Source generation for platform invokes

Article • 05/09/2023

.NET 7 introduces a source generator for P/Invokes that recognizes the LibraryImportAttribute in C# code.

When it's not using source generation, the built-in interop system in the .NET runtime generates an IL stub—a stream of IL instructions that is JIT-ed—at run time to facilitate the transition from managed to unmanaged. The following code shows defining and then calling a P/Invoke that uses this mechanism:

C#

```
[DllImport(
    "nativelib",
    EntryPoint = "to_lower",
    CharSet = CharSet.Unicode)]
internal static extern string ToLower(string str);

// string lower = ToLower("StringToConvert");
```

The IL stub handles marshalling of parameters and return values and calling the unmanaged code while respecting settings on DllImportAttribute that affect how the unmanaged code should be invoked (for example, SetLastError). Since this IL stub is generated at run time, it isn't available for ahead-of-time (AOT) compiler or IL trimming scenarios. Generation of the IL represents an important cost to consider for marshalling. This cost can be measured in terms of application performance and support for potential target platforms that may not permit dynamic code generation. The Native AOT application model addresses issues with dynamic code generation by precompiling all code ahead of time directly into native code. Using `DllImport` isn't an option for platforms that require full Native AOT scenarios and therefore using other approaches (for example, source generation) is more appropriate. Debugging the marshalling logic in `DllImport` scenarios is also a non-trivial exercise.

The P/Invoke source generator, included with the .NET 7 SDK and enabled by default, looks for LibraryImportAttribute on a `static` and `partial` method to trigger compile-time source generation of marshalling code, removing the need for the generation of an IL stub at run time and allowing the P/Invoke to be inlined. Analyzers and code fixers are also included to help with migration from the built-in system to the source generator and with usage in general.

Basic usage

The LibraryImportAttribute is designed to be similar to DllImportAttribute in usage. We can convert the previous example to use P/Invoke source generation by using the LibraryImportAttribute and marking the method as `partial` instead of `extern`:

C#

```
[LibraryImport(
    "nativelib",
    EntryPoint = "to_lower",
    StringMarshalling = StringMarshalling.Utf16)]
internal static partial string ToLower(string str);
```

During compilation, the source generator will trigger to generate an implementation of the `ToLower` method that handles marshalling of the `string` parameter and return value as UTF-16. Since the marshalling is now generated source code, you can actually look at and step through the logic in a debugger.

`MarshalAs`

The source generator also respects the MarshalAsAttribute. The preceding code could also be written as:

C#

```
[LibraryImport(
    "nativelib",
    EntryPoint = "to_lower")]
[return: MarshalAs(UnmanagedType.LPWStr)]
internal static partial string ToLower(
    [MarshalAs(UnmanagedType.LPWStr)] string str);
```

Some settings for MarshalAsAttribute aren't supported. The source generator will emit an error if you try to use unsupported settings. For more information, see Differences from DllImport.

Calling convention

To specify the calling convention, use UnmanagedCallConvAttribute, for example:

C#

```
[LibraryImport(
    "nativelib",
    EntryPoint = "to_lower",
    StringMarshalling = StringMarshalling.Utf16)]
[UnmanagedCallConv(
    CallConvs = new[] { typeof(CallConvStdcall) })]
internal static partial string ToLower(string str);
```

Differences from `DllImport`

LibraryImportAttribute is intended to be a straightforward conversion from DllImportAttribute in most cases, but there are some intentional changes:

- CallingConvention has no equivalent on LibraryImportAttribute. UnmanagedCallConvAttribute should be used instead.
- CharSet (for CharSet) has been replaced with StringMarshalling (for StringMarshalling). ANSI has been removed and UTF-8 is now available as a first-class option.
- BestFitMapping and ThrowOnUnmappableChar have no equivalent. These fields were only relevant when marshalling an ANSI string on Windows. The generated code for marshalling an ANSI string will have the equivalent behavior of `BestFitMapping=false` and `ThrowOnUnmappableChar=false`.
- ExactSpelling has no equivalent. This field was a Windows-centric setting and had no effect on non-Windows operating systems. The method name or EntryPoint should be the exact spelling of the entry point name. This field has historical uses related to the A and W suffixes used in Win32 programming.
- PreserveSig has no equivalent. This field was a Windows-centric setting. The generated code always directly translates the signature.
- The project must be marked unsafe using AllowUnsafeBlocks.

There are also differences in support for some settings on MarshalAsAttribute, default marshalling of certain types, and other interop-related attributes. For more information, see our documentation on compatibility differences.

See also

- P/Invoke
- LibraryImportAttribute
- SYSLIB diagnostics for P/Invoke source generation

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback