UNIX Code Migration Guide

UNIX Application Migration Guide

Chapter 14: Migrating Fortran

Larry Twork, Larry Mead, Bill Howison, JD Hicks, Lew Brodnax, Jim McMicking, Raju Sakthivel, David Holder, Jon Collins, Bill Loeffler
Microsoft Corporation

October 2002

Applies to:
Microsoft® Windows®
UNIX applications
FORTRAN

Summary: Migrating Fortran code from a UNIX environment to Windows brings a different set of challenges. Chapter 14: Migrating Fortran looks at the Fortran migration options and factors that you will need to consider when making such a move. (9 printed pages)

Contents

Introduction

This chapter examines the process of migrating Fortran code to the Microsoft® Windows® operating system. This migration may be to either the Microsoft Win32® subsystem or the Interix subsystem.

Fortran is relatively easy to port between platforms, largely due to the high degree of standardization between Fortran compilers and to the types of applications for which Fortran is typically used.

The main difficulties with migrating Fortran code are often associated with either an application's integration with other languages or its use of third-party libraries.

This chapter covers how you should:

- Gather and analyze data regarding your Fortran application
- Plan the migration of your Fortran application

Data Gathering and Analysis

When you migrate Fortran code, you need to gather a range of data so that you can determine the best migration approach. The data that you need to collect includes the following:

- The Fortran level (77, 90, or 95) used
- The graphical user interface (GUI) requirements
- Any required third-party libraries
- Whether the application provides any cross-language support
- The best development and build environment to use for the migration

You need to know how your Fortran code is being used, for example:

- Is it a stand-alone application?
- Does it expose interfaces for other languages?
- Does it call interfaces in other languages?

Fortran is commonly used in computationally intensive applications. Often this means that Fortran is used in loosely coupled, high performance grid computing environments. Typically, distributed grid computing requires integration with dynamic scheduling and high performance message passing. In most cases, Fortran modules are not part of the core infrastructure services, so they need to integrate with libraries that perform these functions.

It is also important to understand the target development environment for the Fortran migration. Possible target environments include:

- Migration of the development environment from UNIX to Windows and Win32.
- Continued development on UNIX with Windows as a cross-platform port.
- Migration of the development environment from UNIX to a UNIX style development environment on Windows, such as Interix.

The considerations above need addressing, along with the actual source code migration.

Using Third-Party Libraries

Fortran modules can call or be called by other languages, in addition to running as stand-alone modules. When Fortran is integrated with other languages, it is important to understand the naming and calling conventions used as these can have a big impact on the migration.

The necessity to address differences in calling conventions between Fortran and C/C++ is not unique to a migration from UNIX to Windows. The Windows platform offers a wide range of third party libraries to perform functions that could eliminate the need of some of the UNIX based custom code. The use of these types of third-party C/C++ libraries in Windows eases the overall migration task, and is an advantage in migrating to Windows.

This section focuses on integrating Fortran with C and C++ libraries in the Microsoft Win32 subsystem. It begins by examining the default calling and naming conventions for Fortran and how these conventions work with typical Win32 C and C++ libraries, and with Win32 APIs. Table 1 lists the defaults calling and naming conventions for Fortran, C/C++, and Win32 APIs.

Table 1. Fortran, C/C++, and Win32 calling and naming conventions

Source Type	Arguments	Procedure Case	Stack Cleanup	Argument Suffix
Fortran	By reference	Procedure name in all uppercase	The procedure being called is responsible for removing arguments from the stack before returning to the caller.	Yes
C/C++	By value	Procedure name in all lowercase	The procedure doing the call is responsible for removing arguments from the stack after the call is over.	No
Win32 API (STDCALL)	By value	Procedure name in all lowercase	The procedure being called is responsible for removing arguments from the stack before returning to the caller.	Yes

During a migration, a developer usually encounters both the C/C++ calling convention and the **STDCALL** convention. Some of the common development libraries can take care of some of the differences in calling conventions. Other libraries require explicit declarations.

Differences in calling conventions can be handled in a number of ways, including:

- The Fortran Interface statement
- C function declarations
- Compatibility layers
- Modular code that uses Fortran libraries

These are explained in the next four sections.

The Fortran Interface statement

You can use the Fortran **Interface** statement to transform Fortran calling conventions into C style conventions. The example shown below takes a call to the C function **CLibFunction**, which calls a C function that passes an integer value. An alias attribute is used to take care of the case difference and extra underscore.

```
INTERFACE
 SUBROUTINE CLIBFUNCTION(I)
     !MS$ATTRIBUTES C, ALIAS:'_CLibFunction' :: CLIBFUNCTION
     INTEGER I
 END SUBROUTINE CLIBFUNCTION
END INTERFACE
```

Attributes can be defined with the **Interface** statement to adjust the Fortran calling conventions so that they match existing C libraries.

C function declarations

You can also use a combination of C function declarations, function naming, and argument definitions to resolve calling convention differences. The example below shows how a **STDCALL** function declaration handles stack calling conventions and suffixes in a manner similar to default Fortran conventions.

```
extern "C" void __stdcall FLIBUNCTION (int n);
```

This type of translation is especially useful when dealing with C++ name mangling (where the compiler adds characters to function names). Implementing the STDCALL convention tells the compiler that this function is not subject to C++ name mangling.

Compatibility layers

Another means of resolving calling convention differences is to use a compatibility layer to translate between Fortran and C/C++. Using this approach, C/C++ libraries could expose **STDCALL** type interfaces, while actually calling the C/C++ routines using C calling conventions.

The strategy you use for third-party library integration also depends on whether the Fortran or C/C++ code can be modified. For example, introducing Fortran **Interface** statements is only a viable option if the developer can modify the Fortran source. The same is true regarding changing the function declarations in C/C++ source.

This makes the concept of a compatibility layer the most flexible solution. However, this solution requires that you develop and maintain additional source code.

Fortran modules

As with most languages, modularity allows for easier cross-platform development. Because Fortran is most often used for high perform computations, platform-specific routines may already exist in external, non-Fortran routines.

Even though Fortran is often isolated to computationally specific routines, platform-specific APIs—such as threading, synchronization and GUI functions—can also be used in Fortran. The Windows APIs for threading, synchronization, and GUI functions are typically made available using Fortran modules.

For example, Fortran modules could exist for Windows GUI functions, threading and OpenGL graphics. The Fortran modules encapsulate the C and **STDCALL** style functions to the Windows kernel and Win32 libraries. To enhance portability, platform-specific code should either be encapsulated in Fortran modules, or through a call layer to C and C++ functions.

The Fortran module feature requires a Fortran 90 or greater compiler.

Fortran GUI applications often use OpenGL for high performance graphics. OpenGL provides a cross-platform API for GUI development with minimal platform-specific code requirements. On Windows, OpenGL is a C library, and can be used either from a Fortran module, or through a custom OpenGL GUI extraction layer.

Your migration choice will largely depend on whether the existing UNIX Fortran application extracts the GUI calls or uses a Fortran module. Either strategy can be migrated to Windows.

Integrating Fortran with POSIX Applications

The Fortran application that you are migrating may be required to integrate with other POSIX style applications. In this situation, the target Windows environment can be either the Windows POSIX subsystem (Interix), or a UNIX emulator running on the Win32 subsystem. Microsoft Interix is the full-featured POSIX subsystem on Windows. MKS NuTCRACKER and Cygwin are examples of UNIX emulators.

The Fortran considerations for using either the Interix POSIX subsystem or a UNIX emulator are the same as for C/C++ migrations.

> **Note** As of the time of publication, the GNU Fortran 77 compiler, **f77**, is the only Fortran compiler available for Interix.

Development Tools and Resources

Microsoft supplies the GNU Fortran 77 compiler with the Interix subsystem. Microsoft does not supply or sell a Fortran compiler for Win32. For migrations that require Fortran 90 or 95 features, a third-party Fortran compiler is required to target Win32, such as the Intel Visual Fortran compiler or the Lahey/Fujitsu Fortran compiler.

When you migrate Fortran applications, you must ensure that you implement the necessary development tools, including a source code control system and build analysis and management tools You should also consider your cross-platform build and debug environments. The Fortran version that you use for development must be compatible with the other development tools you use during the migration.

For example, if your migration targets the Microsoft® Visual Studio® .NET development system, your Fortran compiler should integrate with Visual Studio .NET.

Design and Validation

When you migrate a Fortran application from UNIX to Windows, the design and capabilities of the Fortran code must be an integral part of your migrated application's design. Considerations such as performance, library interoperability and feature set, can determine the overall success of the project.

Sizing the Fortran Migration

The effort required for a Fortran migration will depend largely on the answers to the following questions:

- Is the code modular?
- Will platform specific code need migration within Fortran?
- What third-party libraries will Fortran code need?
- Is GUI or graphics support required?
- Do feature/function abstraction layers already exist in UNIX?

Because the features and functions needed for a Windows migration are likely to already exist on UNIX, the answers to these questions are likely to exist too. If the code is already modular with feature or function abstraction layers, the code itself will likely move across easily as a port to Win32. In this case, the bulk of your effort will be in choosing any required third-party libraries, the cross-language calling conventions, and the integrated development environment tools.

Assessing and Mitigating Risk

Fortran adds complexity and consequently risk to a migration because of following:

- You may require a third-party Fortran compiler.
- You may need call-level integration between the Fortran and C/C++ code.
- You will need a cross-language build and debug strategy for Windows.

You can mitigate Fortran migration risks by:

- Defining the Windows development environment, including the Fortran compiler and your integration strategy for C/C++ code and third-party libraries.
- Implementing modularity of the Fortran code and putting platform-specific features into a C/C++ compatibility layer. This will enhance the ability of the code to migrate from UNIX to Windows and is essential if the application needs to target both the UNIX and Windows platforms.

Migration Planning

This section describes how you should go about planning a Fortran migration. In particular, this section provides information on how to scope a Fortran migration to Win32 and provides a brief look at other migration strategies.

Scoping the Fortran Migration

At first glance, an ANSI Fortran application migration can have most (if not all) of the same migration combinations as a C or a C++ migration. A Fortran application can:

- Be multi-user

- Have a GUI for user interaction
- Use platform specific features

However, most Fortran applications perform specialized functions where the Fortran language is particularly suitable. For example, Fortran is particularly suited as a language for computationally intensive mathematical operations. Other languages, such as C and C++ provide more widely used features and libraries for such things as process and thread support, and GUI features. For this reason, Fortran source code often performs only the computationally intensive functions in an application, leaving the process management and user interaction to C and C++.

Using Fortran in this manner removes most of the platform-specific issues from an ANSI Fortran migration. This means that in most cases, you can port the Fortran code of an application from UNIX to Win32 with minimal changes. The C/C++ code usually requires the majority of the migration effort.

Porting Fortran to Interix

Before rewriting a Fortran application for Win32, you should consider other migration strategies. Porting to Interix represents another possible strategy.

For porting UNIX style source to Interix, the GNU Fortran 77 compiler is provided. Since the level is Fortran 77, applications that need Fortran 90 support, such as Module support, cannot be ported to Interix. Additionally, since POSIX subsystem libraries cannot be mixed with Win32 subsystem libraries, Win32 versions of C and C++ that need Fortran libraries cannot use them from Interix. This leaves stand alone Fortran 77 applications that interoperate with **stdin** and **stdout** as the best candidates for a UNIX style port to Interix.

Porting UNIX Fortran Source to Win32

Most Fortran migrations are a port, not a rewrite, to Win32. However, Fortran migrations involve the migration, and integration of the other language modules in the application. Techniques and strategies for using C and C++ tools and source from Fortran are needed to complete the application migration.

As a result of this, most of the discussion in this section focuses on how to integrate Fortran code with C and C++ modules or libraries on the Windows platform.

Using C/C++ Libraries or Fortran Modules

Fortran applications can access cross-platform libraries either by using C and C++ libraries, or through Fortran modules (available with Fortran 90 and up). If the cross-platform libraries are in C or C++, there is little (if any) difference between this type of strategy and a port to Win32 strategy. Currently, there are not a large number of third-party Fortran module suppliers. This is due to the limited market for Fortran and the fact that Fortran compilers are provided by third parties. Microsoft does not provide a Fortran compiler. Fortran modules are typically supplied by the compiler vendor or are created in house.

Porting Fortran to Windows

Finally, it is possible to rewrite an application written entirely in Fortran to target Windows. The key task here is to identify how platform-specific features are implemented in Fortran. This is typically done with Fortran modules. If this is the case, you must identify or develop a corresponding Fortran module for each feature on Windows that exists on the source platform. For example, if an OpenGL module and threading module are used on UNIX, you need to identify or develop a corresponding OpenGL and threading module to use on Windows.

Debugging Fortran from Visual Studio

Although your Fortran application must be developed in an environment outside of Visual Studio 6.0, you may need to include your Fortran code as a library or object module in a C or C++ project in Visual Studio 6.0. In most cases, this requires the Visual Studio 6.0 debugger to step through the Fortran code, as well as the C and C++ code. This section explains how you can integrate Fortran libraries and object modules in Visual Studio 6.0 projects and debug the Fortran code when you debug other portions of your project.

Prior to debugging, you may need to include Fortran modules or libraries in projects that contain C and C++ source files. Although Microsoft does not provide a Fortran compiler with Visual Studio 6.0, you can include Fortran modules in Visual Studio 6.0 projects. To do this, .compile the Fortran library or Fortran module with an option that produces a program debug database. Visual Fortran provides this option with the compiler option **-Zi**. This is similar to the process you use to create a program debug database for C and C++ programs. This creates a file with a .pdb extension that contains the debug symbols.

Once you have a debug version of your Fortran module or library, you can include it in a C or C++ project. The easiest way to do this is to add the debug version of the Fortran project as input for the Visual Studio linker.

To add the Fortran project

1. On the **Project** menu, click **Settings**.
2. In the Settings for: list, select Win32 Debug.
3. Select the **Link** tab.
4. In the **Object/library modules** box, type the name of the Fortran object module or library you wish to include.

After you add the Fortran project, you need to enter a search path so that the linker can find the module or library you added. The path must include the current working directory and the directories specified in the **Options** dialog box.

To add a new folder to the search path

1. On the **Tools** menu, click **Options**.
2. Click the **Directories** tab.
3. In the Show Directories for: list, select Library files.
4. Click the empty box at the bottom of the list, and then type the path to the new folder.

Visual Studio does not provide an option to specify the path for the .pdb file for your Fortran module. Therefore, you need to place the .pdb file in the same directory as the library or object module you wish to include.

You can also add a Fortran module or library directly to the project.

To add a Fortran object module or library directly

1. On the **Project** menu, click **Add to Project**, and then click **Files**. The **Insert Files into Project** box appears.
2. In the **Files of Type** list, select **.obj** or **.lib**, depending on whether you need to insert an object module (.obj) or library (.lib).
3. Select the file and path for the Fortran object module or library you wish to add from the main list displayed.

Now you are ready to start debugging. You can start the debugger and step through code in either the C/C++ source, or the Fortran source.

You can also use binaries created with the **Zi** compiler option with other Microsoft debug tools, such as WinDBG.

You may need to include a Fortran object module or library in a Visual Studio 6.0 project where a debug version of the object module or library is not available, or you may not need to include the Fortran routines in your debug session. If you do not need to debug the Fortran modules in a Visual Studio 6.0 project, you can include the release versions of either the object module or library as part of the link. You can continue to debug the C or C++ code. However, the debugger will not step into the Fortran source when a Fortran routine is entered. To accomplish this, include the release version of the Fortran object module or library in the **All Configurations** section of the **Project Settings** dialog box.

If you do need to include Fortran code as part of your debugging, remember to also include the release versions of the Fortran object module or library in the **Win32 Release** section of the **Project Setting** dialog box.

Summary

The primary tasks in a Fortran code migration center on integration with libraries and the development environment. The actual Fortran code can usually be migrated or ported with little or no changes, provided that the level of Fortran compiler on the source platform is the same as the target platform. For example, if your application was developed on the source platform using Fortran 90, the target platform also needs a Fortran 90 compiler or Fortran 90 compatibility mode.

Issues related to source files and source control migration from UNIX to Windows are similar for Fortran as they are for C and C++. These issues are covered in Chapter 7, Creating the Development Environment.

patterns & practices
proven practices for predictable results
