I. User's Guide to the Flux Coupler

The following subsections describe how to use the Flux Coupler on one of NCAR's Cray Y-MP 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. Understanding what this script does will give a good basic understanding of how the entire coupled system works. The basic idea of running a coupled model is quite simple: first a message passing daemon is started (a daemon that handles the interprocess communication) and then the Flux Coupler and all component models are started sequentially as background processes. Upon startup, the coupler and the component models first connect themselves to the message passing system and then data can begin 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 will start the message passing daemon, the Flux Coupler, and the component models appropriately. 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 four separate subscripts called "setup scripts." These subscripts, "drv.setup.csh," "atm.setup.csh," "ice.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 an example nqs script for running the CSM coupled model
#=======================================================================

#-----------------------------------------------------------------------
# (a) Set NQS options
#-----------------------------------------------------------------------
# QSUB -q  reg                          # select a batch queue
# QSUB -lT 99:10:00 -lt  60:00:00       # set CPU time limits
# QSUB -lM 41Mw  -lm 31Mw  -lQ 14Mw     # set memory limits
# QSUB -eo                              # combine stdout & stderr
# QSUB -s  /bin/csh                     # requested shell
# QSUB                                  # no more QSUB options

#-----------------------------------------------------------------------
# (b) Set env variables available to model setup scripts (below):
#     CASE, CASESTR, RUNTYPE, SSD, MAXCPUS, ROOTDIR, MSGLIB
#-----------------------------------------------------------------------
setenv CASE     b001.02               # case name
setenv CASESTR 'CSM coupled model'    # short descriptive text string
setenv ROOTDIR  '/DOE/csm'            # MSS directory path name
setenv RUNTYPE  initial               # run type
setenv MSGLIB   MCL1.0                # msg passing library
setenv SSD      TRUE                  # SSD available
setenv MAXCPUS  8                     # number of CPUs available

#-----------------------------------------------------------------------
# (c) Specify input, output, & execution directories
#     o the component model setup.csh scripts must be in $NQSDIR
#     o create execution directories:  $EXEDIR/[atm|drv|ice|ocn]
#     o stdout & stderr output is saved in $LOGDIR
#-----------------------------------------------------------------------
set EXEDIR = /usr/tmp/doe/$CASE               # model executes here
set NQSDIR = /crestone/u1/doe/csm/nqs/$CASE   # runs scripts are here
set LOGDIR = $NQSDIR                          # stdoutput goes here

mkdir -p $EXEDIR                      # create the execution directory
mkdir $EXEDIR/drv                     # drv executes in this directory
mkdir $EXEDIR/atm                     # atm executes in this directory
mkdir $EXEDIR/ocn                     # ocn executes in this directory
mkdir $EXEDIR/ice                     # ice executes in this directory

#-----------------------------------------------------------------------
# (d) Start the message passing daemon
#-----------------------------------------------------------------------
if ($MSGLIB == "MCL1.0" ) then
   setenv PVM_ROOT /usr/local/pvm3
   setenv PVM_ARCH CRAY
   if (`ps -ef | grep pvmd3 | grep $LOGNAME | wc -l` < 1) then
      set USERID = `id -u`
      rm /tmp/pvm*.$USERID
      cd $EXEDIR
      ln -s /usr/bin/pvmgs .
      pvmd3 &
      echo 'Started new PVM3 daemon'
   else
      echo 'A PVM3 daemon already running'
   endif
else
   echo 'unknown MSGLIB:' $MSGLIB
   exit -1
endif

#-----------------------------------------------------------------------
# (e) Prepare component models for execution
#     o the component model setup.csh scripts must be in $NQSDIR
#     o these setup scripts have access to env variables set above
#     o see assign man page for details of Cray specific assign function
#-----------------------------------------------------------------------
setenv FILENV ./.assign   # allow separate .assign files for each model
set PID = $$
cd $EXEDIR/drv ; $NQSDIR/drv.setup.csh  >&!  drv.log.$PID
cd $EXEDIR/atm ; $NQSDIR/atm.setup.csh  >&!  atm.log.$PID
cd $EXEDIR/ice ; $NQSDIR/ice.setup.csh  >&!  ice.log.$PID
cd $EXEDIR/ocn ; $NQSDIR/ocn.setup.csh  >&!  ocn.log.$PID
```

```
#-----------------------------------------------------------------------
# (f) Execute component models simultaneously (allocating CPUs & SSD)
#-----------------------------------------------------------------------
cd $EXEDIR
cat >!  execute.sh << EOF
#!/bin/csh -xvf
   cd $EXEDIR/atm
   env NCPUS=$MAXCPUS SDSLIMIT=28000  atm < atm.parm >>&!  atm.log.$PID &
   #                  SSD:    (28000 blocks)*(512w/block)=14.34Mw
   cd $EXEDIR/ocn
   env NCPUS=$MAXCPUS SDSLIMIT=00000  ocn < ocn.parm >>&!  ocn.log.$PID &
   cd $EXEDIR/ice
   env NCPUS=2        SDSLIMIT=00000  ice < ice.parm >>&!  ice.log.$PID &
   cd $EXEDIR/drv
   env NCPUS=$MAXCPUS SDSLIMIT=00000  drv < drv.parm >>&!  drv.log.$PID &
   wait
EOF
chmod u+x execute.sh

ja $TMPDIR/jacct         # start CRAY job accounting utility
execute.sh               # execute all component models
ja -tsclh $TMPDIR/jacct  # stop  CRAY job accounting utility

#-----------------------------------------------------------------------
# (g) save model output (stdout & stderr) to $LOGDIR
#-----------------------------------------------------------------------
gzip -v */*.log.$PID            # compress stdout output files
cp   -p */*.log.$PID* $LOGDIR   # save a copy of the output files

#=======================================================================
# end of example nqs shell script
#=======================================================================
```

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

(a) Set NQS options

The Network Queuing System (NQS) is a special facility available only 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 "drv.setup.csh," "atm.setup.csh," "ice.setup.csh," and "ocn.setup.csh." Seven environment variables are set in the NQS script and 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 models 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 the User's Guide section on the coupler setup script in subsection *II.*

- *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.*

- *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.

- *MAXCPUS*: an integer specifying the maximum number of CPUs each component can use for the coupled run.

- *ROOTDIR*: a Mass Storage System (MSS) directory path name. Component models should send their MSS data to a subdirectory of this directory. Generally $ROOTDIR will be something like "/DOE/csm/$CASE/" and models will create their own history 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.*

- *MSGLIB*: the message passing library being used. In general, component models 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. For an example, see subsection *II.*

(c) Specify input, output, and execution directories

Here we specify the directory where the model setup scripts are found (these are input shell script files used to create the model executables), where stdout and stderr output data is saved when the simulation finishes, and create the directory structure used during model execution.

- *EXEDIR*: a parent directory where the coupled system is run. Four subdirectories, atm, drv, ice, and ocn are created beneath $EXEDIR. All executables and all their related data files are located in their respective subdirectories.

- *NQSDIR*: this is where the coupler and component model setup scripts "drv.setup.csh," "atm.setup.csh," "ice.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) Start message passing daemon

In general, the coupled system may be able to use a variety of underlying message passing systems. Currently only one message passing library is available, the MCL 1.0 library developed in NCAR's Scientific Computing Division (SCD). This library is built upon and requires PVM3. PVM3, in turn, requires a daemon. This part of the NQS script checks to see if a PVM3 daemon is already running: if not, a daemon is started. Alternate message passing libraries may not require a daemon.

(e) Prepare component models for execution

Here the coupler and component model setup scripts "drv.setup.csh," "atm.setup.csh," "ice.setup.csh," and "ocn.setup.csh" are invoked. The purpose of these setup scripts is to build their respective executable codes, gather any necessary input data files, and do whatever bookkeeping is necessary to document exactly what executable code and data files are being used. It is highly 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.

(f) Execute component models simultaneously

In section (e), 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 us to specify different numbers of CPUs and different amounts of SSD to the different executable codes. 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.

(g) 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 "drv.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/drv, 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, $ROOTDIR, $RUNTYPE, $MSGLIB, $SSD, and $MAXCPUS 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    drv.setup.csh
# Purpose      build an executable drv (aka "driver" or "Flux Coupler")
# Assumptions  these env variables have been set by a parent shell:
#              CASE, CASESTR, RUNTYPE, MAXCPUS, SSD, MSDIR, MSGLIB
#===============================================================================

#-------------------------------------------------------------------------------
# (a) Build an executable (executable is called "drv")
#-------------------------------------------------------------------------------
set SRCDIR = /crestone/u1/nieman/csm/drv-3.a  # source code directory
make -f  $SRCDIR/Makefile.$MSGLIB             # make an executable

#-------------------------------------------------------------------------------
# (b) Create an input parameter namelist file
#-------------------------------------------------------------------------------
if      ($RUNTYPE == 'initial' ) then
   set rest_type   =  'initial'
   set rest_bfile  =  '/invalid/rest_bfile'
   set rest_date   =  00000831
else if ($RUNTYPE == 'continue') then
   set rest_type   =  'continue'
   set rest_bfile  =  '/invalid/rest_bfile'
   set rest_date   =  -999
else if ($RUNTYPE == 'branch'  ) then
   set rest_type   =  'branch'
   set rest_bfile  =  '/DOE/csm/b001.01/drv/r0002-01-01'
   set rest_date   =                          00020101
else if ($RUNTYPE == 'regen'   ) then
   set rest_type   =  'regen'
   set rest_bfile  =  "$ROOTDIR/$CASE/drv/r1999-01-01"
   set rest_date   =                        19990101
else
   echo  'unknown RUNTYPE = ' $RUNTYPE ; exit -1
endif

cat >!  drv.parm << EOF
E$drv_parm
   rest_str    = '$CASE $CASESTR '
   rest_dir    = '$ROOTDIR/drv/ '
   rest_type   = '$rest_type '
```

```
   rest_bfile  = '$rest_bfile '
   rest_date   =  $rest_date
   rest_freq   = 'monthly'
   stop_option = 'newyear'
   hist_freq   = 'monthly'
   msg_group   = '$CASE'
   $
EOF

cat drv.parm

#-------------------------------------------------------------------------
# (c) List source code, SCCS information, and directory contents
#-------------------------------------------------------------------------
ls -lF $SRCDIR/Makefile.$MSGLIB
ls -lF $SRCDIR/*.[fh]

grep -i SCCS $SRCDIR/Makefile.$MSGLIB | egrep -i '(module|edit)'
grep -i SCCS $SRCDIR/*.[fh]           | egrep -i '(module|edit)'

pwd
ls -alFt

#=========================================================================
# End of example drv.setup.csh
#=========================================================================
```

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 quite simply by first identifying the source code directory, and then invoking a Makefile residing in that directory. Notice how the environment variable $MSGLIB is used to select the appropriate Makefile with respect to the message passing library being used. In general, several Makefiles may be available for building a coupler that uses one of several different message passing libraries. Recall from the discussion in the User's Guide subsection *I* that the variable $MSGLIB is one of several environment variables set in the top level NQS script specifically for use by this, and other, setup scripts. The details of how to build the executable 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) 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 $ROOTDIR are used to construct a descriptive text string found in history files, an output data path name, and a message passing group name. 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 "drv.setup.csh" script works. For a detailed discussion of coupler input parameters, along with several example input namelists, see section J.

(c) List source code and directory contents

Here we make a detailed listing of the source code used, a list of Source Code Control System (SCCS) 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.

III. Preparing Component Models for a Coupled Run

Preparing the component models (atmosphere/land, ocean, sea–ice) 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," and "ocn.setup.csh," which are called by the parent NQS batch job shell script described in subsection *I*. The purpose of these setup scripts is to build their respective executable codes, gather any necessary input data files, and do whatever is necessary to document what executable code and input data files are being used.

Each component model has a separate subdirectory, $EXEDIR/atm, $EXEDIR/ocn, 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, $ROOTDIR, $RUNTYPE, $MSGLIB, $SSD, $MAXCPUS are input from the parent NQS script and are available for use, see subsection *I*. While the setup scripts may be dependent on these environment variables, it is not recommended that the executable codes be dependent on these 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, when invoked, will use the Cray compiler (cf77) and the Cray loader (segldr) to create object files and link them into an executable code. While a pre–processor is not required to make object files, the Cray compiler will pre–process certain "CMIC$" compiler directives that will parallelize various code segments.

The original source code was developed using the Source Code Control System (SCCS), but only one checked out version of each source code file is available with the Flux Coupler distribution. This information can be used to identify which version of the coupler code is contained in any given distribution.

A user may wish to modify their copy of the coupler source code at will. If one wishes to modify the coupler source code, it is strongly recommended that one first study

the section, Pseudo–Code for the Flux Coupler, in this document in conjunction with studying the source code file called "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 the text string "MACHINE DEPENDENCY" in the source code files should lead you to all the code which is not standard FORTRAN 77.

The source code Makefile is called "Makefile.MCL1.0" which, when invoked "make -f Makefile.MCL1.0," will use the Cray compiler and the loader to make object code and an executable binary. The Makefile suffix refers to the message passing library that will be used by the executable code. The particular message passing library used has no effect on any numerical computation. Five source code files also contain an MCL1.0 suffix. Only these files need be altered or replaced if one wishes to use an alternate message passing library.

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 history data files.

(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.
- *Faod_o_taux* is an atmosphere/ocean flux field computed by the coupler on the ocean grid, and the field is zonal wind stress.
- *Faod_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.