Using Mac F2C Generated Code

Stand-Alone Programs

The C/C++ code produced by Mac F2C has special compile and link requirements. These are discussed in detail in the chapters focused on using Mac F2C with specific compilers. However, the general requirements are easily stated (some of these requirements can be relaxed; refer to the compiler-specific chapters):

Compile Requirements
- Most of these only apply to 68K code generation
- 4-byte integers
- 8-byte doubles
- far code
- far data

Link Requirements
- All libraries must be chosen compatible with above compiler settings
- ANSI libraries
- Math libraries (if not included in the ANSI libraries)
- Console libraries (for a minimal Macintosh interface)
- C++ support libraries (for C++ code only)
- Mac F2C support libraries (libI77 and libF77)
- F2Cmain.c or F2Cmain.cp

The easiest way to use Mac F2C generated code is to create a new project file using one of the Mac F2C project models or stationary (depending on your environment). Then add the code files generated by Mac F2C to the project and bring everything up-to-date. You are ready to run: that's all there is to it!

Use the "C" versions of project stationary to work with C code generated by Mac F2C and the "C++" versions of project stationary to work with C++ code generated by Mac F2C.

The enclosed AT&T Computing Science Technical Report No. 149 is your compiler and language reference manual. I encourage you to read it.

Subroutines and Functions

You can also use Mac F2C to translate individual FORTRAN subroutines and functions and then call them from your own C or C++ code. The compile and link requirements specified in the above section remain the same, except that instead of linking against F2Cmain.c or F2Cmain.cp, you link against your own C code. Also, you only need to link against libI77 and libF77 if your FORTRAN code does any I/O or calls any of the FORTRAN intrinsic functions (including exponentiation, the ** operator).

If you include either libI77 or libF77, you need to modify your C or C++ code to accomodate the fatal error trapping mechanism used by these libraries. These libraries use the ANSI longjmp() function to escape out whenever they encounter a fatal run-time error (the best example is an illegal or ill-formed format statement). For this mechanism to work, your code needs to declare and set a global jump buffer before any calls to code that might call functions in libI77 or libF77 (i.e., before any calls to code translated by Mac F2C). You can do this very simply as follows:

```
#include <setjmp.h>
#ifdef __cplusplus
    extern "C"
#endif
 jmp_buf      gRecoverToConsole;
#ifdef __cplusplus

#endif

int main()

    if ( setjmp(gRecoverToConsole) == 0 )

         /* Your code here, including calls to FORTRAN code */
         /* translated by Mac F2C */

    else

        /* Fatal run-time error in libI77 or libF77 */
        /* Handle appropriately */
```

Obviously, you can put this setjmp() wrapper individually around each call to a translated FORTRAN function instead of the entire code as in my example above. Although wrapping each call is more work, it will help debugging because you will be able to clearly identify the source of any any run-time errors

Making Mac F2C-generated Code Multitask

Functions have been built into the support library libI77 that allow you to make code generated by Mac F2C multitask under the MacOS cooperative multitasking environment.

Using Symantec, THINK or Metrowerks:

Cooperative multitasking does not happen automatically. To make FORTRAN code multitask cooperatively, you need to insert calls such as the following into your FORTRAN source code:

```
      CALL DOMULTITASK( N )
```

where N is the number of ticks (60ths of a second) that you are willing to let your program sleep while other programs execute.

If you prefer, you can instead insert the following function calls in the C/C++ code generated by Mac F2C:

```
      DoMultiTask( N );
```

where N is as above. You can mix and match the two methods if you wish. However, note that if you modify the generated C code, your modifications will be destroyed if you re-translate the FORTRAN code.

Using MPW:

MPW tools created with the F2C for MPW libraries provide cooperative multitasking for your tools. This has been coupled with the I/O of the library. You will notice that while your program is performing io the cursor will be a spinning beach ball. This allows other programs to have some of the CPU and allows your tool to be put in the background. If you have some computationally intensive code that does not perform I/O you can call the MPW SpinCursor routine explicitly to free up the CPU for other programs. The documentation for this routine is provided with MPW. To call it from FORTRAN use the something like this:

```
      CALL SPINCURSOR( N )
```

where N is the increment to an internal counter (see the MPW SpinCursor doucmentation).   You can simply use 1.   You can also use the same subroutine call as for Symantec, THINK, and CodeWarrior, namely:

```
CALL DOMULTITASK( N )
```

In the MPW case, the call to DOMULTITASK will simply call through to SPINCURSOR. However, under MPW, N is interpreted as an increment to the internal counter (instead of as the number of ticks your application is willing to sleep).