

# Tema 11. Interacción con código no gestionado

Formación específica, cursos verano 2008  
ETS de Informática Aplicada  
Universidad Politécnica de Valencia

# Ejemplo introductorio

- ◆ Reproducción de un sonido utilizando el método WCE\_PlaySound que ofrece la librería CoreDll.dll

```
Using System.Runtime.InteropServices;
```

```
[DllImport("CoreDll.DLL", EntryPoint="PlaySound", SetLastError=true)]  
private extern static int WCE_PlaySound(string szSound, IntPtr hMod, int flags);
```

```
// Construct the Sound object to play sound data from the specified file.  
public Sound (string fileName) { m_fileName = fileName; }
```

```
// Play the sound.  
public void Play () {  
    if (m_fileName != null)  
        WCE_PlaySound(m_fileName, IntPtr.Zero,  
                      (int) (Flags.SND_ASYNC | Flags.SND_FILENAME));  
}
```

# .NET CF Interop

- ◆ ¿Por qué es importante?
  - El desarrollo para dispositivos móviles exige mayor proximidad con el HW que el desarrollo para PC
- ◆ Idea básica
  - Permitir que los desarrolladores exploten las capacidades de sus dispositivos desde su código (gestionado)
  - Permitir la notificación de eventos de código no gestionado al gestionado

# Arquitectura general

- ◆ Las aplicaciones gestionadas lo son como procesos Windows CE
  - Sujetas a las mismas limitaciones que éstos
    - ◆ 32/64 MB de memoria
  - Las DLLs gestionadas se cargan en el espacio del proceso o se utilizan desde la caché de librería (GAC)
  - Las DLLs no gestionadas sólo se cargan si se utilizan

# ¿Qué soporte se ofrece?

- ◆ Invocación a funciones en DLLs
  - Soporte P/Invoke
- ◆ No se soporta
  - Invocación directa de interfaces COM
    - ◆ Necesidad de utilizar DLLs no gestionadas que sirvan de interfaz
  - Controles ActiveX

# Pasos a seguir para realizar una llamada a código nativo

- ◆ Decidir qué es lo que se necesita
- ◆ Buscar una manera de hacerlo a través de código gestionado
  - Menor coste que hacerlo utilizando P/Invoke
- ◆ Buscar en la SDK cuál es la llamada no gestionada a realizar
  - Es necesario conocer el API

# Pasos a seguir para realizar una llamada a código nativo

- ◆ Crear una clase gestionada que encapsule la invocación P/Invoke
- ◆ Declarar como estática y externa la función no gestionada a invocar
  - Prestar especial atención a que los parámetros a utilizar en la llamada sean los correctos
- ◆ Declarar la función como externa
  - C# – Utilizar el atributo DllImport
  - VB .NET – Utilizar la palabra reservada Lib
- ◆ Crear un método público que realice la invocación P/Invoke

# Declaración de la función externa

- ◆ Declaración de la función y sus parámetros

```
[DllImport("coredll.dll")]  
public static extern unsigned int GetTickCount ();
```

- La función es declarada como estática y externa en C#
- Uso del atributo DllImport

# Lo mismo en VB

- ◆ Utilizar la palabra reservada Lib para declarar la función como externa

```
Declare Function GetTickCount Lib "coredll.dll" () _  
As Integer
```

# ¿Qué DLL utilizar?

- ◆ La DLL de Windows CE es COREDLL.DLL
  - Casi todo lo que se utiliza está aquí
- ◆ Otra DLLs
  - AYGShell.dll – Funciones Pocket PC shell
  - CommCtrl.dll – Common control lib
  - WinSock.dll – Windows Sockets
  - Phone.dll – Control del teléfono
  - SMS.dll – API de mensajería SMS

# Utilización de DllImport

- ◆ Los parámetros del atributo son
  - EntryPoint: Nombre de la función nativa
    - ◆ Puede ser distinto del dado a nivel .NET CF
  - SetLastError: Mantiene el último valor de error

```
[DllImport ("\\windows\\UnManaged.dll",  
          EntryPoint="TestProcW",  
          SetLastError=true)]
```

```
public static extern int TestProc (int a, byte[] b);
```

# En ejecución

- ◆ Una llamada a la función se traduce a nivel CLR en:
  - Cargar la DLL (uso de LoadLibrary)
  - Obtener el punto de llamada (GetProcAddress)
  - Invocar la función
- ◆ Si hay algún problema en la realización de alguno de estos pasos se genera a nivel aplicativo (gestionado) la excepción `MissingMethodException`
  - Las excepciones internas no tratadas por el código no gestionado invocado conllevan la terminación de la aplicación

# Aspectos a considerar

- ◆ Aislar las invocaciones a llamadas P/Invoke en clases propias
- ◆ Declarar las funciones P/Invoke como privadas
- ◆ Encapsular las invocaciones en cláusulas try/catch

# Data Marshaling

- ◆ Tipos simples
  - Traducción directa
- ◆ Objetos simples y ValueTypes
  - Objetos que contienen tipos simples
  - Se ofrecen métodos de soporte al marshaling
- ◆ Complex Objects and Value Types
  - El marshaling es manual

# Diferencias en el Marshaling

- ◆ Los objetos sólo se “marshalean” a un nivel de profundidad
  - Los objetos que contienen objetos deben “marshalearse” manualmente
- ◆ Los parámetros de tipo coma flotante o enteros de 64 bits deben ser enviados a la plataforma como datos binarios
- ◆ No se pueden utilizar delegados con código nativo

# Marshaling de tipos simples

## ◆ Correspondencia

C#	Visual Basic	Native Code
int	Integer	int (32-bit integer)
short	Short	short (16-bit)
bool	Boolean	BYTE (8-bit)*
char	Char	wchar (16-bit)
String	String	wchar *
Array int []	Array Integer ()	Data of the Array int *
long	Long	Not Supported (64-bit)
double	Double	Not Supported (floating point)

\*bool and Boolean do NOT always convert to a Win32 BOOL

# Marshaling de objetos simples

Parameter Type	Marshaling Action
Array	Point at the array data
String Builder*	Make a copy and point at the string data (LPSTR)
String	Point at the string data (LPSTR)
ByRef String*	Make a copy of the string and point at the data (LPSTR)
Other objects	Point at the member data

\*Execution Engine will resize managed strings after native function call

# Marshaling de objetos simples

- ◆ Las clases se tratan secuencialmente
- ◆ Sólo las clases que contienen tipos simples serán automáticamente "marsheladas"

```
public class Rect
{
    int left;
    int top;
    int right;
    int bottom;
}
```

C#



```
typedef struct _RECT
{
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT;
```

Native Code - C

# Marshaling de objetos complejos

- ◆ Marshal class
  - Located in System.Runtime.InteropServices
  - Proporciona funcionalidad de soporte al marshaling de datos
- ◆ Permite tener una copia de objetos manejados en código nativo
- ◆ Permite leer y escribir datos directamente en memoria
- ◆ Permite manipular datos sin formato (uso de byte arrays)

# Areas típicas de uso de P/Invoke

- ◆ API de registro del sistema
- ◆ Comunicación entre procesos
- ◆ Gestión de bloques (grandes) de memoria
- ◆ Uso de HW
- ◆ Uso de la interfaz COM
  - Obex, Pocket Outlook (POOM), ...