Sample: Using Tlbimp Customization

Table of Contents

[Introduction 2](#_Toc223929237)

[*ChangeManagedName* Functionality 2](#_Toc223929238)

[Getting Started: IComparable 2](#_Toc223929239)

[Sample XML Rule: ChangeManagedNameSample.xml 3](#_Toc223929240)

[Creating a Rule File with *TlbimpRuleFileEditor* 3](#_Toc223929241)

[Tying it Together with *Tlbimp* 3](#_Toc223929242)

[*ResolveTo* Functionality 4](#_Toc223929243)

[Getting Started: IEnumVARIANT 4](#_Toc223929244)

[Sample XML Rule: IEnumVARIANT.xml 5](#_Toc223929245)

[Creating a Rule File with *TlbimpRuleFileEditor* 5](#_Toc223929246)

[Tying it Together with *Tlbimp* 6](#_Toc223929247)

[*AddAttribute* Functionality 6](#_Toc223929248)

[Getting Started: IOldInterface 6](#_Toc223929249)

[Sample XML Rule: AddAttributeSample.xml 7](#_Toc223929250)

[Creating a Rule File with *TlbimpRuleFileEditor* 7](#_Toc223929251)

[Tying it Together with *Tlbimp* 8](#_Toc223929252)

[*PreserveSig* Functionality 9](#_Toc223929253)

[Getting Started: FunctionNoThrow 9](#_Toc223929254)

[Sample XML Rule: PreserveSig.xml 10](#_Toc223929255)

[Creating a Rule File with *TlbimpRuleFileEditor* 10](#_Toc223929256)

[Tying it Together with *Tlbimp* 10](#_Toc223929257)

[*ConvertTo* Functionality 11](#_Toc223929258)

[Getting Started: BinarySearch 11](#_Toc223929259)

[Sample XML Rule: ConvertToSample.xml 12](#_Toc223929260)

[Creating a Rule File with *TlbimpRuleFileEditor* 12](#_Toc223929261)

[Tying it Together with *Tlbimp* 13](#_Toc223929262)

# Introduction

The new type library importer tool (Tlbimp) allows you to customize the interop assembly generation result by feeding an xml config file (also called rule file) through the command line like the following:

> tlbimp ATlb.tlb /config:AXMLConfigFile.xml

So far, the customization enables the several scenarios, including ChangeManagedName, ResolveTo, AddAttribute, PreserveSig and ConvertTo etc. The following sections introduce how to use these functionalities one by one by walking through different samples.

# *ChangeManagedName* Functionality

In the conversion process, Tlbimp gives every type in type library a managed name. At most of time, the naming rule is simple, that is the managed name follows the type name in type library. If the name doesn’t meet your needs, you can also use **ChangeManagedName** action to specify the exact managed name you want for the particular type in type library. Actually, whenever you want to give a customized managed name for a type in the type library, using ChangeManagedName action is recommended. It is also very useful when you have name collision issues. Because CLR uses Guid to find the right type in COM, changing the type’s managed name will not cause type resolution issue.

## Getting Started: IComparable

By default, the Tlbimp tool generates the managed names following their type name in type library. In the sample “ChangeManagedNameSample.tlb” attached with the tool, we have an interface named “IComparable”, which will be converted to managed type “ChangeManagedNameSample.IComparable”. While, there is a well known interface named “System.IComparable” in the mscorlib.dll. In this case, if you import System namespace into your managed project (e.g. in C#, it is“using System;” ), which is very common, you have to write the full name, “ChangeManagedNameSample.IComparable” whenever you want use the type. Life will be easier if you change the managed name of “ChangeManagedNameSample.IComparable”. The following content shows you how to use the **ChangeManagedName** action to rename the “IComparable” to “IMyComparable” so as to solve the problem.

To achieve this, you need to create an XML config file (also called rule file) to tell Tlbimp how to perform the customized naming rule. The config file can be created either by hand or by using the UI customization tool TlbimpRuleFileEditor.exe.

## Sample XML Rule: ChangeManagedNameSample.xml

Below is a sample XML config file. Note that Category, Condition and Action are key parts of a rule.

<Rules>

<Rule Name="Rule #1" Category="Type">

<Condition>

<And>

<NativeName Operator="Equal" Value="IComparable" />

</And>

</Condition>

<Action Name="ChangeManagedName">

<Parameter Key="NewName" Value="IMyComparable" />

</Action>

</Rule>

</Rules>

In the above snippet, there is one rule named “Rule #1”, which means

For each type, if its name in the type library is equal to “IComparable”, then a **ChangeManagedName** action is performed. This type’s managed name is changed to “IMyComparable”.

## Creating a Rule File with *TlbimpRuleFileEditor*

As an alternative to crafting and editing the customization XML files by hand, you can use *TlbImpRuleFileEditor.exe* to create and edit rule files. For example, to create a rule file for the ChangeManagedNameSample.xml file, you may perform the following steps:

1. Open “TlbImpRuleFileEditor”; Press “New Rule File” button on the Toolbar to start.
2. Double click on the root node of “Type Library” tree,. An “open file dialog” is shown.
3. Select the “ChangeManagedNameSample.tlb” file in the sample. The content of the selected tlb file is shown in the “Type Library” tree.
4. Drag and drop the “IComparable : Interface” node to the blank space of “Rule File”. An “Add Rule” dialog is shown.
5. Select “ChangeManagedName” for the “Action” combo box, and Press OK. A rule is added in the “Rule File” tree.
6. Right click on the “Action : ChangeManagedName” node, and select “Modify Action…” item. “Change Managed Name Action Dialog” is shown.
7. Input “IMyComparable” in the “New Name” input box, and press “OK”. The “Change Managed Name Action Dialog” is closed and the “New Name” parameter of “Action : ChangeManagedName” is set with “IMyComparable”.
8. Save the file as “ChangeManagedNameSample.xml”

## Tying it Together with *Tlbimp*

Using the XML config file from above, invoke the following on the command line

> tlbimp2 ChangeManagedNameSample.tlb /config:ChangeManagedNameSample.xml

It generates the customized result, in which the managed name of tlb type “IComparable” is specified to “IMyComparable”.

# *ResolveTo* Functionality

In the conversion process, when Tlbimp comes across an unknown type library type, Tlbimp will attempt to resolve this new type to an existing managed type or create a new managed type. If you want Tlbimp to resolve to a different existing managed type, you can use the **ResolveTo** action to specify which managed type to resolve to for this particular type library type. Because many types in the stdole.tlb have corresponding managed type in the namespace “System.Runtime.InteropServices.ComTypes” in mscorlib assembly, this action makes it much easier to redirect resolution. It will also be very useful in the case that two versions of the same type exist.

## Getting Started: IEnumVARIANT

By default, the Tlbimp tool generates the signature “[stdole]stdole.IEnumVARIANT” or “[mscorlib]System.Collections.IEnumerator” for the type library type “IEnumVARIANT”. Since you may not desire this behavior, we will show how to use the **ResolveTo** action to enable the Tlbimp tool to instead generate the signature “[mscorlib]System.Runtime.InteropServices.ComTypes.IEnumVARIANT” for “IEnumVARIANT”.

To achieve this, you need to create an XML config file (also called rule file) to tell Tlbimp how to perform the customized type resolution. The config file can be created either by hand or by using the UI customization tool TlbimpRuleFileEditor.exe.

## Sample XML Rule: IEnumVARIANT.xml

Below is a sample XML config file. Note that Category, Condition and Action are key parts of a rule.

<Rules>

<Rule Name="Rule #1" Category="Type">

<Condition>

<And>

<NativeName Operator="Equal" Value="IEnumVARIANT" />

</And>

</Condition>

<Action Name="ResolveTo">

<Parameter Key="AssemblyName" Value="mscorlib" />

<Parameter Key="ManagedTypeFullName"

Value="System.Runtime.InteropServices.ComTypes.IEnumVARIANT" />

</Action>

</Rule>

</Rules>

In the above snippet, there is one rule named “Rule #1”, which means

For each type, if its name in the type library is equal to “IEnumVARIANT”, then a **ResolveTo** action is performed. This type is resolved to “[mscorlib] System.Runtime.InteropServices.ComTypes.IEnumVARIANT”.

## Creating a Rule File with *TlbimpRuleFileEditor*

As an alternative to crafting and editing the customization XML files by hand, you can use *TlbImpRuleFileEditor.exe* to create and edit rule files. For example, to create a rule file for the IEnumVARIANT.xml file, perform the following steps:

1. Open “TlbImpRuleFileEditor”; Press “New Rule File” button on the Toolbar to start.
2. Right click on the “Untitled” tree node, and select “Add Rule...”.
3. Select “Type” for “Category”, “ResolveTo” for “Action” Combo Box, and press “OK”. A rule is created with an Empty And condition node and Uninitialized ResolveTo action node.
4. Right click on the “Action : ResolveTo” node, and select “Modify Action…”. “ResolveTo Action Wizard” dialog is shown.
5. Press “…” in the wizard. “Managed Type Selector” dialog is shown.
6. Input “mscorlib” in “Assembly Name” text box. Press “Load” button.
7. Input “IEnumVARIANT” in the filter Text Box. Only a few items with substring “IEnumVARIANT” is shown.
8. Double-click on “System.Runtime.InteropServices.ComTypes.IEnumVARIANT” node in the assembly tree. The “Managed Type Selector” dialog is closed. And, the “Assembly Name” and “Managed Type” textboxes in the wizard are filled with information of the selected type.
9. Press OK, in the “ResolveTo Action Wizard”. “Action : ResolveTo” node is updated.
10. Click on “Empty” tree node. The “In place condition editor” is shown.
11. Select “NativeName” in the Combo Box. The editor is expanded.
12. Select “Equal” in the second Combo Box. Input “IEnumVARIANT” in the third Combo Box.
13. Click on the blank space or press Enter to confirm the modification and save the file

## Tying it Together with *Tlbimp*

Using the XML config file from above, invoke the following on the command line

> tlbimp2 IEnumVARIANT.tlb /config:IEnumVARIANT.xml

This should generate the customized result, in which type IEnumVARIANT is resolved to [mscorlib]System.Runtime.InteropServices.ComTypes.IEnumVARIANT.

# *AddAttribute* Functionality

If you want to add some custom attribute to the conversion result of a particular type in type library, you can use the AddAttribute action on this type.

## Getting Started: IOldInterface

In the sample “AddAttributeSample.tlb” attached with the tool, there is an interface named “IOldInterface”, which is an obsolete interface, and is not recommended to use. Therefore, in the managed code, “System.ObsoleteAttribute” is required for this type. However, it’s tricky to modify the interop assembly created by Tlbimp. In this case, to add “System.ObsoleteAttribute” to the type “IOldInterface” at the Tlbimp conversion process, AddAttribute action can help.

To achieve this, you need to create an XML config file (also called rule file) to tell Tlbimp how to add the customized attribute to the particular type. The config file can be created either by hand or by using the UI customization tool TlbimpRuleFileEditor.exe.

## Sample XML Rule: AddAttributeSample.xml

Below is a sample XML config file. Note that Category, Condition and Action are key parts of a rule.

<Rules>

<Rule Name="Rule #1" Category="Type">

<Condition>

<And>

<NativeName Operator="Equal" Value="IOldInterface" />

</And>

</Condition>

<Action Name="AddAttribute">

<Parameter Key="AssemblyName" Value="mscorlib" />

<Parameter Key="TypeName" Value="System.ObsoleteAttribute" />

<Parameter Key="Constructor" Value="Void .ctor(System.String, Boolean)" />

<Parameter Key="Data" Value="01 00 21 50 6C 65 61 73 65 20 75 73 65 20 49 4E 65 77 49 6E 74 65 72 66 61 63 65 20 69 6E 73 74 65 61 64 2E 01 00 00" />

</Action>

</Rule>

</Rules>

In the above snippet, there is one rule named “Rule #1”, which means

For each type, if the type name is equal to “IOldInterface”, then an **AddAttribute** action is performed, which adds an attribute to the type. The parameters in the AddAttribute action tells the details of the attribute. The attribute type is from “mscorlib” assembly. The type name of the attribute is “System.ObsoleteAttribute”. Constructor “Void .ctor(System.String, Boolean)" is used to construct this attribute instance. And the values of the constructor’s parameters are encoded in the byte array, and stored in the “Data” parameter of AddAttribute action. With all the information above, an attribute instance is created and added to the type, whose name is “IOldInterface”.

Notes: The format of byte array in the “Data” parameter is documented in MSDN. You can use the following editor to write the rule file including the encoding of “Data” parameter.

## Creating a Rule File with *TlbimpRuleFileEditor*

As an alternative to crafting and editing the customization XML files by hand, you can use *TlbImpRuleFileEditor.exe* to create and edit rule files. For example, to create a rule file for the AddAttributeSample.xml file, perform the following steps:

1. Open “TlbImpRuleFileEditor”; Press “New Rule File” button on the Toolbar to start.
2. Double click on the root node of “Type Library” tree,. An “open file dialog” is shown.
3. Select the “AddAttributeSample.tlb” file in the sample. The content of the selected tlb file is shown in the “Type Library” tree.
4. Drag and drop the “IOldInterface : Interface” node to the blank space of “Rule File”. An “Add Rule” dialog is shown.
5. Select “AddAttribute” for the “Action” combo box, and Press OK. A rule is added in the “Rule File” tree, and the condition part is filled with the information of the type dragged, automatically.
6. Right click on the “Action : AddAttribute” node, and select “Modify Action…” item. The “AddAttribute Action Wizard” is shown.
7. Press “…” button in the “Attribute” section. “Managed Type Selector” is shown.
8. Input “mscorlib” in the “Assembly Name” input box, and press “Enter” button. All type in the mscorlib assembly is listed in the following box.
9. Input “Obsolete” in the “Filter” input box, and double-click the “System.Obsolete” node. The “Managed Type Selector” is closed, and the information is filled in the “AddAttribute Action Wizard”.
10. Select “Void .ctor(System.String, Boolean)” constructor, and press “…” button in the “Data” section. “AddAttribute Data Input Helper” is shown.
11. Input “Please use INewInterface instead.” for the first parameter, and input “True” for the second parameter, and press “OK”. “AddAttribute Data Input Helper” is closed, and the inputted parameter values are encoded to byte array.
12. Press “OK” button. “AddAttribute Action Wizard” is closed, and the parameter values of “Action : Attribute” node are all set.
13. Save the file as “AddAttributeSample.xml”.

## Tying it Together with *Tlbimp*

Using the XML config file from above, invoke the following on the command line

> tlbimp2 AddAttributeSample.tlb /config:AddAttributeSample.xml

In the result, the “IOldInterface” is added with an “Obsolete” attribute.

Using Reflector, you will see:

[ComImport, Obsolete("Please use INewInterface instead.", true), InterfaceType(ComInterfaceType.InterfaceIsIUnknown), Guid("18CE9E10-86F8-4639-B86B-7D15C9114866")]

public interface **IOldInterface**

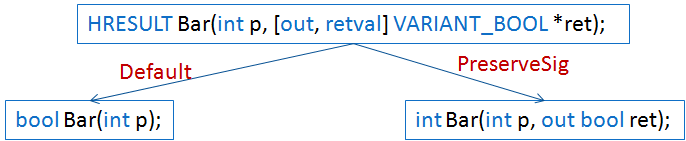
{

…

}

# *PreserveSig* Functionality

In a normal conversion process for a function, Tlbimp uses the [out, retval] parameter as the return value to substitute HResult return value. When error occurred, it throws an exception. However, sometimes, you may want to handle the error code by yourself,. If we perform a function conversion with **PreserveSig**, the original signature will be preserved, and TlbImp will not put the [out, retval] parameter in the return place in the function signature.



Besides the signature is preserved, the function will be added with a “preservesig” attribute.

## Getting Started: FunctionNoThrow

In the sample “PreserveSigSample.tlb” attached with the tool, we have a function named “FunctionNoThrow”, as:

HRESULT FunctionNoThrow([in] int p, [out, retval] VARIANT\_BOOL \*ret);

By default, the function will be converted as:

bool FunctionNoThrow(int p);

And, when you call it in C# code, he should write code like:

bool ret;

try {

ret = FunctionNoThrow(p);

catch {

// if the HRESULT return value is not 0, we get here.

……

}

Here, try-catch way may cause performance problems. If you want to get the HRESULT code, and handle the error yourself. You can convert the function in a PreserveSig way. The function will be converted as following, if PreserveSig action is performed on the function:

int FunctionNoThrow(int p, out bool ret);

And, now the code may look like:

bool ret;

int errCode = FunctionNoThrow(p, out ret);

if (errCode != 0) {

// FunctionNoThrow call did not succeed.

……

}

……

To achieve this, you need to create an XML config file (also called rule file) to tell Tlbimp how to perform the customized function conversion. The config file can be created either by hand or by using the UI customization tool TlbimpRuleFileEditor.exe.

## Sample XML Rule: PreserveSig.xml

Below is a sample XML config file. Note that Category, Condition and Action are key parts of a rule.

<Rules>

<Rule Name="Rule #1" Category="Function">

<Condition>

<And>

<NativeName Operator="Equal" Value="FunctionNoThrow" />

</And>

</Condition>

<Action Name="PreserveSig" />

</Rule>

</Rules>

In the above snippet, there is one rule named “Rule #1”, which means

For each function, if its name in the type library is equal to “FunctionNoThrow”, then a **PreserveSig** action is performed. This function will be converted in a signature-preserved way.

## Creating a Rule File with *TlbimpRuleFileEditor*

As an alternative to crafting and editing the customization XML files by hand, you can use *TlbImpRuleFileEditor.exe* to create and edit rule files. For example, to create a rule file for the PreserveSig.xml file, perform the following steps:

1. Open “TlbImpRuleFileEditor”; Press “New Rule File” button on the Toolbar to start.
2. Double click on the root node of “Type Library” tree,. An “open file dialog” is shown.
3. Select the “PreserveSigSample.tlb” file in the sample. The content of the selected tlb file is shown in the “Type Library” tree.
4. Drag and drop the “FunctionNoThrow : Function” node to the blank space of “Rule File”. An “Add Rule” dialog is shown.
5. Select “PreserveSig” for the “Action” combo box, and Press OK. A rule is added in the “Rule File” tree, and the condition part is filled with the information of the function dragged automatically.
6. Save the file as “PreserveSigSample.xml”

## Tying it Together with *Tlbimp*

Using the XML config file from above, invoke the following on the command line

> tlbimp2 PreserveSigSample.tlb /config:PreserveSigSample.xml

It generates the customized result, in which the signature of the function “FunctionNoThrow” is preserved.

# *ConvertTo* Functionality

When converting a signature of a function in the type library into a managed function signature, Tlbimp tries to give us the best match it thinks. However, some result may not be the same as wanted, because Tlbimp does not have all information. For example, by default, “HRESULT Func(int \*intArray, int size)” is converted to “void Func (ref int intArray, int size)”. But sometimes, it may need to convert the “int \*” to a fixed array (for example, “void Func (int intArray[20])”) or convert “int \*” to an “int[]” whose length is specified by the second parameter “int size”. **ConvertTo** action can convert specified function parameter to the expected managed signature.

## Getting Started: BinarySearch

In the sample “ConvertToSample.tlb” attached with the tool, we have a function named “BinarySearch”, as:

HRESULT BinarySearch([in] int \*p, [in] int size, [in] int element, [out, retval] int \*index);

This function is used to find integer “element” in the integer array “p”, the size of which is “size”. If found “element” in the array, return the index of first match, otherwise, return -1 for “index”.

By default, the function is converted as:

int BinarySearch(ref int p, int size, int element);

This is because Tlbimp does not have the information about the array length. The only fact Tlbimp knows is that “p” is a pointer to an integer. If you want to convert the signature of “p” to an integer array with the length “size”, ConvertTo action can help.

To achieve this, you need to create an XML config file (also called rule file) to tell Tlbimp how to perform the customized signature conversion. The config file can be created either by hand or by using the UI customization tool TlbimpRuleFileEditor.exe.

## Sample XML Rule: ConvertToSample.xml

Below is a sample XML config file. Note that Category, Condition and Action are key parts of a rule.

<Rules>

<Rule Name="Rule #1" Category="Signature">

<Condition>

<And>

<NativeParentFunctionName Operator="Equal" Value="BinarySearch" />

<NativeParameterIndex Operator="Equal" Value="1" />

</And>

</Condition>

<Action Name="ConvertTo">

<Parameter Key="Direction" Value="[In]" />

<Parameter Key="ByRef" Value="False" />

<Parameter Key="ManagedType" Value="LPArray" />

<Parameter Key="MarshalAs" Value="(default)" />

<Parameter Key="Attributes" Value="[SizeParamIndex=1]" />

</Action>

</Rule>

</Rules>

In the above snippet, there is one rule named “Rule #1”, which means

For each signature, if the function name it belongs to is equal to “BinarySearch” and it is the first parameter of the function, then a **ConvertTo** action is performed. This signature will be converted to a customized managed signature. The parameters in the ConvertTo action tells the details of the managed signature. It has a In attribute on it. It is not by ref. The type is an array. It does not need MarshAs to other type. And most importantly, the SizeParamIndex is set to 1, which indicates the second parameter of the function, “int size”.

Notes: SizeParamIndex is a field of MarshAs attribute, and is zero-based. That is, “SizeParamIndex=0” points the first parameter of the function, while, “NativeParameterIndex” is one of our conditions to tell the index of signatures. Because the return part of the function is also a signature, “NativeParameterIndex=0” points the return of the function, and “NativeParameterIndex=1” points the first parameter of the function.

## Creating a Rule File with *TlbimpRuleFileEditor*

As an alternative to crafting and editing the customization XML files by hand, you can use *TlbImpRuleFileEditor.exe* to create and edit rule files. For example, to create a rule file for the PreserveSig.xml file, perform the following steps:

1. Open “TlbImpRuleFileEditor”; Press “New Rule File” button on the Toolbar to start.
2. Double click on the root node of “Type Library” tree,. An “open file dialog” is shown.
3. Select the “ConvertToSample.tlb” file in the sample. The content of the selected tlb file is shown in the “Type Library” tree.
4. Drag and drop the “INT \* p : Signature” node to the blank space of “Rule File”. An “Add Rule” dialog is shown.
5. Select “ConvertTo” for the “Action” combo box, and Press OK. A rule is added in the “Rule File” tree, and the condition part is filled with the information of the function dragged automatically.
6. Right click on the “Action : ConvertTo” node, and select “Modify Action…” item. The “ConvertTo Action Wizard” is shown.
7. Select “LPArray” for the “ManagedType” combo box; Choose “[In]” radio box; Check “Enable Size Control”. The three radio boxes in the “Additional Array Attribute” section are enabled.
8. Select “SizeParamIndex” radio box; Input “1” in the numeric input box; Press OK. The “ConvertTo Action Wizard” is closed.
9. Save the file as “ConvertToSample.xml”

## Tying it Together with *Tlbimp*

Using the XML config file from above, invoke the following on the command line

> tlbimp2 ConvertToSample.tlb /config:ConvertToSample.xml

It generates the customized result.

Using ILDASM, you will see:

int32 BinarySearch([in] int32[] marshal([ + 1]) p,

[in] int32 size,

[in] int32 element)