

Calling 32-bit NAG C DLL functions from Visual Basic

George Levy, NAG Ltd, Oxford

Abstract

Calling NAG routines from Visual Basic (or Excel) can provide a convenient way quickly to develop graphical user interfaces and packages that use mathematical routines. The aim of this report is to show how the complete range of NAG C DLL routines can be called directly from Visual Basic, and thus allow programmers/package builders maximum flexibility when incorporating calls to NAG routines into their Visual Basic code.

Keywords DLLs Visual Basic Excel Visual C++

1 Introduction

The PC Windows environment is constructed so that all applications make calls to Dynamic Link Libraries (DLLs). These DLLs may contain system library routines provided by a particular computer vendor (e.g., the DLLs for Windows 95 supplied by Microsoft) or may be customized third party DLLs which provide a set of specialised functions (e.g., graphical, mathematical, text processing etc).

NAG has developed 32-bit mathematical DLLs containing the routines in its Fortran 77 and C numerical libraries [1], [2]. It is envisaged that these routines will be mainly used by mathematical/scientific programmers who use Visual C++, Fortran PowerStation or Visual Basic to build their applications.

In particular, calling NAG routines from Visual Basic (or Excel [3]) can provide a convenient way quickly to develop graphical user interfaces and packages that use mathematical routines. These interfaces, which may use dialog boxes, menus etc, are simple to use and do not require the end-user to have any knowledge of the underlying DLL routine, or even computer languages such as Fortran 77, C and Visual Basic [4], [5].

This report is intended for Visual Basic programmers/package builders, and is concerned with the issues involved in calling the NAG C DLL routines from Visual Basic.

The main problems associated with using the DLL routines are

- Data type matching
- Function declarations
- Null pointers
- Function arguments
- Two-dimensional array arguments
- Allocation of storage
- Software limitations

One method of dealing with these issues is to do most of the programming in C and use an auxiliary DLL to pass the results back to Visual Basic. This has the disadvantage that, since the NAG C DLL routines cannot be accessed directly from Visual Basic, minor variations to routine arguments may require the auxiliary DLL to be completely rebuilt.

Here we use an alternative approach and illustrate how the full range of C Library routines can be called directly from Visual Basic, and thus give programmers maximum flexibility when using the NAG C DLL.

A rudimentary understanding of pointers is required and, in certain circumstances, an auxiliary DLL needs to be built (template C code is provided for this).

Appendix C contains Visual Basic example code fragments which have been tested (from Windows 95) using both Excel 7.0 and Visual Basic 4.0.

2 Data type matching

This section considers the Visual Basic data types required to match those which occur in the routine argument lists of a (32-bit) NAG C DLL. (More detail concerning data types can be found in the NAG C Library manuals [1].)

Fundamental types

A brief summary of the fundamental types is:

C types	Size in bytes	Visual Basic types
char	1	String * 1
Integer	4	Long
Boolean	4	Long
int	4	Long
double	8	Double

In addition the NAG defined type **Complex** is:

```
typedef struct {double re,im;} Complex;
```

Boolean & enumeration types

In Visual Basic all Boolean or enumeration variables should be declared as type **Long**. For example the C code:

```
typedef enum {Nag_MeanInclude, Nag_MeanZero} Nag_IncludeMean;
typedef enum {Nag_RK_2_3=1, Nag_RK_4_5, Nag_RK_7_8} Nag_RK;

Nag_IncludeMean mean;
Nag_RK rk;
Boolean printit, stopit;

mean = Nag_MeanZero;
mean = Nag_MeanInclude;
printit = TRUE;
stopit = FALSE;
rk = Nag_RK_7_8;
rk = Nag_RK_2_3
```

can be written in Visual Basic as:

```
Dim mean As Long
Dim rk As Long
Dim printit As Long
Dim stopit As Long

mean = 1
mean = 0
printit = 1
```

```

stopit = 0
rk = 3
rk = 1

```

where **FALSE** = 0 and **TRUE** = 1. The above example illustrates that, by default, enumerators in a given C enumeration type declaration start at zero and increase by 1 as the declaration is read from left to right. However, if a given enumerator is assigned a value then subsequent enumerators continue the progression from the assigned value.

Structures

Many of the DLL routines use structures in their argument lists. Sometimes, for example in the d02 and e04 chapters, these structures are rather complicated and may contain other structures. When the structure is relatively simple it is often convenient to define a Visual Basic user-defined type and then make a direct call to the DLL routine. Complicated structures are most easily dealt with by the use of an auxiliary DLL (this is illustrated in the Optimize example in Appendix C).

A structure that is required by nearly all of the routines is the type **NagError**. This is defined as:

```

typedef struct {
    int code;
    Boolean print;
    char message[512];
    void (*handler)(char*, int*,char*);
    Integer errnum;
} NagError;

```

The corresponding Visual Basic user-defined type is:

```

Type NagErrorType
    code As Long
    printm As Long
    Message(511) As String * 1
    handler As Long
    errnum As Long
End Type

```

where both **int**, **Integer** and **Boolean** types have been replaced by **Long** and the pointer to the handler function has been replaced by a structure member of type **Long**.

As another example the type **Nag_Spline** (used by routines e02bac and e02bbc) is defined as:

```

typedef struct {
    Integer n;
    double *lamda;
    double *c;
    Integer init1;
    Integer init2;
} Nag_Spline;

```

The corresponding Visual Basic user-defined type is:

```
Type NagSpline  
  n As Long  
  lamda As Long  
  c As Long  
  init1 As Long  
  init2 As Long  
End Type
```

where both pointers to type **double** have been replaced by integers of type **Long**.

3 Function declarations

The C DLL routines are called by using the following syntax:

```
Declare Function name Lib 'library name' Alias 'decorated name' (arguments) As return type
```

or

```
Declare Sub name Lib 'library name' Alias 'decorated name' (arguments)
```

The routine arguments can be obtained from the appropriate NAG header file and the 'decorated name' is generated from the DLL routine name using the following convention:

An underscore (`_`) is prefixed to the routine name. The name is followed by the at-sign (`@`) character, followed by the number of bytes in the argument list. Appendix A lists the decorated names for the functions exported from the NAG C DLL.

For instance the ANSI C declarations for routines `s17agc` and `g01aac` are:

```
double s17agc(double x, NagError *fail);  
  
void g01aac(Integer n, double x[], double wt[], Integer *nvalid,  
            double *xmean, double *xsd, double *xskew, double *xkurt,  
            double *xmin, double *xmax, double *wsum, NagError *fail);
```

and they would be declared in Visual Basic as:

```
Declare Function s17agc Lib "nagcl04_noopt" Alias "_s17agc@12" (ByVal x As Double, _  
                    ifail As NagErrorType) As Double  
  
and  
  
Declare Sub g01aac Lib "nagcl04_noopt.dll" Alias "_g01aac@48" (ByVal n As Long, _  
                    x As Double, wt As double, nvalid As Long, xmean As Double, _  
                    xsd As Double, _  
                    xskew As Double, xkurt As Double, xmin As Double, xmax As Double, wsum As Double, _  
                    ifail As NagErrorType)
```

In C, pointers are used to pass arguments by reference (e.g., `double *xsd`, `Integer *nvalid`, `double x[]`, etc); here the notation `[]` is used to denote an array argument. When arguments are passed by value in C the syntax *type variable name* (e.g., `Integer n`, `double x`, etc) is used.

In Visual Basic, by default, all arguments are passed by reference; the keyword **ByVal** is required to pass an argument by value.

4 Null pointers

Many of the NAG C Library routines make use of null pointers to indicate that an argument is to be ignored and default action is to be taken.

For example C routine `g01aac` has a pointer argument `wt` which allows the routine to perform statistical computations involving weighted data. If this argument is set to the null pointer then unweighted calculations are performed; all the weights are assumed to be 1. In Visual Basic this can be accomplished by declaring `g01aac` as follows:

```
Declare Sub g01aac Lib "nagcl04_noopt.dll" Alias "_g01aac048" (ByVal n As Long, _
    x As Double, ByVal wt As Long, nvalid As Long, xmean As Double, xsd As Double, _
    xskew As Double, xkurt As Double, xmin As Double, xmax As Double, _
    wsum As Double, ifail As NagErrorType)
```

where the declaration `wt As Long` (instead of `wt As double`) has been used to allow this argument to be used as a pointer. The routine calls

```
Call g01aac(n, x(0), ByVal 0&, nvalid, xmean, xsd, xskew, xkurt, xmin, xmax, wsum, pfail)
Call g01aac(n, x(0), 0, nvalid, xmean, xsd, xskew, xkurt, xmin, xmax, wsum, pfail)
```

are both valid and result in unweighted calculations being performed.

The use of null pointers to indicate default behaviour can also be accomplished when the routines have function arguments. For example routine `d02ejc` has the function arguments `fcn`, `pederv`, `g` and `output`, and an ANSI C function declaration:

```
d02ejc (Integer neq, NAG_D02EJC_FUN fcn, NAG_D02EJC_PFUN pederv, double *t, double y[],
    double tend, double tol, Nag_ErrorControl err_c, NAG_D02EJC_OUTFUN output,
    NAG_D02EJC_GFUN g, Nag_User *comm, NagError *fail);
```

If it is declared in Visual Basic as:

```
Declare Sub d02ejc Lib "nagcl04_noopt.dll" Alias "_d02ejc056" (ByVal neq As Long, _
    ByVal ptr_fcn As Long, ByVal ptr_pederv As Long, x As Double, y As Double, _
    ByVal xend As Double, ByVal tol As Double, ByVal err_c As Long, _
    ByVal ptr_output As Long, ByVal ptr_g As Long, comm As Nag_User, _
    ifail As NagErrorType)
```

then the calls

```
ptr = d02ejc_fcn
Call d02ejc(neq, ptr, ByVal 0&, x, y(0), xend, tol, err_c, ByVal 0&, _
    ByVal 0&, comm, pfail)
Call d02ejc(neq, ptr, 0, x, y(0), xend, tol, err_c, 0, 0, comm, pfail)
```

both result in the internal evaluation of the Jacobian `pederv`. The next section gives more detail concerning function arguments; the ODE example section illustrates the use of the routine `d02ejc`.

5 Function arguments

In contrast to C, Visual Basic procedures are not allowed to have function arguments. This limitation creates a problem when using and declaring DLL routines that require function arguments (these routines are mainly in the d and e chapters of the NAG C Library). A solution to this problem is to create an auxiliary DLL which both defines the required function arguments and also exports pointers to them. These function pointers can then be used in the routine argument lists of DLL procedures called from Visual Basic. The Visual Basic code fragments and auxiliary DLL template C code for the ODE, Optimize and Integrate examples given in Appendix C illustrate this.

6 Two-dimensional array arguments

In Visual Basic multi-dimensional arrays are stored by columns (as in Fortran) rather than by rows, which is the C convention. This means that care must be taken when a DLL routine has matrix (two-dimensional array) arguments.

For example, assume that a 3-by-2 matrix

$$A = \begin{pmatrix} 11 & 12 \\ 21 & 22 \\ 31 & 32 \end{pmatrix}$$

is stored in a Visual Basic 2-dimensional array `a` in the natural manner, as in the following code fragment:

```
Dim a(2,1) As Double

a(0,0) = 11
a(1,0) = 21
a(2,0) = 31
a(0,1) = 12
a(1,1) = 22
a(2,1) = 32
```

The array `a` consists of 6 elements stored in column order, as follows:

11 21 31 12 22 32

However, routines in the NAG C DLL follow the standard C convention that 2-dimensional arrays are stored in *row* order. Suppose the array `a` were passed to a NAG C routine (`f02wec`, say, as in the SVD example in Appendix C):

```
Call f02wec(3,2,a(0,0), ... )
```

where the first two arguments specify the numbers of rows and columns in the matrix. The routine would treat the array as representing a 3-by-2 matrix stored in *row* order:

$$\begin{pmatrix} 11 & 21 \\ 31 & 12 \\ 22 & 32 \end{pmatrix}$$

which is *not* the intended matrix A .

One solution (which is used in Appendix C) is to store the matrix in a 1-dimensional array `a1`, with the element $a_{i,j}$ stored in `a1((i-1)*tda+j-1)`, where `tda` is the *trailing dimension* of the matrix (in this case 2).

```
Dim a1(5) As Double
```

```
Dim tda As Long  
tda = 2
```

```
a1(0) = 11  
a1(1) = 12  
a1(2) = 21  
a1(3) = 22  
a1(4) = 31  
a1(5) = 32
```

```
Call f02wec(3,2,a1(0),tda ... )
```

Another solution is to store the *transpose* of the matrix A in a 2-dimensional array `at`, with `tda` now being the *leading* dimension of the array `at`:

```
Dim at(1,2) As Double
```

```
Dim tda As Long  
tda = 2
```

```
at(0,0) = 11  
at(0,1) = 21  
at(0,2) = 31  
at(1,0) = 12  
at(1,1) = 22  
at(1,2) = 32
```

```
Call f02wec(3,2,at(0,0),tda, ... )
```

The Visual Basic array `at` can be larger than is needed to store the 2-by-3 matrix A^T ; in order that the C routine accesses the correct array elements it is essential that `tda` is set to the correct value:

```

Dim at(3,5) As Double

Dim tda As Long
tda = 4

. . .

Call f02wec(3,2,at(0,0),tda, ... )

```

7 Allocation of storage

The DLL utility routines `x04bec` and `x04bfc` have been provided to facilitate pointer manipulation from within Visual Basic; their Visual Basic declarations are:

```

Declare Sub set_pointer Lib "nagcl04_noopt.dll" Alias _
    "_x04bec@12" (ptr As Long, ar As Double, n As Long)
Declare Sub get_pointer Lib "nagcl04_noopt.dll" Alias _
    "_x04bfc@12" (ptr As Long, ar As Double, n As Long)

```

The routine `set_pointer` allocates $8n$ bytes of internal storage to the pointer `ptr` and copies n elements (of type **double**) from the Visual Basic array `ar`.

The routine `get_pointer` copies n elements from the internal memory associated with pointer `ptr`, to the Visual Basic array `ar` (of type **double**).

A further utility routine `x04bdc` is used to free the memory allocated by `x04bec`; it can be declared as:

```

Declare Sub free Lib "nagcl04_noopt" Alias "_x04bdc@4" (ptr As Long)

```

The use of `set_pointer` can be illustrated by slightly modifying the example given in the section on null pointers. In the following code extract the first call to `g01aac` uses the weights supplied in the array `wt`, while the second results in unweighted computations.

```

set_pointer(ptr,wt(0),n)
Call g01aac(n, x(0), ptr, nvalid, xmean, xsd, xskew, xkurt, xmin, xmax, wsum, pfail)
ptr = 0
Call g01aac(n, x(0), ptr, nvalid, xmean, xsd, xskew, xkurt, xmin, xmax, wsum, pfail)

```

The Spline example, in Appendix C, illustrates the use of both `get_pointer` and `set_pointer` for variable assignment and retrieval within a structure of type **Nag_Spline**.

8 Software limitations

It is recommended that the unoptimised NAG C DLL, 'nagcl04_noopt', be used within Excel/Visual Basic; tests have shown that errors occur when some functions within the optimised DLL, 'nagcl04', are called from Visual Basic.

Functions in Chapter a02 which pass the structure **Complex** by value cannot be called

from Visual Basic. This is because user-defined types may not be passed from Visual Basic to a C DLL by value.

In Excel, function names of the form `c0[1-9][a-z][a-z][a-z]` (e.g., `c02afc`) are not permitted; this means that care must be exercised when using the Visual Basic alias facility on routines in Chapters `c02`, `c05` and `c06`.

9 Discussion of examples

This section provides brief comments on the example code fragments of Appendix C. All the examples use the unoptimized NAG C DLL, 'nagcl04_noopt', and some make use of an auxiliary DLL called 'examples_interfaces'.

The example code has been written so that it can be used from either Excel 7.0 or Visual Basic 4.0. It has been successfully tested with both these packages under Windows 95.

Spline example

This example illustrates the use of the (previously defined) type **Nag_Spline** which is passed to the DLL routines `e02bac` and `e02bbc`. This structure contains two pointers of type **double**, `c` and `lamda`. The memory associated with pointer `lamda` must be allocated before `e02bac` is called while the memory associated with `c` is allocated internally by the routine.

SVD example

In this example the DLL routine is given two Visual Basic declarations. The function `f02wec` is used to calculate both the singular values and also the left and right singular vectors of a matrix *A*; it also requires the declaration of 'dummy' arrays for certain array arguments that are not referenced. The function `f02wec_ptr` is used to calculate the singular values only, and contains the declarations `ByVal q As Long`, `ByVal b As Long` and `ByVal pt As Long` within its argument list. This enables assignment of null pointers to these arguments and thus avoids the use of 'dummy' array arguments when the routine is called. The example also illustrates how data is assigned to the input array `a`; note that `tda` is the second (trailing) dimension of matrix *A*.

ODE example

In the example `d02ejc` first computes a solution with a user-defined Jacobian and then solves the same problem using internal evaluation of the Jacobian. Use is made of pointer arguments to access functions which have return type **void**.

Optimize example

This example requires a variable of type **Nag_E04_Opt** to be initialised and passed to routine `e04ucc`. The type **Nag_E04_Opt** is too complicated to be defined within Visual Basic and so an auxiliary DLL is used. In this approach a variable of type **Nag_E04_Opt** is declared within the auxiliary DLL and a function, called `e04_options_ptr`, is used to export a pointer to the variable. Structure initialization and option setting can then be performed

using routines `e04xxc` and `e04xyc` before `e04_options_ptr` is supplied as an argument to the optimizer routine `e04ucc`.

Integrate example

The example illustrates how several integrals can be evaluated through the use of different function pointers which are exported from an auxiliary DLL. In this case the pointers are used to access functions which have return type **double**.

Acknowledgements

The author acknowledges the work of A Butterworth and S Datardina; he would also like to thank J Du Croz and D Sayers for their comments.

References

- [1] *The NAG C Library Manual, Mark 4*, NAG Ltd, Oxford, UK, 1996.
- [2] *The NAG Fortran Library Manual, Mark 17*, NAG Ltd, Oxford, UK, 1996.
- [3] *Microsoft Office for Windows 95*, Microsoft, 1995.
- [4] B W Kernighan and D M Ritchie *The C Programming Language*, Prentice Hall Software Series, 1988.
- [5] *Visual Basic 4*, Microsoft, 1996.

Appendix A - Decorated function names

The following is a list of the decorated function names exported by the NAG C DLLs.

_a00aac@0	_a00abc@4	_a02bac@16	_a02bbc@16	_a02bcc@16	_a02cac@32	_a02cbc@32
_a02ccc@32	_a02cdc@32	_a02cec@16	_a02cfc@16	_a02cgc@32	_a02chc@32	_a02dac@16
_a02dbc@16	_a02dcc@16	_a02ddc@20	_a02dec@24	_a02dfc@32	_a02dgc@16	_a02dhc@16
_a02djc@16	_a02dkc@16	_a02dlc@16	_a02eic@32	_a02ejc@24		
_c02afc@20	_c02agc@20	_c05adc@44	_c05nbc@28	_c05pbc@36	_c05zbc@28	_c06eac@12
_c06ebc@12	_c06ecc@16	_c06ekc@20	_c06fpc@20	_c06fpz@12	_c06fqc@20	_c06frc@24
_c06fuc@28	_c06gbc@12	_c06gcc@12	_c06gqc@16	_c06gsc@24	_c06gzc@12	_c06hac@20
_c06hbc@20	_c06hcc@24	_c06hdc@24				
_d01ajc@56	_d01akc@56	_d01alc@64	_d01amc@52	_d01anc@68	_d01apc@76	_d01aqc@64
_d01asc@56	_d01bac@32	_d01bbc@40	_d01fcc@44	_d01gac@24	_d01gbc@56	_d02cjc@52
_d02ejc@56	_d02gac@64	_d02gbc@72	_d02nmc@104	_d02nsc@24	_d02nvc@76	_d02pcc@44
_d02pdc@32	_d02ppc@4	_d02pvc@64	_d02pwc@16	_d02pxc@44	_d02pyc@24	_d02pzc@24
_d02qfc@40	_d02qwc@68	_d02qyc@4	_d02qzc@32	_d02rac@80	_d02xkc@72	_d02zac@20
_e01bac@20	_e01bec@20	_e01bfc@32	_e01bgc@36	_e01bhc@40	_e01dac@28	_e01sac@32
_e01sbc@24	_e01szc@4	_e02bac@28	_e02bbc@20	_e02bcc@24	_e02bdc@12	_e02bec@48
_e02dcc@56	_e02ddc@60	_e02dec@24	_e02dfc@28	_e04ccc@28	_e04dgc@32	_e04fcc@44
_e04gbc@44	_e04hcc@28	_e04jbc@44	_e04kbc@44	_e04mfc@48	_e04nfc@60	_e04ucc@60
_e04xxc@4	_e04xyc@24	_e04xzc@12	_e04yac@36	_e04ycc@32		
_f01bnc@20	_f01mcc@28	_f01qcc@24	_f01qdc@44	_f01qec@32	_f01rcc@24	_f01rdc@44
_f01rec@32	_f02aac@20	_f02abc@28	_f02adc@28	_f02aec@36	_f02afc@24	_f02agc@32
_f02awc@20	_f02axc@28	_f02bjc@56	_f02wec@72	_f02xec@72	_f03abc@24	_f03aec@28
_f03afc@28	_f03ahc@28	_f04adc@36	_f04agc@40	_f04ajc@32	_f04akc@32	_f04arc@24
_f04awc@40	_f04mcc@48	_f06eac@20	_f06efc@20	_f06pac@52	_f06pbc@60	_f06pcc@48
_f06pdc@52	_f06pec@44	_f06pfc@32	_f06pgc@36	_f06phc@28	_f06pjc@32	_f06pkc@36
_f06plc@28	_f06pmc@40	_f06ppc@32	_f06pqc@28	_f06prc@40	_f06psc@36	_f06sac@68
_f06sbc@76	_f06scc@64	_f06sdc@68	_f06sec@60	_f06sfc@32	_f06sgc@36	_f06shc@28
_f06sjc@32	_f06skc@36	_f06slc@28	_f06smc@48	_f06snc@48	_f06spc@32	_f06sqc@28
_f06src@48	_f06ssc@44	_f06yac@60	_f06ycc@56	_f06yfc@48	_f06yjc@48	_f06ypc@48
_f06yrc@56	_f06zac@76	_f06zcc@72	_f06zfc@56	_f06zjc@56	_f06zpc@48	_f06zrc@64
_f06ztc@72	_f06zuc@64	_f06zwc@72				

-g01aac@48 -g01alc@16 -g01bjc@32 -g01bkc@28 -g01blc@32 -g01cec@12 -g01ddc@28
-g01dhc@24 -g01eac@16 -g01ebc@24 -g01ecc@24 -g01edc@32 -g01eec@48 -g01efc@32
-g01fac@16 -g01fbc@24 -g01fcc@20 -g01fdc@28 -g01fec@36 -g01ffc@36 -g01hac@28
-g02brc@36 -g02bxc@56 -g02cac@52 -g02cbc@72 -g02dac@100 -g02dcc@64 -g02ddc@60
-g02dec@44 -g02dfc@24 -g02dgc@64 -g02dkc@52 -g02dnc@52 -g02fac@36 -g02gac@120
-g02gbc@112 -g02gcc@116 -g02gdc@120 -g02gkc@48 -g02gnc@56 -g02hac@116 -g02hkc@60
-g05cac@0 -g05caz@4 -g05cbc@4 -g05ccc@0 -g05cfc@8 -g05cgc@12 -g05dac@16
-g05dbc@8 -g05ddc@16 -g05dyc@8 -g05eac@32 -g05ecc@16 -g05edc@20 -g05ehc@12
-g05ejc@20 -g05exc@24 -g05eyc@4 -g05ezc@8 -g05fec@28 -g05ffc@28 -g05hac@52
-g07cac@80 -g07dac@28 -g07dbc@92 -g07ddc@44 -g10cac@24 -g11aac@44 -g12aac@36
-g13abc@32 -g13acc@32 -g13bec@56 -g13bjc@76 -g13bxc@4 -g13byc@12 -g13bzc@4
-g13cac@68 -g13cbc@60 -g13ccc@72 -g13cdc@60 -g13cec@52 -g13cfc@48 -g13cgc@52
-g13eac@88 -g13ebc@88 -g13ecc@88 -g13edc@84 -g13ewc@40 -g13exc@40 -g13xzc@4

_h03abc@56

_m01cac@16 _m01csc@28 _m01ctc@28 _m01cuc@20 _m01dsc@28 _m01esc@24 _m01fsc@36
_m01zac@12

_s10aac@8 _s10abc@12 _s10acc@12 _s11aac@12 _s11abc@8 _s11acc@12 _s13aac@12
_s13acc@12 _s13adc@8 _s14aac@12 _s14abc@12 _s14bac@36 _s15abc@8 _s15acc@8
_s15adc@8 _s15aec@8 _s17acc@12 _s17adc@12 _s17aec@12 _s17afc@12 _s17agc@12
_s17ahc@12 _s17ajc@12 _s17akc@12 _s18acc@12 _s18adc@12 _s18aec@12 _s18afc@12
_s18ccc@12 _s18cdc@12 _s18cec@8 _s18cfc@8 _s19aac@12 _s19abc@12 _s19acc@12
_s19adc@12 _s20acc@8 _s20adc@8 _s21bac@20 _s21bbc@28 _s21bcc@28 _s21bdc@36

_x02ahc@0 _x02ajc@0 _x02akc@0 _x02alc@0 _x02amc@0 _x04bdc@4 _x04bec@12
_x04bfc@12 _x04bgc@12 _x05aac@0 _x05bac@0

Appendix B - Enumeration types

The following is a list of the enumeration types used in the NAG C DLLs (information concerning structure types can be found in the Nag_types.h header file).

Chapter c06

```
typedef enum {Nag_Convolution, Nag_Correlation} Nag_VectorOp;
typedef enum {Nag_ForwardTransform, Nag_BackwardTransform} Nag_TransformDirection;
```

Chapter d01

```
typedef enum {Nag_Alg, Nag_Alg_loga, Nag_Alg_logb, Nag_Alg_loga_logb} Nag_QuadWeight;
typedef enum {Nag_UpperSemiInfinite, Nag_LowerSemiInfinite, Nag_Infinite} Nag_BoundInterval;
typedef enum {Nag_Cosine, Nag_Sine} Nag_TrigTransform;
typedef enum {Nag_Legendre, Nag_Rational, Nag_Laguerre, Nag_Hermite} Nag_GaussFormulae;
typedef enum {Nag_OneIteration, Nag_ManyIterations} Nag_MCMethod;
```

Chapter d02

```
typedef enum {Nag_Relative, Nag_Absolute, Nag_Mixed} Nag_ErrorControl;
typedef enum {Nag_UserInitMesh, Nag_DefInitMesh} Nag_MeshSet;
typedef enum {Nag_RK_range, Nag_RK_onestep} Nag_RK_task;
typedef enum {Nag_RK_2_3=1, Nag_RK_4_5, Nag_RK_7_8} Nag_RK_method;
typedef enum {Nag_ErrorAssess_off, Nag_ErrorAssess_on} Nag_ErrorAssess;
typedef enum {Nag_Sol, Nag_Der, Nag_SolDer} Nag_SolDeriv;
```

Chapters e01, e02, f01, f02 and f04

```
typedef enum {Nag_RC, Nag_Shep} Nag_2d_Scat_Method;
typedef enum {Nag_LeftDerivs, Nag_RightDerivs} Nag_DerivType;
typedef enum {Nag_ElementsIn, Nag_ElementsSeparate } Nag_WhereElements;
typedef enum {Nag_Supplied, Nag_NotSupplied } Nag_InitRotation;
typedef enum {Nag_LDLTX, Nag_LDX, Nag_DLTX, Nag_LLTX, Nag_LX, Nag_LTX} Nag_SolveSystem;
```

Chapter f06

```
typedef enum {NoTranspose, Transpose, ConjugateTranspose } MatrixTranspose;
typedef enum {UpperTriangle, LowerTriangle } MatrixTriangle;
typedef enum {UnitTriangular, NotUnitTriangular } MatrixUnitTriangular;
typedef enum {LeftSide, RightSide } OperationSide;
typedef enum {BottomPivot, TopPivot, VariablePivot, FixedPivot } PivotType;
typedef enum {ForwardSequence, BackwardSequence } SequenceDirection;
typedef enum {OneNorm, FrobeniusNorm, MaxAbsValue } NormType;
typedef enum {General, UpperTriangular, LowerTriangular, SymmetricUpper,
              SymmetricLower, HermitianUpper, HermitianLower } MatrixType;
```

Chapter g01

```
typedef enum {Nag_LowerTail, Nag_UpperTail, Nag_TwoTailSignif,
              Nag_TwoTailConfid, Nag_TwoTail} Nag_TailProbability;
typedef enum {Nag_RankScores, Nag_NormalScores, Nag_BlomScores,
              Nag_TukeyScores, Nag_WaerdenScores, Nag_SavageScores} Nag_Scores;
typedef enum {Nag_AverageTies, Nag_LowestTies, Nag_HighestTies,
              Nag_RandomTies, Nag_IgnoreTies} Nag_Ties;
```

Chapter g02

```
typedef enum {Nag_WeightedEstimate, Nag_UnweightedEstimate} Nag_IncludeWeight;  
typedef enum {Nag_MeanInclude, Nag_MeanZero} Nag_IncludeMean;  
typedef enum {Nag_ObservAdd, Nag_ObservDel} Nag_UpdateObserv;  
typedef enum {Nag_AboutMean, Nag_AboutZero} Nag_SumSquare;  
typedef enum {Nag_FirstCall, Nag_Update} Nag_Initialize;  
typedef enum {Nag_Expo, Nag_Iden, Nag_Log, Nag_Sqrt, Nag_Reci,  
              Nag_Logistic, Nag_Probit, Nag_Compl } Nag_Link;  
typedef enum {Nag_RegNotSet = -1, Nag_HuberReg, Nag_MallowsReg, Nag_SchweppesReg} Nag_RegType;  
typedef enum {Nag_PsiNotSet = -1, Nag_Lsq, Nag_HuberFun, Nag_HampelFun, Nag_AndrewFun,  
              Nag_TukeyFun} Nag_PsiFun;  
typedef enum {Nag_SigmaNotSet = -1, Nag_SigmaRes, Nag_SigmaConst, Nag_SigmaChi} Nag_SigmaEst;  
typedef enum {Nag_CovNotSet = -1, Nag_CovMatAve, Nag_CovMatObs} Nag_CovMatrixEst;  
typedef enum {Nag_SigmaSimul, Nag_SigmaBypas} Nag_SigmaSimulEst;
```

Chapters g05, g07, g10 and g12

```
typedef enum {Nag_PDF, Nag_CDF} Nag_DiscreteDistrib;  
typedef enum {Nag_PopVarEqual, Nag_PopVarNotEqual} Nag_PopVar;  
typedef enum {Nag_4253H, Nag_3RSSH} Nag_Smooth_Type;  
typedef enum {Nag_Freq, Nag_NoFreq} Nag_FreqTime;
```

Chapter g13

```
typedef enum {Nag_CriteriaNotSet = -1, Nag_LeastSquares, Nag_Exact, Nag_Marginal} Nag_Likelihood;  
typedef enum {Nag_Rectangular, Nag_Bartlett, Nag_Tukey, Nag_Parzen} Nag_LagWindow;  
typedef enum {Nag_NoCorrection, Nag_Mean, Nag_Trend} NagMeanOrTrend;  
typedef enum {Nag_Unlogged, Nag_Logged} Nag_LoggedSpectra;  
typedef enum {Nag_next_state, Nag_curr_state} Nag_state;  
typedef enum {Nag_ab_prod, Nag_ab_sep} Nag_ab_input;  
typedef enum {Nag_UH_Observer, Nag_LH_Observer} Nag_ObserverForm;  
typedef enum {Nag_UH_Controller, Nag_LH_Controller} Nag_ControllerForm;
```

Chapter m01

```
typedef enum {Nag_Ascending, Nag_Descending} Nag_SortOrder;  
typedef enum {Nag_First, Nag_Last} Nag_SearchMatch;
```

Other Chapters: d01, d02, e02, e04 and g13

```
typedef enum {Nag_StartNotSet = -1, Nag_Cold, Nag_Warm, Nag_Hot,  
              Nag_NewStart, Nag_ReStart, Nag_Continue } Nag_Start;  
typedef enum {Nag_PrintNotSet = -1, Nag_NoPrint, Nag_Soln, Nag_Iter, Nag_Iter_Long,  
              Nag_Soln_Iter, Nag_Soln_Iter_Long, Nag_Soln_Iter_Const, Nag_Soln_Iter_Diag,  
              Nag_Soln_Iter_Full} Nag_PrintType;  
typedef enum {Nag_ChkNotSet = -1, Nag_NoCheck, Nag_SimpleCheck, Nag_CheckObj, Nag_CheckCon,  
              Nag_CheckObjCon, Nag_XSimpleCheck, Nag_XCheckObj, Nag_XCheckCon,  
              Nag_XCheckObjCon} Nag_GradChk;  
typedef enum {Nag_D_NotSet = -1, Nag_D_NoPrint, Nag_D_Full} Nag_Print_Deriv;  
typedef enum {Nag_ObjCheck, Nag_ConCheck, Nag_DiffInt} Nag_CheckType;  
typedef enum {Nag_DerivNotSet = -1, Nag_SomeG_SomeJ, Nag_AllG_SomeJ, Nag_SomeG_AllJ,  
              Nag_AllG_AllJ} Nag_DerivSet;  
typedef enum {Nag_Deriv, Nag_NoDeriv} Nag_FunType;  
typedef enum {Nag_LinFunNotSet = -1, Nag_Lin_Deriv, Nag_Lin_NoDeriv} Nag_LinFun;  
typedef enum {Nag_InitNotSet = -1, Nag_Init_None, Nag_Init_F_G_H, Nag_Init_All, Nag_Init_H_S} Nag_InitType;  
typedef enum {Nag_BoundNotSet = -1, Nag_Bounds, Nag_BoundsZero, Nag_BoundsEqual,  
              Nag_NoBounds, Nag_NoBounds_One_Call} Nag_BoundType;  
typedef enum {Nag_ProbTypeNotSet = -1, Nag_FP, Nag_LP, Nag_QP1, Nag_QP2, Nag_QP3, Nag_QP4} Nag_ProblemType;  
typedef enum {Nag_EndStateNotSet = -1, Nag_Feasible, Nag_Optimal, Nag_Deadpoint, Nag_Weakmin,  
              Nag_Unbounded, Nag_Infeasible, Nag_Too_Many_Iter, Nag_Hess_Too_Big} Nag_EndState;
```

Appendix C - Examples

This section contains Visual Basic code fragments that illustrate how to call various C DLL functions; it also gives the C DLL function prototypes and, where required, template C code to create the auxiliary DLL. Note that the fragments presented here are *incomplete* and are reproduced for illustrative purposes only. Complete code for these examples (and for a few more) can be obtained from the URL <http://www.nag.co.uk/public.html> – this code has been tested using both Excel 7.0 and Visual Basic 4.0.

Spline example

C DLL function prototypes

```
extern void __stdcall e02bac(Integer m, double x[], double y[], double weights[], double *ss,
    Nag_Spline *spline, NagError *fail);
extern void __stdcall e02bbc(double x, double *s, Nag_Spline *spline, NagError *fail);
```

Visual Basic declarations

```
Declare Sub e02bac Lib "nagcl04_noopt.dll" Alias "_e02bac@28" (ByVal m As Long, x As Double, _
    y As Double, weights As Double, ss As Double, spline As NagSpline, ifail As NagErrorType)
Declare Sub e02bbc Lib "nagcl04_noopt.dll" Alias "_e02bbc@20" (ByVal x As Double, s As Double, _
    spline As NagSpline, fail As NagErrorType)
```

Visual Basic code

```
Static Aspline As NagSpline
Static nl As Long
Static lamda(14) As Double
Static c(8) As Double
Static ptr As Long
Dim pfail As NagErrorType

Aspline.n = nl

' allocate and assign nl elements to Aspline.lamda
Call set_double_pointer(Aspline.lamda, lamda(0), nl)
Call e02bac(m, x(0), y(0), weights(0), ss, Aspline, pfail)

' copy nl elements from Aspline.lamda to the Visual Basic array lamda, and
' 8 elements from the (internally allocated) pointer Aspline.c
Call get_double_pointer(Aspline.lamda, lamda(0), nl)
Call get_double_pointer(Aspline.c, c(0), 8)
For i = 0 To 7
Print #1, Format(c(i), " ##0.0000;-##0.0000")
Next i
For i = 0 To m - 1 Step 1
    xarg = x(i)
    Call e02bbc(xarg, fit, Aspline, pfail)
    Print #1, i, Format(x(i), " ##0.0000;-##0.0000"), Format(fit, "##0.0000;-##0.0000")
    If (i < m - 1) Then
        j = j + 1
        xarg = (x(i) + x(i + 1)) * 0.5
        Call e02bbc(xarg, fit, Aspline, pfail)
        Print #1, Format(xarg, " ##0.0000;-##0.0000"), Format(fit, "##0.0000;-##0.0000")
    End If
    j = j + 1
Next i

' free allocated memory
Call free(Aspline.lamda)
Call free(Aspline.c)
```

SVD example

C DLL function prototypes

```
extern void __stdcall f02wec(Integer m, Integer n, double *a, Integer tda, Integer ncolb, double *b,  
    Integer tdb, Boolean wantq, double *q, Integer tdq, double *sv, Boolean wantp, double *pt,  
    Integer tdpt, Integer *iter, double *e, Integer *failinfo, NagError *fail);
```

Visual Basic declarations

Standard C style declaration

```
Declare Sub f02wec Lib "nagcl04_noopt.dll" Alias "_f02wec072" (ByVal m As Long, ByVal n As Long, _  
    a As Double, ByVal tda As Long, ByVal ncolb As Long, b As Double, ByVal tdb As Long, _  
    ByVal wantq As Long, q As Double, ByVal tdq As Long, sv As Double, ByVal wantp As Long, _  
    pt As Double, ByVal tdpt As Long, iter As Long, e As Double, failinfo As Long, _  
    ifail As NagErrorType)
```

Allow the use of null pointers

```
Declare Sub f02wec_ptr Lib "nagcl04_noopt.dll" Alias "_f02wec072" (ByVal m As Long, ByVal n As Long, _  
    a As Double, ByVal tda As Long, ByVal ncolb As Long, ByVal b As Long, ByVal tdb As Long, _  
    ByVal wantq As Long, ByVal q As Long, ByVal tdq As Long, sv As Double, ByVal wantp As Long, _  
    ByVal pt As Long, ByVal tdpt As Long, iter As Long, e As Double, failinfo As Long, _  
    ifail As NagErrorType)
```

Visual Basic code

```
Static a(m*n-1) As Double  
Static a2(m*n-1) As Double  
Static q(m*n-1) As Double  
Static sv(m-1) As Double  
Static pt(0) As Double  
Static e(m-1) As Double  
Static dum(0) As Double  
Dim pfail As NagErrorType
```

```
pfail.code = 0  
pfail.printm = 1  
ncolb = 0  
tda = n  
For i = 0 To m - 1  
    For j = 0 To n - 1  
        Input #2, a(i * tda + j)  
        a2(i * tda + j) = a(i * tda + j)  
    Next j  
Next i  
tdb = 0  
tdpt = 0  
tdq = n  
wtp = 1  
wtq = 1
```

' calculate the singular values and also the left and right singular vectors

```
Call f02wec(m, n, a2(0), tda, ncolb, dum(0), tdb, wtq, q(0), tdq, sv(0), wtp, _  
    pt(0), tdpt, iter, e(0), info, pfail)
```

```
tdq = 0  
wtq = 0  
wtp = 0
```

' only calculate the singular values, call f02wec_ptr with 3 null pointers

```
Call f02wec_ptr(m, n, a(0), tda, ncolb, ByVal 0%, tdb, wtq, ByVal 0%, tdq, sv(0), wtp, _  
    ByVal 0%, tdpt, iter, e(0), info, pfail)
```

ODE example

C DLL function prototypes

```
typedef void (__stdcall * NAG_D02EJC_FUN)(Integer neq, double x, double y[], double *f,
    Nag_User *comm);
typedef void (__stdcall * NAG_D02EJC_PFUN)(Integer neq, double x, double y[], double pw[],
    Nag_User *comm);
typedef void (__stdcall * NAG_D02EJC_OUTFUN)(Integer neq, double *xsol, double y[],
    Nag_User *comm);
typedef double (__stdcall * NAG_D02EJC_GFUN)(Integer neq, double x, double y[],
    Nag_User *comm);
extern void __stdcall d02ejc(Integer neq, NAG_D02EJC_FUN fcn, NAG_D02EJC_PFUN pederv, double *t,
    double y[], double tend, double tol, Nag_ErrorControl err_c, NAG_D02EJC_OUTFUN output,
    NAG_D02EJC_GFUN g, Nag_User *comm, NagError *fail);
```

Auxiliary DLL template code

```
#define DllExport __declspec( dllexport )
void local_d02ejc_fcn(Integer neq, double x, double y[], double f[], Nag_User *comm);
void local_d02ejc_pederv(Integer neq, double x, double y[], double pw[], Nag_User *comm);
DllExport long d02ejc_pederv(void) { return (long)(local_d02ejc_pederv);}
DllExport long d02ejc_fcn(void) { return (long)(local_d02ejc_fcn);}
void local_d02ejc_fcn(Integer neq, double x, double y[], double f[], Nag_User *comm) {
    < ... Insert C code ... >
}
void local_d02ejc_pederv(Integer neq, double x, double y[], double pw[], Nag_User *comm) {
    < ... Insert C code ... >
}
```

Visual Basic declarations

```
Declare Sub d02ejc Lib "nagcl04_noopt.dll" Alias "_d02ejc056" (ByVal neq As Long, ByVal ptr_fcn As Long, _
    ByVal ptr_pederv As Long, x As Double, y As Double, ByVal xend As Double, ByVal tol As Double, _
    ByVal err_c As Long, ByVal ptr_output As Long, ByVal ptr_g As Long, comm As Nag_User, _
    ifail As NagErrorType)
Declare Function d02ejc_fcn Lib "examples_interfaces.dll" Alias "_d02ejc_fcn00" () As Long
Declare Function d02ejc_pederv Lib "examples_interfaces.dll" Alias "_d02ejc_pederv00" () As Long
```

Visual Basic code

```
Static err_c As Long
Static comm As Nag_User
Static y(2) As Double
Static fcn_ptr As Long
Static pederv_ptr As Long
Dim pfail As NagErrorType
```

```
' user-defined Jacobian; use pointers d02ejc_fcn and d02ejc_pederv
```

```
xend = x + 2#
fcn_ptr = d02ejc_fcn
pederv_ptr = d02ejc_pederv
Call d02ejc(neq, fcn_ptr, pederv_ptr, x, y(0), xend, tol, err_c, 0, 0, comm, pfail)
Print #1, x, y(0), y(1), y(2)
```

```
' internal evaluation of Jacobian; use pointer d02ejc_fcn
```

```
xend = x + 2#
fcn_ptr = d02ejc_fcn
Call d02ejc(neq, fcn_ptr, ByVal 0&, x, y(0), xend, tol, err_c, ByVal 0&, ByVal 0&, comm, pfail)
Print #1, x, y(0), y(1), y(2)
```

Optimize example

C DLL function prototypes

```
typedef void (__stdcall * NAG_E04UCC_FUN)(Integer, double *, double *, double *, Nag_Comm *);
typedef void (__stdcall * NAG_E04UCC_CONFUN)(Integer, Integer, Integer *, double *, double *,
double *, Nag_Comm *);
extern void __stdcall e04ucc(Integer n, Integer nclin, Integer ncnlin, double a[], Integer tda,
double bl[], double bu[], NAG_E04UCC_FUN objfun, NAG_E04UCC_CONFUN confun, double x[],
double *objf, double objgrad[], Nag_E04_Opt *options, Nag_Comm *user_comm, NagError *fail);
extern void __stdcall e04xxc(Nag_E04_Opt *opt);
extern void __stdcall e04xyc(const char *name, const char *opt_file, Nag_E04_Opt *opt,
Boolean print, const char *outfile, NagError *fail);
```

Auxiliary DLL template code

```
#define DllExport __declspec( dllexport )
Nag_E04_Opt options;
void local_e04ucc_objfun(Integer n, double x[], double *objf, double objgrd[], Nag_Comm *comm);
void local_e04ucc_confun(Integer n, Integer ncnlin, Integer needc[], double x[], double conf[],
double conjac[], Nag_Comm *comm);
DllExport long e04ucc_objfun(void) { return (long)(local_e04ucc_objfun);}
DllExport long e04ucc_confun(void) { return (long)(local_e04ucc_confun);}
DllExport long e04_options_ptr(void) {
    < ... Insert C code ... >
return (long>(&options);
}
void local_e04_objfun(long n, double *x, double *objf, double *g, Nag_Comm *comm) {
    < ... Insert C code ... >
}
void local_e04ucc_objfun(Integer n, double x[], double *objf, double objgrd[], Nag_Comm *comm) {
    < ... Insert C code ... >
}
void local_e04ucc_confun(Integer n, Integer ncnlin, Integer needc[], double x[], double conf[],
double conjac[], Nag_Comm *comm) {
    < ... Insert C code ... >
}
}
```

Visual Basic declarations

```
Declare Sub e04ucc Lib "nagcl04_noopt.dll" Alias "_e04ucc@60" (ByVal n As Long, _
    ByVal nclin As Integer, ByVal ncnlin As Integer, a As Double, ByVal tda As Integer, _
    bl As Double, bu As Double, ByVal objfun_ptr As Long, ByVal confun_ptr As Long, _
    x As Double, objf As Double, g As Double, ByVal options_ptr As Long, comm As Any, _
    fail As NagErrorType)
Declare Sub e04xxc Lib "nagcl04_noopt.dll" Alias "_e04xxc@4" (ByVal options_ptr As Long)
Declare Sub e04xyc Lib "nagcl04_noopt.dll" Alias "_e04xyc@24" (ByVal routine_name As String, _
    ByVal opt_file As String, ByVal options_ptr As Long, ByVal iprint As Long, _
    ByVal outfile As String, ifail As NagErrorType)
Declare Function e04_options_ptr Lib "examples_interfaces.dll" Alias "_e04_options_ptr@0" () As Long
Declare Function e04_objfun Lib "examples_interfaces.dll" Alias "_e04_objfun@0" () As Long
Declare Function e04ucc_confun Lib "examples_interfaces.dll" Alias "_e04ucc_confun@0" () As Long
```

Visual Basic code

```
Static a(ncclin * n)
Static bu(totalvars)
Static x(n)
Static bl(totalvars)
Static objgrd(n)
Static printit As Long
Static objf As Double
Static local_e04_options_ptr As Long
Static objfun_ptr As Long
Static confun_ptr As Long
Dim pfail As NagErrorType
```

```

tda = n
pfail.code = 0

' initialise options using options structure pointer, e04_option_ptr

local_e04_options_ptr = e04_options_ptr
Call e04xxc(local_e04_options_ptr)
printit = 1

' read the options file for e04ucc

local_e04_options_ptr = e04_options_ptr
Call e04xyc("e04ucc", "e04ucce.d", local_e04_options_ptr, printit, "e04ucce.r", pfail)

' call the optimizer using function pointers e04ucc_objfun, e04ucc_confun and options structure
' pointer, e04_options_ptr

objfun_ptr = e04ucc_objfun
confun_ptr = e04ucc_confun
local_e04_options_ptr = e04_options_ptr
Call e04ucc(n, nclin, ncnlin, a(0), tda, bl(0), bu(0),-
    objfun_ptr, confun_ptr, x(0), objf, objgrd(0), local_e04_options_ptr, ByVal 0%, pfail)

```

Integrate example

C DLL function prototypes

```
typedef double (__stdcall * NAG_D01AJC_FUN)(double);
extern void __stdcall d01ajc(NAG_D01AJC_FUN f, double a, double b, double epsabs, double epsrel,
    Integer max_num_subint, double *result, double *abserr, Nag_QuadProgress *qp, NagError *fail);
```

Auxiliary DLL template code

```
#define DllExport __declspec( dllexport )
double local_d01ajc_fun1(double x);
double local_d01ajc_fun2(double x);
DllExport long d01ajc_fun1(void) { return (long)(local_d01ajc_fun1);}
DllExport long d01ajc_fun2(void) { return (long)(local_d01ajc_fun2);}
double local_d01ajc_fun1(double x) {
    < ... Insert C code ... >
    return ( < ... Insert C code ... > );
}
double local_d01ajc_fun2(double x) {
    < ... Insert C code ... >
    return ( < ... Insert C code ... > );
}
```

Visual Basic declarations

```
Declare Sub d01ajc Lib "nagcl04_noopt.dll" Alias "_d01ajc@56" (ByVal ptr As Long, _
    ByVal a As Double, ByVal b As Double, ByVal epsabs As Double, ByVal epsres As Double, _
    ByVal max_num_sunint As Long, result As Double, abserr As Double, _
    qp As NagQuadProgress, fail As NagErrorType)
Declare Function d01ajc_fun1 Lib "examples_interfaces.dll" Alias "_d01ajc_fun1@0" () As Long
Declare Function d01ajc_fun2 Lib "examples_interfaces.dll" Alias "_d01ajc_fun2@0" () As Long
```

Visual Basic code

```
Static qp As NagQuadProgress
Static a As Double
Static b As Double
Static epsabs As Double
Static epsrel As Double
Static result As Double
Static max_num_subint As Long
Static abserr As Double
Static pi As Double
Static fun_ptr As Long
Dim pfail As NagErrorType

pi = 4# * Atn(1#)
a = 0#
b = 2 * pi
max_num_subint = 200
epsabs = 0#
epsrel = 0.0001
pfail.printm = 1
pfail.code = 0

' integrate function 1; use pointer d01ajc_fun1

fun_ptr = d02ajc_fun1
Call d01ajc(fun_ptr, a, b, epsabs, epsrel, max_num_subint, result, abserr, qp, pfail)

' integrate function 2; use pointer d01ajc.fun2

fun_ptr = d02ajc_fun2
Call d01ajc(fun_ptr, a, b, epsabs, epsrel, max_num_subint, result, abserr, qp, pfail)
```