I. User's Guide to the Flux Coupler

The following subsections describe how to use the Flux Coupler on one of NCAR's Cray Supercomputers. The coupled system can be,and has been, run on other platforms which have sufficient CPU speed,incore memory, and output data transfer and storage capability. In subsection I there is an example of a coupled model batch job script.The accompanying explanation gives a top-level view of how to execute the entire coupled system. The next two subsections, II and III, provide example scripts that illustrate how to prepare the Flux Coupler and the various component models (ocean, sea-ice, atmosphere, land) for execution. Such preparations are deferred to separate and independent "setup scripts", one for each executable. Subsection IV explains how the Coupler source code is maintained and how to build an executable code. The final subsection V, describes the contents of the Coupler's output history data and the history data format, netCDF.

I. The Coupled Model Batch Job Script

In this section we examine a CSM coupled model batch job shell script. The basic purpose of this script is very simple: first the Coupler and the component model codes are prepared for execution, and then the Coupler and all component models are executed simultaneously as separate processes. Upon startup, the component models first establish a connection to the Coupler and then data begins to flow between them. All component models will continue to advance forward in time until they are signaled to stop by the Coupler. When the coupled system stops is determined solely by the Coupler (see section J).

The example Network Queuing System (NQS) shell script shown below is a complete script, suitable for actual use. This script first prepares each component for execution and then executes the components simultaneously. While the specifics shown below are a recommended method for running the coupled system on NCAR's Cray computers, variations on this method will also work, some of which may be more appropriate for different situations.

The coupled model batch job script below is a top-level NQS batch job script which in turn calls five separate subscripts called "setup scripts." These subscripts, "cpl.setup.csh," "atm.setup.csh," "ice.setup.csh," "lnd.setup.csh," and "ocn.setup.csh," are responsible for building their respective executable codes and gathering any required input data files, also see subsections II and III. The setup scripts receive input data from the parent NQS script by means of several environment variables. After calling the setup subscripts, the parent NQS script executes the Coupler and all component models simultaneously as background processes. The parent script waits for these background processes to complete; when they do, the coupled run has completed and the NQS script saves stdout output and terminates. Following the example is a more detailed explanation of what is being done in the various parts of the NQS script.

```
#=======================================================================
#  This is a CSM coupled model NQS batch job script
#=======================================================================
#-----------------------------------------------------------------------
# (a) Set NQS options
#-----------------------------------------------------------------------
# QSUB -q reg                              # select batch queue
# QSUB -lT   5:10:00 -lt   5:00:00         # set CPU time limits
# QSUB -lM 35Mw  -lm 20Mw                  # set memory limits
# QSUB -mb -me -eo                         # combine stderr & stout
# QSUB -s  /bin/csh                        # select shell script
# QSUB                                     # no more QSUB options
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
# (b) Set env variables available to model setup scripts (below)
#     CASE, CASESTR, RUNTYPE, ARCH  , MAXCPUS, MSGLIB, SSD,
#     MSS , MSSDIR , MSSRPD , MSSPWD, RPTDIR
#-----------------------------------------------------------------------
setenv CASE     test.00          # case name
setenv CASESTR '(CSM test)'      # short descriptive text string
setenv RUNTYPE  initial          # run type
setenv ARCH     CRAY             # machine architecture
setenv MAXCPUS  8                # max number of CPUs available
setenv MSGLIB   MPI              # message passing library
setenv SSD      TRUE             # is SSD available?
setenv MSS      TRUE             # is MSS available?
setenv MSSDIR   /DOE/csm/$CASE   # MSS directory path name
setenv MSSRPD   365              # MSS file retention period
setenv MSSPWD   'rosebud'        # MSS file password
setenv RPTDIR   $HOME            # where restart pointer files are saved
#-----------------------------------------------------------------------
# (c) Specify input, output, and execution directories
#     o the component model setup.csh scripts must be in $NQSDIR
#     o stdout & stderr output is saved in $LOGDIR
#-----------------------------------------------------------------------
set EXEDIR = /usr/tmp/doe/$CASE           # model runs here
set NQSDIR = ~doe/$CASE                    # model setup scripts are here
set LOGDIR = $NQSDIR                      # stdout output goes here
#-----------------------------------------------------------------------
# (d) Prepare component models for execution
#     o create execution directories:  $EXEDIR/[atm|cpl|lnd|ice|ocn]
#     o execute the component model setup scripts located in $NQSDIR
#       (these scripts have access to env variables set above)
#     o see the man page for details about the Cray assign function
#-----------------------------------------------------------------------
setenv FILENV ./.assign    # allow separate .assign files for each model
set LID = "`date +%y%m%d-%H%M%S`"  # create a unique log file ID
mkdir -p $EXEDIR
cd       $EXEDIR
foreach model (cpl atm ice lnd ocn)
mkdir $EXEDIR/$model
cd    $EXEDIR/$model
$NQSDIR/$model.setup.csh >&!  $model.log.$LID || exit 2
end
#-----------------------------------------------------------------------
# (e) Execute models simultaneously (allocating CPUs)
#-----------------------------------------------------------------------
rm $TMPDIR/*.$LOGNAME.*  # rm any old msg pipe files
ja $TMPDIR/jacct         # start Cray job accounting
cd $EXEDIR/cpl
env NCPUS=$MAXCPUS cpl -l 0 -n 5 -t 600 < cpl.parm >>&!  cpl.log.$LID &
cd $EXEDIR/atm
env NCPUS=$MAXCPUS atm -l 1      -t 600 < atm.parm >>&!  atm.log.$LID &
cd $EXEDIR/ocn
env NCPUS=$MAXCPUS ocn -l 2      -t 600 < ocn.parm >>&!  ocn.log.$LID &
cd $$EXEDIR/ice
env NCPUS=2        ice -l 3      -t 600 < ice.parm >>&!  ice.log.$LID &
cd $EXEDIR/lnd
env NCPUS=$MAXCPUS lnd -l 4      -t 600 < lnd.parm >>&!  lnd.log.$LID &
ja -tsclh $TMPDIR/jacct  # end   Cray job accounting
#-----------------------------------------------------------------------
# (f) save model output (stdout & stderr) to $LOGDIR
#-----------------------------------------------------------------------
cd $EXEDIR
gzip -v */*.log.$LID
cp   -p */*.log.$LID* $LOGDIR
#=======================================================================
# End of nqs shell script
#=======================================================================
```

Items (a) through (f) in the above job script are now reviewed.

(a) Set NQS options

The Network Queuing System (NQS) is a special facility available under UNICOS (Cray's version of UNIX). NQS is a batch job facility: you submit your job to NQS and NQS runs your job. The QSUB options set here select the queue, the maximum memory required, the maximum time required, the combining of the NQS script's stdout and stderr, and the shell to interpret the NQS script. See the qsub man page on a UNICOS computer for more information.

(b) Set environment variables for use in the model setup scripts

While the executing of the Coupler and the component models is explicitly done in this NQS script, the building of executables from source code, the gathering of necessary input data files, and any other pre-execution preparation is deferred to the subscripts "cpl.setup.csh," "atm.setup.csh," "ice.setup.csh," "lnd.setup.csh," and "ocn.setup.csh." The 12 environment variables set in the NQS script may be used by the setup scripts to prepare the respective codes for execution. These environment variables are specifically intended to be used as input to the component model setup scripts - these variables are not intended to be accessed by component model executables. It is strongly suggested that component model binaries do not contain a hard-coded dependence on these environment variables. The environment variables are:

- CASE: a string of one to eight characters which is the case name. Component models may also want to include this case name in their output data files to help identify the origin of that data. CASE must consist of valid UNIX filename characters. For an example, see subsection II of this User's Guide.
- CASESTR: a string of zero or more characters which briefly describes the case. There is no specific restriction on length, but most component models probably wouldn't use more than the first 20 to 30 characters. Component models may want to include this case string in their output data files to help identify the origin of that data. For an example, see subsection II.
- RUNTYPE: one of "initial," "continue," "regen," or "branch." Component models may need this information to create an appropriate input parameter namelist file. For an example, see subsection II.
- ARCH: the machine architecture. In general, the CSM may be run on a variety of computer architectures. $ARCH specifies which architecture is being used.
- MAXCPUS: an integer specifying the maximum number of CPUs each component can use for the coupled run.
- MSGLIB: the message passing library being used. In general, the system may be able to use a variety of underlying message passing systems. If this is the case, $MSGLIB specifies which library is to be used.

- SSD: one of "TRUE" or "FALSE". This indicates whether or not a Solid-state Storage Device (SSD) is available for use (i.e., whether the model can be run out-of-core or not). The SSD is site specific to NCAR. See the assign man page on an NCAR UNICOS system for more details.
- MSS: one of "TRUE" or "FALSE". This indicates whether or not the Mass Storage System (MSS) is available for use. The MSS is site specific to NCAR. See the mswrite man page on an NCAR UNICOS system for more details.
- MSSDIR: a Mass Storage System (MSS) directory path name. Component models should send their MSS data to a subdirectory of this directory. Generally $MSSDIR will be something like "/DOE/csm/$CASE/" and models will create their own MSS data subdirectory, e.g., "/DOE/csm/$CASE/ocn/". The MSS is site specific to NCAR. See the mswrite man page on an NCAR UNICOS system for more details. For an example, see subsection II.
- MSSRPD: an integer specifying the MSS retention period in days. See the mswrite man page on an NCAR UNICOS system for more details.
- MSSPWD: a character string to be used as the MSS write-permission password. See the mswrite man page on an NCAR UNICOS system for more details.
- RPTDIR: a character string specifying an NFS-mounted directory into which component model restart pointer files are to be saved.

(c) Specify input, output, and execution directories

Here we specify the directory where the model will run, the directory where the model setup scripts are found, and the directory where stdout and stderr output data will be saved when the simulation finishes.

- EXEDIR: a parent directory where the coupled system is run. Five subdirectories, atm, cpl, ice, lnd, and ocn, are created beneath $EXEDIR. All executables, source code, and all related data files are located in their respective subdirectories.
- NQSDIR: the NFS-mounted directory where the Coupler and component model setup scripts "cpl.setup.csh," "atm.setup.csh," "ice.setup.csh," "lnd.setup.csh," and "ocn.setup.csh" must be located.
- LOGDIR: this is where stdout and stderr output files are saved when the simulation is complete.

(d) Prepare component models for execution

Here the execution directory and component model subdirectories are created and the Coupler and component model setup scripts are invoked. The purpose of the setup scripts is to build their respective executable codes, document what source code and data files are being used, and gather or create any necessary input data files. It is recommended that each component model have it's own, separate setup script. This natural decomposition of code allows the persons responsible for a given model to create an appropriate setup script for their model without being confused by the details of another model. Setting $FILENV to ./.assign allows each executable to create and use it's own, independent assign file. Assign is a UNICOS specific file I/O utility that may or may not be used by the various executables. See the assign man page on a UNICOS system for more details.

(e) Execute component models simultaneously

In section (d), via the setup scripts, all necessary pre-execution preparations were taken care of. At this point, all models are ready to be run. In this section we execute the Coupler and all component models simultaneously as background processes. Command line environment variables allow one to specify different numbers of CPUs to the different component models. The "-l" command line options are used by the message passing system, MPI, to assign logical process numbers to the component models. The "-t 600" options tell MPI how many seconds to wait for a message to be received before assuming that an error has occured the message will never be sent. The ja command is a UNICOS job accounting utility which provides data on CPU time used, memory used, etc. See the ja man page on a UNICOS system for more details.

(f) Save model output (stdout & stderr)

A separate stdout output file, combined with the stderr output, from each component model is compressed and saved to the directory $LOGDIR.

II. Preparing the Flux Coupler for a Coupled Run

Here we describe all the necessary pre-execution preparation of the Coupler. This is done by a cshell setup script called "cpl.setup.csh," which is called by the parent NQS batch job shell script described in subsection I. The purpose of the setup script is to build an executable Coupler code, gather and/or create any necessary data files, and to document exactly which code and data files are being used in a given simulation.

The Coupler has its own, separate subdirectory, $EXEDIR/cpl, see subsection I, in which the setup script is run, in which the Coupler executable resides, and in which all the Coupler's input and output files are kept. The environment variables $CASE, $CASESTR, $RUNTYPE, $ARCH, $MSGLIB, $MAXCPUS, $SSD, $MSS, $MSSDIR, $MSSRPD, $MSSPWD, and $RPTDIR are input from the parent NQS script and are available for use, see subsection I.

Note that the example NQS script in subsection I makes use of separate and independent setup subscripts for the Coupler and each component model. While this is not required, it is a natural decomposition of code, and is highly recommended. Because the Coupler, or any other component model, has its own setup script, the persons responsible for this code can edit this file as necessary without being confused by extraneous details about other component models, and without the danger of inadvertently introducing errors into the top-level NQS run script or the setup scripts of the other component models.

An example Coupler setup script is given below:

```
#=======================================================================
# File name   cpl.setup.csh
# Purpose     prepares Flux Coupler (aka ``cpl") for execution
# Assumptions these env variables have been set by a parent shell:
#             CASE, CASESTR, RUNTYPE, ARCH,   MSGLIB, MAXCPUS, SSD,
#             MSS,  MSSDIR,  MSSRPD,  MSSPWD, RPTDIR
#=======================================================================
#-----------------------------------------------------------------------
# (a) Build an executable
#-----------------------------------------------------------------------
mkdir src
cd    src
set   SRCDIR = ~/csm/cpl4.0
cp -fp $SRCDIR/* .
cp -fp dims.h.T31_x3 dims.h
Makeprep
make EXEC=cpl4
cd ..
ln -s  src/cpl4 cpl
#-----------------------------------------------------------------------
# (b) document the source code used
#-----------------------------------------------------------------------
pwd ; ls -alF  src
grep CVS src/*.[hF]
#-----------------------------------------------------------------------
# (c) Create an input parameter namelist file
#-----------------------------------------------------------------------
if      ($RUNTYPE == 'initial' ) then
set rest_type   =  'initial'
set rest_bfile  =  '/unused/rest_bfile'
set rest_date   =  00010101
else if ($RUNTYPE == 'continue') then
set rest_type   =  'continue'
set rest_bfile  =  '/unused/rest_bfile'
set rest_date   =  -999
else if ($RUNTYPE == 'branch'  ) then
set rest_type   =  'branch'
set rest_bfile  =  '/KAUFF/csm/b003.01/cpl/r0010-01-01'
set rest_date   =  00010101
else if ($RUNTYPE == 'regen'   ) then
set rest_type   =  'regen'
set rest_bfile  =  "$MSSDIR/cpl/r1999-01-01"
set rest_date   =  19990101
else
echo  'unknown RUNTYPE = ' $RUNTYPE ; exit -1
endif
cat >!  cpl.parm << EOF
&cpl_parm
case_name   = '$CASE '
case_desc   = '$CASESTR '
rest_type   = '$rest_type '
rest_dir    = '$MSSDIR/cpl/ '
rest_pfile  = '$RPTDIR/cpl.$CASE.rpointer '
rest_bfile  = '$rest_bfile '
rest_date   =  $rest_date
rest_freq   = 'monthly'
stop_option = 'newyear'
hist_freq   = 'monthly'
/
EOF
cat cpl.parm
#=======================================================================
# End of Coupler setup shell script
#=======================================================================
```

Items (a) through (c) in the above Coupler setup script are now reviewed.

(a) Build an executable

The goal here is to build an executable Coupler binary in the current working directory. This is done by first identifying a source code directory, copying all files from that directory, selecting a resolution-dependent "dims.h" file, executing a script that creates the necessary include files needed by the makefile, and invoking a makefile. The dimensions of the component models (i.e. the number of x and y grid points) must be known to the Coupler at compile time and are all specified in one resolution-dependent file: dims.h. Notice how an appropriate dims.h file is selected. In general, several dims.h.* files may be available for building a Coupler binary suitable for various component model resolutions. The details of how to build the executable (eg. preprocessor options and compiler options) are contained in the standard Makefile. See the User's Guide subsection IV, or a make man page on any UNIX system for more information.

(b) Document the source code used

Here we make a detailed listing of the source code used, a list of revision control system (CVS) information, and list of the contents of the current working directory. This information can be used to identify what Coupler source code, input namelist, etc. were used in a particular simulation. This is not necessary but is strongly suggested.

(c) Create an input parameter namelist file

Here we are creating a Coupler input parameter namelist file. To create an appropriate namelist we must know and specify whether the run is an initial run, a continuation run, a branch run, or a regeneration run. Checking the value of the environment variable $RUNTYPE gives this information. See the User's Guide section on Coupler Input Parameters for a complete description of input namelist variables. The environment variables $CASE, $CASESTR, and $MSSDIR are used to construct a descriptive text string found in history files and to create history and restart data path names. Recall from the discussion in the User's Guide subsection I that these environment variables are set in the NQS script specifically for use by this, and other, setup scripts.

This particular input namelist was selected just to illustrate how the "cpl.setup.csh" script works. For a detailed discussion of Coupler input parameters, along with several example input namelists, see section J.

III. Preparing Component Models for a Coupled Run

Preparing the component models (atmosphere, ice, land, ocean) for a coupled run is analogous to preparing the Coupler, see subsection II. Such preparation is done in the component model setup scripts "atm.setup.csh," "ice.setup.csh," "lnd.setup.csh," and "ocn.setup.csh," which are called by the parent NQS batch job shell script described in subsection I. The purpose of the setup scripts is to build their respective executable codes, document what source code and data files are being used, and gather or create any necessary input data files.

Each component model has a separate subdirectory, $EXEDIR/atm, $EXEDIR/ocn, $EXEDIR/lnd, or $EXEDIR/ice, see subsection I, in which the setup script is run, and where the executable and all input and output files are kept. As in the Coupler setup script, the environment variables $CASE, $CASESTR, $RUNTYPE, $ARCH, $MSGLIB, $MAXCPUS, $SSD, $MSS, $MSSDIR, $MSSRPD, $MSSPWD, and $RPTDIR are set by the parent NQS script and are available for the setup scripts to use, see subsection I.

These environment variables are not intended to be accessed by the component model codes themselves. It is recommended that the executable codes use input namelist parameters instead of accessing environment variables.

If the UNICOS specific assign function is used prior to executing a code, the assigning of I/O units should be done here in the setup script. Setting environment variable FILENV = ./.assign in the top level run script, see subsection I, allows each component model to do it's own I/O unit assignments, using it's own .assign file, without conflicting with the I/O unit assignments of other models. See the assign man page on a UNICOS system for more details.

IV. Source Code Maintenance

The distribution FORTRAN source code for the Flux Coupler comes with a Makefile which uses the Cray f90 pre-processor, compiler, and loader to create object files and link them into an executable code. Makefiles tend to be very site-specific, so the user is expected to modify the Makefile as necessary if the Cray f90 system is not available.

The original source code was developed using the CVS revision control system, but only one "tagged" version of the Coupler is available within any source code distribution. This information can be used to identify the code contained in a particular distribution.

A user may wish to modify their copy of the Coupler source code. If one wishes to modify the Coupler source code, it is strongly recommended that one first study the Pseudo-Code section in this document in conjunction with studying the source code file "main.F." This should provide a good overview of how the Coupler works, a necessary prerequisite for successful code modification.

The Coupler source code is written almost entirely using standard FORTRAN 77. Perhaps the most notable exception is the use of the library calls "msread" and "mswrite" which rely on NCAR's site specific Mass Storage System for storing large output files. The use of the Coupler on non-NCAR computers will necessitate code modification of the source code to remove such site specific or machine specific code. Searching for preprocessor "if-defs," e.g. "#if (defined CRAY)", in the source code files should lead you to the code which is not standard FORTRAN.

V. Output Data

The Flux Coupler outputs three types of data:
- stdout outpt (ASCII text)
- restart files (machine specific binary)
- history files (netCDF, a common data format)

The Coupler's stdout, combined with stderr, is saved by the top level NQS batch job script to a directory specified by the the environment variable $LOGDIR, see subsection I. The Coupler's restart and history files are stored on NCAR's Mass Storage System (MSS). Restart files are in a machine dependent binary representation, whereas history files are in netCDF format. While most of the coupled run output data to be analyzed is assumed to be created by the various component models, the Coupler also creates output history files.

Stdout (standard out) output consists mostly of brief messages that indicate how the simulation is progressing and whether any error conditions have been detected. Stdout also contains a record of the values of all Coupler input parameters. If global diagnostics have been activated, by setting the appropriate Coupler input parameter, stdout will also contain some diagnostic information

Restart files are in a machine dependent binary representation. Because all fields from the restart file can be saved onto a history file, normally there is no need to examine the contents of a restart file. If users want to examine a restart file, they will have to look at the Coupler source code to see how the file was written, and write their own program to read the file and examine its contents.

NetCDF was chosen as the history data format because many common visualization tools already accept this format as input, thus facilitating the acquisition of a visualization or analysis utility to view Coupler history files. NetCDF (network Common Data Form) is an interface for array-oriented data access and a library that provides an implementation of the interface. The netCDF library also defines a machine independent format for representing scientific data. Together the interface, library, and format support the creation, access, and sharing of scientific data. The netCDF software was developed at the Unidata Program Center in Boulder, Colorado. The freely available source can be obtained by anonymous FTP from ftp://ftp.unidata.ucar.edu/pub/netcdf/ or from other mirror sites.

Because netCDF files are self-describing, the most complete and accurate description of the contents of a Coupler history file (or any netCDF file) will always come from the netCDF data file itself. The netCDF tool "ncdump" will generate the CDL text representation of a netCDF file on standard output, optionally excluding some or all of the variable data in the output.

Three types of data are found in Coupler netCDF history data files: global attributes, domain data, and two dimensional state and flux data.

(1) Global attributes

This includes the case name corresponding to the history data and the date the data was created.

(2) Model domain data

This includes the coordinates of the grid cells of all model domains, as well as a domain mask for each surface model. Each model has two sets of strictly increasing latitude and longitude coordinates, one corresponding to grid cell centers, xc(i) and yc(j), and one corresponding to grid cell edges, xe(i) and ye(j). A state variable S(i,j) is understood to be a point value located at (xc(i),yc(j)), which is located within a grid cell bounded by latitude coordinates xe(i) and xe(i+1) and longitude coordinates ye(j) and ye(j+1). A flux field F(i,j) can be thought of as a point value located at (xc(i),yc(j)), but more accurately it is an area average value that applies uniformly over the grid cell containing that point.

Four sets of coordinate arrays are found in the history data:

- xc_a(i), yc_a(j), xe_a(i), and ye_a(j) are the center and edge coordinates for the atmosphere model grid.
- xc_i(i), yc_i(j), xe_i(i), and ye_i(j) are the center and edge coordinates for the sea-ice model grid.
- xc_l(i), yc_l(j), xe_l(i), and ye_l(j) are the center and edge coordinates for the land model grid.
- xc_o(i), yc_o(j), xe_o(i), and ye_o(j) are the center and edge coordinates for the ocean model grid.

Each surface model (land, sea-ice, ocean) also has a corresponding domain mask. The domain mask is an integer array such that if mask(i,j) $\neq 0$, then the indices (i,j) correspond to a grid cell that is in the active model domain, i.e., S(i,j) is a valid model state and F(i,j) is a valid flux value. Conversely, if mask(i,j) $= 0$, then S(i,j) and F(i,j) are undefined.

Three masks are found in the history data:

- mask_l(i,j) is the land model domain mask.
- mask_i(i,j) is the ice model domain mask.
- mask_o(i,j) is the ocean model domain mask.

There is no atmosphere domain mask because all atmosphere data points are assumed to be valid points within the atmosphere domain.

(3) Two dimensional state and flux data

This includes model state variables, component flux fields, and merged input flux fields. The naming convention for two dimensional variables follows the convention introduced in section D of this document. Some examples of state variable names are:

- Sa_a_t is an atmosphere state variable on the atmosphere grid, and the variable is temperature.
- Sa_a_u is an atmosphere state variable on the atmosphere grid, and the variable is zonal velocity.
- Sa_o_u is an atmosphere state variable on the ocean grid, and the variable is zonal velocity.
- So_o_t is an ocean state variable on the ocean grid, and the variable is temperature.

Some examples of flux field variable names are:

- Faoa_a_prec is an atmosphere/ocean flux field computed by the atmosphere on the atmosphere grid, and the field is precipitation.
- Faoa_o_prec is an atmosphere/ocean flux field computed by the atmosphere on the ocean grid, and the field is precipitation.
- Faoc_o_taux is an atmosphere/ocean flux field computed by the Coupler on the ocean grid, and the field is zonal wind stress.

- Faoc_a_taux is an atmosphere/ocean flux field computed by the Coupler on the atmosphere grid and the field is zonal wind stress.

Each variable in the netCDF history file has long_name and units attributes which further describe the variable. See section D for a more complete description of this variable naming convention, and section C for a more complete description of these variables.