Using Mac F2C With THINK C/C++ for 68K

Before you can use the code produced by Mac F2C, you must set up and build all of the required support libraries. There are also special rules that must be followed when using code produced by Mac F2C. The process and rules are slightly different for each compiler. These instructions are for setting up Mac F2C for use with THINK C/C++ for 68K. Refer to other chapters for instructions on how to set things up for use with Symantec C/C++ for PPC or Metrowerk's CodeWarrior C/C++ compilers.

Users upgrading from earlier versions need to re-install all libraries, project files, and supporting files. All files, including library source code, have been updated in version 1.3.

Important Changes for Users Upgrading

• Do not place the source code to the Mac F2C Libraries in the folder containing the THINK Project Manager application or in any of it's sub-folders). There are name conflicts between the files in libI77/libF77 and some of the new universal header files and PPC support files. This means you must build the F2C libraries first and then copy the built libraries into the THINK Project Manager tree.

• To accomodate PPC versions, the THINK libraries `libI77a`, `libI77b`, and `libF77` have been renamed `libI77a 68K`, `libI77b 68K`, and `libF77 68`. You may either modify your existing projects or simply rename these libraries to their old names after you have built them.

All of the THINK project files shipped with Mac F2C are for the THINK Project Manger (TPM) version 7.0.5 (the version that ships with Symantec C/C++ Version 8, Release 4).

If you are using THINK v7.0.4 (the latest version you can update to for free), you can find 7.0.4 versions of all the project files and stationary files in the folder `TPM 7.0.4 Project Files` (located in the `TPM Support` folder). Simply replace all the project files and stationary files referred to in these instructions with the corresponding files of the same name from the `TPM 7.0.4 Project Files` folder.

If you have THINK v6 or an earlier v7, use the free updaters provided by Symantec to bring your copy of THINK up to v7.0.4.

If you are using a version of THINK older than v7, please refer to the additional instructions in the chapter "THINK pre-v7 and Other Compilers".

Setting Up Mac F2C Using the Installer

The easiest way to set up Mac F2C is to use the installer included with Mac F2C. This installer will only work correctly if you have System 7.5 (or higher) and have THINK v7. If you do not meet both of these requirements, please follow the instructions for manual installation found in the following section. If you are using an earlier version of THINK, please also refer to the additional instructions in the chapter "THINK pre-v7 and Other Compilers".

The installer is stand-alone AppleScript application called Mac F2C Installer. To run the installer, simply double click on it, and answer the dialogs it presents. The installer will do the following:

- Create a folder called Mac F2C Support within the folder containing the THINK Project Manager.
- Copy the Mac F2C versions of the THINK standard libraries.
- Build the appropriate Mac F2C libraries.
- Move the built libraries to the Mac F2C Support folder.
- Copy the Mac F2C project models to THINK's (Project Models) folder.
- Translate and build the appropriate versions of the test application.

When the installer is finished, you will find completed test applications in the Test Project ƒ folder. You should run these to verify correct operation of Mac F2C and its libraries. After that, you are ready to go. To compile translated FORTRAN code, simply open a new project in the THINK Project Manager, select the appropriate version of Mac F2C project models, and add the translated files.

The remaining sections provide step-by-step instructions for installing Mac F2C manually, a more detailed description of how to test your Mac F2C installation, and additional information on using code generated by Mac F2C with THINK Project Manager.

Setting Up Mac F2C Manually

All the files and folders you need are contained in the folder TPM Support (which is located within the folder Symantec/THINK Support). Unless otherwise indicated, all files and folders referred to in the following sections are located in this folder. TPM refers to the THINK Project Manager.

Step 1: Bring all the libraries up-to-date

The Mac F2C libraries come without binaries, so you have to build them according to the following algorithm:

FOR the project files:
(1) libI77a 68K (in Mac F2C Libraries folder)
(2) libI77b 68K (in Mac F2C Libraries folder)
(3) libF77 68K (in Mac F2C Libraries folder)
REPEAT the following steps:
(a) Double-click on the project file.
(b) In the THINK Project Manager's Source menu, select the Make command.
(c) Uncheck the Quick Scan check-box.
(d) Click on the Use Disk button.
(e) Click on the Make button.
END FOR-REPEAT

Step 2: Move things to the recommended locations

For easiest and smoothest operation, files should be installed as follows:

• The folder For 'Standard Libraries' contains four TPM project files: ANSI F2C, unix F2C, IOStreams F2C, and CPlusLib F2C. Drag these files (not the folder itself) to the Standard Libraries folder located in the same folder as the TPM application. If you don't have the THINK C++ compiler (or you don't plan to use Mac F2C with it) you do not need IOStreams F2C or CPlusLib F2C.

• Create a folder called Mac F2C Support inside the folder within the folder that contains the THINK Project Manager (TPM) application.

• Drag the three TPM project files in the folder Mac F2C Libraries (libI77a 68K, libI77b 68K, and libF77 68K; the same ones you brought up-to-date in Step 2 above) to the Mac F2C Support folder which you just created.

WARNING: Do not place the source code for these libraries (found in the folders libF77 Sources and libI77 Sources) in the folder containing the THINK Project Manager application or in any of it's sub-folders). The source code for these libraries has name conflicts with Apple's Universal Headers (e.g., a file called fp.h appears in both but the two are NOT equivalent files). Otherwise any of your

code that #includes any of the conflicted files may inadvertently access the wrong file.

• The folder For '(Project Models)' (located one level up in the TPM Support Folder) contains two model project folders called TPM Mac F2C C Project and TPM Mac F2C C++ Project. Drag both folders to the (Project Models) folder located in the same folder as the TPM (if you don't have THINK C++ or don't plan to use Mac F2C with the C++ compiler, you don't need the TPM Mac F2C C++ Project folder).

Verifying Correct Operation of Mac F2C

The folder Test Project ƒ contains the following files:

test.f -- a sample FORTRAN program.

F2Cmain.c -- the main program required to run programs produced by Mac F2C.

F2Cmain.cp -- the main program required to run programs produced by Mac F2C.

f2c.h -- an include file required to compile programs produced by Mac F2C.

test.c (Output) -- what you should get when you translate the sample FORTRAN code files.

Test.68K.π -- a TPM v7.0.4 project to run the sample C program.

test.c (C++ Output) -- what you should get when you translate the sample FORTRAN code files and select the C++ output option.

Test++.68K.π -- a TPM v7.0.4 project to run the sample C++ program.

Test.PPC.π and Test++.PPC.π -- SPM v8.0.1 project files to run the sample program.

Test++.*.μ -- CodeWarrior project files to run the sample program.

Translate the sample FORTRAN program Test.f simply by dragging it onto Mac F2C. Do not change any of the options (use Factory Defaults). Once you have

done this you can compare it with Test.c (C Output) file to verify that you got the same thing. If so, double click on the TPM project Test.68K.π and run it to verify correct operation.

If you also plan to use Mac F2C C++ output with the Codewarrior, you can run a second test to verify correct operation with the C++ compiler. Start Mac F2C and in the C Options dialog, select C++ code. Do not change any of the other options. Translate Test.f. Compare it with Test.cp (C++ Output) to verify that you got the same thing. If so, double click on the CPM project Test++.68K.π and run it to verify correct operation.

Using C Code Generated by Mac F2C

The C code produced by Mac F2C has the following compile and link requirements when using THINK to generate code for the 68K:

- the header file:
  f2c.h
- the F2C libraries:

libI77a 68K
libI77b 68K

libF77 68K

- the THINK libraries:

    ANSI F2C
    unix F2C
- for C++ code only, the THINK libraries:
    IOStreams F2C
    CPlusLib F2C
- 4-byte integers (C only)
- 8-byte doubles
- Native Floating-Point Format
- Far Code
- Far Data
- C++ ANSI Conformance unchecked (C++ only)

In addition, if you compile a stand-alone FORTRAN program (instead of only some FORTRAN subroutines) you must include F2Cmain.c in your project (or F2Cmain.cp if you use C++; the two files are identical). This is because the original main routine in the FORTRAN program becomes a function that is called by F2Cmain.c. In addition, F2Cmain.c performs a series of initializations (primarily related to error catching) prior to executing the main FORTRAN program.

The model project provided (TPM Mac F2C C Project) is a folder that contains everything you need to compile and run code produced by Mac F2C using the C compiler. This folder has a copy of F2Cmain.c, f2c.h, and a project file that includes the appropriate libraries and option settings.

The C++ model project (TPM Mac F2C C++ Project) is a folder that contains everything you need to compile and run code produced by Mac F2C using the C++ compiler. This folder has a copy of F2Cmain.cp, f2c.h, and a project file that includes the appropriate libraries and option settings.

If you have THINK C version 7.0 or better, simply create a new project by using the New Project command in the THINK Project Manager's File menu and selecting TPM Mac F2C C Project or TPM Mac F2C C++ Project as the model for the new project. Add your code files as appropriate, bring it up-to-date (you may need to use the Make command and the Use Disk option the first time), and run.

If you have an earlier version of THINK C, first replace the project file provided in Mac F2C C Project with one made with your version of THINK (I have included a list of the project contents for your convenience—see the additional instructions in the chapter "THINK pre-v7 and Other Compilers" if you have trouble doing this). Do not include any objects at this time. To start a new

project, duplicate the entire Mac F2C C Project folder, change the names of files and folders as appropriate, add your code files, and bring everything up-to-date.

If you compile a FORTRAN subroutine or function that you want to call from a C program, look at the output C code to see the appropriate calling protocol. You may or may not need to include the F2C support libraries (libF77 68K, libI77a 68K, and libI77b 68K). In rare cases, you may also need to copy some of the initialization code from F2Cmain.c to your calling program.

Special 68K Considerations

Please read the following section carefully if you intend to use Mac F2C with the THINK compiler generating 68K code:

As noted above, code produced by Mac F2C MUST be compiled with 4-byte integers. This requirement cannot be relaxed. The other requirements (8-byte doubles, native floating-point format, far code, and far data) can sometimes be relaxed:

IF you do not use doubles in any situation where their size relative to reals matters (e.g., if you do not use doubles in equivalence and common statements), then your code probably does not require 8-byte doubles. You need to verify this on a case-by-case basis.

This requirement exists because Mac F2C follows FORTRAN sizing rules when compiling FORTRAN code: sizeof(real) == sizeof(integer) and sizeof(double) == 2*sizeof(real). FORTRAN real is compiled as C float and FORTRAN double as C double, so doubles have to be 8-bytes long for equivalence and common statements to be properly aligned. There are a few other cases where the size of double variables matters; see AT&T Computing Science Technical Report No. 149 (included with Mac F2C) for a detailed discussion.

IF you compile your program with the option Local variables are automatic and you do not have large static data structures, you might not need Far Data. You need to verify this on a case-by-case basis.

Mac F2C creates large static data structures for I/O. If you create local variables in the global area (static instead of automatic) or if you have other static data, you will almost certainly require Far Data. The I/O data structures can be large enough that you may require Far Data for that reason alone.

IF your program is not very large and doesn't have a large number of subroutines, you probably will not need Far Code. You need to verify this on a case-by-case basis.

Mac F2C tends to produced redundant copies of utility code (especially code for performing array indexing). It can also produce large numbers of auxiliary functions. The result is that Far Code is often required. Compile first with Far Code, then check the code size and jump table to see if you can relax this requirement.

IF your program will not be compiled under THINK C++ (i.e., you chose K&R C or ANSI C output instead of C++ output) and you will not link with code produced by THINK C++, you do not need native floating-point format. The native floating-point format option is selected only to guarantee compatibility with the THINK C++ compiler should you chose to use the output of Mac F2C with C++ code.

If you change the 8-byte doubles, native floating-point format, Far Code, or Far Data options, remember to also change them in all the libraries, specifically libI77a 68K, libI77b 68K, libF77 68K, ANSI F2C, unix F2C, IOStreams F2C, and CPlusLib F2C (the latter two for C++ only).

I urge all users to read the enclosed AT&T Computing Science Technical Report No. 149. Consider it your compiler and language reference manual. You can print the report by downloading it to any PostScript printer. You can use Apple's LaserWriter Utility application to do this or you can use any of the many equivalent utilities.