Chapter 2

# Using the CM Fortran CMSSL Interface

This chapter contains information about running CM Fortran programs that call CMSSL routines. The following topics are included:

- creating a CM Fortran CMSSL program
- a note about aligning arrays
- using the CMSSL safety mechanism
- on-line sample code and man pages

## 2.1 Creating a CM Fortran CMSSL Program

To use CMSSL from within a CM Fortran program, follow these steps:

1. Read the *CMSSL Release Notes* for information specific to the CMSSL release you are using and for updates to the manual. Important information such as switches for linking with CMSSL may change from release to release.
2. Include the header file **`/usr/include/cm/cmssl-cmf.h`** if you are calling a CMSSL function or a CMSSL subroutine that uses predefined symbolic constants.
3. Place calls to CMSSL routines into CM Fortran code.
4. Use the CM Fortran **`cmf`** command to compile your code.
5. Use the appropriate **`cmf`** command line switch to link with CMSSL.


*Copyright © 1994 Thinking Machines Corporation*

The remainder of this chapter discusses these steps in detail.

### 2.1.1 Including the CMSSL Header File

A CM Fortran program that calls CMSSL can access the appropriate header file if you place the following line at the top of any program unit that makes a CMSSL call:

```
INCLUDE '/usr/include/cm/cmssl-cmf.h'
```

This file declares the return values of CMSSL functions and defines symbolic constants used as parameter values for some CMSSL routines.

The **INCLUDE** line is *required* only in CM Fortran code that contains CMSSL function calls or uses predefined CMSSL symbolic constants. However, we recommend that you include the header file wherever you use CMSSL. It is easier to do this at the outset than to remember to add the **INCLUDE** line should you add a CMSSL function call to your code in the future. Also, in the future, the library is likely to make greater use of symbolic constants, which require the definitions provided in the header file.

If the CM Fortran compiler cannot find the CMSSL include files, check your partition manager for the existence of a path to the appropriate directory. If the files appear to be missing, consult your system administrator or your Thinking Machines Corporation customer service representative.

### 2.1.2 Calling CMSSL Routines

Before invoking a CMSSL routine from within a CM Fortran program, make sure you are using compatible versions of CM Fortran and CMSSL. To find out which version of the CM Fortran compiler is required, see the *CMSSL Release Notes* shipped with the CMSSL version you are using. Treat the CMSSL routine as you would any other subroutine or function.

### 2.1.3 Compiling and Linking

After writing a CM Fortran program that calls CMSSL routines, compile it and link it with the library. Compiling a CM Fortran CMSSL program is the


Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

same as compiling other CM Fortran programs: use the **cmf** command. To compile the program *program* on a CM-5 and link it with CMSSL, you can use either of the alternatives indicated below.

### Linking with CMSSL Using −lcmssl

If the CM Fortran program *program* calls CMSSL routines, you can compile it and link it with CMSSL using the **-lcmssl** switch. This switch links with the appropriate CMSSL library based on the execution modal you specify in the **cmf** command line. To use this alternative, issue one of the following command lines at the UNIX prompt:

- In the CM Fortran (SPARC) nodes model:

  **cmf -cm5 -sparc -o** *program program*.**fcm -lcmssl**

- In the CM Fortran vector-units model:

  **cmf -cm5 -vu -o** *program program*.**fcm -lcmssl**

- In the CM Fortran single-node model:

  **cmf -cm5 -vu -node -o** *program program*.**fcm -lcmssl**

  In the single-node model, add **-cmmd_root /usr/cmmd/3.0** before **-lcmsslcm5vu-node** if CMMD Version 3.0 (or higher) is not the default on your system.

- To compile and link a CM Fortran global/local program that makes calls to CMSSL routines in both the global and the local program units:

  **cmf -cm5 -vu -o** *program program*.**fcm -lcmssl -local** *local*.**fcm** *program*.**proto -local -lcmssl**

### Linking with a Selected CMSSL Library Explicitly

Alternatively, if you want to link with the appropriate CMSSL library explicitly, issue one of the following command lines at the UNIX prompt:

- In the CM Fortran (SPARC) nodes model:

  **cmf -cm5 -sparc -o** *program program*.**fcm -lcmsslcm5**


Copyright © 1994 Thinking Machines Corporation

- In the CM Fortran vector-units model:

  **cmf -cm5 -vu -o** *program program*.**fcm -lcmsslcm5vu**

- In the CM Fortran single-node model:

  **cmf -cm5 -vu -node -o** *program program*.**fcm**
  **-lcmsslcm5vu-node**

  In the CM Fortran single-node model, add **-cmmd_root /usr/cmmd/3.0** before **-lcmsslcm5vu-node** if CMMD Version 3.0 (or higher) is not the default on your system.

- To compile and link a CM Fortran global/local program that makes calls to CMSSL routines in both the global and the local program units:

  **cmf -cm5 -vu -o** *program program*.**fcm -lcmsslcm5vu**
  **-local** *local*.**fcm** *example*.**proto -local**
  **-lcmsslcm5vu-node**

#### Using the Correct Version of CMOST

CMSSL is a layered product. Any CMSSL version requires a specific CMOST version and a specific CM Fortran version. If these dependencies are not observed, proper operation of CMSSL routines is unlikely. Consult the most current version of the *CMSSL Release Notes* to find out which versions of CM Fortran and CMOST are required by the current version of CMSSL.

### 2.1.4 Executing CMSSL Programs

Execute a CM Fortran CMSSL program just as you would any compiled CM Fortran program.

## 2.2 A Note about Aligning Arrays

Many CMSSL routines fail when supplied with a CM array that has been aligned (using the CM Fortran **CMF$ ALIGN** directive) to an array of higher rank. CM Fortran reuses the geometry of the **ALIGN** target, rather than the


Version 3.2, April 1994
Copyright © 1994 Thinking Machines Corporation

**ALIGN** source, causing the CMSSL array rank checks to fail. It is recommended that you avoid using arrays that are aligned to arrays of higher rank.

## 2.3 Using the CMSSL Safety Mechanism

You can use the CMSSL safety mechanism in two ways:

- by setting the environment variable **CMSSL_SAFETY**
- by using the calls **CMSSL_get_safety** and **CMSSL_set_safety** in a program

### 2.3.1 Safety Mechanism Features

The CMSSL safety mechanism offers two basic features: it synchronizes the CM-5 parallel processing elements and partition manager so that you can pinpoint the area of code that generated an error, and it performs error checking and reports errors at several levels of detail.

#### Synchronization

The CM-5 parallel processing elements and partition manager operate asynchronously with respect to one another. Without the CMSSL safety mechanism, an error that occurs in the parallel processing elements is not reported to the partition manager until the next time the partition manager requests information from or checks the status of the elements. Such a request or status check is known as an *implicit synchronization* because it has the side effect of synchronizing the processing elements and partition manager, allowing the processing elements to report any accumulated errors. When an implicit synchronization occurs, there is no way to tell exactly when the reported error occurred, or which module of code produced it.

The CMSSL safety mechanism addresses this problem by forcing *explicit synchronization* between the parallel processing elements and the partition manager before, after, and within each CMSSL call in your code. The safety mechanism traps and reports errors, indicating when the errors occurred in relation to the synchronization points.


Copyright © 1994 Thinking Machines Corporation

### Error Checking and Reporting

The safety mechanism can perform error checking and generate run-time error information at several levels of detail. You can turn safety checking on at any level during all or part of a program. One level checks for errors in the usage and arguments of the CMSSL calls in your program; a more detailed level also checks for errors generated by internal CMSSL routines. Examples of errors found and reported by the safety mechanism include the following:

- A supplied or returned data element that should be numerical is not; for example, it is identified as a "Not a Number" (NaN) or as infinity. (NaNs are defined in the IEEE Standard for Binary Floating-Point Arithmetic.)
- The code generates a division by 0 (for example, because of bad data, a user error, or an internal software problem).
- The code references a memory location that it has not initialized. The safety mechanism identifies this kind of error by writing NaNs to all allocated processing element memory. If the code references a memory location without first explicitly assigning it a numerical value, the NaN at that location causes further errors that make the original erroneous reference easy to find. (This is the same strategy used by CM Fortran safety checking when you include the **−safety=10** option on the **cmf** command line.)

As more debugging checks and safety levels are added in future releases, CMSSL safety checking will become more exhaustive.

## 2.3.2 Levels of Error Checking

The CMSSL safety mechanism currently provides the following levels:

0 (**off**) Turns off the safety mechanism. Explicit synchronization and error checking are not performed. This level is appropriate for production runs of code that has already been thoroughly tested.

1 (**on**) Checks for and reports errors caused by incorrect usage or arguments in high-level-language CMSSL calls. Performs explicit synchronization before and af-



Copyright © 1994 Thinking Machines Corporation

ter each call and locates each error with respect to the synchronization points. This safety level is appropriate during program development or during runs for which a small performance penalty can be tolerated.

9 (**full**) Checks for and reports all level 1 errors, and in addition any errors generated by the lower levels of code that are called by the high- level-language CMSSL calls. Performs explicit synchronization in these lower levels of code and locates each error with respect to the synchronization points. This level performs all implemented error checking and exacts a very high performance price. It is appropriate for detailed debugging when a problem occurs. If you cannot analyze and correct the problem, provide your local site coordinator, applications engineer, or Thinking Machines Corporation customer service representative with the output generated by level 9 safety checking.

At levels 1 and 9, some safety mechanism error messages are displayed at the terminal when you run the program; other information appears in the backtrace when you use a debugger such as Prism.

If you report a software problem to your local site coordinator, applications engineer, or Thinking Machines Corporation customer service representative, you may be asked to run your program with the CMSSL safety mechanism enabled at a level other than 0, 1, or 9. These additional levels are used for pinpointing problems in the internal software or for obtaining internal status information.

### 2.3.3 Setting the CMSSL Safety Environment Variable

To set the CMSSL safety level using the **CMSSL_SAFETY** environment variable, issue the command

```
setenv CMSSL_SAFETY { 0 | 1 | 9 | off | on | full }
```

choosing one of the listed options. As indicated above, 0 is equivalent to **off**, 1 to **on**, and 9 to **full**.

The advantage of using the **CMSSL_SAFETY** environment variable is that you can set or change the safety level without recompiling your code.


Copyright © 1994 Thinking Machines Corporation

### 2.3.4 Using CMSSL Safety from within a Program

To set the CMSSL safety level, issue the following call and specify the desired level in the integer argument *n*:

**cmssl_set_safety** (*n*)

To obtain the current CMSSL safety level, issue the following call:

**cmssl_get_safety** ( )

The advantage of using these calls from within a program is that you can set or obtain the safety level at any point within your code. However, you must recompile the code each time you change these calls.

**NOTE**

The inner product, 2-norm, outer product, matrix vector multiplication, vector matrix multiplication, and matrix multiplication routines described in Chapter 3 perform error checking only when the CMSSL safety mechanism is on. Therefore, we strongly recommend that you turn CMSSL safety on when testing new programs that call these routines.

## 2.4 On-Line Sample Code and Man Pages

Included with CMSSL are sample on-line programs that demonstrate how to call each CMSSL routine. You are encouraged to experiment with these sample programs. Also included with CMSSL are on-line man pages for all routines.



Copyright © 1994 Thinking Machines Corporation

The on-line sample programs are located in subdirectories of the CMSSL examples directory. The default location for the examples directory is

**/usr/examples/cmssl**.

Examples for the operation *operation* are included in the subdirectory

*operation***/cmf**

or

*operation***/***sub-operation***/cmf**

of the examples directory. For example, the sample code for the routine that performs eigenvector analysis using the Jacobi method is located in the sub-directory

**eigen/jacobi/cmf**

of the examples directory. If you do not find the on-line examples in **/usr/examples/cmssl**, check with your system administrator (or the person who installs CMSSL at your site) to find out where they were installed.

To read the on-line man page for a routine, enter the command

**man** *routine_name*

at the UNIX prompt.


Copyright © 1994 Thinking Machines Corporation