IVI-3.14: Primary Interop Assembly Specification

August 4, 2014 Edition
Revision 1.3 Draft

Important Information

The Primary Interop Assembly Specification (IVI-3.14) is authored by the IVI Foundation member companies. For a vendor membership roster list, please visit the IVI Foundation web site at `www.ivifoundation.org`.

The IVI Foundation wants to receive your comments on this specification. You can contact the Foundation through the web site at `www.ivifoundation.org`.

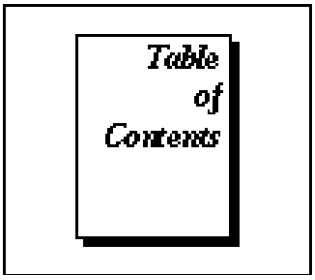
Warranty

The IVI Foundation and its member companies make no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The IVI Foundation and its member companies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Trademarks

Product and company names listed are trademarks or trade names of their respective companies.

No investigation has been made of common-law trademark rights in any work..



Primary Interop Assembly Specification

Primary Interop Assembly Revision History

This section is an overview of the revision history of the Primary Interop Assembly specification.

Table 1. Primary Interop Assembly Specification Revisions

Revision Number	Date of Revision	Revision Notes
Revision 0.1	August, 2002	Original draft.
Revision 0.5	January, 2003	Issues and action items removed from text though they all appear in their own sections..
Revision 0.6	February, 2003	Changes based on discussions at the IVI meeting in Jan. 2003. Also added a section 5 to capture some resolved issues which don't fit well elsewhere.
Revision 0.7	February, 2003	The format of the xml file changed because we added the namespace option to tlbimp. Changed batch file. Resolved all know issues.
Revision 0.8	March, 2003	Draft submitted to Technical Committee for final approval.
Revision 1.0	March, 2003	First approved version.
Revision 1.1	March, 2008	Update obsolete references. Add information about creating policy files. Add information about tools for creating interop and policy files. Allow manual creation of PIAs. Move "how-to" information to appendices. Editorial change to update the IVI Foundation contact information in the Important Information section to remove obsolete address information and refer only to the IVI Foundation web site.
Revision 1.2	November 17, 2008	Add support for 64-bit drivers.
Revision 1.3	August 4, 2014	Specify that PIAs be registered in a manner equivalent to using "regasm <PIA>" rather than "regasm <PIA> /codebase".

1 Overview of the Primary Interop Assembly Specification

1.1 Introduction

This document provides detailed documentation on components supplied by the IVI Foundation to facilitate the use of IVI-COM drivers on the Microsoft .NET Framework development platform. It also provides information on what an IVI-COM driver supplier is required to include with the driver to facilitate that driver's use on the Microsoft .NET Framework development platform.

1.2 Primary Interop Assembly Overview

To use a COM server in the Visual Studio .NET programming environment, the programmer adds a reference to the server. If a primary interop assembly (PIA) is not registered for the COM server, the development environment creates an interop assembly for the project. If the COM server has a registered PIA, the development environment uses that PIA.

By supplying a set of PIAs, the IVI Foundation ensures that .NET programmers are using the same PIAs and not inadvertently introducing type incompatibilities into their programs. A fundamental rule of .NET is that a type is scoped by the assembly that defines the type. A type with the same name and definition that is defined in two different assemblies is considered to be two different types by the .NET runtime. By following the rules for generating a single PIA for an IVI-COM driver, a driver supplier gives the programmer the same benefits for that driver.

1.3 References

Several other documents and specifications are related to this specification. These other related documents are as follows:

- IVI Charter Document
- IVI-3.1: Driver Architecture Specification
- IVI-5: Glossary

1.4 Definition of Terms and Acronyms

Primary Interop Assembly – A .NET assembly which contains type information about a COM server. It is abbreviated PIA. Primary interop assemblies are provided by the same publisher as the type library they describe, and provide the official definitions of the types defined with that type library. Primary interop assemblies are always signed by their publisher to ensure uniqueness. Avoid using interop assemblies that are not primary interop assemblies.

IntelliSense File – An xml file which contains help information about the contents of a PIA. Its name must be the same as the PIA except with a .xml suffix.

Publisher Policy File – A publisher policy file is an XML-based assembly, created by a component developer and released with a component upgrade, that sets policy on the version of the component that should be used. The file redirects assembly binding to use the assembly shipped with the publisher policy file. You can configure an application to ignore the policy file and use the version of a component that the application was built with.

1.5 Implementation

The current installation packages for the IVI Foundation Common Components, including the primary interop assemblies, is available from the IVI Foundation website at http://www.ivifoundation.org.

2 IVI Common Components

The IVI Foundation supplies two sets of common components. The first is the IVI Shared Components, which includes the primary interop assemblies for the instrument class type libraries and several IVI-COM servers. The second is the IVI VISA Shared Components, which includes the primary interop assembly for VISA COM.

2.1 IVI Keys

The IVI Foundation created a key pair using the Strong Name tool supplied by Microsoft as part of the Microsoft .NET Framework. The actual command used was:

```
sn –k IviPublicPrivate.snk
```

The file containing the private and public keys is kept in a secure location by the IVI Foundation members. These keys are not used by IVI-COM driver developers. Only those developing IVI shared components need access to these keys.

2.2 Namespaces and File Names

PIA namespaces for IVI instrument classes shall be constructed as follows, where <*class*> is the IVI class name without the leading Ivi.

```
Ivi.<class>.Interop
```

PIA namespaces for other COM components are Ivi.Driver.Interop, Ivi.ConfigServer.Interop, and Ivi.SessionFactory.Interop, and Ivi.Visa.Interop.

PIA filenames for IVI instrument classes and other IVI-COM components shall be the PIA namespace followed by ".dll", as follows:

```
Ivi.<componentName>.Interop.dll
```

2.3 Creating PIAs With TlbImp.exe

PIAs may be created manually, or they may be created using the TlbImp.exe utility. When using TlbImp.exe to create PIAs for IVI shared components, certain conventions must be observed.

After January 1, 2009, IVI Shared Component PIAs and IVI VISA Shared Component PIAs shall be created using TlbImp.exe version 2.0.50727, the version distributed with .NET Framework 2.0

The **/primary** option shall be used, since the IVI Foundation is the publisher of the shared component PIAs.

The **/out** option shall be used to determine the PIA assembly file name.

The **/namespace** option shall be used to determine the namespace of all types in the PIA.

The **/keyfile** option shall be used with the IVI public/private key file so that the PIA is strong signed.

If the type library in question references other IVI type libraries, the **/reference** option shall be used with the PIA filename created from the referenced type library.

The **/asmversion** option shall be used to specify the PIA version if the default version created by TlbImp is not correct.

The **/nologo** option may be used to prevent TlbImp from displaying its logo.

The **/machine:x86** option shall be used to create 32-bit PIAs. The **/machine:x64** shall be used to create 64-bit PIAs. In many cases, both of these options may create the same PIA, indicated by the fact that it will

have a Processor Type of "MSIL" rather than "x86" or "x64". If both PIAs are the same, they are entirely implemented in Microsoft Intermediate Language (MSIL), which is not 32-bit or 64-bit specific.

The **/sysarray** option shall not be used because none of the instrument class type libraries have any multi-dimensional arrays. Omitting this option allows arrays to retain their fundamental type, such as double, rather than becoming the more generic type of System.Array.

Refer to Appendix A for more information on creating PIAs with TlbImp.exe

2.4 Creating PIAs Manually

PIAs may be created manually if the resulting PIA includes the manual equivalents of the TlbImp.exe options specified above. Manually creating PIAs allows the developer more flexibility, but also requires specialized expertise. At the time of writing, all IVI Shared Component PIAs and IVI VISA Shared Component PIAs are created using TlbImp.exe.

2.5 IntelliSense Files

For each PIA, there shall be a corresponding IntelliSense file. These files have the same name as the PIA except their suffix is .xml.

2.6 Publisher Policy Files

When new versions of IVI PIAs are intended to be backwards compatible with older versions, publisher policy files must be created to redirect references to the older version of the PIA to the newer version. The publisher policy file specifies assembly redirection and code base settings, and uses the same format as an application configuration file. The publisher policy file is compiled into an assembly and placed in the global assembly cache. Refer to Appendix A for information on creating publisher policy files.

The following filename convention shall be used for publisher policy file names: <*namespace*>.config.

The following filename convention shall be used for publisher policy assembly file names: policy.<*major version*>.<*minor version*>.<*namespace*>.dll.

A separate policy file shall be provided for each previous major/minor version being redirected to the current version. For example, suppose that the IVI Foundation previously released version 1.0 and 1.5 of a component, and is now releasing version 1.6. Version 1.6 is backwards compatible with 1.0 and 1.5, and replaces both of them. In this scenario, both of the following files are provided:

- `Policy.1.0.<namespace>.dll`
- `Policy.1.5.<namespace>.dll`

2.6.1 Configuration Server Example

When the IVI Foundation approved version 1.6 of the Configuration Server type library, version 1.0 of the PIA was no longer distributed. In order to allow programs that had been built with references to version 1.0 of the Configuration Server PIA to continue to work without being rebuilt, the IVI Foundation provided a publisher policy assembly to redirect references to version 1.0 of the PIA to version 1.6. The file name of this assembly is:

`Policy.1.0.Ivi.ConfigServer.Interop.dll`

See Section Appendix B, Creating Publisher Policy Files, for more information on creating publisher policy assemblies.

2.6.2 VISA COM Example

When the IVI Foundation approved version 3.0 of the VISA COM type library, version 1.0 of the PIA was no longer distributed. In order to allow programs that had been built with references to version 1.0 of the

VISA COM PIA to continue to work without being rebuilt, the IVI Foundation provided a publisher policy assembly to redirect references to version 1.0 of the PIA to version 3.0. The file name of this assembly is:

`Policy.1.0.Ivi.Visa.Interop.dll`

Refer to Appendix B, Creating Publisher Policy Files for more information related to publisher policy files and assemblies.

2.6.3 Linker Versions for Policy Files

IVI policy files shall be created using version 8.00.50727 of al.exe. This is the version of the linker distributed with version 2.0 of the .NET framework.

2.7 PIA Registration

2.7.1 Shared Component Registration

The IVI and VISA Shared Components shall install each PIA in a manner that is equivalent to running the following command:

`gacutil –I <PIA>`

The IVI and VISA Shared Components shall register each PIA with the corresponding COM type library and COM class in a manner that is equivalent to running the following command:

`regasm <PIA>`

2.7.2 Registration file

The file `IviPiaRegistration.bat` is provided by the IVI Shared Components in the `<IVIStandardRootDir>\Bin\Primary Interop Assemblies` directory. The file `VisaPiaRegistration.bat`, which is identical, is also provided by the IVI VISA Shared Components in the `<VXIPNPPATH>\VisaCom\Primary Interop Assemblies` directory. This file enables users to install PIAs in the GAC and register them properly after shared components are installed, if necessary.

The batch file runs these commands:

```
gacutil –i <PIA>
regasm <PIA>
```

for each PIA in the directory.

The contents of `IviPiaRegistration.bat` are:

```
@Rem This batch file puts all Primary Interop Assembly in this directory
@Rem into the Global Assembly Cache using gacutil. It assumes every dll
@Rem is a PIA. It also runs regasm on them if Visual Studio .NET is installed.


@Rem Run this batch file if the .NET Framework or Visual Studio .NET is
@Rem installed after the IVI Shared Components are installed.


@Rem The first batch parameter can specify a directory different from
@Rem the current directory.
set dir=%~1
```

```
if "%~1"=="" set dir="."

for %%f in ("%dir%\*.Interop.dll") do gacutil -nologo -i "%%f"

if not defined VSCOMNTOOLS goto end
@Rem Running regasm allows the development environment to properly add a
@Rem reference to the assembly and to make IntelliSense work.
@Rem
@Rem The environment variable, VSCOMNTOOLS, is defined by the Visual
@Rem Studio .NET installation. If it's defined, that directory contains
@Rem a batch file, vsvars32.bat,which sets the PATH so reagsm can execute.
 call %VSCOMNTOOLS%vsvars32.bat
 for %%f in ("%dir%\Ivi.*.Interop.dll") do regasm -nologo"%%f"
:end
```

2.8 Legacy PIA Considerations for Shared Components

2.8.1 Legacy Tlbimp Considerations

In one limited scenario, PIAs built with TlbIMp.exe version 1.0 or 1.1 may be required. This scenario may be described as follows.

1. An older (pre-2.0) version of IVI Shared Components is installed
2. A user program is built which references both the .NET 1.0 or 1.1 CLR and one or more of the IVI-COM PIAs.
3. .NET version 2.0 or later, and version 2.0 of the IVI Shared Components are installed, and the existing 1.0 or 1.1 versions of .NET are not uninstalled.
4. The user program is not rebuilt.

In this case, the user program will load .NET 1.0 or 1.1 into the process, and after that, the process will try to run every .NET component with the .NET 1.0 or 1.1 runtime. The program will fail when it tries to access any PIAs built with TlbImp 2.0, since the .NET 1.0 or 1.1 runtimes cannot execute .NET 2.0 code.

To eliminate this error, which could affect .NET users who upgrade to IVI Shared Components 2.0 or later, the IVI Shared Components shall install all PIAs delivered with IVI Shared Components 1.5.1 into the GAC. As this is sufficient to address the scenario described above, these PIAs shall not be installed into the IVI PIA directory.

Likewise, the IVI VISA Shared Components shall install all PIAs delivered with IVI VISA-COM Common Components 1.5.1 into the GAC. As this is sufficient to address the scenario described above, these PIAs shall not be installed into the VISA-COM PIA directory.

2.8.2 Legacy Regasm Considerations

Prior to 2014, this specification indicated that IVI and VISA Shared Components would register each PIA with the corresponding COM type library and COM classes in a manner that is equivalent to running the following command:

```
regasm <PIA> /codebase
```

Regasm with the `/codebase` option adds the `PrimaryInteropAssemblyCodeBase` registry value to the COM type library registry key and the CodeBase registry value to subkeys of the CLSID\<GUID> key for classes defined in the type library. The issue is with the type library PIA codebase value, however. This registry value is the physical PIA file path of a PIA for the COM type library. In the case of the IVI and VISA shared components, the value prior to 2014 was the location of the 32-bit version of the PIA. This worked for both 32-bit and 64-bit development in Visual Studio 2005 and 2008. However, this prevents 64-bit development in Visual Studio 2010, 2012, and 2013 because these versions of Visual Studio require references to a PIA of the same bitness as the current project type.

As a result, the type library `PrimaryInteropAssemblyCodeBase` registry value will no longer be added to IVI and VISA shared component type library keys with the release of version 2.3 of the IVI Shared Components and version 5.4 of the VISA Shared Components. Without this key, all versions of Visual Studio resolve references to the PIA by using the GAC. This means that the reference is automatically resolved with the correct bitness of PIA from the GAC. However, the PIA IntelliSense help files are not (and should not) be installed to the GAC, with the resulting downside that this help is not available to developers in Visual Studio IntelliSense. In addition, the CodeBase registry value to subkeys of the CLSID\<GUID> key for classes defined in the type library will also be omitted. The net effect is that IVI and VISA Shared Components now register each PIA with the corresponding COM type library and COM classes in a manner that is equivalent to running the following command:

```
regasm <PIA>
```

3 IVI-COM Drivers

An IVI-COM driver supplier follows essentially the same steps the IVI Foundation took to create a PIA for IVI-COM driver. An IVI-COM driver is not required to include a PIA in its installation, though its inclusion is recommended.

3.1 Keys

The driver supplier creates a key pair for its organization using the Strong Name tool. The driver supplier keeps the private key secret and secure and publishes the public key. The same keys can and should be used to sign all the IVI-COM driver primary interop assemblies supplied by the organization. The command executed should look like:

```
sn –k <keyfile>.snk
```

where <keyfile> is an appropriate name for the organization.

The public key can be extracted by executing the command:

```
sn –p <keyfile>.snk <keyfile>Public.snk
```

The public key is stored in the file, <keyfile>Public.snk.

3.2 Namespaces and File Names

PIA namespaces for IVI instrument drivers shall be constructed as follows, where <*driver*> is the name of the driver and <*vendor*> is the name of the driver vendor.

```
<vendor>.<driver>.Interop
```

PIA filenames for IVI instrument drivers shall be the PIA namespace followed by ".dll", as follows:

```
<vendor>.<driver>.Interop.dll
```

3.3 Creating PIAs With TlbImp.exe

PIAs may be created manually, or they may be created using the TlbImp.exe utility. When using TlbImp.exe to create PIAs for IVI-COM drivers, certain conventions must be observed.

After January 1, 2009, if a 64-bit IVI-COM driver is provided, the PIAs for both the 32-bit and 64-bit drivers shall be created using TlbImp.exe version 2.0.50727 (delivered with .NET 2.0). Otherwise, TlbImp.exe version 1.0.3705 (.Net 1.0), 1.1.4322 (.Net 1.1), or 2.0.50727 may be used.

The **/primary** option shall be used, since the driver vendor is the publisher of the driver PIAs.

The **/out** option shall be used to determine the PIA assembly file name.

The **/namespace** option shall be used to determine the namespace of all types in the PIA.

The **/keyfile** option shall be used with the IVI public/private key file so that the PIA is strong signed.

If the type library in question references other IVI type libraries, the **/reference** option shall be used with the PIA filename created from the referenced type library.

The **/asmversion** option shall be used to specify the PIA version if the default version created by TlbImp is not correct.

The **/nologo** option may be used to prevent TlbImp from displaying its logo.

If TlbImp 2.0.50727 is used, the **/machine:x86** option shall be used to create 32-bit PIAs. The **/machine:x64** shall be used to create 64-bit PIAs. In many cases, both of these options may create the same PIA, indicated by the fact that it will have a Processor Type of "MSIL" rather than "x86" or "x64". If

both PIAs are the same, they are entirely implemented in Microsoft Intermediate Language (MSIL), which is not 32-bit or 64-bit specific.

The **/sysarray** option shall not be used because none of the instrument class type libraries have any multi-dimensional arrays. Omitting this option allows arrays to retain their fundamental type, such as double, rather than becoming the more generic type of System.Array.

Running `tlbimp` with these options requires that the IVI PIAs be properly installed and registered as done by the IVI Shared Components installer. The driver must have a reference to `IviDriverLib` and likely has a reference to an IVI instrument class. These references are resolved by `tlbimp` when the IVI PIAs are registered and in the gac. Alternatively, the **/reference:*filename*** option can be used to resolve references.

Refer to Appendix A for more information on creating PIAs with TlbImp.exe

3.4 Creating PIAs Manually

PIAs may be created manually if the resulting PIA includes the manual equivalents of the TlbImp.exe options specified above. Manually creating PIAs allows the developer more flexibility, but also requires specialized expertise.

3.5 Publisher Policy Files

When new versions of IVI PIAs are intended to be backwards compatible with older versions, publisher policy files must be created to redirect references to the older version of the PIA to the newer version. The publisher policy file specifies assembly redirection and code base settings, and uses the same format as an application configuration file. The publisher policy file is compiled into an assembly and placed in the global assembly cache. Refer to Appendix A for information on creating publisher policy files.

The following filename convention shall be used for publisher policy file names: <*namespace*>.config.

The following filename convention shall be used for publisher policy assembly file names: policy.<*major version*>.<*minor version*>.<*namespace*>.dll.

A separate policy file is required for each previous major/minor version being redirected to the current version. For example, suppose that a vendor previosuly released version 1.0 and 1.5 of a driver, and is now releasing version 1.6. Version 1.6 is backwards compatible with 1.0 and 1.5, and replaces bth of them. In this scenario, both of the following files must be proveded:

- `Policy.1.0.<namespace>.dll`
- `Policy.1.5.<namespace>/dll`

3.5.1 Linker Versions for Policy Files

IVI driver policy files shall be created using the version of al.exe that corresponds to the version of TlbImp.exe used to create the PIA. For example, if version 2.0.50727 of TlbImp was used to create the PIA, version 8.00.50727 of al.exe must be used to create any policy files that apply to that PIA. The following table shows the versions of al.exe that correspond to the versions of TlbImp.exe mentioned in Section 3.3.

Table 3.1: Allowed versions of TlbImp.exe and al.exe.

.NET Framework Version	TlbImp.exe Version	al.exe Version
1.0	1.0.3705	7.0.9466
1.1	1.1.4322	7.0.9466
2.0-3.5	2.0.50727	8.0.50727

3.6 IntelliSense File

If an IVI-COM driver's installation includes a PIA, it shall also include an IntelliSense file. The name of the file shall be <vendor>.<driver>.Interop.xml.

The format of the IntelliSense file shall be the same as that produced by the C# compiler when run with the /doc option.

This file may be created using the C# compiler, by editing the xml directly, or through the use of other tools. Those tools are beyond the scope of this specification.

The file shall always have this form:

```
<doc>
   <assembly>
      <name>NameSpace</name>
   </assembly>
   <members>
      <member name="T: NameSpace.InterfaceName">
         <summary>Description of the interface</summary>
      </member>
      <member name="M: NameSpace.InterfaceName.MethodName">
         <summary>Description of the method</summary>
      </member>
      <member name="P: NameSpace.InterfaceName.PropertyName">
         <summary>Description of the property</summary>
      </member>
      <member name="T: NameSpace.EnumerationName">
         <summary>Description of the enumeration</summary>
      </member>
      <member name="F: NameSpace.EnumerationName.EnumerationValue">
         <summary>Description of the enumeration value</summary>
      </member>
   <members>
<doc>
```

NameSpace is the same as the value given to the namespace option for tlbimp and the name of PIA and IntelliSense files without a suffix. The case of all the letters in NameSpace must match exactly with those in the namespace option value.

The <members> block shall contain a <member> entry for every interface, enumeration, enumeration value, method, and property defined in the driver's instrument specific portion of its type library. In a <member> entry, the name string for an interface or enumeration has a "T:" prefix; a method has an "M:" prefix; a property has a "P:" prefix; an enumeration value has a "F" prefix. For methods and properties, the "InterfaceName" portion of the string is the actual name of the interface which contains the method or property.

The <member> entry for a method may also contain <param> or <returns> blocks. For each parameter to the method there is a <param> entry of the form:

```
<param name="ParameterName">Description of parameter.</param>
```

When a method has one or more parameters, the method's name string shows the parameters' types. It might look like:

```
<member name="M: NameSpace.InterfaceName.MethodName(System.String)">
```

If the type returned by the method is something other than void, the <member> block contains a <returns> entry which look like:

```
<returns>Description of value returned.</returns>
```

The complete <member> block for a method with a parameter which returns a value might look like:

```
<member name="M: NameSpace.InterfaceName.CountChar(System.String)">
    <summary>Counts the number of characters in a string</summary>
    <param name="StringToCount">A string parameter.</param>
```

```
    <returns>Number of characters in StringToCount.</returns>
</member>
```

The IntelliSense files distributed with the IVI PIAs provide examples.

3.7 PIA Registration

If the Microsoft .NET Framework is installed after the driver PIAs are installed, the user will need to run these commands:

```
gacutil –i <PIA>
regasm <PIA>
```

for each PIA in the directory. Driver developers should install a batch file to do this task if the `IviPiaRegistration.bat` file is not sufficient.

3.8 Legacy PIA Considerations for Drivers

3.8.1 Legacy Tlbimp Considerations

In one limited scenario, PIAs built with TlbIMp.exe version 1.0 or 1.1may be required. This scenario may be described as follows.

1. A version of the IVI-COM driver is installed, with PIAs built with TlbImp 1.0 or 1.1.
2. A user program is built which references both the .NET 1.0 or 1.1 CLR and the IVI-COM driver PIA.
3. .NET version 2.0 or later is installed, and a newer version of the IVI-COM driver whose PIAs were built with TlbImp 2.0 is installed, and the existing 1.0 or 1.1 versions of .NET are not uninstalled.
4. The user program is not rebuilt.

In this case, the user program will load .NET 1.0 or 1.1 into the process, and after that, the process will try to run every .NET component with the .NET 1.0 or 1.1 runtime. The program will fail when it tries to access the PIAs built with TlbImp 2.0, since the .NET 1.0 or 1.1 runtimes cannot execute .NET 2.0 code.

To eliminate this error, which could affect .NET users who upgrade to IVI Shared Components 2.0 or later, the IVI-COM driver installer should install the last deployed PIA built with TlbImp 1.0 or 1.1 into the GAC. As this is sufficient to address the scenario described above, the PIA shall not be installed into the IVI PIA directory.

3.8.2 Legacy Regasm Considerations

Prior to 2014, this specification recommended that IVI-COM driver installers register each driver PIA with the corresponding COM type library in a manner that is equivalent to running the following command:

```
regasm <PIA> /codebase
```

Regasm with the `/codebase` option adds the `PrimaryInteropAssemblyCodeBase` registry value to the COM type library registry key and the CodeBase registry value to subkeys of the CLSID\<GUID> key for classes defined in the type library. The issue is with the type library PIA codebase value, however. This registry value is the physical PIA file path of a PIA for the COM type library. In the case of ICI-COM drivers, the value prior to 2014 was the location of the 32-bit version of the PIA. This worked for both 32-bit and 64-bit development in Visual Studio 2005 and 2008. However, this prevents 64-bit development in Visual Studio 2010, 2012, and 2013 because these versions of Visual Studio require references to a PIA of the same bitness as the current project type.

As a result, after January 1, 2015 the `PrimaryInteropAssemblyCodeBase` registry value shall no longer be added to type library keys for IVI-COM drivers. Without this key, all versions of Visual Studio resolve references to the PIA by using the GAC. This means that the reference is automatically resolved with the correct bitness of PIA from the GAC. However, the PIA IntelliSense help files are not (and should not) be installed to the GAC, with the resulting downside that this help is not available to developers in Visual Studio IntelliSense. In addition, the CodeBase registry value to subkeys of the CLSID\<GUID> key for classes defined in the type library will also be omitted. The net effect is that IVI-COM drivers will now register each PIA with the corresponding COM type library and COM classes in a manner that is equivalent to running the following command:

```
regasm <PIA>
```

Drivers created prior to January 1, 2015 may continue to register PIAs as previously allowed by this specification.

4 Additional Considerations

The .NET development and runtime environments necessitated some additional rules for IVI primary interop assemblies.

4.1 Adding References

In .NET installations, the registry contains a key HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/.NETFramework/AssemblyFolders. This key contains sub-keys whose values are directory paths. When a programmer adds a reference to an assembly, the application development environment search all these directories for assemblies and lists what it finds under the .NET tab. The COM tab lists all the COM servers installed on the computer.

Since the IVI primary interop assemblies are for COM, they appear only under the COM tab. At some future time the IVI Foundation expects to describe how to build native .NET drivers. Those assemblies would appear under the .NET tab. Also, when a reference is added under the COM tab, additional PIAs needed to resolve references are automatically added. When done under the .NET tab, these references are not added and the user must add them individually

An installer, whether for IVI Shared Components, VISA-COM, or an IVI driver, shall not add a registry entry under AssemblyFolders.

4.2 PIAs in Two Places

The IVI PIAs, VISA-COM PIA, and IVI Driver PIAs are all installed into both the global assembly cache and the directory specified in IVI-3.17, *Installation Requirements Specification*. If Visual Studio references are added using the .NET tab in the Add References dialog, references to the PIA at development time will be resolved from the GAC. In this case the XML IntelliSense comments will not be visible to developers in IntelliSense. If Visual Studio references are added using the Browse tab in the Add References dialog, references to the PIA at development time will be resolved to the particular file referenced. In this case the XML IntelliSense comments will be visible. In addition, if someone builds a makefile and uses CSC directly, the reference to a PIA must be done via a directory.

Appendix A: Creating PIAs

The process of creating primary interop assemblies can be difficult to track down in Microsoft documentation. This appendix is provided as an overview of the process, and to give developers the information they need to quickly track down more information in the documentation. It also provides IVI specific examples and considerations.

A.1 TlbImp

From the Microsoft documentation on tlbimp.exe:

The Type Library Importer converts the type definitions found within a COM type library into equivalent definitions in a common language runtime assembly. The output of tlbimp.exe is a binary file (an assembly) that contains runtime metadata for the types defined within the original type library.

The **/keyfile:*filename*** option signs the resulting assembly with a strong name using the publisher's official public/private key pair found in *filename*.

The **/primary** option produces a primary interop assembly for the specified type library. Information is added to the assembly indicating that the publisher of the type library produced the assembly. By specifying a primary interop assembly, a publisher's assembly is differentiated from any other assemblies that are created from the type library using tlbimp.exe. The **/primary** option should be used only by the publisher of the type library imported with tlbimp.exe. Note that a primary interop assembly must be signed with a strong name.

The **/reference:*filename*** option specifies the assembly file to use to resolve references to types defined outside the current type library.

/namespace:*namespace* option specifies the namespace in which to produce the assembly.

/asmversion:*version* option specifies the version of the assembly. This is optional, and is needed only if the default version created by TlbImp is not correct.

/nologo is needed only to prevent TlbImp from displaying its logo.

For TlbImp 2.0 or higher, the **/machine:x86** option specifies 32-bit PIAs. The **/machine:x64** option specifies 64-bit PIAs. In many cases, both of these options may create the same PIA, indicated by the fact that it will have a Processor Type of "MSIL" rather than "x86" or "x64". If both PIAs are the same, they are entirely implemented in Microsoft Intermediate Language (MSIL), which is not 32-bit or 64-bit specific.

/out:*filename* option specifies the name of the output file, assembly, and namespace in which to write the metadata definitions. The **/out** option has no effect on the assembly's namespace if the type library specifies the Interface Definition Language (IDL) custom attribute that explicitly controls the assembly's namespace. If you do not specify this option, Tlbimp.exe writes the metadata to a file with the same name as the actual type library defined within the input file and assigns it a .dll extension. If the output file is the same name as the input file, the tool generates an error to prevent overwriting the type library."

/sysarray option specifies to the tool to import a COM style SafeArray as a managed System.Array Class type.

A.2 TlbImp Examples

A.2.1 IVI Driver PIA

```
tlbimp IviDriverTypeLib.dll /machine:x86 /keyf:IviPublicPrivate.snk
/namespace:Ivi.Driver.Interop /pr /nologo /out:Ivi.Driver.Interop.dll
```

Similar commands would be used to create PIAs for VISA COM, the IVI Configuration Server, and the IVI-COM Session Factory.

A.2.2 Instrument Class PIAs

When creating PIAs for instrument class type libraries, the command must be modified to include references to the IVI Driver PIA. As an example, the command used to create a 64-bit PIA for the IVI DMM class type library is

```
tlbimp IviDmmTypeLib.dll /machine:x64 /keyf:IviPublicPrivate.snk
/namespace:Ivi.Dmm.Interop /pr /nologo /out:Ivi.Dmm.Interop.dll
/ref:Ivi.Driver.Interop.dll
```

A.2.3 Specifying PIA Version

When the assembly version default is not correct, use the /asmversion key to specify the correct version. As an example, the command used to create the 32-bit PIA for version 1.6.0.0 of the Configuration Server is

```
tlbimp IviConfigServer.dll /machine:x86 /keyf:IviPublicPrivate.snk
/namespace:Ivi.ConfigServer.Interop /pr /nologo /asmversion:1.6.0.0
/out:Ivi.ConfigServer.Interop.dll
```

A.2.4 Instrument Driver PIAs

The type library for an IVI-COM driver is contained in the driver's executable file. The vendor creates the 64-bit PIA by executing the command:

```
tlbimp <driver>.dll /machine:x64 /keyfile:<keyfile>.snk /primary
/namespace:<vendor>.<driver>.Interop /out:<vendor>.<driver>.Interop.dll
```

or

```
tlbimp <driver>.dll /machine:x64 /keyfile:<keyfile>.snk /primary
/asmversion:w.x.y.z /namespace:<vendor>.<driver>.Interop
/out:<vendor>.<driver>.Interop.dll
```

where <driver> is the name of the driver, <vendor> is the name of the vendor, <keyfile> is the name of the vendor's pubic/private key file, and w.x.y.z is the correct version for the PIA, with w being the type library major version, and x being the type library minor version.

Appendix B: Creating Publisher Policy Files

The process of creating publisher policy files can be difficult to track down in Microsoft documentation. This appendix is provided as an overview of the process, and to give developers the information they need to quickly track down more information in the documentation.

B.1 Publisher policy files Creating the Publisher Policy File.

Publisher policy files are XML files, formatted according to a schema defined by Microsoft. The following example shows a publisher policy file that redirects any version of the Configuration Server between 1.0 and 1.5 to 1.6.

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <dependentAssembly>
            <assemblyIdentity name="Ivi.ConfigServer.Interop"
                              publicKeyToken="a128c98f1d7717c1"
                              culture="neutral" />
            <!-- Redirecting to version 1.6.0.0 of the assembly. -->
            <bindingRedirect oldVersion="1.0.0.0 - 1.5.65535.65535"
                             newVersion="1.6.0.0"/>
         </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

The assembly identity name (which corresponds to the namespace of the PIA), public key token, and culture identify a specific assembly installed in the GAC. The public key token shown is the IVI public key token.

B.2 Create a publisher policy assembly.

Publisher policy assemblies are DLLs created from publisher policy files using the linker. For example, to create the 64-bit assembly for the Configuration Server publisher policy file in the previous section, the command is

```
al /link:Ivi.ConfigServer.Interop.config /keyfile:IviPublicPrivate.snk
/v:1.6.0.0 /platform:x64 /out:policy.1.0.Ivi.ConfigServer.Interop.dll
```

The IVI filename convention for publisher policy assembly file names (policy.<*major version*>.<*minor version*>.<*namespace*>.dll) is also the required Microsoft convention.

The major/minor revision in the name is actually used, so that the file created by the above command will only redirect version(s) 1.0.x.x of the Configuration Server PIA to version 1.6.0.0. To redirect other versions, you must create other publisher policy assemblies. They can be created from the same publisher policy .config file, but will only be applied to the major/minor versions of the PIAs referenced in the file name.

The /platform option is only available with version 8.0. 50727 of al.exe (distributed with .NET 2.0) or later.

B.3 Add the publisher policy assembly to the global assembly cache.

As for PIAs, publisher policy assemblies are added to the GAC using gacutil. It is not necessary to run regasm, since the publisher policy assemblies do not need to appear as references to developers.

The command is simple.

```
gacutil –i <Publisher Policy Assembly>
```

Before this command is run, make sure that both the publisher policy .config file and the publisher policy assembly are in the same directory. The publisher policy assembly will not install correctly into the GAC if the publisher policy .config file is not present.