Marchalling

...

Axel Buendia
(axel.buendia@cnam.fr)

When C++ merges with C#

- C#
  - interpreted language
  - no strong control of structure sizes
  - no strong control of memory management
- C++
  - compiled language
  - strong control (sometimes not so strong) of structure sizes
  - strong control of memory management
- Use cases:
  - using a dynamic library from C#
  - using a C++ class in C#

Using dll from C#: call function

lib.cpp

```
extern "C" __declspec(dllexport)
int Addition (int a, int b) {
    return a+b;
}
```

- extern "C" is used to fix the call type
- __declspec(dllexport) is used to export the function (make it accessible from the dll)

main.cs

```
[DllImport("lib.dll")]
public static extern int Addition (int a, int b);
```

- DllImport is used to specify in which file to look for the function
- the dll file has to be near the executable
- dll functions are public static and extern
- because function has been declared extern "C" no mangling is applied

Using dll from C#: use class

lib.cpp

```
#include "lib.h"

CPerson::CPerson(string name, string firstname){
    Name = name;
    FirstName = firstname;
}

CPerson::~CPerson(void) {}

void CPerson::Print(){
    cout << "My name is " << FirstName << " " << Name << endl;
}

extern "C" __declspec(dllexport) CPerson* CPersonNew ( {
    return new CPerson("Buendia", "Axel");
}

extern "C" __declspec(dllexport) void CPersonDelete (CPerson* cp) {
    delete cp;
}
```

- factories are needed to allow C++ side creation of the instance
- recycles are needed to free the C++ side created instances
- the factory uses constant parameters, we will see later how to pass parameters

Using dll from C#: use class

lib.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;

class __declspec(dllexport) CPerson {
private:
     string    Name;
     string    FirstName;
public:
     CPerson(string name, string firstname);
     ~CPerson(void);
     void Print();
}

extern "C" __declspec(dllexport) void CPersonDelete (CPerson* cp);
extern "C" __declspec(dllexport) CPerson* CPersonNew ();
```

- the class is exported with __declspec(dllexport)

Using dll from C#: use class

```
[DllImport("lib.dll", EntryPoint = "CPersonNew")]
     public static extern IntPtr NewCPerson();

[DllImport("lib.dll", EntryPoint = "CPersonDelete")]
     public static extern void DeleteCPerson(IntPtr cp);
```

main.cs

- during link time, the Print method cannot be found
- this is due to C++ name mangling
- name mangling or name decoration is used to distinguish methods that could have the same name but differ from their parameters or class, it is a way to add more information in the exported name of a method
- to get decorated names, use dumpbin.exe (in visual C++):
  - dumpbin.exe /exports lib.dll

```
5    4 000110EB ?Print@CPerson@@QAEXXZ = @ILT+230(?Print@CPerson@@QAEXXZ)
6    5 00011203 CPersonDelete = @ILT+510(_CPersonDelete)
7    6 0001112C CPersonNew = @ILT+295(_CPersonNew)
```

- CPersonDelete and CPersonNew are exported without any decoration
- Print is exported with its decoration, just use the decorated name in the EntryPoint

```
[DllImport("lib.dll", EntryPoint = "?Print@CPerson@@QAEXXZ", CharSet = CharSet.Unicode,
CallingConvention = CallingConvention.ThisCall)]
        public static extern void PrintCPerson(IntPtr this);
```

Using dll from C#: use class

main.cs

```
[DllImport("lib.dll", EntryPoint = "?Print@CPerson@@QAEXXZ", CharSet = CharSet.Unicode,
CallingConvention = CallingConvention.ThisCall)]
        public static extern void PrintCPerson(IntPtr this);
```

- CharSet is used to define the encodage of string, not used yet
- CallingConvention is used to define the calling convention, here we use the ThisCall which includes the implicit pointer *this*
    - To learn more about calling conventions of the C++ look at http://www.codeproject.com/Articles/1388/Calling-Conventions-Demystified

Using dll from C#: use class

how to pass string parameters

lib.h

```
#pragma once
#include <wchar.h>

class __declspec(dllexport) CPerson {
private:
     wchar_t   Name;
     wchar_t   FirstName;
public:
     CPerson(wchar_t name, wchar_t firstname);
     ~CPerson(void);
     void Print();
}

extern "C" __declspec(dllexport) void CPersonDelete (CPerson* cp);
extern "C" __declspec(dllexport) CPerson* CPersonNew (wchar_t* name, wchar_t* firstname);
```

- std::string is replaced par wchar_t (a more basic type)
- some types are automatically marshalled, see blittable types
- try to stay on basic types, arrays can be tricky to pass

Using dll from C#: use class

how to pass string parameters

lib.cpp

```
#include "lib.h"

CPerson::CPerson(wchar_t name, wchar_t firstname){
    wcscpy_s(Name, name);
    wcscpy_s(FirstName, firstname);
}

CPerson::~CPerson(void){}

void CPerson::Print(){
    wprintf(L"My name is %s %s\n", FirstName, Name);
}


extern "C" __declspec(dllexport) void CPersonDelete (CPerson* cp){
    delete cp;
}

extern "C" __declspec(dllexport) CPerson* CPersonNew (wchar_t* name, wchar_t* firstname){
    return new CPerson(name, firstname);
}
```

- wcscpy_s is used to copy the wchar
- wprintf is used to output the wchar

Using dll from C#: use class

main.cs

how to pass string parameters

```
[DllImport("lib.dll", EntryPoint = "CPersonNew", CharSet = CharSet.Unicode)]
     public static extern IntPtr NewCPerson();

[DllImport("lib.dll", EntryPoint = "CPersonDelete", CharSet = CharSet.Unicode)]
     public static extern void DeleteCPerson(IntPtr cp);

[DllImport("lib.dll", EntryPoint = "?Print@CPerson@@QAEXXZ", CharSet = CharSet.Unicode,
CallingConvention = CallingConvention.ThisCall)]
     public static extern void PrintCPerson(IntPtr this);

static void Main(string[] args){
     IntPtr cp = NewCPerson("Buendia", "Axel");
     PrintCPerson(cp);
     DeleteCPerson(cp);
}
```

- be careful about the mangling of Print which could have changed (should not)
- the wchar_t is automatically marshalled to C# string

Other interactions

- C++/CLI
  - Compile C++ in a managed way
  - strong interoperability
  - used to call C# from native C++
- COM
  - declare COM component
  - strong interoperability with any language
  - more complex to declare, have to be registered (registry)
- final comments:
  - undname.exe is a tool that undecorate names retrieving the original declaration
  - try to use only basic types