Calling the NAG Fortran Library for Windows x64 DLLs from VB.NET

Users who have Microsoft Visual Studio 2005 or 2008 may use the DLLs provided with the NAG Fortran Library for Windows XP/Vista/7 x64 (FLW6I22DC_nag.dll and FLW6I22DC_mkl.dll) in conjunction with VB.NET.

In this report, we will see the rules one has to follow to use the NAG routines from VB.NET and then we will illustrate those rules with examples.

The examples and Declare statements referred to in this report use the self-contained DLL, FLW6I22DC_nag.dll. To call NAG Fortran Library routines in a different DLL *which uses the same calling conventions* (for example, FLW6I22DC_mkl.dll), simply replace "FLW6I22DC_nag.dll" in the Declare statements by the name of the DLL you wish to use.

Note that the NAG DLLs will need to be on your path and, if you are using (say) an MKL-based DLL such as FLW6I22DC_mkl.dll, the MKL DLLs will also need to be on the path. Please refer to the Users' Note for details.

Contents

1 - Rules to call the routines in FLW6I22DC_nag.dll from VB.NET

To call a routine contained in the DLL, you will need to define it in a `Declare` statement. This `Declare` statement contains the name of the routine, the name of the DLL in which to look for it and the definition of its arguments.

The NAG Fortran Library Documentation is available at
http://www.nag.co.uk/numeric/FL/FLdocumentation.asp
and contains all the necessary information about routine arguments.

The definition of those arguments has to follow some rules:

- All scalar values are passed by reference (`ByRef`).
- All arrays are passed by value (`ByVal`) and have one of the following decorations: `<[In]()>`, `<[Out]()>` or `<[In](),Out[]()>` depending on what is expected from the Fortran point of view. Moreover, VB.NET and Fortran don't store arrays in memory in the same way (Fortran is column-wise whereas VB.NET is row-wise). All arrays in VB.NET must be transposed before calling Fortran subprograms.
- When the Fortran expects a scalar argument of the type `CHARACTER*`, the actual VB.NET argument is a `String`, passed by value (`ByVal`). The length of this `String` must be passed at the end of the argument list.
- When the Fortran expects an array argument of the type `CHARACTER*`, the actual VB.NET argument is a single `String` in which all the array elements are concatenated. It is passed by value (`ByVal`). The length of one element of the array must be passed at the end of the argument list.
- When the Fortran expects a call-back function as argument, an interface must be declared

on the VB.NET side by defining a `Delegate` Function and the routine argument is passed by value (`ByVal`) as being of the type of this `Delegate` function. When calling the Fortran routine from VB.NET, the function implementing the `Delegate` interface will be passed as an argument using the keyword `AddressOf`.

For these declarations to be recognised by VB.NET, the following line is needed at the top of the VB.NET source code:

```
Imports System.Runtime.InteropServices
```

This zip file contains a VB.NET module including the `Declare` statements for all the library routines.

You may also need the Intel Fortran Runtime DLLs included in this zip file if you don't have the Intel Fortran Compiler installed.

Here are a few examples illustrating how to call NAG Fortran Library routines from VB.NET.
The source code for all these examples is available here.

2 - Calling a routine with scalar arguments and a call-back function

The NAG Fortran Library routine called here is D01BDF. According to the NAG Fortran Library documentation, this routine has the following Fortran prototype:

```
SUBROUTINE D01BDF(F, A, B, EPSABS, EPSREL, RESULT, ABSERR)
double precision  F, A, B, EPSABS, EPSREL, RESULT, ABSERR
EXTERNAL          F
```

All the arguments are scalar except for `F` which is an `EXTERNAL` function returning a ***double precision*** value.

More precisely, the documentation indicates that the Fortran interface for `F` is:

```
double precision FUNCTION F(X)
double precision            X
```

The first thing to do is to declare the VB.NET prototype for this call-back function. In this case it will be a `Delegate Function` returning a `Double` and taking a `Double` as argument. Its declaration will be:

```
Delegate Function D01BDF_F_DELEGATE( _
    ByRef X As Double _
  ) as Double
```

Now that the `Delegate` function is known, the `Declare` statement for D01BDF can be easily written:

```
Declare Sub D01BDF Lib "FLW6I22DC_nag.dll" ( _
    ByVal F As D01BDF_F_DELEGATE, _
    ByRef A As Double, _
    ByRef B As Double, _
    ByRef EPSABS As Double, _
    ByRef EPSREL As Double, _
    ByRef RESULT As Double, _
    ByRef ABSERR As Double _
   )
```

All the scalar arguments are passed by reference and the call-back function F is passed by value as being of the type of the `Delegate` function.

Below is a short VB.NET program calling D01BDF:

```
Option Explicit On
Imports System.Math
Imports System.Text
Imports System.Runtime.InteropServices
Imports System.Reflection
Imports System.IO
Imports Microsoft.VisualBasic

Module Module1

Delegate Function D01BDF_F_DELEGATE( _
    ByRef X As Double _
  ) as Double

Declare Sub D01BDF Lib "FLW6I22DC_nag.dll" ( _
    ByVal F As D01BDF_F_DELEGATE, _
    ByRef A As Double, _
    ByRef B As Double, _
    ByRef EPSABS As Double, _
    ByRef EPSREL As Double, _
    ByRef RESULT As Double, _
    ByRef ABSERR As Double _
   )

Sub Main()
        Const A As Double = 0.0#, B As Double = 1.0#
        Dim epsabs As Double, epsrel As Double
        Dim myres As Double, abserr As Double
        Dim strResult As New StringBuilder

        epsabs = 0.0
        epsrel = 0.0001

        'THE KEYWORD AddressOf IS USED TO PASS FUN AS ARGUMENT
        Call D01BDF(AddressOf FUN, A, B, epsabs, epsrel, myres, abserr)

        strResult.Append("D01BDF Example Results" & vbCrLf)
        strResult.Append(vbCrLf & "Result of Integration" & vbCrLf)
        strResult.Append(myres)
        MessageBox.Show(strResult.ToString)
        strResult = Nothing

    End Sub

    Function FUN(ByRef x As Double) As Double
        FUN = x * x * Sin(10 * PI * x)
        Exit Function
    End Function
End Module
```

3 - Calling a routine with array arguments

Here the Fortran routine that is to be called is G02BAF. According to the NAG Library Documentation this routine is declared in the following manner:

```
SUBROUTINE G02BAF(N, M, X, LDX, XBAR, STD, SSP, LDSSP, R, LDR, IFAIL)
INTEGER           N, M, LDX, LDSSP, LDR, IFAIL
double precision  X(LDX,M), XBAR(M), STD(M), SSP(LDSSP,M), R(LDR,M)
```

Once again according to the documentation, the `INTENT` attributes for the arrays are:

- `X INTENT(IN)`
- `XBAR INTENT(OUT)`

- `STD INTENT(OUT)`
- `SSP INTENT(OUT)`
- `R INTENT(OUT)`

With all those details, it is possible to determine the following `Declare` statement for G02BAF:

```
Declare Sub G02BAF Lib "FLW6I22DC_nag.dll" ( _
    ByRef N As Integer, _
    ByRef M As Integer, _
    <[In]()> ByVal X As Double(,), _
    ByRef LDX As Integer, _
    <[Out]()> ByVal XBAR As Double(), _
    <[Out]()> ByVal STD As Double(), _
    <[Out]()> ByVal SSP As Double(,), _
    ByRef LDSSP As Integer, _
    <[Out]()> ByVal R As Double(,), _
    ByRef LDR As Integer, _
    ByRef IFAIL As Integer _
   )
```

Here the Fortran `INTENT` attributes have been specified in the VB.NET source code with `<[IN]()>` or `<[OUT]()>`. If the Fortran `INTENT` attribute was `INTENT(INOUT)`, the matching VB.NET source code would be `<[IN](),OUT[]()>`.

Below is the conversion of Fortran example program g02bafe.f showing the use of G02BAF from VB.NET.

```
Option Explicit On
Imports System.Math
Imports System.Text
Imports System.Runtime.InteropServices
Imports System.Reflection
Imports System.IO

  Imports Microsoft.VisualBasic

Module Module1
    Declare Sub G02BAF Lib "FLW6I22DC_nag.dll" _
      (ByRef N As Integer, ByRef M As Integer, _
               <[In]())> ByVal X As Double(,), _
                         ByRef LDX As Integer, _
             <[Out]()> ByVal XBAR As Double(), _
              <[Out]()> ByVal STD As Double(), _
             <[Out]()> ByVal SSP As Double(,), _
                       ByRef LDSSP As Integer, _
               <[Out]()> ByVal R As Double(,), _
    ByRef LDR As Integer, ByRef IFAIL As Integer)

    Sub Main()
        Dim M As Integer = 3
        Dim N As Integer = 5
        Dim IA As Integer = N
        Dim ISSP As Integer = M
        Dim ICORR As Integer = M

        Dim I As Integer
        Dim IFAIL As Integer
        Dim J As Integer

        Dim A(M - 1, IA - 1) As Double
        Dim AMEAN(M - 1) As Double
        Dim CORR(M - 1, ICORR - 1) As Double
        Dim SSP(M - 1, ISSP - 1) As Double
        Dim STD(M - 1) As Double

        'START DATA INITIALISATION - THE ARRAY A IS TRANSPOSED

        A(0, 0) = 2.0
        A(0, 1) = 4.0
        A(0, 2) = 9.0
        A(0, 3) = 0.0
        A(0, 4) = 12.0

        A(1, 0) = 3.0
        A(1, 1) = 6.0
        A(1, 2) = 9.0
        A(1, 3) = 12.0
        A(1, 4) = -1.0

        A(2, 0) = 3.0
        A(2, 1) = 4.0
        A(2, 2) = 0.0
        A(2, 3) = 2.0
        A(2, 4) = 5.0

        'END DATA INITIALISATION

        Console.WriteLine("G02BAF Example Program Results")
        Console.WriteLine("Number of variables (columns) = " & M)
        Console.WriteLine("Number of cases (rows) = " & N)
        Console.WriteLine()
        Console.WriteLine("Data matrix is:")
        For J = 1 To M
            Console.Write(" " & J)
        Next

        Console.WriteLine()

        For I = 0 To N - 1
            For J = 0 To M - 1
                Console.Write(Format(A(J, I), "###0.0") & "  ")
            Next
            Console.WriteLine()
        Next

        IFAIL = 1
        'CALL TO THE NAG LIBRARY ROUTINE
        Call G02BAF(N, M, A, IA, AMEAN, STD, SSP, ISSP, CORR, ICORR, IFAIL)

        If (IFAIL <> 0) Then
            Console.WriteLine("Routine fails, IFAIL = " & IFAIL)
        Else
            Console.WriteLine("Variable   Mean  St. dev.")
            For I = 0 To M - 1
                Console.Write("  " & Format((I + 1), "###0"))
                strResult.Append("            ")
                Console.Write(Format(AMEAN(I), "###0.000"))
                strResult.Append("     ")
                Console.WriteLine(Format(STD(I), "###0.000"))
            Next
            Console.WriteLine()
            Console.WriteLine _
                   ("Sums of squares and cross-products of derivations")
            For I = 0 To M - 1
                For J = 0 To M - 1
                    Console.Write(Format(SSP(J, I), "###000.00") & "  ")
                Next
                Console.WriteLine()
            Next
            Console.WriteLine("Correlation coefficients")
            Console.WriteLine()
            For I = 0 To M - 1
                For J = 0 To M - 1
                    Console.Write(Format(CORR(J, I), "###0.000") & "   ")
                Next
                Console.WriteLine()
            Next
        End If
        Console.ReadLine()

    End Sub

End Module
```

The results can then be compared with the results provided with the NAG Fortran Library or alternatively in the NAG Fortran Library Documentation.

4 - Calling a routine with CHARACTER arguments

The NAG Fortran Library routine which will be studied here is M01CCF. According to the NAG Fortran Library Documentation, its Fortran protoype is:

```
SUBROUTINE M01CCF(CH, M1, M2, L1, L2, ORDER, IFAIL)
INTEGER           M1, M2, L1, L2, IFAIL
CHARACTER*(*)     CH(M2)
CHARACTER*1       ORDER
```

It is then possible to define the matching `Declare` statement:

```
Declare Sub M01CCF Lib "FLW6I22DC_nag.dll" ( _
    ByVal CH As string, _
    ByRef M1 As Integer, _
    ByRef M2 As Integer, _
    ByRef L1 As Integer, _
    ByRef L2 As Integer, _
    ByVal ORDER As string, _
    ByRef IFAIL As Integer, _
    ByVal CHLength As Integer, _
    ByVal ORDERLength As Integer _
    )
```

In this statement, there are two arguments of type `String`, the first one matching a `CHARACTER*(*)` Fortran array and the second one a simple `CHARACTER*1` Fortran variable. The corresponding lengths to be passed are at the end of the argument list, in order of appearance.

The first length must be the length of one element of the array concatenated in `CH` and the second is the actual length of `ORDER`.

The example below illustrates the use of this `Declare` statement:

```
Option Explicit On
Imports System.Math
Imports System.Text
Imports System.Runtime.InteropServices
Imports System.Reflection
Imports System.IO
Imports Microsoft.VisualBasic

Module module1

Declare Sub M01CCF Lib "FLW6I22DC_nag.dll" ( _
    ByVal CH As string, _
    ByRef M1 As Integer, _
```

```
        ByRef M2 As Integer, _
        ByRef L1 As Integer, _
        ByRef L2 As Integer, _
        ByVal ORDER As string, _
        ByRef IFAIL As Integer, _
        ByVal CHLength As Integer, _
        ByVal ORDERLength As Integer _
        )

    Sub Main()

            Dim ch(10) As String
            Dim ch_len As Integer
            Dim i As Integer, ifail As Integer, l1 As Integer, l2 As Integer, m As
    Integer
            Dim strResult As New StringBuilder
            Dim ch_all As String

            ch(0) = "A02AAF   289"
            ch(1) = "A02ABF   523"
            ch(2) = "A02ACF   531"
            ch(3) = "C02ADF   169"
            ch(4) = "C02AEF   599"
            ch(5) = "C05ADF  1351"
            ch(6) = "C05AGF   240"
            ch(7) = "C05AJF   136"
            ch(8) = "C05AVF   211"
            ch(9) = "C05AXF   183"
            ch(10) = "C05AZF  2181"



            m = 11
            l1 = 7
            l2 = 12
            ifail = 1
            ch_all = ""

            'THE ARRAY TO SORT IS CONCATENATED IN ONE STRING
            For i = 0 To m - 1
                ch_all = ch_all & ch(i)
            Next

            'ch_len IS THE LENGTH OF ONE ELEMENT OF CH
            ch_len = 12

            M01CCF(ch_all, 1, m, l1, l2, "Reverse ASCII", ifail, ch_len,13)

            For i = 0 To m - 1
                ch(i) = ch_all.Substring(i * ch_len, ch_len)
            Next

            strResult.Append("M01CCF Example Results" & vbCrLf & vbCrLf)
            strResult.Append("Records sorted on columns " & l1.ToString)
            strResult.Append(" to " & l2.ToString & vbCrLf & vbCrLf)

            For i = 0 To m - 1
                strResult.Append(ch(i).ToString & vbCrLf)
            Next

            strResult.Append(vbCrLf & "ifail on exit = " & _
                                     ifail.ToString & vbCrLf)
            MessageBox.Show(strResult.ToString)
            strResult = Nothing

        End Sub
    End Module
```