

P/Invoke Interop Assistant - Overview

Tricky marshaling rules make Interop an advanced feature that many developers struggle with. This is especially true for Platform Invocation Services which, unlike COM Interop that uses Tlbimp, do not provide any automated way of generating managed proxy entry points based on some formal description of the unmanaged side. Developers end up declaring the entry points manually, often just trying to guess the signatures on a trial-and-error basis until their particular scenario seems to work. As often as not, this method of generating PInvoke signatures results in insidious bugs such as memory corruption or unexpected product behavior, that do not get caught until late into the product cycle, if at all.

The P/Invoke Interop Assistant provides automatic generation of managed code from C headers as well as the reverse generation of unmanaged signatures based on managed inputs. Users can take advantage of the tools to easily generate Windows and custom managed entry points and verify that the stack layout produced by the marshaler at runtime will really match what is expected by the unmanaged side. The SigExp is also capable of providing additional information such as memory allocation and marshaling details.

The Assistant is supported on Windows XP SP2, Windows Vista and above. It comprises of two tools that complement each other:

[SigImp: Unmanaged to Managed Signature Converter](#) [SigExp: Managed to Unmanaged Signature Converter](#)

If you have any suggestion about the tool, want to file bugs or give your comments, please send email to sigtool@microsoft.com.

SigImp: Unmanaged to Managed Signature Converter

The converter generates managed signatures and definitions in either C# or VB.NET from unmanaged:

- Type
- Procedure (Function)
- Constant
- Snippet of manually-entered unmanaged function signature

The converter can be invoked using the P/Invoke Interop Assistant application or as a command line tool. In the Assistant, switch to the *SigImp Search* tab, choose the kind (Type, Procedure, Constant, All), the target language (C# or VB.NET) and enter the name of the unmanaged entity. You can also enter unmanaged signature manually if *SigImp Translate Snippet* tab is selected. The C# or VB.NET code appears in the box located at the right side of the window.

The command line tool can be used to process entire C++ header files in the same fashion as the Assistant *SigImp Translate Snippet* tab. It has the following usage:

[GUI Usage](#) [Command line usage](#)
[Unsupported scenarios and other special cases](#)

GUI Usage

The tabs of *SigImp Search* and *SigImp Translate Snippet* offer two ways to convert unmanaged signatures to the managed signatures.

SigImp Search allows you to choose the managed language that you want to generate, and select a native type, procedure, or constant to do the generation. The UI consists of the following elements:

- **Name:** The keyword used to search the unmanaged signature names.
- **Kind:** Select one item in the Kind, and the unmanaged signature list will be filtered.
- **Language:** Choose Csharp or VB.NET, and the managed code will be generated in the selected language.
- **The unmanaged signature list:** It's at the left-bottom, which displays a list of supported types, methods, and/or constants collected from common Windows SDK header files. You can select one or more items to generate the managed signature(s).
- **Generated managed code output:** It's on the right of the UI, which shows the generated results.
- **Generate button:** Click the button and the code editor on the right will display the generated results.
- **Auto Generate:** If it is checked, the generated results will be generated immediately when you select items in the signature list.
- **Limitation:** At most 5 items can be auto generated simultaneously each time.

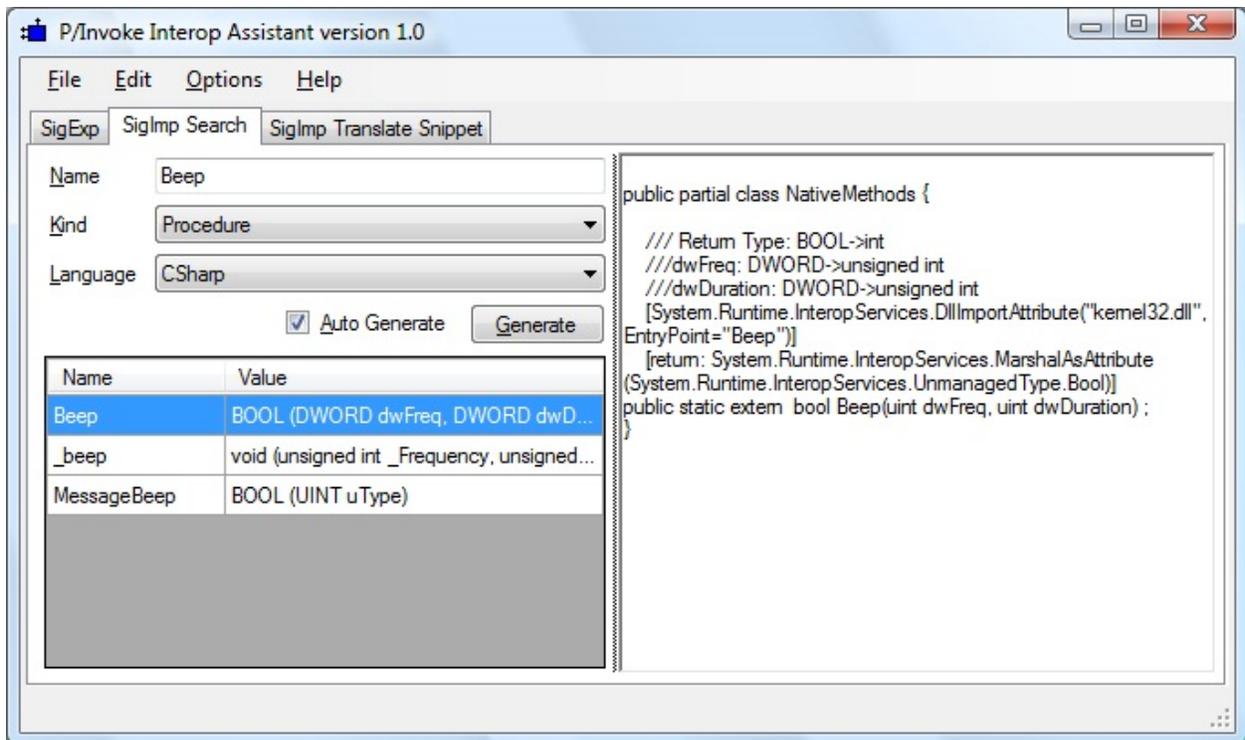


Figure 1: P/Invoke Interop Assistant GUI – SigImp Search

In *SigImp Translate Snippet*, you can write your native code snippet to generate the managed code. The UI description and usage is as follows:

- **Language:** Choose Csharp or VB.NET, and the managed code will be generated in the selected language.
- **Native Code Snippet editor:** You can write your native code snippet here.
- **Errors area:** It's at the left-bottom, which will display compile errors and warnings in real time.
- **Generated managed code output:** It's on the right of the UI, which shows the generated results.
- **Generate button:** Click the button and the code editor on the right will display the generated results.
- **Auto Generate:** If the box is checked, the generated results will be generated immediately when you are typing in the Native Code Snippet editor.

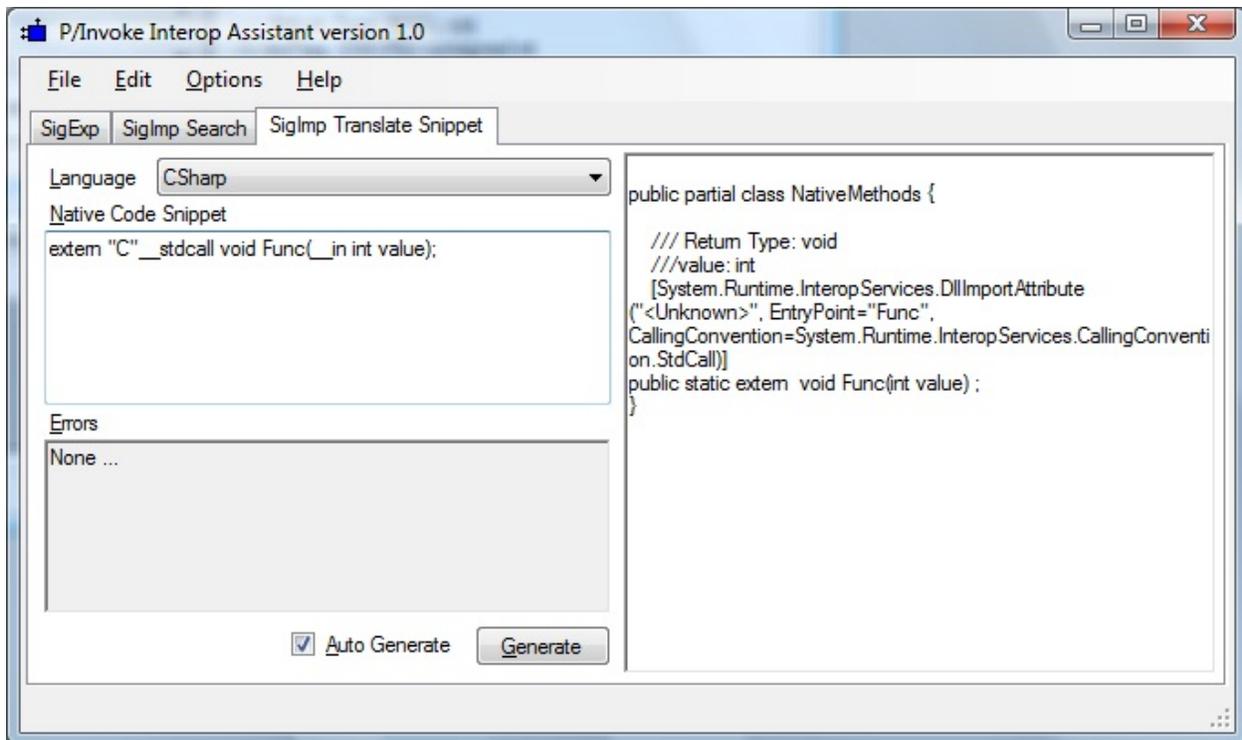


Figure 2: P/Invoke Interop Assistant GUI – SigImp Translate Snippet

Command Line Usage

The command line tool has the following usage:

```
sigimp [options] [Header File Names]  
  /genCode:[yes/no]           Whether or not to generate a  
  /genPreProc:[yes/no]       Generate the preprocessor co  
  /lang[uage]:lang           Language to generate into; vl  
  /out:filename               Output file name (default Na  
  /useSdk:[yes/no]           Whether or not to add common  
  /lib:name,name              List of libraries to resolve  
  /includePath:path,...      Include File Path  
  /nologo                     Prevent logo display  
  /?                           Help screen
```

Notice: The command line tools have a special interpretation of backslash characters when they are followed by a quotation mark character ("), as follows:

- 2n backslashes followed by a quotation mark produce n backslashes followed by a quotation mark.
- (2n) + 1 backslashes followed by a quotation mark again produce n backslashes followed by a quotation mark.
- n backslashes not followed by a quotation mark simply produce n backslashes.

For example, in the command line, you may need to specify a file path like the following:

```
"C:\Header Files\Header File.h"
```

```
"C:\Header Files"
```

```
"C:\Header Files\\"
```

Unsupported Scenarios and Other Special Cases

In the current version, the tool doesn't support converting unmanaged signatures in the following scenarios:

- Const value types, such as *const int a = 1*
- Explicit cast macro definition, such as *#define A (char)1*
- Macro functions, such as *#define Min(a,b)?a:b*
- Macros involving expressions, such as *#define A (1+2-0.1)*
- Nested macros such as *#define A TEXT("AA")*
- Typedefs that are equivalent to primitive types, such as *typedef int A*
- Functions with a variable number of arguments
- C++ language features, such as classes, templates, and namespaces
- The DLL name in `DllImportAttribute` might be unknown if the functions are implemented in .lib file, the functions are user defined, or in other cases that the DLL name cannot be determined.

Check MSDN for help in manually generating signatures for the above scenarios. Some other special cases are:

- The built-in types and APIs are from Visual Studio 2005 with Service Pack 1. For the types and APIs from other versions, you may need to copy the necessary declarations from the corresponding header files, and paste it into the *SigImp Translate Snippet* tool to see the results.
- When no calling convention is specified in unmanaged code, it's recommended that you review the generated code and make sure that the calling convention is correct.

SigExp: Managed to Unmanaged Signature Converter

The converter consumes managed assemblies which it reflects on to find all P/Invoke declarations, types imported from COM, and delegates. It can be invoked using the P/Invoke Interop Assistant application or as a command line tool. In the Assistant, switch to the *SigExp* tab, open an assembly, and choose the method or type to convert. The corresponding C declarations appear in the *Unmanaged signature* box and tips in the *Additional information* box. The *Options* menu contains settings that influence the output (refer to the description below).

[GUI Usage](#) [Command line usage](#)

GUI Usage

The UI of SigExp is split into three areas:

- **The left area:** displays the signatures in the user-loaded assembly.
- **The unmanaged signature area:** shows the unmanaged signature generated from the selected managed signature selected in the left area.
- **The additional information area:** offers additional information about the behavior of the marshaler for the particular managed signature, including hints and warnings addressing common issues.

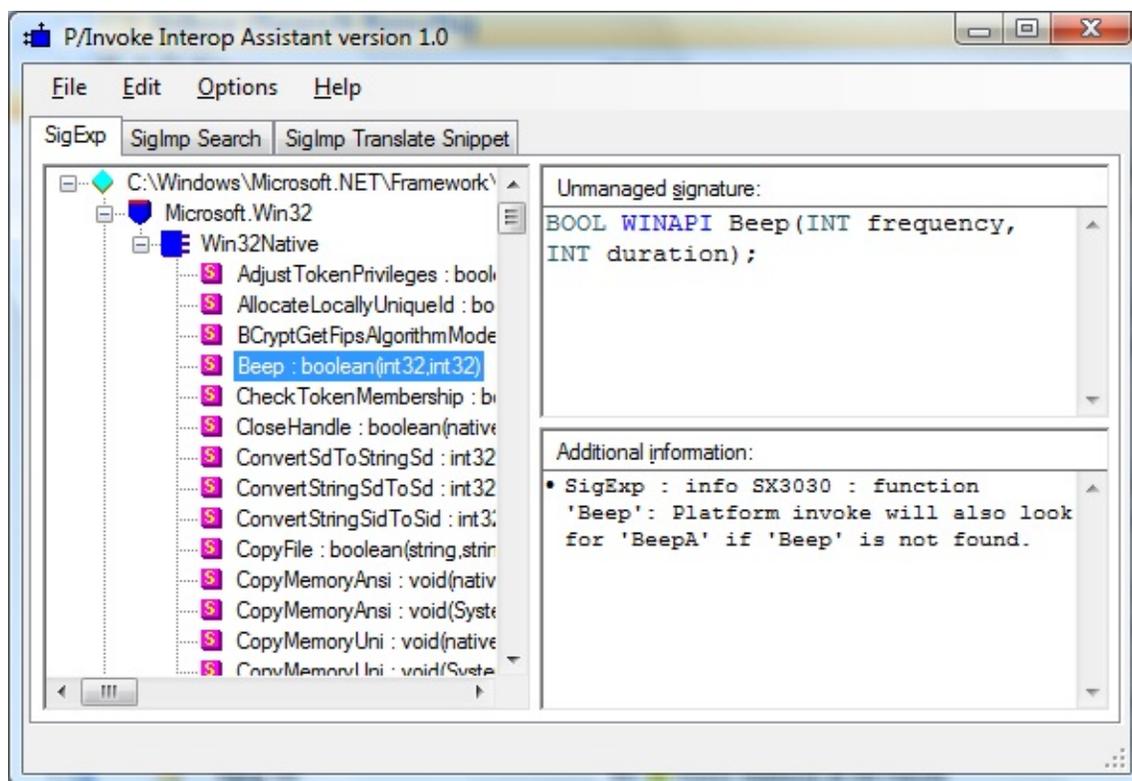


Figure 3: P/Invoke Interop Assistant GUI – SigExp

To load an assembly, you can either:

- Choose File -> Open from the menu, and select a particular assembly, or
- Drag the assembly file and drop it to the left area.

Command Line Usage

The command line tool has the following usage:

sigexp [**<options>**] **/file** **<path_to_assembly>** [**<def_1>**,**<def_2>**]

- /32** - generate signatures for 32-bit target platform
- /64** - generate signatures for 64-bit target platform

- /unicode** - generate signatures for Unicode target platform
- /ansi** - generate signatures for ANSI target platform

- /wintypes** - use standard Windows types like 'LPCSTR' (Default)
- /plaintypes** - use plain C++ types like 'const char *'

- /color** - use colorful console output (Default)
- /bw** - use boring B&W; output

- /direction** - annotate ptr parameters with marshal direction
- /nologo** - prevents displaying of logo
- /nomsg** - prevents displaying of error, warning, and info messages
- /notypes** - prevents displaying of complex types definitions
(will only display the signature itself)

- /?** - This help screen.

<def_n> are designations of types or methods in the assembly. If **<def_n>** is specified, signatures of all interop methods and delegates will be displayed.

Example:

```
sigexp /direction /file mscorlib Microsoft.Win32.Win32Native
```