



C# Integration of Wall Risk Engine (NORM libraries)

August 2013

C# Integration Overview

The purpose of this document is to describe how to call the C API of NORM in C# on a .NET platform through a simple example.

The IT requirements are as follows:

- Operating System: Windows Server, Windows XP or higher
- Visual Studio 2008 or higher
- .Net framework: 2.0, 3.0, 3.5, 4.0 or 4.5

Configuration

You need to modify the *app.config* file of your C# project as follows:

A screenshot of a Visual Studio code editor window. The title bar shows three tabs: 'app.config', 'Program.cs', and 'exempleAppelNORM'. The 'app.config' tab is active, displaying XML code. The code is as follows:

```
<?xml version="1.0"?>
<configuration>
  <startup>

    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/></startup>
  <runtime>
    <NetFx40_PInvokeStackResilience enabled="1"/>
  </runtime>
</configuration>
```

The line containing the `<NetFx40_PInvokeStackResilience enabled="1"/>` element is highlighted with a red rectangular box.




This parameter indicates that the verification of the stack state before and after the call to the unmanaged code should be deactivated for .NET frameworks 4.0 and higher.

Delivered short example

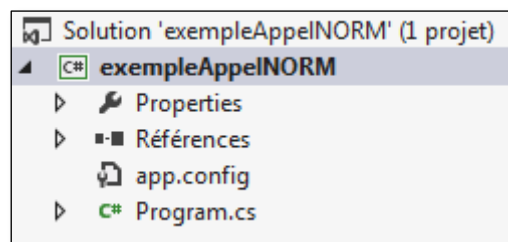
The short example delivered consists in a Visual Studio project that should be opened in a Visual Studio Environment via the .sln file.

This example shows how to call in C# a NORM function (NORMmodelingCov) from the C dll. This short program computes a covariance matrix based on series of returns.

For this example to run, the NORM libraries (.dll files) should be copied in a folder that is in the PATH system variable.

	exempleAppelNORM	22/08/2013 14:00	Dossier de fichiers	
	exempleAppelNORM.sln	21/08/2013 12:58	Microsoft Visual S...	1 Ko
	exempleAppelNORM.suo	22/08/2013 12:03	Visual Studio Solu...	40 Ko

Files of the C# short example



C# project in Visual Studio

Calling NORM from a C# program

First you need to add the following reference at the beginning of your main program file.

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

InteropServices reference

The System.Runtime.InteropServices reference allows to call methods from an unmanaged library.

The declaration of the dll is done as follows:

```
namespace exempleAppelNORM
{
    class Program
    {
        // import NORMAM dlls
        [DllImport("normam-modeling-c-3.1.dll", EntryPoint = "NORMmodelingCov")]
        // declare external functions
        public static extern int NORMmodelingCov(
            ref int returnsSize,
            ref int nbSec,
            double[,] secReturns,
            double[,] covMatrix,
            ref int info
        );

        public static double[,] computeCovarianceMatrix(double[,] returns) {...}

        static void Main(string[] args){...}
    }
}
```

Importing the NORM functions

The DllImport command indicates that the attributed method is exposed by an unmanaged dynamic-link library (DLL) as a static entry point.

The function should be declared as a public static external function:

Public static extern *type* *functionName*(*variables*);

All the NORM functions return an integer corresponding to an error code of the API. See the general NORM documentation for more information about this error code.

All the input and output variables should be passed by address to the NORM function:

- Variable with no dimension should be declared as : ref int *variableName*
- Arrays should be defined as: *type* [] *variableName* or *type*[,] *variableName* depending on the dimension of the array.

The variables are always passed to the NORM functions in the following order:

1. Inputs
2. Outputs of the calculation
3. Error code output for the calculation (info)

Only one call to DllImport is necessary for each NORM dll, with as many *EntryPoint* parameters as functions you want to use in the module. In this example, only the *NORMmodelingCov* function is needed.

NORM functions need to have as an input the size of each input array: *returnsSize* and *nbSec* in the example above correspond to the dimensions of the input array *secReturns*). This is not the case in C#, so you can avoid unnecessary variables by encapsulating the NORM function in a C# function as follows:

```
public static double[,] computeCovarianceMatrix(double[,] returns) {
    int dataSize = returns.GetLength(0);
    int nbAssets = returns.GetLength(1);
    double[,] covMatrix = new double[nbAssets, nbAssets];
    int info = 0;
    int res;
    res = NORMmodelingCov(ref dataSize, ref nbAssets, returns, covMatrix, ref info);
    if (res != 0)
    {
        if (res < 0)
            throw new Exception("ERROR: NORMmodelingCov encountered a problem. See info parameter for more details");
        else
            throw new Exception("WARNING: NORMmodelingCov encountered a problem. See info parameter for more details");
    }
    return covMatrix;
}
```

Encapsulation function in C#

This encapsulation can also be useful in case you do not need to use all the output variables of the NORM function in your C# program.

Finally, in your main program, you just need to call this C# function as follows:

```
static void Main(string[] args)
{
    // header
    Console.WriteLine("*****");
    Console.WriteLine("NORMmodelingCov in C#");
    Console.WriteLine("*****");

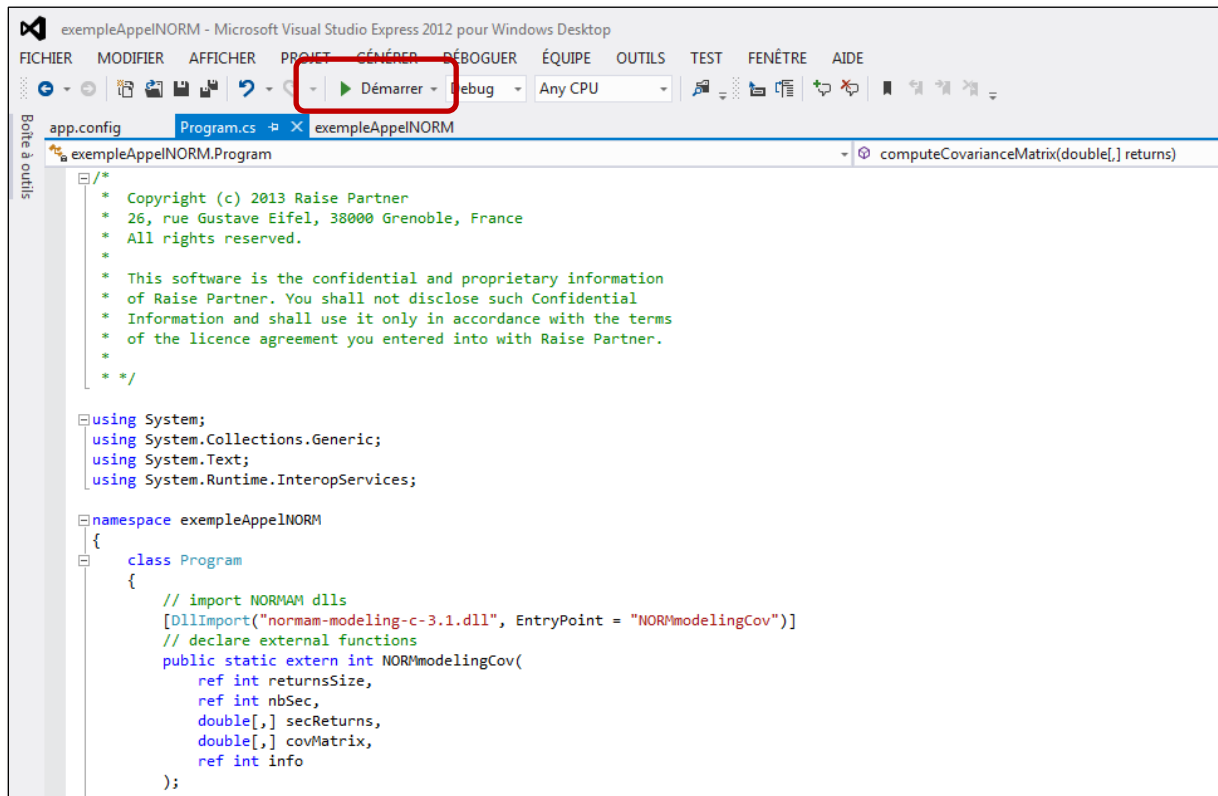
    // data
    double[,] returns = { {0.05, -0.1, 0.6}, {-0.001, -0.4, 0.56}, {0.7, 0.001, 0.12}, {-0.3, 0.2, -0.1},
                          {0.1, 0.2, 0.3}};

    // core
    double[,] myCovMatrix = computeCovarianceMatrix(returns);
    int n = myCovMatrix.GetLength(0);

    Console.WriteLine("Covariance matrix:");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            Console.Write(myCovMatrix[i, j] + "\t");
        }
        Console.WriteLine();
    }
    Console.WriteLine("\nType enter to exit");
    Console.ReadKey(true);
}
```

Main routine

And run it by clicking on the play button:



Launching the C# example