CERES Software Bulletin 95-13

CERESlib Access and the CERES makemake Utility, September 18, 1995

-Joe Stassi (j.c.stassi@larc.nasa.gov)

1.0 Purpose:

This bulletin has a dual purpose. First, it describes how to load the CERESlib library routines into Fortran 90 programs. This requires adding proper flags and object files to the code compile and link commands.

This bulletin also explains the CERES makemake utility which automates the creation of Fortran 90 Makefiles. These Makefiles automatically include the compilation and link requirements for the CERESlib interface. A simple example with an exercise is provided at the end of the bulletin to illustrate the simplicity of using makemake. For those who do not particularly care about the other details, it is permissible at this time to jump directly to the example and exercise in Section 4.0.

2.0 Accessing CERESlib Libraries

The procedures outlined in Sections 2.2 and 2.3 below are automated by the makemake utility described in Section 3.0.

2.1 Checking for $CERESLIB Directory Visibility

The CERES project system administrators have set up directories, mounts, and environment variables so that any machine needing visibility to CERESlib will have it without requiring any special procedures by the user of that machine.

The CERESlib routines are located under the $CERESLIB directory. To test whether your machine has $CERESLIB visibility, type the following command.

```
ls $CERESLIB
```

If you get “Undefined variable,” or any response other than a directory listing which includes the makemake and cereslib.a files, then your machine has not been configured for CERESlib. Please direct any questions to your system administrator or to the e-mail address at the top of this memo.

2.2 Compilation Flag

To USE CERESlib modules, the compiler needs the CERESlib module interface information at compilation. This information is located in the $CERESLIB/Mod subdirectory. To make this information available to the compiler, add the -I$CERESLIB flag to the compilation command.

```
f90 -I$CERESLIB/Mod ...
```

This flag is required whenever compiling files which contain references to CERESlib modules.

2.3 Linking Object Libraries

If a program USEs CERESlib modules, then the linker needs to know where to find the object code for those modules. The CERESlib object code is contained within three separate library files within the $CERESLIB directory: data_products.a, toolkit.a, and cereslib.a. The CERESlib contents and documentation are still under development and will be available soon under the ASD Home Page on WWW.

To make the object libraries available at link time, add them to the end of the f90 link command. Here is how the libraries should be listed on the link command line.

```
$CERESLIB/data_products.a
$CERESLIB/toolkit.a
$CERESLIB/cereslib.a
```

Only the libraries which contain modules used in the program need be added at link time, though it doesn't hurt to include all three. Whenever all three are listed, they should be listed in the order shown above. The order is important because of interdependencies between the libraries; the data_products.a library uses the toolkit.a library, and both of these use the cereslib.a module.

```
f90 -o myprog.exe myprog.f90 ... $CERESLIB/cereslib.a
```

3.0 Using makemake

3.1 What is a Makefile?

A Makefile is like a script file in that it defines a sequence of shell commands to be executed. What distinguishes a Makefile is that it allows the user to define dependencies so that certain commands are performed only if their "target" has been outdated.

The natural and most common application for Makefiles is code compilation. With a Makefile, a user can list the sequence of commands necessary for creating an executable program from a collection of source code files. The Makefile will not only contain the instructions for compiling the individual parts and then linking them together--this can be done with a script file--but it will also define the interdependencies between the parts and update only those portions which have been outdated. Thus, if a program consists of 100 different routines separated into individual source code files, and one routine is modified, the Makefile will recompile only the modified routine and any routines which are dependent upon it. This is more elegant than a compilation script which would recompile all 100 routines regardless of whether or not they were affected by the change.

3.2 What is makemake?

Makemake is a perl script, retrieved from the Fortran Market on the Internet, which creates Makefiles for f90 programs. Unlike f77 programs where compilation order of the individual parts is irrelevant, the USE facility in f90 requires that USE'd modules get compiled before any routines or modules which USE them. In a program consisting of several different modules, the interdependencies between modules can get complicated, and the effort required to construct a Makefile which properly describes these interdependencies can be quite significant. Makemake automates this process for you.

The CERES makemake script, available in the $CERESLIB directory, is actually a modified version of the one retrieved from Internet. Modifications have been made to handle some peculiarities of the NAG 90 compiler related to the .mod files. But more significantly, the makemake script has been modified to automatically add compilation and link flags necessary for loading CERESlib routines.

3.3 Aliasing the makemake Command

To alias the makemake command, add the following line to the .cshrc file in the top directory of your account.

```
alias makemake   '$CERESLIB/makemake'
```

This will define a "makemake" command on your account which points to the makemake script in the $CERESLIB directory. This command will be automatically defined whenever you begin a new session or open a new window, though it will be necessary for you to source your .cshrc file in order to activate the command in any current sessions.

3.4 Creating an Executable with a makemake Makefile

Here are procedures for creating and using makemake Makefiles.

a) Place all source files associated with a particular program into a <u>separate</u> directory. (Note: the makemake utility will not work properly if more than one (main) program is present in the directory.)

b) Create a Makefile by invoking the makemake command in that directory. In the following example, the Makefile target is called prog1.exe.

```
makemake prog1.exe
```

c) A Makefile should now exist in the directory. Use it to create the executable file prog1.exe.

```
make
```

The above procedures work for any f90 program regardless of its dependency on CERESlib routines. This, of course, is contingent upon the availability of a f90 compiler on your workstation, as well as code which is free from syntax errors.

3.5 A Word about ECS Toolkit INCLUDE Files

ECS Toolkit INCLUDE files have been given the ".f" extension, which was an unfortunate choice since the makemake utility interprets these files as compilable f77 files. If these files are within your source code directory when you execute makemake, then the resulting Makefile will attempt to compile these files into object code.

Solution: *Remove from the Makefile all ".o" references corresponding to ECS Toolkit ".f" INCLUDE files. These references will be found near the top of the Makefile on the line starting with "OBJS_F = ".*

4.0 A Simple Example: Approximating the Earth’s surface area.

In this section, an example is given which specifically uses the CERESlib interface. As a suggested exercise, the reader should use makemake to compile and run this program on his/her machine. The steps for this exercise are outlined in Section 4.4.

4.1 Algorithm

The test_area program was written to validate the region_area function found in the CERESlib reference_grid module. It uses the following algorithm.

a) Using the region_area function, sum the area of all 26,410 regions on the globe.

b) Calculate an independent estimate of the earth’s total surface area using the spherical area formula, $A = 4\pi R^2$, where R = the mean radius of the earth.

c) Compare the region area sum to the surface area estimate. Find the percent error of the region area sum, assuming that the surface area estimate is correct.

4.2 Program Area_test.f90

Here is the source code for the area_test program.

```
!-----------------------------------------------------!
 PROGRAM area_test
   USE ceres_constants, only : PI, EARTH_RADIUS_MEAN
   USE f90_kind, only : real8
   USE reference_grid, only : number_of_zones, &
        number_of_regions, region_area
   IMPLICIT NONE
   REAL(real8) :: region_area_sum, earth_surface_area
   REAL(real8) :: error, percent_error
   INTEGER :: zone_no

   !*** sum the areas from all regions on the globe
   region_area_sum = 0._real8
   DO zone_no = 1, number_of_zones()
      region_area_sum = region_area_sum + &
           number_of_regions(zone_no) * region_area(zone_no)
   END DO

   !*** estimate earth surface area using spherical formula
   earth_surface_area = 4 * PI * EARTH_RADIUS_MEAN ** 2

   !*** output the results
   WRITE(*,'(/)')
   WRITE(*,'(/a,f15.4,a)') 'Earth surface area  = ',&
        earth_surface_area, ' sq km'
   WRITE(*,'(a,f15.4,a)') 'Sum of region areas = ', &
        region_area_sum, ' sq km'

   !*** calculate and output the error
   error = earth_surface_area - region_area_sum
   percent_error = error / earth_surface_area * 100.
   WRITE(*,'(a)') '------------------------------------------'
   WRITE(*,'(a,f15.4,a)') 'error               = ', error, ' sq km'
   WRITE(*,'(a,f15.4,a)') 'percentage error   = ', &
        percent_error, '%'
   WRITE(*,'(/)')

   STOP
 END PROGRAM area_test
!----------------------------------------------------------!
```

4.3 CERESlib Routines and PARAMETERS in Area_test

The area_test program uses the following CERESlib PARAMETERS and routines.

a) From ceres_constants module: `PI and EARTH_RADIUS_MEAN`

b) From f90_kind module : `real8`

c) From reference_grid module: `number_of_zones,region_area, and number_of_regions`

4.4 Exercise

This exercise demonstrates the “invisible to the user” interface to CERESlib provided by the CERES makemake utility.

Preliminary “one time” procedures

a) Verify $CERESLIB directory visibility. (see Section 2.1)

b) Alias the makemake command. (see Section 3.3)

Exercise steps

a) Place the file area_test.f90 into a separate directory in your account. The file can be obtained from the subdirectory $CERESLIB/Source/Example.

b) Type makemake at.exe (to create a Makefile)

c) Type make (to create the executable file at.exe)

d) Type at.exe (to run the program)