.2 AllowMultipleAssemblyLevelAttributes

When this is set before calling Merge, then if the CopyAttributes property (Section 2.7) is also set, any assembly-level attributes names that have the same type are copied over into the target directory as long as the definition of the attribute type specifies that "AllowMultiple" is true.

Default: false

Command line option: `/allowMultiple`

2.3 AllowWildCards

When this is set before calling Merge, any wild cards in file names are expanded and all matching files will be used as input. Note that because the wild card matching is done by a call to `Directory.GetFiles`, it does not allow the characters ".." to appear in a file name. So if you want to specify a relative path containing ".." to move up a directory, you will have to use it with the "/lib" option (Section 2.20). That option does allow the use of ".." to move up directories.

Default: false

Command line option: `/wildcards`

2.4 AllowZeroPeKind

When this is set before calling Merge, then if an assembly's PeKind flag (this is the value of the field listed as .corflags in the Manifest) is zero it will be treated as if it was ILonly. This can be used to allow C++ assemblies to be merged; it does not appear that the C++ compiler writes the value as ILonly. However, if such an assembly has any non-IL features, then they will probably not be copied over into the target assembly correctly. So please use this option with caution.

Default: false

Command line option: `/zeroPeKind`

2.5 AttributeFile

If this is set before calling Merge, then it specifies the path and filename to an *atttribute assembly*, an assembly that will be used to get all of the assembly-level attributes such as Culture, Version, etc. It will also be used to get the Win32 Resources from. It is mutually exclusive with the CopyAttributes property (Section 2.7). When it is not specified, then the Win32 Resources from the primary assembly are copied over into the target assembly. If it is not a full path, then the current directory is used.

Default: null

Command line option: `/attr:`*`filename`*

2.6 Closed

When this is set before calling Merge, then the "transitive closure" of the input assemblies is computed and added to the list of input assemblies. An assembly is considered part of the transitive closure if it is referenced, either directly or indirectly, from one of the originally specified input assemblies and it has an external reference to one of the input assemblies, or one of the assemblies that has such a reference. Complicated, but that is life...

Default: false

Command line option: `/closed`

2.7 CopyAttributes

When this is set before calling Merge, then the assembly level attributes of each input assembly are copied over into the target assembly. Any duplicate attribute overwrites a previously copied attribute. If you want to allow duplicates (for those attributes whose type specifies "AllowMultiple" in their definition), then you can also set the AllowMultipleAssemblyLevelAttributes (Section 2.2). The input assemblies are processed in the order they are specified. This option is mutually exclusive with specifying an attribute assembly, i.e., the property AttributeFile (Section 2.5). When an attribute assembly is specified, then no assembly-level attributes are copied over from the input assemblies.

Default: false

Command line option: `/copyattrs`

2.8 DebugInfo

When this is set to true, ILMerge creates a .pdb file for the output assembly and merges into it any .pdb files found for input assemblies. If you do not want a .pdb file created for the output assembly, either set this property to false or else specify the /ndebug option at the command line.

Default: true

Command line option: `/ndebug`

2.9 DelaySign

When this is set before calling Merge, then the target assembly will be delay signed. This can be set only in conjunction with the /keyfile option (Section 2.13).

Default: false

2.10 ExcludeFile

This property is used only in conjunction with the Internalize property (Section 2.12). When this is set before calling Merge, it indicates the path and filename that will be used to identify types that are not to have their visibility modified. If Internalize is true, but ExcludeFile is "", then all types in any assembly other than the primary assembly are made non-public. Setting this property implicitly sets Internalize to true.

The contents of the file should be one regular expression per line. The syntax is that defined in the .NET namespace System.Text.RegularExpressions for regular expressions.

The regular expressions are matched against each type's full name, e.g., "System.Collections.IList". If the match fails, it is tried again with the assembly name (surrounded by square brackets) prepended to the type name. Thus, the pattern "\[A\].*" excludes all types in assembly A from being made non-public. (The backslashes are required because the string is treated as a regular expression.) The pattern "N.T" will match all types named T in the namespace named N no matter what assembly they are defined in.

It is important to note that the regular expressions are *not* anchored to the beginning of the string; if this is desired, use the appropriate regular expression operator characters to do so.

Default: ""

Command line option: `/internalize[:excludeFile]`

2.11 FileAlignment

This controls the file alignment used for the target assembly. The setter sets the value to the largest power of two that is no larger than the supplied argument, and is at least 512.

Default: 512

Command line option: `/align:n`

2.12 Internalize

This controls whether types in assemblies *other than* the primary assembly have their visibility modified. When it is true, then all non-exempt types that are visible outside of their assembly have their visibility modified so that they are not visible from outside of the merged assembly. A type is exempt if its *full name* matches a line from the ExcludeFile (Section 2.10) using the .NET regular expression engine.

Default: false

Command line option: `/internalize[:excludeFile]`

2.13 KeyFile

When this is set before calling Merge, it specifies the path and filename to a .snk file. The target assembly will be signed with its contents and will then have a strong name. It can be used with the DelaySign property (Section 2.9) to have the target assembly delay signed. This can be done even if the primary assembly was fully signed.

Default: null

Command line option: `/keyfile:filename`

2.14 Log

When this is set before calling Merge, then log messages are written. It is used in conjunction with the LogFile property. If Log is true, but LogFile is null, then log messages are written to Console.Out. To specify this behavior on the command line, the option "/log" can be given without a log file.

Default: false

Command line option: `/log[:logfile]`

2.15 LogFile

When this is set before calling Merge, it indicates the path and filename that log messages are written to. If Log is true, but LogFile is null, then log messages are written to Console.Out.

Default: null

Command line option: `/log[:logfile]`

2.16 Merge

Once all desired options are set, this method performs the actual merging.

2.17 OutputFile

This must be set before calling Merge. It specifies the path and filename that the target assembly will be written to.

Default: null

Command line option: `/out:filename`

2.18 PublicKeyTokens

This must be set before calling Merge. It indicates whether external assembly references in the manifest of the target assembly will use full public keys (false) or public key tokens (true).

Default: true

Command line option: `/useFullPublicKeyForReferences`

2.19 SetInputAssemblies

When used programmatically, each element of the array should contain the path and filename of an input assembly. The first element of the array is considered to be the primary assembly.

2.20 SetSearchDirectories

If specified, this sets the directories to be used to search for input assemblies. When used programmatically, each element of the array should contain a directory name. When specified on the command line, use a separate "/lib" option for each directory.

Command line option: `/lib:`*`directory`*

2.21 SetTargetPlatform

This method sets the .NET Framework for the target assembly to be the one specified by `platform`. Valid strings for the first argument are "v1", "v1.1", "v2", and "v4". The "v" is case insensitive and is also optional. This way ILMerge can be used to "cross-compile", i.e., it can run in one version of the framework and generate the target assembly so it will run under a different assembly. The second argument is the directory in which mscorlib.dll is to be found.

Command line option: `/targetplatform:`*`version`*`,`*`platformdirectory`*

2.22 StrongNameLost

Once merging is complete, this property is true if and only if the primary assembly had a strong name, but the target assembly does not. This can occur when an .snk file is not specified, or if something goes wrong trying to read its contents.

2.23 TargetKind

This controls whether the target assembly is created as a library, a console application or as a Windows application. When it is not specified, then the target assembly will be the same kind as that of the primary assembly. (In that case, the file extensions found on the specified target assembly and the primary assembly must match.) When it is specified, then the file extension of the target assembly must match the specification.

The possible values are ILMerge.Kind.{Dll, Exe, WinExe}

Default: ILMerge.Kind.SameAsPrimaryAssembly

Command line option: `/target:(library|exe|winexe)`

2.24 UnionMerge

When this is true, then types with the same name are all merged into a single type in the target assembly. The single type is the union of all of the individual types in the input assemblies: it contains all of the members from each of the corresponding types in the input assemblies. It cannot be specified at the same time as `/allowDup`.

Default: false

Command line option: `/union`

2.25 Version

When this has a non-null value, then the target assembly will be given its value as the version number of the assembly. When specified on the command line, the version is read in as a string and should look like "6.2.1.3" (but without the quote marks). The version must be a valid assembly version as defined by the attribute AssemblyVersion in the System.Reflection namespace.

Default: null

Command line option: `/ver:`*`version`*

2.26 XmlDocumentation

This property controls whether XML documentation files are merged to produce an XML documentation file for the target assembly.

Default: false

Command line option: *`/xmldocs`*

3 Command Line Usage

The full command line for ILMerge is:

```
ilmerge [/lib:directory]* [/log[:filename]] [/keyfile:filename
[/delaysign]] [/internalize[:filename]]
[/t[arget]:(library|exe|winexe)] [/closed] [/ndebug] [/ver:version]
[/copyattrs [/allowMultiple]] [/xmldocs] [/attr:filename]
([/targetplatform:<version>[,<platformdir>]]|v1|v1.1|v2|v4)
[/useFullPublicKeyForReferences] [/zeroPeKind] [/wildcards]
[/allowDup[:typename]]* [/allowDuplicateResources] [/union] [/align:n]
/out:filename <primary assembly> [<other assemblies>...]
```

All options that take arguments can use either ":" or "=" as a separator. Options can be in any order, but all of the options must precede the list of input assemblies.

4 Troubleshooting

4.1 Input assembly not merged in correctly

A common problem is that after merging some assemblies, you get an error message stating that an input assembly was not merged in correctly because it is still listed as an external reference in the merged assembly. The most common cause of this problem is that one of the input assemblies, B, has an external reference to the incorrectly merged assembly, A, and also an external reference to another assembly, C, that itself has an external reference to A. Suppose the reference to C is to a method that takes an argument whose type is defined in A. Then the type signature of the method call to the method in C still refers to the type from A even in the merged assembly. After all, ILMerge cannot go and modify the assembly C so that it now depends on the type as it is defined in the output assembly after it has been merged!

The solution is to use the closed option (Section 2.6) to have ILMerge compute the transitive closure of the input assemblies to prevent this kind of “dangling reference”. In the example, that would result in all three assemblies A, B, and C, being merged. There is no way to merge just A and B without there still being an external reference to A.

4.2 Merged assembly causes a runtime error (not present in the unmerged assemblies)

The most frequent cause of problems when executing the target assembly is that ILMerge has no way to merge resources. Therefore resources are just copied over from the input assemblies into the target assembly. If these resources encode references to types defined in the input assemblies, then at runtime your program may fail because the type returned from the resource does not match the type as it is defined in the target assembly. You may even see a message that “type ‘T’ cannot be converted to type ‘T’”. This looks confusing because the messages do not show the assembly that each type is defined in. The actual error is that one type is defined in the input assembly while the other is defined in the target assembly.

I do not know of any way to have ILMerge do the right thing in such cases. There is no general way to decode resources and change any type references they contain.

5 Dependencies

ILMerge is a stand-alone assembly dependent only on the v2.0 .NET Framework. (It actually uses two other assemblies: the assembly that makes up CCI itself, System.Compiler.dll, and AssemblyResolver.dll, which provides a small component for finding and loading assemblies when requested by the CCI Reader. But those assemblies have been merged into ILMerge using ILMerge before it is distributed.

6 Acknowledgements

Without Herman Venter, this tool could not have been written. Not only because it completely depends on CCI to provide the underlying functionality for reading, transforming, and writing IL, but because of all of his help and support.