مكتبات الربط الديناميكي: التطوير و الإستخدام

بن العيد

20/05/2008

بسم الله الرحمن الرحيم

"... وَمَنْ يُّعْرِضْ عَن ذِكْرِ رَبِّهِ، نَسْلُكْهُ عَذَابًا صَعَدًا".

الجن (17).

صدق الله العظيم

الفهرس

1. مقدمة

نظرا لأهمية إستخدام مكتبات الربط الديناميكي في تطوير البرمجيات و ما تتسم به من ثراء معرفي أشارككم في هذه الأسطر خبرتي المتواضعة في هذا المجال. هذا الدرس، إن صحت تسميته كذلك، يخص بالأساس الجانب التطبيقي.

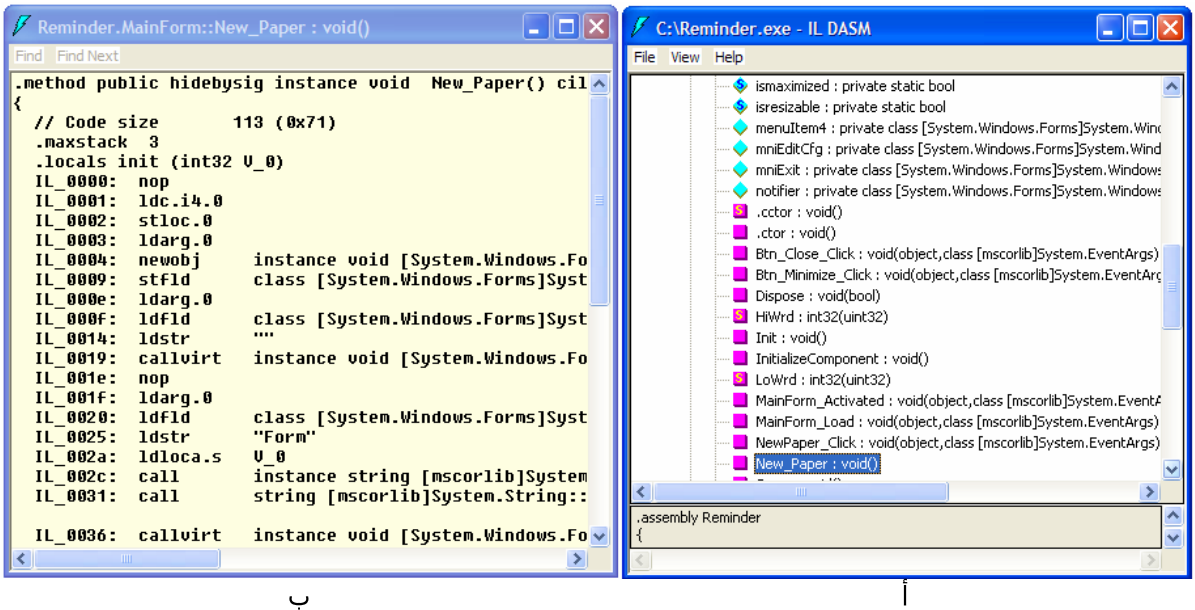
ينقسم هذا الدرس إلى ثلاثة أجزاء رئيسية؛ أولها يخص فوائد إستخدام مكتبات الربط الديناميكي كالحماية و الحاجة للربط بين لغات برمجة مختلفة. ثانيها يخص مراحل تطوير مكتبة ديناميكية بلغة السي مرورا بالإعدادات التي يمكن أن نحتاج لتعديلها لتحديد مسار إنشاء المكتبة أو إدراج ملفات خارجية و غير ذلك. ثالثها يهتم بكيفية إستدعاء المكتبة المطورة في الجزء الثاني في لغات برمجة مختلفة و التعديلات التي يتوجب إدراجها لشفرة المكتبة إذا إستوجب ذلك.

2. الغاية من مكتبات الربط الديناميكي

قبل الشروع في الحديث عن تطوير مكتبات الربط الديناميكي, توقف قليلا للإطلاع على بعض فوائد إستخدامها في تطوير و حماية البرمجيات.

1.2. الحماية

يطول الحديث عن حماية البرمجيات و خاصة المطورة في منصة الدوت نت. هذه الأخيرة جائت بكافة العدة اللازمة لتطوير البرمجيات في أسرع وقت و بإتقان كبير و في الآن ذاته جائت بالأداة ildasm[1] لتكشف الستار عن إمكانية تطبيق الهندسة العكسية على ملفات التشغيل و المكتبات المطورة بإحدى لغاتها. هاته الأداة متواجدة في Microsoft Visual Studio 8\SDK\v2.0\Bin\ildasm.exe و لك أن تتبع هذا المسار آخذا بعين الإعتبار إصدار الفيزوال ستوديو و عدة البرمجة اللذان تعتمدهما.



ب أ

الصورة رقم1: الأداة IL Disassembler.

[1] ILDasm: Intermediate Language Disassembler

الصورة رقم 1.أ تجسد الأداة إثر فتح ملف تشغيل لتطبيق مطور بلغة السي شارب. تظهر هذه الأداة أسماء الدوال الأصناف و غيرها من تفاصيل التي يمكن التمييز بينها من خلال الرموز المسندة لكل منها و التي أبرزتها في الجدول رقم 1. الصورة الثانية (ب) تجسد بدورها الشفرة الخاصة بالعنصر الذي تم النقر عليه مزدوجا و لرؤية النوع أنقر Ctrl+M, أما إذا أردت نسخ الشفرة كاملة لملف التشغيل أو المكتبة التي تم فتحها بهذة الأداة فيمكنك إختيار File\Dump لحفظه في ملف il. في المسار الذي تختاره [5].

الـرمـز	الـوصف	
	مزيد من المعلومات	More Information
	حقل	Field
	حقل ثابة	Static Field
	صنف	Class
	مجال	Namespace
	دالة	Method
	دالة ثابتة	Static Method
	خاصية	Property
	وسيط	Interface
		Structure
	حدث	Event

الجدول رقم 1: فهرس الرموز [5].

في المقابل لهذه الأداة توجد أداة أخرى وهي ilasm و تمكن من إسترجاع ملف التشغيل أو المكتبة التي تم حفظ الشفرة المكونة لها.

الإشكال هنا يكمن في أن ملفات التشغيل لا تحتوي ما يسمى ب executable machine code إنما شفرة لغة وسيطة خاصة بمايكروسوفة (IL أو MSIL[2]) و كافة مترجمات منصة الدوت نت تولد شفرة MSIL و هو ما يفسر تشابه كافة لغات هذه المنصة من حيث النجاعة.

إقترح عدة مزودين إصدار من الأداة obfuscator خاصة بمنصة الدوت نت بعد أن سبقتهم به الجافا. هاته الأداة لا توفر حماية كاملة للشفرة إنما تجعل شفرة اللغة الوسيطة (IL) أصعب في القراءة، أي كما يقال حل المشكل بتأجيل النظر فيه إلى أن جاء البرنامج الشهير Reflector الذي تجاوز ILDasm ليظهر الشفرة الأصلية للملف المطور في منصة الدوت نت بوضوح لا غبار عليه. لذلك فإن إخفاء الجزء الهام من وضائف البرنامج المطور بإحدى لغات الدوت نت في مكتبة ربط ديناميكي مطورة بالسي أو السي++ سيجعله محمي و لايمكن الإطلاع عليه، و لكن يمكن رؤية كيفية إستدعاء الدوال المعرفة في المكتبة أي يمكن إعادة إستخدام المكتبة إذا لم يكن هناك مانع مثل ربط جميع وضائفها بدالة تسجيل (Registration).

[2] MSIL: Microsoft Intermediate Language.

2.2. الربط بين لغات البرمجة

من بين الدوافع للربط بين لغات البرمجة:

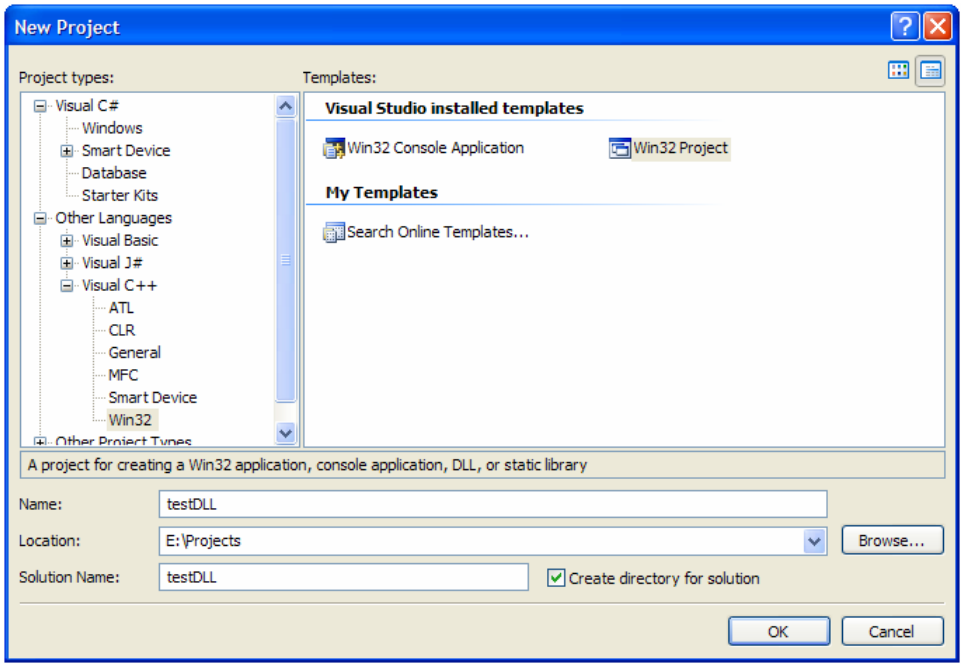
- إعادة إستغلال شفرة مطورة بلغة مغايرة للغة التي نعتمدها. فنادرا ما يكون المطور متقنا لعدة لغات في آن واحد و لا يمكن الوثوق بنجاعة برامج التحويل بين لغات البرمجة لأنها عادة ما تكون آلية و "غبية". لذلك إدراج تلك الشفرة في مكتبة ربط ديناميكي إن أمكن يحل المشكل خاصة إن كانت تلك الشفرة مطورة بالسي، السي++ أو إحدى لغات الدوت نت.
- عجز لغة البرمجة المعتمدة على إنجاز وضيفة ما مثل التحكم في السواقات و سير عمل المعالج و غير ذلك. فالسي، سي++ و لغة الآلة تمكن بيسر مثل هذه الوضائف و إذا أردناها في تطبيق مطور بالجافا مثلا، فلا مفر من إدراجها في مكتبة ربط ديناميكي.
- سرعة عمل البرنامج المطور عامل من عوامل نجاحه أو فشله، مما يتوجب على المطور حذف التعليمات الزائدة و الحلقات المفرغة أو العقيمة و إستبدالها بما يزيد في نجاعة البرنامج و سرعته. من الأسباب الجاعلة للبرنامج بطيء سوء إختيار اللغة المناسبة لإنجاز وضيفة ما، لذلك يمكن كتابة ذلك الجزء بلغة أنسب و تجعله أسرع. الحديث هنا لايخص تطبيقات بسيطة إنما مشاريع يكون فيها الوقت عامل أساسي كالبرمجيات الخاصة بالشبكات و معالجة الصور.
- المزج بين لغتي برمجة للإستفادة بإيجابيات كلتيهما، لغات الدوت نت مثلا تمكن و بسهولة من تصميم واجهات رسومية حرفية على عكس ال MFC التي تتسم بالتعقيد. لذلك يمكن كتابة الوضائف الرئيسية بلغة السي أو السي++ و تصميم الواجهة الرسومية بالسي شارب مثلا.

إثر هذا الإستعراض لفوائد إستخدام مكتبات الربط الديناميكي ننتقل للجزء الموالي و الذي يهدف لشرح مراحل إنشاء مكتبة بلغة السي.

2.تطوير مكتبات الربط الديناميكي

الغاية من هذا التطبيق شرح كيفية إنشاء مكتبة ربط ديناميكي بلغة السي و السي++ مع إيضاح بعض المسائل المتعلقة بالمشاكل التي يمكن أن يواجهها المطور كتغير مسار إنشاء المكتبة, إدراج مكتبات و ملفات خارجية و غيرها.

المرحلة الأولى في تطوير المكتبة تتمثل في إختيار لغة البرمجة، نوع التطبيق و أيضا نظام التشغيل الموجهة له هذه المكتبة لأنه كما تجسد الصورة رقم 2 من الممكن أيضا إنشاء مكتبة ربط ديناميكي لحواسيب الكف و الهواتف الذكية عند تظليل الإختيار Smart Device.



الصورة رقم 2: إنشاء مشروع جديد

المرحلة التالية تتمثل في إختيار التطبيق من نوع DLL[3] من قائمة الإختيارات التي توفرها المجموعة Win32 Project أي سننشأ مكتبة ربط ديناميكي لحاسوب المكتب بلغة السي و من نوع Win32 علما أنه من الممكن إنشاء مكتبات بلغة ال MFC أو ال ATL.

الشفرة الأولية التي يولدها الفيزوال ستوديو تتمثل في ما يلي:

```
#include "stdafx.h"

#ifdef _MANAGED
#pragma managed(push, off)
#endif

BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call,
                       LPVOID lpReserved )
{ return TRUE;}

#ifdef _MANAGED
#pragma managed(pop)
#endif
```

إذا جربت ترجمة الشفرة السابقة فستتحصل على المكتبة الديناميكية و تكون في المجلد Debug التابع للمشروع ولكننا أنشأنا إلى حد الآن مكتبة "عقيمة" لأنها خالية من المخرجات. المقصود بالترجمة هنا القيام بما يسمى Build لأن عملية Debug تخص المشاريع التي تولد ملفات تشغيل.

سنجرب الآن إدراج الدالة FirstMethod للشفرة الأولية التي يولدها الفيزوال ستوديو. هذه الدالة ستكون من مخرجات المكتبة. هذه الدالة من نوع int و ترجع متغير من نوع int أيضا.

```
extern "C" __declspec(dllexport) int _stdcall FirstMethod()
  { int output=7;
    return output;
  }
```

[3] DLL: Dynamic Library Link.

تعريف الدوال بهذه الطريقة كاف إذا أردنا إستدعائها في منصة الدوت نت و لكن إذا أردنا إستدعائها في شفرة مطورة بلغة السي++ فينبغي تعديل معرفات الدالة و إضافة تعليمات تجعلها المكتبة مستعملة من قبل السي و السي++ في الآن ذاته.

```
#include "stdafx.h"
#include "stdio.h"

#ifdef __cplusplus    // used in C++,
extern "C" {          //export the C interface
#endif

#ifdef _MANAGED
#pragma managed(push, off)
#endif

BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call, LPVOID
lpReserved )
  { return TRUE; }

#ifdef _MANAGED
#pragma managed(pop)
#endif

__declspec(dllexport) int __cdecl FirstMethod()
  { printf("this is an output of the API FirstMethod. \n");
    int output=7;
    return output;
  }

#ifdef __cplusplus
}
#endif
```

لو كان لك أكثر من دالة منتمية إلى صنف أو عدة أصناف فليس من الضروري تعريفها كلها بالطريقة ذاتها للدالة و إنما هذا التعريف يخص فقط الدوال التي نريد "تصديرها" من المكتبة. في المثال التالي عرفت الدالة debugMethod و التي سيتم إستدعائه في الدالة FirstMethod دون أن تكون إحدى مخرجات المكتبة:

```
……
#endif

void debugMethod()
  {   printf("this is an output of the API FirstMethod. \n");
  }

__declspec(dllexport) int __cdecl FirstMethod()
  { debugMethod();
        return 7;
  }
....
```

أحيانا نحتاج لإدراج مكتبات lib. أو ملفات h. للمشروع و هي عادة ما تكون مكتبات خارجية أو مطورة من قبل مطورين مستقلين(Third Party). و رغم إدراج هاته الملفات في مجلد المشروع فإن رسالة

الخطأ بعدم العثور على هذه الملفات يتواصل. لذلك فمن الممكن تحديد مسار المجلد الذي يحتوي المكتبات lib. في الخانة الخاصة بذلك في
Configuration Properties\Linker\General\ Additional Library Directories. أما الملفات h. التي نحتاج لإضافتها للمشروع فينبغي تحديد مسار المجلد الذي يجمعها في الخانة الخاصة بها في
Configuration Properties\C/C++\Additional Include Directories.
أما إذا كانت المكتبات من توابع الفيزوال ستوديو سوى كانت من جنس التطبيق أم لا شرط أن تكون هناك صلاحية إدراجها فيكفي إضافة إسمها أو أسمائها إن كانت أكثر من مكتبة في الخانة
Configuration Properties\Linker\Input\Additional Dependencies و يمكن القيام بالعملية السابقة برمجيا و كمثال على ذلك:

```
#pragma comment(lib,"Iphlpapi.lib")
#pragma comment(lib,"ws2.lib")
```

ذكرت سابقا أنه من الضروري مراعاة تطابق المكتبة المدرجة مع نوع التطبيق و نظام التشغيل المطورة فيه. فالإنتقال من لغة برمجة إلى أخرى و من نظام تشغيل إلى آخر يدفع في المطور الرغبة في إدراج معارفه السابقة في البيئة الجديدة. إعتمدت كثيرا على الصنف CString التابع لل ATL في تطوير تطبيقات من نوع MFC و نظرا ليسر إستخدامه صرت أستخدمه أيضا في التطبيقات من نوع Win32 بعد إدراج كل من المكتبة atlsd.lib و الملف atlstr.h معا إذا كنت بصدد تطوير مكتبة أما إذا كان تطبيق من نوع فيكفي إدراج الملف atlstr.h:

```
#include "stdafx.h"
#include <atlstr.h>
#include <conio.h>

void _tmain(int argc, _TCHAR* argv[])
{ CString Item("This is a trial.");
  Item=Item.Mid(0,8); // Return only the first 8 Chars
  printf("%S",Item.GetString());
  getch();
 }
```

3. إستدعاء المكتبة

إستدعاء المكتبة المطورة في تطبيقات من نفس لغة البرمجة أو لغة أخرى لا يقل تعقيدا من تطويرها حسب لغة البرمجة. و الصعوبة تكمن أساسا في نجاح إستدعائها و تنفيذ دوالها، بالإضافة إلى التوافق بين أنواع المتغيرات المدخلة و المخرجة لهذه الدوال في كلتا لغتي البرمجة; المطورة بها و المستدعاة فيها.

1.3. إستدعاء المكتبة في شفرة سي /سي++

```
// testarabteam2000.cpp : Defines the entry point for the console
application.

#include "stdafx.h"
#include <windows.h>
#include <stdio.h>

typedef int (__cdecl *MYPROC)(LPWSTR);

void _tmain(int argc, _TCHAR* argv[])
{
    HINSTANCE Instance_To_Load_Lib;
    MYPROC PROC_To_Load_Method;
    BOOL bRet_free_Lib, bRet_To_Load_Method = FALSE;

    // Load DLL
    Instance_To_Load_Lib = LoadLibrary(TEXT("testDLL"));

    // Get function Address if Load DLL succeded
    if (Instance_To_Load_Lib != NULL)
    { PROC_To_Load_Method = (MYPROC) GetProcAddress(Instance_To_Load_Lib,
                                                    "FirstMethod");

     // If the function address is valid, call the function.
     //call the target method if Address is valid
     if (NULL != PROC_To_Load_Method)
      {   bRet_To_Load_Method = TRUE;
          (PROC_To_Load_Method) (L"DLL function found\n");
      }

      //Free the loaded DLL.
      bRet_free_Lib = FreeLibrary(Instance_To_Load_Lib);
    }

    // Failed to load the DLL.
    if (! bRet_To_Load_Method)
    printf("Library not found.\n Please check the existance of the target dll
in c:\\windows\\system32.\n");

    system("PAUSE");
}
```

إذا تم إستخدام الدالة دون تحديد المسار الكامل للمكتبة و إكتفيت بذكر إسمها فسيتم البحث في مسار تواجد المكتبات التي يستخدمها نظام التشغيل أي المجلد System32. أما إذا كانت هناك حاجة لإستدعاء مكتبة أو ملف تشغيل في مسار معين فيمكن تحديده بإستخدام الدالة [4] SetDllDirectory أو مباشرة عبر إدراج المسار الكامل للمكتبة كما يلي:

```
Instance_To_Load_Lib = LoadLibrary(TEXT("c:\\testDLL"));
```

الحل الأيسر لتجاوز مشاكل مسار المكتبة يتمثل في نسخ المكتبة المطورة في المجلد System32 لنظام التشغيل رغم أن البعض يعارض هذا الحل نظرا لإمكانية توافق إسم المكتبة مع مكتبة أخرى متواجدة مسبقا أو توليد مشاكل عند تنصيب أحدى البرمجيات تعتمد مكتبة لها نفس الإسم, هو أمر

مستبعد لأن أغلب البرمجيات التي يتم تنصيبها تستقل بذاتها عبر إنشاء مجلد خاص بها تنسخ فيه ملفاتها و مكتباتها.

يحدد برنامج التطوير الفيزول ستوديو المسار الذي تنسخ فيه المكتبات و الملفات التابعة لها و أسمائها بالكيفية التالية:

```
$(OutDir)\$(ProjectName).dll
```

للإطلاع على ذلك راجع إعدادات المشروع Configuration Properties\Linker\General\Output File.

إسم المكتبة سيكون بنفس إسم المشروع المنشأ لذلك إذا أردت تغيره فيكفي أن تحدد إسمها عوض المتغير $(ProjectName). أما المسار الكامل للمكتبة فيحدده المتغير $(OutDir) الذي تعادل قيمته مسار المجلد Debug التابع للمشروع. يوجد مجلدان يحملان إسم Debug في مجلد المشروع. مجلد تُنسخ فيه المكتبة مع ملف lib. و ملفات أخرى و مجلد آخر يحتوي ملفات و قتية manifest. و ملفات بنتائج الترجمة و معطيات حولها و غير ذلك.

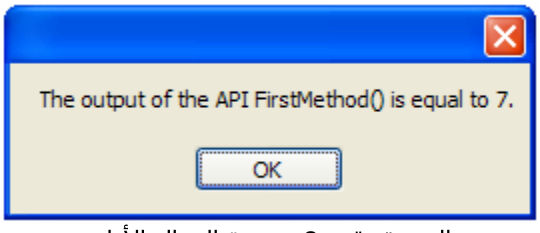
كما سبق و ذكرت أننا سنعتمد مجلد نظام التشغيل كمسار للمكتبة و لذلك ينبغي تغير الإعدادات من $(OutDir)\$(ProjectName).dll إلى C:\Windows\System32\$(ProjectName).dll. الغاية من هذا التغير تجنيب المطور عمل ممل متمثل في نسخ المكتبة يدويا إلى المسار المعني كلما غير في الشفرة و أراد التأكد من النتيجة.

2.3. إستدعاء المكتبة في شفرة دوت نت

إستدعاء مكتبات الربط الديناميكي في منصة الدوت نت يكون بإستخدام الدالة DllImport [3] المدرجة في المجال System.Runtime.InteropServices، وفي المثال التالي قمت بإستدعاء المكتبة testDLL.dll التي تم تعريفها في الجزء السابق لإستخدام الدالة FirstMethod.

```
[System.Runtime.InteropServices.DllImport("testDLL.dll")]
 public static extern int FirstMethod();

 private void MainForm_Load(object sender, EventArgs e)
  { MessageBox.Show(
          String.Format(
                  "The output of the API FirstMethod() is equal to {0}.",
                  FirstMethod()));
  }
```

النوع extern يُستخدم لتعريف الدوال الخارجية المدرجة في التطبيقات المطورة بلغة السي شارب ويقابله النوع Shared في ال VB.Net. في هذا المثال البسيط قمت بإستدعاء المكتبة و قراءة القيمة المخرجة من الدالة المعرفة فيها وإظهارها كما هو مجسد في الصورة رقم 3.



الصورة رقم :3 ترجمة المثال الأول.

عند تطوير المكتبة إعتمدت دالة واحدة كمُخرج للمكتبة و هو بمثابة نموذج، حيث أن عدد الدوال غير محدد و بإمكانك إستخدام ما شئت مع مراعاة الضرورة. لأن كل إستخدام لإحدى دوال المكتبة بمقابل و هو سرعة عمل البرنامج، إستهلاك موارد الجهاز من ذاكرة و معالج و غير ذلك.

نعود مجددا للصنف CString لإبراز كيفية تمرير نص من إحدى دوال المكتبة إلى التطبيق المطور في بيئة الدوت نت ونعيد تعريف الدالة debugMethod بالكيفية التالية:

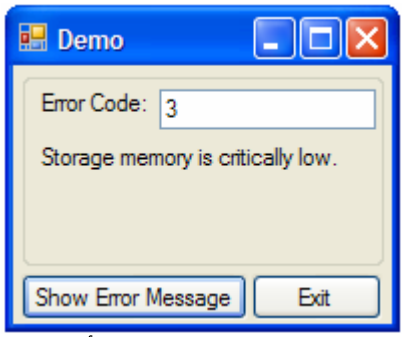
```
__declspec(dllexport) BSTR __cdecl debugMethod(int ErrorCode)
{  CString ErrorMsg("Undefined error");
      switch(ErrorCode)
      { case 1: ErrorMsg=_T("File not found."); break;
        case 2: ErrorMsg=_T("File in use.");  break;
        case 3: ErrorMsg=_T("Storage memory is critically low."); break;
    }
      return ErrorMsg.AllocSysString();
}
```

هذه المرة الدالة تأخذ رقم/عدد طبيعي كمدخل و ترجع نص الخطأ المقابل, من الجهة المقابلة يتغير تعريف الدالة في التطبيق المطور في بيئة الدوت نت بالكيفية التالية:

```
' 1الجزء
Imports System.Runtime.InteropServices
Public Class Form1

' 2 الجزء
  <DllImport("testDLL.dll", EntryPoint:="debugMethod")> _
  Public Shared Function OurSecondMethod(ByVal ErrorNbr As Integer) As IntPtr
  End Function
  <DllImport("testDLL.dll")> _
  Public Shared Function FirstMethod(ByVal ErrorNbr As Integer) As Integer
  End Function

  Private Sub btn_show_Click(ByVal sender As System.Object, ByVal e As _
   System.EventArgs) Handles btn_show.Click
' 3الجزء
     Dim ErrorCode As Integer = Int32.Parse(txb_ErrorCode.Text)
     Dim ptrErrorText As IntPtr = OurSecondMethod(ErrorCode)
     Dim strErrorText As String = Marshal.PtrToStringBSTR(ptrErrorText)
     Marshal.FreeBSTR(ptrErrorText)
     lbl_ErrorText.Text = strErrorText
    End Sub

    Private Sub btn_Exit_Click(ByVal sender As System.Object, ByVal e As _
     System.EventArgs) Handles btn_Exit.Click
        Application.Exit()
    End Sub
End Class
```

لغة الدوت نت هذه المرة هي ال VB.Net مع إضافة إختيار آخر من مجموعة الإختيارات التي يمكن إدراجها في الدالة DllImport و هو EntryPoint (الجزء 2). يمكن هذا الإختيار من تحديد إسم الدالة التي نود إستخدامها من المكتبة مسندين لها إسم مغاير كما هو في المثال حيث أن إسم الدالة هو debugMethod و لكن في التطبيق أسندنا لها إسم مغاير. نلاحظ أيضا أنه تم إستدعاء المكتبة مرتين لأننا في حاجة لكلتا الدالتين معا.

تم تعريف المجال المعرف للدالة DllImport بشكل مستقل و مخالف للمثال السابق لأننا سنحتاجه أيضا لتعريف الصنف Marshal. هذا الأخير يوفر الدالة PtrToStringBSTR التي تمكن من قراءة القيمة المسندة لمؤشر و تحويلها إلى نص لذلك تم تعريف الدالة debugMethod في المكتبة بنوع BSTR و النتيجة مجسدة في الصورة التالية:



الصورة رقم 3: ترجمة المثال الأول الثاني

3.3. إستدعاء المكتبة في شفرة جافا

الآن و قد تعرفنا على كيفية إستدعاء المكتبة المطورة في كل من منصة الدوت نت و السي++ سننتقل إلى لغة برمجة أخرى كثيرة الإستعمال و هي الجافا. الإشكال الأساسي هنا هو أننا نريد الربط بين لغتي برمجة لشركتين مختلفتين و لذلك لابد من وسيط بين اللغتين، هذا الوسيط يسمى JNI[4].

```
#include <jni.h>
```

في هذا المثال سنستخدم دالة ثانية لتتبع الإستثناآت التي يمكن أن تحدث و التي يختص بها الصنف java/lang/RuntimeException في الجافا.

```
void ThrowException(JNIEnv * env, char * strErr)
{
	jclass newExcCls;
	newExcCls = env->FindClass("java/lang/RuntimeException");
	if (newExcCls == NULL)
		return;
	env->ThrowNew(newExcCls, strErr);
}
```

هاته الدالة لن تكون من مخرجات المكتبة إنما ستستخدم ضمنيا و متابعة المشاكل التي يمكن أن تطرأ في سير عمل الدوال الرئيسية.

[4] JNI: Java Native Interface

الوسيط JNI هو عبارة عن غلاف يقوم بالتقديم لبيئة الجافا و التحويل من متغيرات السي و السي++ إلى ما يقابلها في الجافا. و هو ما يجسده المثال التالي:

```
JNIEXPORT jstring JNICALL debugMethod(JNIEnv * env, int ErrorCode)
{ jstring strRet = NULL;
  char ErrorMsg[30];
  switch(ErrorCode)
   { case 1:  sprintf(ErrorMsg,"File not found."); break;
     case 2:  sprintf(ErrorMsg,"File in use."); break;
     case 3:  sprintf(ErrorMsg,"Storage memory is critically low."); break;
     default: ThrowException(env, "Undefined Exception"); break;
   }
  strRet = env->NewStringUTF((const char *) ErrorMsg);
  return strRet;
}
```

لغة البرمجة الجافا توفر بدورها دالة لاستدعاء المكتبات و هي الدالة LoadLibrary التابعة للصنف System و لتعريف المترجم بأن الدالة المعرفة خارجية و متواجدة في مكتبة فلابد من استخدام النوع native وهو ما يماثل استعمال النوع extern في السي شارب و shared في ال VB.NET.

```
public class PInvokeClass
{
static public native String debugMethod(int ErrorCode)throws RuntimeException;
static public native int FirstMethod()throws RuntimeException;
  static {
          System.loadLibrary("testDLL");
         }
}
```

4.خاتمة

في نهاية هذا الدرس إن صحت تسمية كذلك، يصبح المطور مطلع على أهمية إستخدام مكتبات الربط الديناميكي في تطوير البرمجيات و حمايتها خاصة مع ضعف مستوى الحماية في كل من منصة الدوت نت و الجافا. علما أنها ليست الحل الوحيد لحماية الشفرة إنما من الممكن أيضا المزج بين كل من السي/السي ++ و الدوت نت في تطبيقات من نوع MFC [6] ...

المراجع

[1] http://msdn.microsoft.com/en-us/library/ms683152(VS.85).aspx

[2] http://msdn.microsoft.com/en-us/library/ms684175(VS.85).aspx

[3] http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.dllimportattribute.aspx

[4] http://msdn.microsoft.com/en-us/library/ms686203(VS.85).aspx

[5] ASP.NET في ستين دقيقة في اليوم، جلاڤ جونسون، ص 102، 2003.

[6] Extending MFC Applications with the .NET Framework

لكل الإستفسارات، الإقتراحات و التصويب المتعلق بمحتويات هذا الكتاب بالإمكان الإتصال بي عبر البريد الإلكتروني التالي:

Daly.Hammadi@laposte.net