Native library loading

Article • 09/09/2023

This article explains which paths the runtime searches when loading native libraries via P/Invoke. It also shows how to use SetDllImportResolver.

Library name variations

To facilitate simpler cross platform P/Invoke code, the runtime adds the canonical shared library extension (`.dll`, `.so` or `.dylib`) to native library names. On Unix-based platforms, the runtime will also try prepending `lib`. These library names variations are automatically searched when you use APIs that load native libraries, such as DllImportAttribute.

> **Note**
>
> Absolute paths in library names (for example, `/usr/lib/libc.so`) are treated as-is and no variations will be searched.

Consider the following example of using P/Invoke:

C#

```
[DllImport("nativedep")]
static extern int ExportedFunction();
```

When running on Windows, the DLL is searched for in the following order:

1. `nativedep`
2. `nativedep.dll` (if the library name does not already end with `.dll` or .`exe`)

When running on Linux or macOS, the runtime will try prepending `lib` and appending the canonical shared library extension. On these OSes, library name variations are tried in the following order:

1. `nativedep.so` / `nativedep.dylib`
2. `libnativedep.so` / `libnativedep.dylib` [1]
3. `nativedep`
4. `libnativedep` [1]

On Linux, the search order is different if the library name ends with `.so` or contains `.so.` (note the trailing `.`). Consider the following example:

C#

```
[DllImport("nativedep.so.6")]
static extern int ExportedFunction();
```

In this case, the library name variations are tried in the following order:

1. `nativedep.so.6`
2. `libnativedep.so.6` [1]
3. `nativedep.so.6.so`
4. `libnativedep.so.6.so` [1]

[1] Path is checked only if the library name does not contain a directory separator character (`/`).

Custom import resolver

In more complex scenarios, you can use SetDllImportResolver to resolve DLL imports at run time. In the following example, `nativedep` is resolved to `nativedep_avx2` if the CPU supports it.

> **Tip**
>
> This functionality is only available in .NET Core 3.1 and .NET 5+.

C#

```
using System;
using System.Reflection;
using System.Runtime.InteropServices;

namespace PInvokeSamples
{
    public static partial class Program
    {
        [LibraryImport("nativedep")]
        private static partial int ExportedFunction();

        public static void Main(string[] args)
        {
            // Register the import resolver before calling the im-
ported function.
```

```
            // Only one import resolver can be set for a given assem-
bly.

NativeLibrary.SetDllImportResolver(Assembly.GetExecutingAssembly(),
DllImportResolver);

            int value = ExportedFunction();
            Console.WriteLine(value);
        }

        private static IntPtr DllImportResolver(string libraryName,
Assembly assembly, DllImportSearchPath? searchPath)
        {
            if (libraryName == "nativedep")
            {
                // On systems with AVX2 support, load a different li-
brary.
                if (System.Runtime.Intrinsics.X86.Avx2.IsSupported)
                {
                    return NativeLibrary.Load("nativedep_avx2", as-
sembly, searchPath);
                }
            }

            // Otherwise, fallback to default import resolver.
            return IntPtr.Zero;
        }
    }
}
```

See also

- Platform Invoke (P/Invoke)