Creating Fortran DLLs

A dynamic-link library (DLL) contains one or more subprogram procedures (functions or subroutines) that are compiled, linked, and stored separately from the applications using them. Because the functions or subroutines are separate from the applications using them, they can be shared or replaced easily.

Like a static library, a DLL is an executable file. Unlike a static library where routines are included in the base executable image during linking, the routines in a DLL are loaded when an application that references that DLL is loaded (run time). A DLL can also be used as a place to share data across processes.

The advantages of DLLs include:

- You can change the functions in a DLL without recompiling or relinking the applications that use them, as long as the functions' arguments and return types do not change.
  
  This allows you to upgrade your applications easily. For example, a display driver DLL can be modified to support a display that was not available when your application was created.

- When general functions are placed in DLLs, the applications that share the DLLs can have very small executables.
- Multiple applications can access the same DLL.

  This reduces the overall amount of memory needed in the system, which results in fewer memory swaps to disk and improves performance.

- Common blocks or module data placed in a DLL can be shared across multiple processes.

To build a DLL in the visual development environment, specify the Fortran Dynamic-Link Library project type. On the command line, specify the /dll option.

You cannot make a QuickWin application into a DLL (see Using QuickWin) and QuickWin applications cannot be used with Fortran run-time routines in a DLL.

This section describes the following aspects of creating Fortran DLLs:

- Coding Requirements for Sharing Procedures in DLLs
- Coding Requirements for Sharing Data in DLLs
- Building and Using Dynamic-Link Libraries

Coding Requirements for Sharing Procedures in
DLLs

A dynamic-link library (DLL) contains one or more subprograms that are compiled, linked and stored separately from the applications using them.

Coding requirements include using the `cDEC$ ATTRIBUTES DLLIMPORT and DLLEXPORT compiler directives. Variables and routines declared in the main program and in the DLL are not visible to each other unless you use DLLIMPORT and DLLEXPORT.

This section discusses aspects of sharing subprogram procedures (functions and subroutines) in a Fortran DLL.

To export and import each DLL subprogram:

1. Within your Fortran DLL, export each subprogram that will be used outside the DLL. Add a `cDEC$ ATTRIBUTES DLLEXPORT directive to declare that a function, subroutine, or data is being exported outside the DLL. For example:

   ```fortran
   SUBROUTINE ARRAYTEST(arr)
   !DEC$ ATTRIBUTES DLLEXPORT :: ARRAYTEST
   REAL(4) arr(3, 7)
   INTEGER i, j
   DO i = 1, 3
     DO j = 1, 7
       arr (i, j) = 11.0 * i + j
     END DO
   END DO
   END SUBROUTINE
   ```

2. Within your Fortran application, import each DLL subprogram. Add a `cDEC$ ATTRIBUTES DLLIMPORT directive to declare that a function, subroutine, or data is being imported from outside the current image. For example:

   ```fortran
   INTERFACE
   SUBROUTINE ARRAYTEST (rarray)
   !DEC$ ATTRIBUTES DLLIMPORT :: ARRAYTEST
   REAL rarray(3, 7)
   END SUBROUTINE ARRAYTEST
   END INTERFACE
   ```

The DLLEXPORT and DLLIMPORT options (for the `cDEC$ ATTRIBUTES directive) define a DLL's interface.

The DLLEXPORT property declares that functions or data are being exported to other images or DLLs, usually eliminating the need for a Linker module definition (.DEF) file to export symbols for the functions or subroutines declared with DLLEXPORT. When you declare a function, subroutine, or data with the DLLEXPORT property, it must be defined in the
same module of the same program.

A program that uses symbols defined in another image (such as a DLL) must import them. The DLL user needs to link with the import LIB file from the other image and use the DLLIMPORT property inside the application that imports the symbol. The DLLIMPORT option is used in a declaration, not a definition, because you do not define the symbol you are importing.

3. Build the DLL and then build the main program, as described in Building and Using Dynamic-Link Libraries.

Fortran and C applications can call Fortran and C DLLs provided the calling conventions are consistent (see Programming with Mixed Languages, especially Visual Fortran/Visual C++ Mixed-Language Programs).

Visual Basic applications can also call Fortran functions and subroutines in the form of DLLs (see Programming with Mixed Languages, especially Calling Visual Fortran from Visual Basic).

For more information:

- About building DLLs, see Building and Using Dynamic-Link Libraries.
- On sharing either common block or module data in a DLL, see Coding Requirements for Sharing Data in DLLs.
- About importing and exporting subprograms using a Sample program, see the Visual Fortran Sample TLS in folder ...\Df98\Samples\Advanced\Win32.

Coding Requirements for Sharing Data in DLLs

A dynamic-link library (DLL) is an executable file that can be used as a place to share data across processes.

Coding requirements include using the cDEC$ ATTRIBUTES DLLIMPORT and DLLEXPORT compiler directives. Variables and routines declared in the program and in the DLL are not visible to each another unless you use DLLIMPORT and DLLEXPORT.

When sharing data among multiple threads or processes, do the following:

- Declare the order, size, and data types of shared data consistently in the DLL and in all procedures importing the DLL exported data.
- If more than one thread or process can write to the common block simultaneously, use the appropriate features of the Windows operating system to control access to the shared data. Such features on Windows NT 4 and Windows 2000 systems include critical sections (for single process, multiple thread synchronization) and mutex objects (for multi-process
This section discusses:

- Exporting and Importing Common Block Data
- Exporting and Importing Data Objects in Modules

**Exporting and Importing Common Block Data**

Data and code in a dynamic-link library is loaded into the same address space as the data and code of the program that calls it. However, variables and routines declared in the program and in the DLL are not visible to one another unless you use the `DEC$ ATTRIBUTES DLLIMPORT and DLLEXPORT` compiler directives. These directives enable the compiler and linker to map to the correct portions of the address space so that the data and routines can be shared, allowing use of common block data across multiple images.

You can use DLLEXPORT to declare that a common block in a DLL is being exported to a program or another DLL. Similarly, you can use DLLIMPORT within a calling routine to tell the compiler that a common block is being imported from the DLL that defines it.

**To export and import common block data:**

1. Create a common block in the subprogram that will be built into a Fortran DLL. Export that common block with a `DEC$ ATTRIBUTES DLLEXPORT` directive, followed by the `COMMON` statement, associated data declarations, and any procedure declarations to be exported. For example:

   ```fortran
   !DEC$ ATTRIBUTES DLLEXPORT :: /X/
   COMMON /X/ C, B, A
   REAL C, B, A
   END...
   ```

   If the Fortran DLL procedure contains only a common block declaration, you can use the `BLOCK DATA` statement:

   ```fortran
   BLOCK DATA T
   !DEC$ ATTRIBUTES DLLEXPORT :: /X/
   COMMON /X/ C, B, A
   REAL C, B, A
   END
   ```

The Fortran procedure to be linked into a DLL can contain a procedure, such as the following:

```fortran
SUBROUTINE SETA(I)
!DEC$ ATTRIBUTES DLLEXPORT :: SETA, /X/
   COMMON /X/ C, B, A
   REAL C, B, A
```
INTEGER I
A = A + 1.
I = I + 1
WRITE (6,*) 'In SETA subroutine, values of A and I:', A, I
RETURN
END SUBROUTINE

2. Refer to the common block in the main image with a DEC$ ATTRIBUTES DLLIMPORT directive, followed by the local data declarations and any procedure declarations defined in the exported DLL. For example:

PROGRAM COMMONX
!DEC$ ATTRIBUTES DLLIMPORT:: SETA, /X/
    COMMON /X/ C, B, A
    REAL C, B, A, Q
    EQUIVALENCE (A,Q)
    A = 0.
    I = 0
    WRITE (6,*) 'In Main program before calling SETA...'
    WRITE (6,*) 'values of A and I:', A, I
    CALL SETA(I)
    WRITE (6,*) 'In Main program after calling SETA...'
    WRITE (6,*) 'values of A and I:', Q, I
    A = A + 1.
    I = I + 1
    WRITE (6,*) 'In Main program after incrementing values'
END PROGRAM COMMONX

3. Build the DLL and then build the main program, as described in Building and Using Dynamic-Link Libraries.

Exporting and Importing Data Objects in Modules

You can give data objects in a module the DLLEXPORT property, in which case the object is exported from a DLL.

When a module is used in other program units, through the USE statement, any objects in the module with the DLLEXPORT property are treated in the program using the module as if they were declared with the DLLIMPORT property. So, a main program that uses a module contained in a DLL has the correct import attributes for all objects exported from the DLL.

You can also give some objects in a module the DLLIMPORT property. Only procedure declarations in INTERFACE blocks and objects declared EXTERNAL or with DEC$ ATTRIBUTES EXTERN can have the DLLIMPORT property. In this case, the objects are imported by any program unit using the module.

If you use a module that is part of a DLL and you use an object from that module that does not have the DLLEXPORT or DLLIMPORT property, the results are undefined.
For more information:

- On building a DLL, see Building and Using Dynamic-Link Libraries.
- On multithread programming, see Creating Multithread Applications.

Building and Using Dynamic-Link Libraries

A dynamic-link library is a collection of source and object code in the same manner as a static library. The differences between the two libraries are:

- The DLL requires an interface specification.
- The DLL is associated with a main project during execution, not during linking.

For more information, see:

- Building Dynamic-Link Libraries
- The DLL Build Output
- Checking the DLL Symbol Export Table
- Building Executables that Use DLLs
- DLL Sample Programs

Building Dynamic-Link Libraries

When you first create a DLL, you follow the general steps described in Defining Your Project. Select Fortran Dynamic-Link Library as the project type when you create a new project in the Microsoft visual development environment.

To debug a DLL, you must use a main program that calls the library routines (or references the data). From the Project Settings menu, choose the Debug tab. A dialog box is available for you to specify the executable for a debug session.

To build the DLL from the Microsoft visual development environment:

1. A Fortran DLL project is created like any other project, but you must specify Fortran Dynamic-Link Library as the project type (see Defining Your Project).
2. Add files to your Fortran DLL project (see Defining Your Project). Include the DLL Fortran source that exports procedures or data as a file in your project.
3. If your DLL exports data, for both the DLL and any image that references the DLL's exported data, consistently specify the project settings options in the Fortran Data category. In the Fortran Data compiler option category, specify the appropriate values for Common Element Alignment (common block data) and Structure Element Alignment (structures in a module). This
sets the /alignment option, which specifies whether padding is needed to
ensure that exported data items are naturally aligned.

For example, in the case of a common block containing four-byte variables,
in the Project Setting dialog box you might specify:

- Open the appropriate workspace
- From the Project menu, click Settings
- Click the Fortran tab
- Select the Fortran Data category
- In the Common Element Alignment box, specify 4.

4. If you need to specify linker options, use the Linker tab of the Project
   Settings dialog box.
5. Build your Fortran DLL project.

The Microsoft visual development environment automatically selects the
correct linker instructions for loading the proper run-time library routines
(located in a DLL themselves). Your DLL is created as a multithreaded-
enabled library. An import library (.LIB) is created for use when you link
images that reference the DLL.

To build the DLL from the command line:

1. If you build a DLL from the command line or use an exported makefile, you
   must specify the /dll option. For example, if the Fortran DLL source code is
   in the file f90arr.f90, use the following command line:

   DF /dll f90arr.f90

   This command creates:

   - A DLL named f90arr.dll.
   - An import library, f90arr.lib, that you must link with applications that
call your DLL.

   If you also specify /exe:file or /link /out:file, you name a .DLL rather than
an .EXE file (the default file extension becomes projectname.DLL instead of
projectname.EXE)

   The /dll option selects as the default the DLL run-time libraries to support
multithreaded operation.

2. If your DLL will export data, the procedures must be compiled and linked
consistently. Consistently use the same /alignment option for the DLL
export procedure and the application that references (imports) it. The goal
is to specify padding to ensure that exported data items are naturally
aligned, including common block data items and structure element alignment (structures in a module).

3. If you need to specify linker options, place them after the `/link` option on the DF command line.

4. Build the application.

For example, if your DLL exports a common block containing four-byte variables, you might use the following command line (specify the `/dll` option):

```
DF /align:commons /dll dllfile.for
```

The `/dll` option automatically selects the correct linker instructions for loading the proper run-time library routines (located in a DLL themselves). Your DLL is created as a multithread-enabled library.

For more information, see:

- The DLL Build Output
- Checking the DLL Symbol Export Table
- Building Executables that Use DLLs
- DLL Sample Programs

**The DLL Build Output**

When a DLL is built, two library files are created:

- An import library (.LIB), which the linker uses to associate a main program with the DLL.
- The .DLL file containing the library's executable code.

Both files have the same basename as the library project by default.

Your library routines are contained in the file `projectname.DLL` located in the default directory for your project, unless you specified another name and location. Your import library file is `projectname.LIB`, located in the default directory for your project.

**Checking the DLL Symbol Export Table**

To make sure that everything that you want to be visible shows up in the export table, look at the export information of an existing DLL file by using QuickView in the Windows Explorer File menu or the following DUMPBIN command:

```
DUMPBIN /exports file.dll
```

**Building Executables that Use DLLs**
When you build the executable that imports the procedures or data defined in the DLL, you must link using the import library, check certain project settings or command-line options, copy the import library so the Linker can locate it, and then build the executable.

**To use the DLL from another image:**

1. Add the import .LIB file with its path and library name to the other image.

   In the visual development environment, add the .LIB import library file to your project. In the Project menu, click Add to project, then Files.

   On the command line, specify the .LIB file on the command line.

   The import .LIB file contains information that your program needs to work with the DLL.

2. If your DLL exports data, consistently use the same project settings options in the Fortran Data category /alignment option as was used to create the DLL. In the Fortran Data compiler option category, specify the appropriate values for Common Element Alignment (common block data) and Structure Element Alignment (structures in a module). This sets the /alignment option, which specifies whether padding is needed to ensure that imported data items are naturally aligned.

3. In the Project Settings dialog box (Fortran tab), make sure the type of libraries specified is consistent with that specified for the Fortran DLL.

4. If you need to specify linker options:
   - In the visual development environment, specify linker options in the Linker tab of the Project Settings dialog box.
   - On the DF command line, place linker options after the /link option.

5. Copy the DLL into your path.

   For an application to access your DLL, it must be located in a directory on the search path or in the same directory as the main project. If you have more than one program accessing your DLL, you can keep it in a convenient directory identified in the environment path. If you have several DLLs, you can place them all in the same directory to avoid adding numerous directories to the path specification.

   When changing your path specification on a Windows Me, Windows 98, or Windows 95 system, you must restart the operating system for the change to take effect. On a Windows NT 4 or Windows 2000 system, you should log out and back in after modifying the system path.

6. Build the image that references the DLL.
When using the visual development environment:

- Make sure you have added the import library (created when you built the DLL file) to the project by (click the FileView tab).
- Like building other projects in the visual development environment, use the Build menu items to create the executable (see Defining Your Project).

When using the command line:

- Specify the import library at the end of the command line.
- If your DLL exports data that will be used by the application being built, specify the same /alignment options that were used to build the DLL.
- If you are building a main application, omit the /dll option.
- When building a Fortran DLL that references another DLL, specify the /dll option.

For example, to build the main application from the command line that references 4-byte items in a common block defined in dllfile.dll:

```bash
DF /align:commons mainapp.f90 dllfile.lib
```

**DLL Sample Programs**

Visual Fortran provides Sample programs are installed in ...\ DF98\SAMPLES when you request the Samples with a custom installation. You can copy the Samples folders from the ...\ DF\SAMPLES folder on the Visual Fortran CD-ROM to your hard disk.

For an example of a DLL, see the Samples folder ...\ DF98\SAMPLES\ADVANCED\WIN32\TLS, which creates a DLL as a subproject. The subproject DLL is used in a second project.

Other Samples that use DLLs are folders in ...\ DF98\SAMPLES\DLL. For example:

- The files associated with Sample DLLEXP2 are in the folder ...\ DF98\SAMPLES\DLL\DLLEXP2. DLLEXP2 demonstrates how COMMON variables defined in a DLL can be shared between multiple programs. To build DLLEXP2, use the makefile.
- The Sample Loadexp1 shows how to dynamically load a DLL by using the LoadLibrary and GetProcAddress Win32 routines. The folder ...\ DF98\SAMPLES\DLL\Loadexp1 contains the source files and a project workspace file.

For a description of the Samples, see Visual Fortran Samples.