P/Invoke

Platform Invocation Services

Juan Ramírez (@ichramm)

Agenda

- Intro con dibujito
- Qué es? Qué hace? Para qué sirve?
- Analizando el problema
  - P/Invoke vs C++/CLI
- DllImport y sus allegados
- Data Marshalling
  - Data Types
  - Estructuras
  - Callbacks

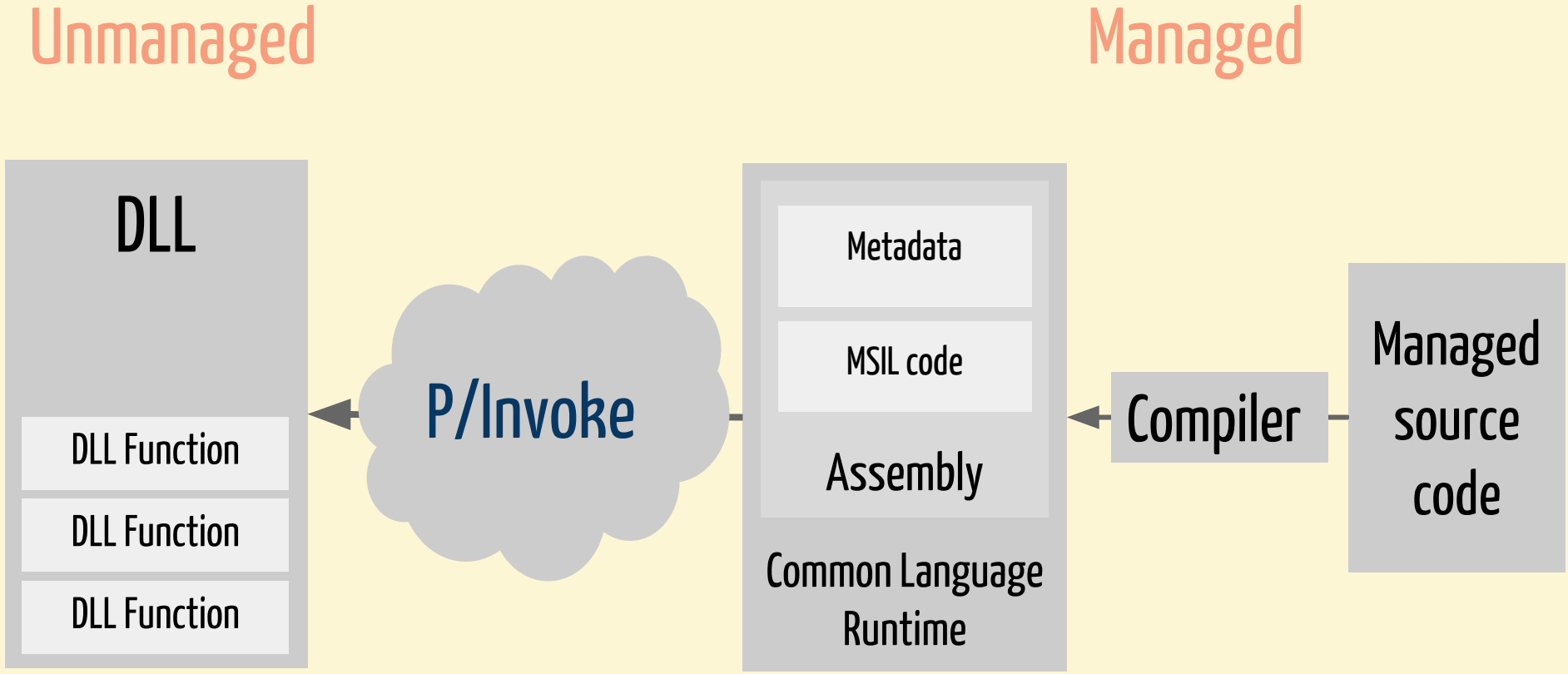
Qué es? Que hace? Para qué sirve?

- Es una funcionalidad del CLR ...

  … que permite invocar código nativo *(unmanaged)* desde .Net *(managed)*.

- Es muy flexible, y con defaults razonables.

“Permite ejecutar native code desde managed code”



y...

PARA QUÉ SIRVE?

Problema

- Se desea utilizar cierta API nativa, que no tiene bindings para .Net.
- Debe ser Portable.
- No se dispone del código fuente de la librería.

Ejemplos (poco concretos)

- API de Windows
- Librería que implementa determinado protocolo.
- Interoperabilidad con distintos frameworks

Y qué pasa con C++/CLI?

- Crear un wrapper sería más sencillo. ✓
- Buena performance, mejor que P/Invoke en algunos casos. ✓
- Estáticamente type-safe. ✓
- Transiciones entre managed y unmanaged son sencillas y seguras. ✓

Y qué pasa con C++/CLI?

- No es portable (solo Windows). ✗
- Se necesita el código fuente de la librería. ✗
- Hay que tocar otro lenguaje, una especie de C++ maquillado. ✗

Y con P/Invoke?

- Solo necesito la DLL. ✓
- Es portable (mono project). ✓
- Solo tengo que tocar C#. ✓

Peeeero...

- No es type-safe. ✘
- Más lento en ciertos casos. ✘
- Se puede volver muy complicado. ✘

En resumen

- Es la solución a un problema

- Permite utilizar funcionalidades que de otra manera no se podrían utilizar.

Basta de cháchara...

GETTING DOWN TO BUSINESS

Ejemplo

```
using System.Runtime.InteropServices;

[DllImport("kernel32.dll")]
private static extern bool CopyFile (
  string lpExistingFileName,
  string lpNewFileName,
  bool   bFailIfExists
);

// Usage:
bool copied = CopyFile(textBox1.Text, textBox2.Text, true);
```

/- `DllImport`

- Indica que el método es expuesto por una DLL nativa (`class DllImportAttribute`).
- El keyword **`extern`** indica que el método está implementado en otro lado.
- En nuestro ejemplo:
  - Dll Nativa: `kernel32.dll`
  - Función : `CopyFile`

Ejemplo

```
[DllImport("kernel32.dll",
    CharSet = CharSet.Unicode)]
private static extern bool CopyFile (
  [MarshalAs(UnmanagedType.LPWStr)] string lpExistingFileName,
  [MarshalAs(UnmanagedType.LPWStr)] string lpNewFileName,
  bool   bFailIfExists
);

// Usage:
bool copied = CopyFile(textBox1.Text, textBox2.Text, true);
```

/- CharSet

- Marshalling de strings
  - `Charset.Ansi    -> char*`
  - `Charset.Unicode -> wchar_t*`

- Name matching
  - `Charset.Ansi    -> CopyFileA`
  - `Charset.Unicode -> CopyFileW`

Ejemplo

```
[DllImport("kernel32.dll",
    CharSet        = CharSet.Unicode,
    ExactSpelling = true)]
private static extern bool CopyFile (
  [MarshalAs(UnmanagedType.LPWStr)] string lpExistingFileName,
  [MarshalAs(UnmanagedType.LPWStr)] string lpNewFileName,
  bool   bFailIfExists
);
```

/- CharSet vs ExactSpelling

- ExactSpelling deshabilita name matching.

- La función **`CopyFile`** no existe en `kernel32.dll`.

- Sí existen **`CopyFileA`** y **`CopyFileW`**

/- CharSet vs ExactSpelling

Posible solución

```
[DllImport("kernel32.dll",
    CharSet       = CharSet.Unicode,
    ExactSpelling = true)]
private static extern bool CopyFileW (
  [MarshalAs(UnmanagedType.LPWStr)] string lpExistingFileName,
  [MarshalAs(UnmanagedType.LPWStr)] string lpNewFileName,
  bool   bFailIfExists
);
```

Ejemplo

```
[DllImport("kernel32.dll",
    CharSet           = CharSet.Unicode,
    SetLastError      = true, // This function sets last error
    CallingConvention = CallingConvention.Cdecl)]
[return: MarshalAs(UnmanagedType.Bool)]
private static extern bool CopyFile (
  [MarshalAs(UnmanagedType.LPWStr)] string lpExistingFileName,
  [MarshalAs(UnmanagedType.LPWStr)] string lpNewFileName,
  [MarshalAs(UnmanagedType.Bool)]   bool   bFailIfExists
);
```

Ejemplo

```
[DllImport("kernel32.dll",
    CharSet           = CharSet.Unicode,
    SetLastError      = true, // This function sets last error
    CallingConvention = CallingConvention.StdCall)]
[return: MarshalAs(UnmanagedType.Bool)]
private static extern bool CopyFile (
  [MarshalAs(UnmanagedType.LPWStr)] string lpExistingFileName,
  [MarshalAs(UnmanagedType.LPWStr)] string lpNewFileName,
  [MarshalAs(UnmanagedType.Bool)]   bool   bFailIfExists
);
```

/-CallingConvention

- Esquema que establece como una función recibe parámetros y devuelve un resultado.
- “... quien limpia el stack”.
- El código necesario lo genera el propio compilador (C/C++).
- El default en .Net es `Winapi` (`StdCall`).

/-CallingConvention

StdCall

- El callee limpia el stack.
- Default en el Windows API.

Cdecl

- El caller limpia el stack.
- Default en Visual C++.

Ejemplo

```
[DllImport("kernel32.dll",
    CharSet           = CharSet.Unicode,
    SetLastError      = true,
    CallingConvention = CallingConvention.StdCall,
    EntryPoint        = "CopyFile")]
[return: MarshalAs(UnmanagedType.Bool)]
private static extern bool NativeCopyFile (
  [MarshalAs(UnmanagedType.LPWStr)] string lpExistingFileName,
  [MarshalAs(UnmanagedType.LPWStr)] string lpNewFileName,
  [MarshalAs(UnmanagedType.Bool)]   bool   bFailIfExists
);
```

/- `EntryPoint`

- Especifica el nombre con el que se va a buscar la función.

- O el índice de la función dentro de la DLL .
  - Ej: `EntryPoint = "#167"`

- El default es el nombre del método marcado con el DllImport.

Ejemplo

```
void CopyFile(string from, string to)
{
    bool copied = NativeCopyFile(from, to, true);
    if (copied == false)
    {
        throw new Win32Exception(Marshal.GetLastWin32Error());
    }
}
```

```
> git checkout example1
```

DOS EJEMPLITOS LOCOS

```
> git checkout example2
```

Data Marshaling

- Se trata de una especie de transformación de datos.

  - Desde el mundo managed al mundo nativo.
  - Desde el mundo nativo al mundo managed.

Data Marshaling / Data Types

Para cada data type de .net existe un data type unmanaged por defecto.

- `int -> int`
- `bool -> BOOL /ojo! son 4 bytes`
- `string -> char * (o TCHAR)`
- `etc`

Data Marshaling / Data Types / `System.IntPtr`

- Se utiliza para representar punteros (o handles).
- IntPtr es ideal para tipos opacos.
- Es platform-specific (x86, x64).

Ejemplo

```
// typedef void * HANDLE; -> winnt.h
HANDLE WINAPI OpenMutex(
    DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName
);

[DllImport("kernel32.dll")]
public static extern IntPtr OpenMutex(
    uint dwDesiredAccess, bool bInheritHandle, string lpName
);
```

Data Marshaling / Data Types / `System.String`

- Ya vimos como pasar strings desde el mundo managed al mundo unmanaged.

- Vamos a ver como se hace al revés.

Data Marshaling / Data Types / System.String

```
DWORD WINAPI GetModuleFileName(
    HMODULE hModule, LPTSTR lpFilename, DWORD nSize
);

[DllImport("kernel32.dll")]
public static extern uint GetModuleFileName(
    IntPtr                              hModule,
    // System.String is not mutable
    StringBuilder                       lpFilename,
    [MarshalAs(UnmanagedType.U4)] int nSize
);
```

Ejemplo

```
public string StartupPath {
  get {
    StringBuilder sb = new StringBuilder(260);
    GetModuleFileName(IntPtr.Zero, sb, sb.Capacity);
    // TODO: Handle error
    return sb.ToString();
  }
}
```

```
> git checkout example3
```

OTRO EJEMPLITO

Data Marshaling / Structs

- Se debe especificar el layout
- Members pueden llevar atributos

```
[StructLayout(LayoutKind.Sequential)]
public class LOGFONT {
    // members...
    [MarshalAs(UnmanagedType.ByValTStr,
        SizeConst = LF_FACESIZE)]
    public string lfFaceName;
```

Data Marshaling / Structs

- El layout se puede configurar explícitamente

```
[StructLayout(LayoutKind.Explicit)]
public class LOGFONT {
    [FieldOffset(0)] public int lfHeight;
    // more members...

}
```

Data Marshaling / Structs

- Ojo con el padding!!
- Cual es el tamaño de este struct?

```
public struct Test {
    public byte  a;
    public int   b;
    public short c;
    public byte  d;
}
```

12

```
> git checkout example4
```

JUMP!

Data Marshaling / Callbacks

- Una forma de llamar funciones de .Net desde el mundo unmanaged.
- Mapean directo con delegates
- Mismas reglas que con las demas funciones.

Ejemplo

```
ESME_HANDLE SMPP_API libSMPP_ClientCreate (
  OnIncomingMessageCallback onNewMessage
);
```

Ejemplo

```
typedef void (*OnIncomingMessageCallback)(
  ESME_HANDLE  hClient,
  const char*  from,
  const char*  to,
  const char*  content, // UTF-8
  unsigned int size
);
```

Ejemplo

```
[UnmanagedFunctionPointer(CallingConvention.Cdecl)]
delegate void IncomingMessageHandler(
    IntPtr hClient,
    string from,
    string to,
    [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 4)]
    byte[] content,
    [MarshalAs(UnmanagedType.U4)] int bufferSize
);
```

Ejemplo

```
[DllImport("inconcertsmpp.dll"
  , CharSet = CharSet.Ansi
  , CallingConvention = CallingConvention.Cdecl)]
static extern IntPtr libSMPP_ClientCreate(
    [MarshalAs(UnmanagedType.FunctionPtr)]
    IncomingMessageHandler onNewMessage
);
```

Ejemplo

```
hClient = libSMPP_ClientCreate(
  new IncomingMessageHandler( delegate(
       IntPtr hClient, string from, string to,
       byte[] content, int bufferSize)
  {
    string text = Encoding.UTF8.GetString(content);
    // ...
  }
);
```

Pero...

- Ese código vuela por los aires
- El garbage collector se lleva el delegate
- Hay que mantener vivas las referencias a esos elementos

Ejemplo

```
m_handler = new IncomingMessageHandler( delegate(
        IntPtr hClient, string from, string to,
        byte[] content, int bufferSize)
  {
    string text = Encoding.UTF8.GetString(content);
    // ...
  };

hClient = libSMPP_ClientCreate(m_handler);
```

Gracias!

Didn't make it:

- Keywords: unsafe, fixed
- Custom Marshallers
- Punteros