Parallel Ocean Program (POP) User Guide

Version 2.0
Los Alamos National Laboratory
LACC 99-18

23 March 2003

Abstract

This manual contains detailed instructions for operating the POP code. It should be the only manual needed to use POP "as is". Topics include:

- How to compile POP, including compile-time options
- How to run POP, including run-time options in *namelist* input
- Procedures for preparing auxiliary input files that are needed if you are setting up a new grid
- Options for model diagnostics
- Options for model output files and formats

Throughout this manual, it is assumed that the operating system is some variant of Unix. However, POP has been run on PCs under windows (see Sec. 6.1 for details)..

To keep the *User's Guide* as short as possible, a *Reference Manual* is provided for those users who seek an in-depth understanding of the code, such as would be needed to make changes to the source code. It contains a thorough description of the code, including the model equations, their discretized form, and the numerical methods used for their solution. The principal modules and subroutines are explained, the organization of the code is outlined, and the principal variables are defined.

Whenever POP is run, the version number and release date of the source code on your local system is output at the beginning of the *log file.* They should agree with the information on the front cover of this manual. If they do not, you can obtain the current version of this manual at
http://climate.acl.lanl.gov/models/pop/documentation.

The Manual revision letter and release date indicate corrections or additions to the manual corresponding to the stated version of the model.

If you obtained POP directly from LANL, technical support is provided by John Davis (jfd@lanl.gov, 505-667-4793) and Phil Jones (pwjones@lanl.gov , 505-667-6386). If you are using POP as a component of the NCAR Climate System Model (CCSM), technical support is provided by NCAR.

We are always interested in learning about your experiences with POP, favorable or otherwise. Send your comments and suggestions for improvements to pop_feedback@acl.lanl.gov. Bug reports should be sent to pop_bugs@acl.lanl.gov.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Here we present a bit of POP history so you may marvel at the once-revolutionary advances and rest secure in the knowledge that we are working diligently (and sometimes even successfully) to continue such advances in the future. If you are bored with such historical ramblings or simply wish to rush into actually running ocean simulations, feel free to skip over this section and perhaps read it later over a nice cappuccino, or a glass of port beside the fire, or maybe a beer at your favorite brew pub.

1.1 Brief history of POP development

The Parallel Ocean Program (POP) was developed at LANL under the sponsorship of the Department of Energy's CHAMMP program, which brought massively parallel computers to the realm of climate modeling. POP is a descendent of the Bryan-Cox-Semtner class of ocean models first developed by Kirk Bryan and Michael Cox [4] at the NOAA Geophysical Fluid Dynamics Laboratory in Princeton, NJ in the late 1960s. POP had its origins in a version of the model developed by Semtner and Chervin [18] [5]. The complete "family tree" of the BCS models is displayed in Figure 1.1 (courtesy of Bert Semtner, Naval Postgraduate School).

Under the CHAMMP program, the Semtner-Chervin version was rewritten in CM Fortran for the Connection Machine CM-2 and CM-5 massively parallel computers. Experience with the resulting model led to a number of changes resulting in what is now known as the Parallel Ocean Program (POP). Although originally motivated by the adaptation of POP for massively parallel computers, many of these changes improved not only its computational performance but the fidelity of the models physical representation of the real ocean. The most

Figure 1.1: Bryan-Cox-Semtner model family tree
Figure currently exists in on-line version only.

significant of these improvements are summarized below. Details can be found in articles by Smith, [19], Dukowicz et al., [7], and Dukowicz and Smith [6]. The model has continued to develop to adapt to new machines, incorporate new numerical algorithms and introduce new physical parameterizations.

1.2 Improvements introduced in POP

1.2.1 Surface-pressure formulation of barotropic mode

The barotropic streamfunction formulation in the standard BCS models required an additional equation to be solved for each continent and island that penetrated the ocean surface. This was costly even on machines like Cray parallel-vector-processor computers, which had fast memory access. To reduce the number of equations to solve with the barotropic streamfunction formulation, it was common practice to submerge islands, connect them to nearby continents with artificial land bridges, or merge an island chain into a single mass without gaps.The first modification created artificial gaps, permitting increased flow, while the latter two closed channels that should exist.

On distributed-memory parallel computers, these added equations were even more costly because each required gathering data from an arbitrarily large set of processors to perform a line-integral around each landmass. This computational dilemma was addressed by developing a new formulation of the barotropic mode based on surface pressure. The boundary condition for the surface pressure at a land-ocean interface point is local, which eliminates the non-local line-integral.

Consequently, the surface-pressure formulation permits any number of islands to be included at no additional computational cost, so all channels can be treated as precisely as the resolution of the grid permits.

Another problem with the barotropic streamfunction formulation is that the elliptic problem to be solved is ill-conditioned if bottom topography has large spatial gradients. The bottom topography must be smoothed to maintain numerical stability. Although this reduces the fidelity of the simulation, it does have the desirable side effect (given the other limitations of the streamfunction approach mentioned above) of submerging many islands, thereby reducing the number of equations to be solved. In contrast, the surface-pressure formulation allows more realistic, unsmoothed bottom topography to be used with no reduction in time step.

1.2.2 Free-surface boundary condition

The original "rigid-lid" boundary condition was replaced by an implicit free-surface boundary condition that allows the air-sea interface to evolve freely and makes sea-surface height a prognostic variable.

1.2.3 Latitudinal scaling of horizontal diffusion

Scaling of the horizontal diffusion coefficient by $\cos^n(\theta)$ was introduced, where θ is latitude, $n = 1$ for Laplacian mixing and $n = 3$ for bi-harmonic mixing. This *optional* scaling prevents horizontal diffusion from limiting the time step severely at high latitudes, yet keeps diffusion large enough to maintain numerical stability.

1.2.4 Pressure-averaging

After the temperature and salinity have been updated to time-step $n+1$ in the baroclinic routines, the density ρ^{n+1} and pressure p^{n+1} can be computed. By computing the pressure gradient with a linear combination of p at three time-levels ($n-1$, n, and $n+1$), a technique well known in atmospheric modeling [3], it is possible to increase the time-step by as much as a factor of two.

1.2.5 Designed for parallel computers

The code is written in Fortran90 and can be run on a variety of parallel and serial computer architectures. Originally, the code was written using a data-parallel approach for the Thinking Machines Connection Machine. Later versions used a more traditional domain decomposition style using MPI or SHMEM for inter-processor communications. The most recent version of the code supports current clusters of shared-memory multi-processor nodes through the use of thread-based parallism (OpenMP) between processors on a node and message-passing (MPI or SHMEM) for communication between nodes. The flexibility of mixing thread-based and message-passing programming models gives the user the option of choosing the best combination of styles to suit a given machine.

1.2.6 General orthogonal coordinates in horizontal

The primitive equations were reformulated and discretized to allow the use of any locally orthogonal horizontal grid. This provides alternatives to the standard latitude-longitude grid with its singularity at the North Pole.

This generalization made possible the development of the displaced-pole grid (Fig.1.2.6), which moves the singularity arising from convergence of meridians at the North Pole into an adjacent landmass such as North America, Russia or Greenland. Such a displaced pole leaves a smooth, singularity-free grid in the Arctic Ocean. That grid joins smoothly at the equator with a standard Mercator grid in the Southern Hemisphere. The most recent versions of the code also support a tripole grid (Fig.1.2.6) in which two poles can be placed opposite each other in land masses near the North Pole to give more uniform grid spacing in the Arctic Ocean while maintaining all the advantages of the displaced pole grids.

Figure 1.2: Displaced-pole grid
Figure currently exists in on-line version only.

Figure 1.3: Tripole grid
Figure currently exists in on-line version only.

1.3 POP applications to date

1.3.1 High-resolution global and regional modeling

In the period 1994-97, POP was used to perform high resolution (0.28° at the Equator) global ocean simulations, running on the Thinking Machines CM5 computer then located at LANL's Advanced Computing Laboratory. (Output from these runs is available at http://climate.acl.lanl.gov. The primary motivation for performing such high-resolution simulations is to resolve mesoscale eddies that play an important role in the dynamics of the ocean. Comparison of sea-surface height variability measured by the TOPEX/Poseidon satellite with that simulated by POP gave convincing evidence that still higher resolution was required (Fu and Smith [9]; Maltrud [15]).

At the time, it was not possible to do a higher resolution calculation on the global scale, so an Atlantic Ocean simulation was done with 0.1° resolution at the Equator. This calculation agreed well with observations of sea-surface height variability in the Gulf Stream. Many other features of the flow were also well simulated [20]. Using the 0.1° case as a benchmark, lower resolution cases were done at 0.2° and 0.4°; a comparison can be found at http://neit.cgd.ucar.edu/oce/bryan/woce-poster.html.

Recently, it has become possible to begin a 0.1° global simulation and such a simulation has been started. In addition, higher resolution North Atlantic simulations have also been initiated.

1.3.2 Coupled models

POP and the Los Alamos sea-ice model (CICE) have been adopted as the ocean and sea ice components of the Community Climate System Model (CCSM) at NCAR. POP and CICE are also being used in coupled model development efforts at Colorado State University and UCLA. Information on CICE can be found at http://climate.acl.lanl.gov/models/cice.

Chapter 2

Installing and Building POP

Now that you've presumably perused the first chapter describing all the great features of POP, you're understandably anxious to leap in and run some simulations yourself. The general procedure for unpacking and building a POP executable will be described here. Exceptions to this procedure for some architectures will be noted.

2.1 Supported architectures

POP uses standard Fortran and will work on any machine with a compliant Fortran compiler (Fortran here denotes Fortran 90 or later – FORTRAN 77, FORTRAN 66, FORTRAN IV or other antiquated dialects are only supported to the extent that F90 is backward compatible). Two levels of parallelism are supported and combinations of these two levels can be used on architectures which support them. For shared-memory parallelism, OpenMP can be used. For distributed-memory parallelism, MPI or SHMEM can be used. In clustered SMP architectures, OpenMP can be used for multiple processors on a node while MPI can be used between nodes.

2.2 Obtaining the POP code

If you have not done so already, you must first actually download a version of the POP code, maybe even the latest version. The preferred place to get the POP code is our main web site at Los Alamos:
http://climate.acl.lanl.gov/models/pop
from which you probably downloaded or are viewing this manual.

If you are running POP as part of the NCAR Climate System Model (CCSM), you should download POP as part of the CCSM distribution.

2.3 Creating a run directory

The POP distribution you obtained is probably in the form of a compressed tar file. You must first uncompress the tar file using

uncompress popXXXX.tar

where XXXX refers to the version number which appears in the file name. Then the tar archive must be unpacked using

tar -xvf popXXXX.tar.

This process will result in a directory named pop with several subdirectories containing source code, templates for various input files, utilities and test codes.

In order to build a version of POP to run, a directory with the appropriate makefiles and input files must be created. In the main POP directory, a script has been provided to create all the necessary structure. Typing

setup_run_dir dirname [model]

will create a directory called *dirname* with all the necessary makefiles and input files. A sub-directory called *compile* will also be created to provide a work area for the compilation process. The optional argument *model* will copy files that are specific to a standard resolution or model setup. One example of such a setup is the *test* case (see Sec. 2.7 which should be used to test the code for the first time.

2.4 Building POP

2.4.1 Make Procedure

The POP *make* procedure consists of several steps that are governed by options in an architecture-specific *archdir.gnu* file. The correct *.gnu* file is chosen based on an environment variable called ARCHDIR, which is a combination of vendor name and communication paradigm. For example, if you are compiling for an SGI Origin you will have the choice of *sgi_serial*, *sgi_mpi*, *sgi_shmem*, *sgi_omp*, or *sgi_mpiomp* depending on whether you want to run serially, in parallel using message-passing (MPI or SHMEM), in parallel using thread-based parallelism (OpenMP) or in parallel using a hybrid of threads and message-passing. The *archdir.gnu* file contains the proper paths and compiler options for the particular architecture chosen. A variety of these *.gnu* files have been provided with the standard distribution in the *input_templates* directory. If a *archdir.gnu* file does not exist for your choice, a file *generic.gnu* exists with comments on how to configure it for your particular machine and environment. Also, if your site has an unusual setup for locations of compilers and various other tools like netCDF, you may have to edit the *archdir.gnu* file to reflect the different setup.

POP version 2.0 or later requires version 3.5 or later of the netCDF library. If not already installed on your system, the netCDF package can be obtained free of charge from

http://my.unidata.ucar.edu/content/software/netcdf/index.html

Once the ARCHDIR environment variable has been set, typing *gmake* should start the *make* process. The *make* process includes a step which runs scripts to generate dependencies for the *makefile*. After the dependencies are generated, the source code passes through a preprocessor and the resulting routines are finally compiled into an executable named **pop**.

If you wish to compile a version of POP suitable for debugging or wish to turn all optimizations off, typing

gmake OPTIMIZE=no

will create an exectutable named **pop_db** for this purpose.

2.5 Domain decomposition

In order to understand some aspects of compiling and running POP, a few words must be said here about how POP breaks up a problem to run on different threads and processors. Note that even the serial versions decompose the domain in order to achieve better performance on cache-based microprocessors.

In POP, the full horizontal domain size *(nx_global,ny_global)* is broken up into domains or blocks. The size of these blocks can be chosen to achieve better performance as described below. Any block size can be chosen, but to avoid padding the domain with extra points, the block size in each direction should be chosen such that it divides the global domain size in that direction evenly.

Once the domain has been decomposed into blocks, the blocks are distributed among the processors or nodes, ignoring blocks that only contain land points. The distribution of blocks across processors or nodes can be performed using either a load-balanced distribution to try to give all processors an equal amount of work or a Cartesian distribution which ensures that the block's north, south, east and west neighbors remain nearest neighbors. A load-balanced distribution is generally better for the baroclinic section of the code; a Cartesian distribution is better for the barotropic solver. Different distributions can be specified for the baroclinic and barotropic parts of the code.

Such a domain decomposition allows some flexibility in tuning the model for the best performance. Generally, a smaller block size will improve processor performance on cache-based microprocessors and a smaller block size should ensure a better load balance and better land point elimination. However, smaller block sizes add complexity to the communication routines (boundary updates, global reductions) and will result in a performance penalty for the barotropic solver. The user will need to experiment with a few combinations to find the best configuration for the simulation being run.

2.6 Compile-time options

There are a few options for POP that must be determined at compile time. Some of these options are set by editing modules; one option requires a preprocessor directive which is handled by a C-language preprocessor (*cpp*) or equivalent (if the Fortran compiler understands such directives). The options below are the *only* options that need to be decided at compile time; all other options are set at run time through a namelist input file.

2.6.1 Domain

The full model size must be set in the domain_size module (in the file *domain_size.F90* located in the run directory). The horizontal extents of the grid *nx_global* and *ny_global* are defined here as well as the number of vertical levels *km*. The number of tracers *nt* also must be defined here and must be at least two to handle potential temperature and salinity which are always assumed to be tracer number one and two, respectively.

2.6.2 Blocks

The size of the blocks for which the domain is decomposed (see Sec. 2.5) is set by two parameters in the domain_size module (contained in the file *domain_size.F90* in the run directory). The parameter *block_size_x* determines the number of physical grid points in the first horizontal dimension; the parameter *block_size_y* determines the number of physical grid points in the (wait for it...) second horizontal dimension. The two parameters *max_blocks_clinic* and *max_blocks_tropic* determine the maximum number of blocks that can be distributed onto a processor or node. An initial guess for these two parameters can be made by dividing the total number of blocks by the number of processors you plan to run on

$$\frac{(nx\_global/block\_size\_x)(ny\_global/block\_size\_y)}{nprocs}. \tag{2.1}$$

Only an initial guess is required; when running the code, the actual numbers required will be output so that the user can set these parameters correctly. Also, note that these parameters can be set higher than required with no penalty other than memory use. For example, the parameters can be set for the lowest processor count you plan to use and then the same executable can be used to run at higher processor counts with no change other than the namelist inputs for number of processors.

As mentioned in previous sections, finding the optimal block size and distribution can require experimentation. However, a starting point for users who are familiar with previous versions of POP is to set the block size such that only one block is distributed on each processor. In this case, *block_size_x* = *nx_global/NPROCX* where *NPROCX* is the value found in the old POP *.gnu* files; values for the y direction are computed analogously. The *max_blocks* parameters are then set to one. After using this configuration, users can experiment with reducing the block size to improve performance.

2.6.3 Coupled model runs

If POP is run in coupled mode, the default interface communicates with the NCAR CCSM Flux Coupler. This requires message-passing calls and linking with additional libraries (including MPI whether MPI is used internally or not). To enable this capability, the coupled option must be requested on the makefile command line:

gmake COUPLED=yes.

This option turns on the COUPLED ifdef flag for the preprocessor so that the code necessary for model coupling is included during the preprocessing phase.

2.6.4 Debugging

Similar to the coupling option, if you wish to create a non-optimized pop executable for use with a debugging tool, you must specify this on the make command line:

gmake OPTIMIZE=no.

By default, this will create an executable named **pop_db** rather than the usual **pop**.

2.7 Testing POP

The POP distribution includes a simple test case that can be used for a variety of purposes, including validation, performance tuning and benchmarking. The key point is that there are no input fields: the model grid, topography, initial state, equation of state coefficients and wind stress (there is no other forcing enabled) are all generated internally. The only file that is read is *pop_in*.

To run the test problem, type

./setup_run_dir test test

and a directory called *test* will be created that contains all of the appropriate files. The default model size is 192x128x20 grid points, though this can be changed arbitrarily by the user. The grid generated internally is an equally-spaced latitude-longitude global grid with idealized landmasses. Make the executable as described in Sec. 2.4. The *pop_in* file defaults to running 20 steps with full diagnostics output every step. Note that for performance benchmarks, the diagnostic frequency should be set to 'never' as the global diagnostics are expensive and typically requested relatively infrequently (e.g. every 10 days) in a production simulation.

To use this test case for validation, the user can compare their ouput with the file *pop/input_templates/pop_sgi.log.test* which contains the output of a 20 step calculation run on 4 processors of an SGI Origin3000. The results from other computer platforms should agree reasonably well – within roundoff for the first step for most of the larger fields (the nearly-zero fields are very sensitive and my not agree to that level of precision).

Once the answers have been validated using the 192x128x20 grid, performance and scaling can be investigated by varying the grid size (in *domain_size.F90*) and the number of processors (in the pop_in file).

Chapter 3

Running POP

This chapter begins with the assumption that the user has followed the instructions in the previous chapter and has successfully built a **pop** executable. A second assumption is that the user might want to actually run an ocean simulation with the code, thereby impressing friends and colleagues and becoming a card-carrying member of the POP user's club. Given these assumptions, the following describes all the many options for configuring an ocean simulation with POP.

POP requires some input data to run correctly. The first requirement is a file called *pop_in* that contains many namelists that determine options and parameter values for the run. A sample input file should have been created in the run directory, but can also be found in the *input_templates* directory.

In addition to the namelist input file, files will be required to initialize grids, preconditioners, fields for output and possibly other options. These will be discussed in more detail below and in later chapters.

The namelists and the model features that they control are presented below. Each namelist appears in a table whose columns contain:

- The name of the namelist, followed by the names of all parameters.

- The range (discrete or continuous) of possible values of the parameter. Square brackets surround the [default value] and numerical ranges will be denoted by (min,max) where the limits are inclusive.

- The units, if applicable, and a description of the parameter.

The namelists are presented in approximately the order in which they are called during the initialization of a POP run. However, some regrouping has been done to bring related topics together. The actual order of the namelists within the *pop_in* file can be arbitrary; the code will search the file for the proper namelist to be read.

3.1 Operational control

This section describes various input options that control the operation control (rather than physical parameters) of the model, including processor configuration, time management and some input/output control.

3.1.1 Processor configuration

The first namelist read by the POP model determines how many processors it should use and how to distribute blocks of the domain across processors (see Sec.2.5).

The number of processors used by the model is governed by the `nprocs_clinic` parameter. The parameter `nprocs_tropic` determines whether the barotropic solver is run using the same or fewer number of processors; specifying a number larger than the clinic value will result in an error.

Table 3.1: Domain namelist

&domain_nml		**options for controlling domain decomposition**
nprocs_clinic	(1,machine maximum)	number of processors to be used for most of the code
nprocs_tropic	(1,nprocs_clinic)	number of processors to be used for barotropic solver
clinic_distribution_type	[‘balanced’], ‘cartesian’	method for distributing blocks across processors for most of the code
tropic_distribution_type	‘balanced’, [‘cartesian’]	method for distributing blocks across processors barotropic solver
ew_boundary_type	[‘cyclic’], ‘closed’	type of boundary in the logical east-west direction for global domain
ns_boundary_type	[‘cyclic’], ‘closed’,’tripole’	type of boundary in the logical north-south direction for global domain
/		

The distribution of blocks across processors is determined by the parameters `clinic_distribution_type` and `tropic_distribution_type`. Typically, the `‘balanced’` choice is best for the baroclinic part of the code and `‘cartesian’` is best for the barotropic solver.

In order to update “ghost cells” and implement proper boundary conditions, some boundary information for the global domain is required. The parameter `ew_boundary_type` determines the type of boundary for the logical east-west direction (i direction). Acceptable values are currently `'cyclic'` and 'closed' for periodic and closed boundaries, respectively. The parameter for the logical north-south direction (j direction) is `ns_boundary_type` and accepts 'cyclic','closed' and 'tripole', where cyclic and closed have the same meaning as the east-west direction and tripole refers to use of a tripole grid.

3.1.2 Input/Output

POP supports both binary and netCDF file formats. The formats for each type of file (e.g. restart, history, movie) are set in individual namelists for those operations. For binary output format, POP can perform parallel input/output in order to speed up IO when writing large files. Because most files read or written by POP utilize direct-access IO with a horizontal slice written to each binary record, the parallel IO routines allow several processors to write individual records to the same file. The user can specify how many processors participate in the parallel IO with some restrictions. The number of processors obviously cannot exceed the total number of processors assigned to the job. In addition, it is not productive to assign more processors than the number of vertical levels as these processors will generally remain idle (or even perform unnecessary work). Lastly, there may be some restrictions based on the particular architecture. Some architectures have a limit on the number of effective IO units that can be open simultaneously. Some architectures (e.g. loose clusters of workstations) may not have a file system accessible to all of the participating processors, in which case the user must set the number of IO processors appropriately. Lastly, note that netCDF does not support parallel I/O, so any netCDF formatted files will be read/written from a single processor regardless of the num_iotasks setting.

The POP model writes a variety of information, including model configuration and many diagnostics, to standard output. Typically standard output would be redirected to a log file using a Unix redirect > operator. However, in some cases this is not possible, so a namelist flag `lredirect_stdout` can be turned on to redirect standard output to a log file. The logfile will have the name `log_filename`.*date.time* where the date and time are the actual wallclock time and not the model simulation time.

During production runs, it is not convenient to have to change the `pop_in` file for every run. Typically, the only changes necessary are the names of any restart input files. To avoid having to change these filenames in the `pop_in` file for every run, an option `luse_pointer_files` exists. If this flag is .true., the names of restart output files are written to pointer files with the name `pointer_filename`.*suffix*, where *suffix* is currently either `restart` or `tavg` to handle restart files and tavg restart files. When a simulation is started from restart, it will read these pointer files to determine the location and name of the actual restart files.

Table 3.2: I/O namelist

&io_nml		**options for controlling I/O**
num_iotasks	(1,min(km, nprocs_clinic))	number of I/O processes for parallel binary I/O
lredirect_stdout	[.false.]	flag to write stdout to log file
log_filename	[‘pop.out’]	root filename (with path) of optional output log file
luse_pointer_files	[.false.]	flag to turn on use of pointer files
pointer_filename	[‘pop_pointer’]	root filename (with path) of pointer files
/		

3.1.3 Time management

The `time_manager_nml` namelist controls the timestep, the length of the current run, the method used to suppress the leapfrog computational mode, and the date on which this *run-sequence* began. A *run-sequence* consists of one or more job submissions, each of which produces a *restart file* that is used to begin the next job in the sequence. A run-sequence is identified by a `runid` that is declared in the first job of the sequence and held fixed throughout the sequence; `runid` is used in generating default names for the model's output files. Similarly, the start date and time for the run sequence (`iyear0...,isecond0`), are set in the first job and held fixed throughout the sequence. An additional variable called `date_separator` can be used to govern the form of the date that is appended to various output files. The `date_separator` is a single character used to separate yyyy, mm, and dd in a date format. A blank character is the default and is translated to no separator (yyyymmdd); a value of '-' would result in the format yyyy-mm-dd.

The timestep is defined using a combination of `dt_option` and `dt_count`. If `steps_per_(day,year)` is chosen, the timestep is computed such that `dt_count` steps are taken each day or year. If `hours` or `seconds` is chosen, the timestep is `dt_count` in hours or seconds (note that `dt_count` is an integer). If `auto_dt` is chosen, the timestep is automatically computed based on the grid size. The time step may be adjusted from these values to accomodate averaging time steps.

In order to control a computational mode resulting from the use of a leapfrog time stepping scheme, either a time-averaging method (‘avg’,‘avgbb’,‘avgfit’) or a Matsuno (‘matsuno’) time step must be specified through the `time_mix_opt` parameter. The frequency (in time steps) for applying one of these methods is defined by the `time_mix_freq` parameter. If ‘avg’ is selected for `time_mix_opt`, the averaging results in only a half timestep being taken every `time_mix_freq` steps. This may result in a non-integral number of steps per day and will result in irregular day boundaries. If an integral number of steps per day is required, two alternative options are provided. Choosing 'avgbb' will enable always taking two half steps back-to-back, thus giving a full time step, but with

Table 3.3: Time manager namelist

&time_manager_nml		**management of time-related quantities**
runid	must be supplied	alphanumeric run-sequence identifier
stop_option	['nstep'], 'nday', 'nyear', 'date'	units of time for 'stop_count'
stop_count	[20]	how long in above units to run this segment (use yyyymmdd for date)
time_mix_opt	['avgfit'], 'avgbb', 'avg', 'matsuno'	Method to suppress leapfrog computational mode.
fit_freq	[1]	When using 'avgfit', the intervals per day into which full and half steps must fit
time_mix_freq	[17]	Requested frequency (in steps) for taking mixing steps
dt_option	['auto_dt'], 'steps_per_day', 'steps_per_year', 'seconds', 'hours'	units for determining timestep (combined with dt_count)
dt_count	[1]	number of timesteps in above units to compute timestep
impcor	[.true.]	If .true., the Coriolis terms treated implicitly
laccel	[.false.]	if .true., tracer timesteps increase with depth
accel_file	['unknown_accel']	file containing vertical profile of timestep acceleration factor
dtuxcel	1.0	factor to multiply momentum timestep for different momentum and tracer timesteps
allow_leapyear	[.false.]	use leap years in calendar
iyear0	[0]	year (yyyy) at start of full run sequence
imonth0	[1]	Month at start of sequence
iday0	[1]	day at start of sequence
ihour0	[0]	hour at start of sequence
iminute0	[0]	Minute at start of sequence
isecond0	[0]	Seconds at start of sequence
date_separator	[' ']	Character to separate yyyy mm dd in date (' ' means *no* separator)
/		

increased diffusion. The 'avgfit' option will compute a number of full and half steps that will fit into a particular time interval. The time interval to be fit is governed by the 'fit_freq' parameter which sets the number of time intervals per day (1=once per day) into which the time steps must fit exactly. The Matsuno scheme does not use half steps, but Matsuno is generally more diffusive than time averaging and has been shown to be unstable in many situations.

The timestep above can be increased for tracers in the deep ocean. If such acceleration is requested (`laccel` = .true.), a profile of the acceleration with depth must be read from the file `accel_file`, an ascii file with a single column of numbers giving the acceleration factor for each vertical level. Another form of acceleration is to take a longer tracer timestep than momentum timestep. This can be specified by changing `dtuxcel` to a factor smaller than 1.0.

3.2 Grid and bottom-topography definition

POP can be configured to use a variety of grids, because it is written to allow any logically rectangular, orthogonal coordinate system on a sphere. Generation grids and topography can be time consuming, but it only needs to be done once for a given run-sequence (or even for a set of run-sequences based on the same grid). Consequently, grids and topography are usually generated off-line and grid information is stored in files that are read in during initialization of each job. A graphical tool for generating and modifying grids and topography will soon be released and can be obtained on request. This tool supports almost-global Mercator grids, global displaced-pole grids, global tripole grids and regional grids.

When generating the grid off-line, a horizontal grid file is created containing arrays of double precision values for:

- latitude (radians) of velocity (U) points
- longitude (radians) of velocity (U) points
- length (cm) of north side of tracer (T) cell
- length (cm) of east side of tracer (T) cell
- length (cm) of south side of velocity (U) cell
- length (cm) of west side of U velocity (U) cell
- angle formed by the south U cell side and latitude circle passing through the soutwest corner of U cell

where north, south, east and west refer to logical directions, not necessarily geographic directions. The vertical grid file is an ASCII file containing the thickness of each grid layer on a separate line.

Options exist to generate both horizontal and vertical grids internally ('internal'). Because any grid size can be easily produced and IO is unnecessary,

Table 3.4: Grid and topography namelist

&grid_nml		**input/generation of grid and bottom topography**
horiz_grid_opt	'file', ['internal']	read horizontal grid from a file OR create simple lat/lon grid
horiz_grid_file	['unknown_grid_file']	filename (with path) of file containing horizontal grid info
sfc_layer_opt	['varthick'], 'rigid', 'oldfree'	surface layer is variable thickness OR rigid lid OR old free surface formulation
vert_grid_opt	'file', ['internal']	read vertical grid structure from file OR compute vertical grid internally
vert_grid_file	['unknown_vert_grid_file']	file containing thickness (cm) of each vertical layer
topography_opt	'file', ['internal']	read discretized bottom topography from file or compute idealized flat-bottom topography internally
topography_file	['unknown_topography_file']	file containing index of deepest level at each gridpoint
region_mask_file	['unknown_region_mask']	file containing region number at each gridpoint
partial_bottom_cells	[.false.]	use partial bottom cells
bottom_cell_file	['unknown_bottom_cell']	file containing thickness (cm) of partial bottom cell for each column
topo_smooth	[.false.]	if .true., smooth topography using 9-point averaging stencil
flat_bottom	[.false.]	if .true., flat bottom is used
lremove_points	[.false.]	if .true., remove isolated or disconnected ocean points
/		

this option is used for the benchmark test problem (see Sec.2.7). The horizontal grid in this case is a simple latitude-longitude grid. The vertical grid uses an algorithm identical to the off-line vertical grid generator located in the tools/grid directory; vertical grids are generated which have relatively shallow surface layers and increasing thickness with depth.

The bottom topography is defined by a field of integers (KMT) which gives the index of the deepest vertical level at each horizontal grid point. This field should also be generated off-line because it often requires modification for narrow channels and other small-scale features. For off-line generated topography, the code expects a binary file with the integer KMT field. Topography can be generated internally, but this option currently only creates an idealized continental outline with a flat bottom; it should not be used for serious ocean simulations. Bottom topography can be modified to include partial bottom cells. If partial bottom cells are selected, the thickness of the bottom cell in each column can be less than the full thickness of the bottom model layer, giving a better representation of the actual bottom topography. For this option, an array containing the thickness of each bottom cell (in cm) must be read from a bottom cell input file.

At present, POP does not support in-flow/out-flow boundary conditions for regional grids. For such regional simulations, closed boudaries are used with buffer zones whose thermohaline properties are maintained by restoring to climatology (see Sec. 3.8.3).

Three options are available for the surface layer. The default is now a variable thickness surface layer ('varthick') in which the thickness of the surface layer adjusts to fresh water input/export. Note that in this formulation, an assumption remains that the changes in thickness should be much smaller than the thickness of the surface layer itself. A second option is the original free surface formulation ('oldfree') where the thickness remains constant, but changes in the free surface height are used in forcing terms. In this case, fresh water fluxes are treated as virtual salinity fluxes and conservation of tracers is not exact (though the residual tracer flux is globally very small). The last option is the rigid lid option ('rigid'). As it implies, there is no variation is surface height with this option. Note that when using this option, the barotropic solver takes much longer to converge so the maximum iteration parameter must be increased to account for this slower convergence. The rigid lid option is not recommended for typical ocean simulations.

For some forcing options (and in future versions for regional diagnostics and river runoff), a region mask is necessary to define various regions. This mask is a simple two-dimensional integer mask that assigns a region number for each horizontal grid point. Negative values of the region number are used to indicate marginal seas. If such a region mask is necessary, it is read from the `region_mask_file`. If the mask is not necessary, `region_mask_file` must be set to 'unknown_region_mask'.

3.3 Initializing the model state

3.3.1 Temperature and salinity distribution

Most jobs are continuations of a run-sequence, so the potential temperature and salinity (and other necessary variables) are read in from a restart file (see Sec. 3.3.2). In this case, the `init_ts_opt` should be set to `'restart'` and the `runid` and `iyear0,...,isecond0` parameters in `time_manager_nml` are reset by the values in the restart file.

Table 3.5: Temperature and salinity initialization

&init_ts_nml		**initial temperature and salinity distribution**
init_ts_option	'restart', 'branch', 'file', 'mean', ['internal']	start from *restart* OR read initial ocean conditions from a *file* OR create conditions from an input *mean* ocean profile OR create initial conditions based on 1992 Levitus mean ocean profile computed *internally*
init_ts_file	'unknown_ts_file'	restart file OR file containing 3D potential temperature and salinity at grid points OR file containing depth profile of potential temperature and salinity OR (ignored for 'internal' or when luse_pointer_files is enabled)
init_ts_file_fmt	['bin'], 'nc'	data format (binary or netCDF) for input init_ts_file ('file' and 'restart' options only)
/		

If you want to start from an old restart file, but also want the new run to start from time zero, the `init_ts_opt` should be set to `'branch'`. In this case, the time information in the restart file is ignored (namelist inputs for the various parameters are used instead), but the ocean state is initialized with the data in the restart file. Note, that this will not give an exact restart because the code will start with an Euler forward step as if it were starting from scratch.

To begin a run from scratch, any of the other three options can be used.The `'file'` option reads a file containing the 3-d fields for potential temperature and salinity and uses those fields for the initial condition. The `'mean'` option reads a mean profile from an ascii input file with potential temperature and salinity given in two columns as a function of depth.This mean state will be spread across the horizontal domain to create horizontally-uniform 3-d fields. The final option is `'internal'` which generates a mean vertical profile by interpolating from the 1992 Levitus mean ocean profile. As in the `'mean'` option, this vertical profile is then spread across the horizontal domain to create the full 3-d fields.

3.3.2 Restart control

As a POP calculation proceeds, restart files are produced at intervals of simulated time specified by the parameter `restart_freq` in namelist `restart_nml`. The root of the restart filename is given by the `restart_outfile` variable. The actual full filename will be *restart_outfile.runid.suffix* where *suffix* is a number that depends on `restart_freq_opt`. If this option is 'nyear', 'nmonth' or 'nday' *suffix* will be the calendar day in yyyymmdd. If the option is 'nstep', *suffix* will be the current step number. In the unlikely case that the user chooses 'nhour' or 'nsecond', *suffix* will be yyyymmdd.[hh,sssss] where hh and sssss denote the current hour or seconds in the current day. A common convention (see Chapter 4) is to give restart files the simple root name 'd' (for dump).

Table 3.6: Restart file namelist

&restart_nml		**generation of restart files**
restart_freq_opt	'nyear', 'nmonth', 'nday', 'nhour', 'nsecond', 'nstep',['never']	units of time for 'restart_freq'
restart_freq	[100000]	number of units between output of restart files
restart_outfile	['d']	root filename (with path prepended, if necessary) for restart files ('runid' and suffixes will be added)
restart_fmt	['bin'],'nc'	data format (binary or netCDF) for restart output files
leven_odd_on	[.false.]	create alternating even/odd restart outputs which overwrite each other
even_odd_freq	[100000]	frequency (in steps) for even/odd output
pressure_correction	[.false.]	if true, corrects surface pressure error due to (possible) different timestep. use .false. for exact restart
/		

In addition to these named restart files, additional restart files may be written using the even/odd convention. If `leven_odd` is set to .true., the model will write restart files every `even_odd_freq steps`. These files will be named *restart_outfile.runid.even,odd* where the code alternately writes to an even or odd file. Any previous file of the same name will be overwritten. This capability is particularly useful for backup in case the simulation terminates prematurely; it provides restart capability without needing to keep many named restart files present in the file system. The model writes alternately to each of two such files so that if an error occurs while writing one restart file, the other restart file will still (presumably) be all right.

Restart files contain only the minimum information required to restart the model exactly, so that the results are the same after a restart as would have been the case had no restart been done.This means that two time-levels (n-1 and n) of the prognostic variables (PT, S, U, V and H) must be saved in 64-bit precision. The restart format is governed by the `restart_fmt` option. Note that using netCDF for restarts on machines that do not use the IEEE binary format standard will result in restart files that are not exact due to internal conversion by the netCDF library. This will prevent exact reproducibility on such machines. Binary format is therefore highly recommended for restart files.

3.4 Computational options

3.4.1 Barotropic mode solver

As part of the implicit solution of the barotropic mode, a two-dimensional elliptic equation for the surface pressure is solved. Three solver methods are available, all iterative (preconditioned conjugate gradient, conjugate residual and Jacobi). Convergence of the iterative solvers is governed by the two parameters `solv_convrg` and `solv_max_iters` as shown in the table below. The convergence criterion `solv_convrg` should be chosen small enough such that the pressure balance (printed as part of the model global diagnostics) agrees to 3-4 digits. The parameter `solv_max_iters` must be chosen large enough to allow the solver to converge (typically a few hundred), but small enough so that the code will terminate in a reasonable time if the solver is unable to converge.

Occasionally, when benchmarking the code, it is useful to fix the number of iterations to give a consistent iteration count between runs. In this case, `solv_convrg` is set to exactly zero and the solver will iterate `solv_max_iters` and continue with the simulation without terminating. This feature should only be used for benchmarking and not for actual ocean simulations.

The `solv_ncheck` provides a means to improve performance by checking for convergence every `solv_ncheck` iterations, thus eliminating an extra global sum on most iterations. Another means for improving performance is to supply a preconditioner to improve convergence. The preconditioner must be computed off-line and must be in the form of a nine point stencil operator. The preconditioner is then supplied in a file named `precond_file` containing the nine

Table 3.7: Barotropic solver namelist

&solver_nml		**control of iterative solver for barotropic mode**
solv_type	[‘pcg’], ‘cgr’, ‘jac’	preconditioned conjugate gradient OR conjugate gradient residual OR jacobi
lprecond	[.false.]	if .true., use preconditioner to reduce number of iterations to convergence
precond_file	[‘unknown_precond_file’]	file containing preconditioner coefficients for solver
solv_convrg	1.00E-12	convergence criterion: $\|\delta X/X\| < solv\_convrg$
solv_max_iters	1000	upper limit on number of iterations allowed
solv_ncheck	10	check for convergence every solv_ncheck iterations
/		

operator weights. The Jacobi method converges very slowly; it is not recommended but is provided as an alternative.

3.4.2 Advection methods

Currently, advection of momentum is always done by leapfrog centered advection with periodic ‘mixing’ steps (see Sec. 3.1.3) For tracer advection, two options are available. The first is standard leapfrog centered advection; the second is a 3rd-order upwinding [14] which, although not monotone, will improve monotonicity at a somewhat increased computational cost.

Table 3.8: Advection namelist

&advect_nml		**advection methods for tracers**
tadvect_ctype	[‘centered’], ‘upwind3’	centered differences OR 3rd-order upwinding
/		

3.4.3 Pressure gradient options

The pressure-averaging technique was explained in section 1.2.4. Because it increases the timestep, it should always be enabled. The option to turn it off is provided only to permit comparisons with and without pressure-averaging or between POP and other codes that do not incorporate this technique.

The pressure gradient term includes a density factor which is assumed to be a constant reference density in Boussinesq models. The depth-dependent pressure effects on this density can be corrected for using simple depth-dependent factors.

Table 3.9: Pressure averaging namelist

&pressure_grad_nml		**averaging of horizontal pressure gradient**
lpressure_avg	[.true.]	use pressure averaging to increase time step
lbouss_correct	[.true.]	applies depth-dependent factor to correct for assumed constant density
/		

3.5 Vertical mixing and convection parameterizations

Several vertical mixing parameterizations are available within the POP model and are described in much more detail in the Reference Manual. The value of `vmix_choice` determines whether a simple constant mixing, a Richardson-number dependent mixing or the KPP mixing parameterization is used. Additional mixing parameters for each of these schemes are set in individual namelists shown below; only the namelist associated with the mixing choice is actually read.

The treatment of convection is also specified in the vertical mixing namelist through the `convection_type` variable. Convection can be treated using either convective adjustment or by specifying large diffusion coefficients in convectively unstable regions. The KPP vertical mixing parameterization *must* use the diffusion option. If convective adjustment is chosen, the number of passes through the vertical column to adjust is determined by the parameter `nconvad`. The treatment of convection by diffusion is governed by the input diffusion coefficients `convect_diff` and `convect_visc`. Note that for constant vertical mixing, you can apply diffusion to tracers only by setting `convect_visc` to zero; this is *not* true for Richardson number mixing or KPP.

Some vertical and horizontal mixing parameterizations (e.g. KPP and Gent-McWilliams to be discussed later) create large vertical mixing coefficients. In addition, when diffusion is used as the method for treating convection, the diffusion coefficients are large. In such cases, implicit vertical mixing must be enabled (`implicit_vertical_mix = .true.`) to avoid severe restrictions on the model time step.

If implicit vertical mixing is chosen, the parameter `aidif` governs the time-centering of the implicit scheme.The `bottom_drag` coefficient is used to compute bottom drag. To simulate geothermal heating at the bottom of the ocean, a

Table 3.10: Vertical mixing namelist

&vertical_mix_nml		**vertical mixing parameterizations**
vmix_choice	‘const’, ‘kpp’,[‘rich’]	method of computing vertical diffusion
implicit_vertical_mix	[.true.]	if true, vertical mixing is solved implicitly in time
aidif	[1.0] 0.5-1.0	time-centering parameter for implicit vertical mixing; use of the default value [1.0] is recommended
bottom_drag	[1.0E-03]	(dimensionless) coefficient used in quadratic bottom drag formula
convection_type	‘adjustment’, [‘diffusion’]	Convection treated by adjustment or by large mixing coefficients
nconvad	2	number of passes through the convective adjustment algorithm
convect_diff	[1000.]	tracer mixing coefficient to use with diffusion option
convect_visc	[1000.]	momentum mixing coefficient to use with diffusion option
bottom_heat_flx	[0.0]	constant (geothermal) heat flux (W/m^2) to apply to bottom layers
bottom_heat_flx_depth	[100000.]	depth (cm) below which to apply bottom heat flux
/		

constant heat flux can be applied below a fixed depth in the ocean. A heat flux of zero turns off this option.

3.5.1 Constant coefficients

Constant vertical mixing simply uses a constant diffusion coefficient for mixing everywhere in the domain.

Table 3.11: Constant vertical mixing namelist

&vmix_const_nml		**constant vertical mixing coefficients**
const_vvc	[0.25]	vertical viscosity coefficient (momentum mixing) (cm^2/s)
const_vdc	[0.25]	vertical diffusivity coefficient (tracer mixing) (cm^2/s)
/		

3.5.2 Richardson-number mixing

The Pacanowski and Philander [1] mixing scheme was developed primarily for use in tropical ocean and, although it is often used elsewhere in the global ocean, the user should be aware of the possible need to adjust its parameters ([17],[10]).

Table 3.12: Richardson-number vertical mixing namelist

&vmix_rich_nml		**Richardson-number mixing (Pacanowski-Philander)**
bckgrnd_vvc	[1.0]	background vertical viscosity (cm^2/s)
bckgrnd_vdc	[0.1]	background vertical diffusivity (cm^2/s)
rich_mix	[50.0]	Coefficient for Richardson-number function
/		

3.5.3 KPP mixing

The k-profile parameterization (KPP) [13] is relatively complex and only the parameters that are routinely changed are shown here in the namelist. It is possible to change other parameters by editing the KPP module, but this should not be necessary and is discouraged. As described previously, KPP utilizes enhanced diffusion for convection so implicit vertical mixing must be enabled and diffusion must be specified as the convection method. Note that the constants `convect_diff, convect_visc` are used for convection within KPP.

A recent change to the KPP implementation is to allow a depth dependent background diffusivity κ and viscosity ν. The form of this dependence is

$$\kappa \quad = \quad \kappa\_1 + \kappa\_2 \arctan((z-d)/L) \tag{3.1}$$

Table 3.13: KPP namelist

&vmix_kpp_nml		**KPP mixing**
bckgrnd_vdc1	[0.1]	base background vertical diffusivity (cm^2/s)
bckgrnd_vdc2	[0.0]	variation in background vertical diffusivity (cm^2/s)
bckgrnd_vdc_dpth	[250000.0]	depth (cm) at which background vertical diffusivity is vdc1
bckgrnd_vdc_linv	[0.000045]	inverse of the length scale (1/L in cm^{-1}) over which diffusivity transition takes place
Prandtl	[10.0]	(unitless) ratio of background vertical viscosity and diffusivity
rich_mix	[50.0]	Coefficient for Richardson-number function
lrich	[.true.]	use Richardson-number for interior mixing
ldbl_diff	[.false.]	add double-diffusive parameterization
lshort_wave	[.false.]	use penetrative shortwave forcing
lcheckekmo	[.false.]	check whether boundary layer exceeds Ekman or Monin-Obukhov limit
num_v_smooth_Ri	[1]	Number of passes to smooth Richardson number
/		

$$\nu = (\mathrm{Pr})\kappa. \tag{3.2}$$

where z is the model depth, d is the depth at which κ reaches κ_1, L is a length scale over which the transition between κ_1 and κ_2 takes place and Pr is the Prandtl number. If a constant diffusivity and viscosity are required, simply set vdc2 to zero and vdc1 to the appropriate diffusivity.

3.6 Horizontal mixing parameterizations

Several horizontal mixing options are available for mixing tracers and momentum. With a few exceptions (discussed later), the choice of tracer mixing can be made independently of the choice of momentum mixing. As with vertical mixing, the main namelist input only selects the choice of mixing options; the actual mixing parameters associated with each option are read from a namelist specific to that option. The `del2` (Laplacian) and `del4` (bi-harmonic) mixing options are *ad hoc* level-oriented parameterizations that mix water-mass properties across sloping isopycnic surfaces. The Gent-McWilliams [11] parameterization remedies this shortcoming by forcing the mixing (of tracers only) to take place along isopycnic surfaces. The principal drawback of the `gent` option is cost; it nearly doubles the running time. For momentum mixing, an anisotropic viscosity parameterization (`aniso`) is also available which assigns different values of viscosity parallel and perpendicular to a given direction, where the direction can be specified as described in a later section. Under the `aniso` option, a Smagorinsky form of viscosity can be specified.

Table 3.14: Horizontal mixing namelist

&hmix_nml		**horizontal mixing methods**
hmix_momentum_choice	['del2'], 'del4', 'anis'	method for horizontal mixing of momentum (Laplacian, biharmonic or anisotropic)
hmix_tracer_choice	['del2'], 'del4', 'gent'	method for horizontal mixing of tracers (Laplacian, biharmonic or Gent-McWilliams)
/		

3.6.1 Laplacian horizontal mixing.

The Laplacian mixing coefficients for tracers `ah` and momentum `am` are specified in separate namelists. The defaults shown in the namelists are only valid for a particular grid size; the user must determine the appropriate values for their particular grid size. The `variable_hmix` option modifies the coefficients `ah` and `am` based on functions of the grid cell areas and will reduce the values for smaller grid cells (`am` and `ah` thus represent the values at the largest grid cells). Currently, the functional form of this scaling can only be changed by editing the modules. The `auto_hmix` option attempts to compute coefficients based on known values for other resolutions. The result may or may not be suitable and the `auto_hmix` option is provided mainly for flexible benchmarking of the code at various resolutions.

Table 3.15: Laplacian momemtum mixing namelist

&hmix_del2u_nml		**Laplacian momentum mixing parameters**
lauto_hmix	[.false.]	computes mixing coefficient based on resolution
lvariable_hmix	[.false.]	scales mixing coeff by grid cell area
am	$\sim 2 \times 10^7$	momentum mixing coefficient (cm^2/s)
/		

3.6.2 Biharmonic horizontal mixing.

The biharmonic mixing coefficients for tracers `ah` and momentum `am` are specified in separate namelists. The defaults shown in the namelists are only valid for a particular grid size; the user must determine the appropriate values for their particular grid size. The `variable_hmix` option modifies the coefficients

Table 3.16: Laplacian tracer mixing namelist

&hmix_del2t_nml		**Laplacian tracer mixing parameters**
lauto_hmix	[.false.]	computes mixing coefficient based on resolution
lvariable_hmix	[.false.]	scales mixing coeff by grid cell area
ah	$\sim 2 \times 10^7$	tracer mixing coefficient (cm^2/s)
/		

`ah` and `am` based on functions of the grid cell areas and will reduce the values for smaller grid cells (`am` and `ah` thus represent the values at the largest grid cells). Currently, the functional form of this scaling can only be changed by editing the modules. The `auto_hmix` option attempts to compute coefficients based on known values for other resolutions. The result may or may not be suitable and the `auto_hmix` option is provided mainly for flexible benchmarking of the code at various resolutions.

Table 3.17: Biharmonic momentum mixing namelist

&hmix_del4u_nml		**Biharmonic momentum mixing parameters**
lauto_hmix	[.false.]	compute mixing coefficient based on resolution
lvariable_hmix	[.false.]	scale mixing coeff by grid cell area
am	$\sim -0.6\times 10^{20}$	momentum mixing coeff (cm^2/s)
/		

Table 3.18: Biharmonic tracer mixing namelist

&hmix_del4t_nml		**Biharmonic tracer mixing parameters**
lauto_hmix	[.false.]	compute mixing coefficient based on resolution
lvariable_hmix	[.false.]	scale mixing coeff by grid cell area
ah	$\sim -0.2\times 10^{20}$	tracer mixing coefficient (cm^2/s)
/		

3.6.3 Gent-McWilliams isopycnic tracer diffusion

Gent-McWilliams (`gent`) mixing operates only on tracer species (potential temperature, salinity and other tracers), so it should be used in conjunction with a different option for `hmix_momentum_choice`, typically either `del2` or `aniso`. No bi-harmonic form of `gent` has been developed and accepted yet, so it is appropriate to use the `del2` values of `ah`. For vertical dependence of the mixing, a profile with the form $\kappa_1 + \kappa_2 \exp(-z/D)$ can be chosen, where D is a depth scale, z is model depth and κ_1 and κ_2 parameters specifiy factors multiplying the diffusivity. Note that this function is multiplied by the diffusivity `ah`; for a constant κ, the first parameter should be set to 1 and the second to 0. Two diffusivities can be specified for the Redi and bolus parts of the GM parameterization; `ah` is used for the Redi part, `ah_bolus` is used for the bolus part. Two different maximum slopes can also be specified to allow different taperings of the Redi and bolus terms. A backgroud horizontal diffusivity `ah_bkg` can be used for bottom cells. If the `gm_bolus` flag is set, the bolus velocity is explicitly calculated and used as part of the velocity field, as opposed to the incorporating this process as part of the horizontal mixing. This last option does not currently work with partial bottom cells.

Table 3.19: Gent-McWilliams horizontal mixing namelist

&hmix_gm_nml		**Gent-McWilliams isopycnic diffusion**
kappa_choice	['constant'], 'variable'	constant or (horizontally) variable kappa
slope_control_choice	['notanh'], 'tanh', 'clip', 'gerdes'	control slope using tanh, algebraic approximation to tanh (notanh), clipping or method of Gerdes
kappa_depth_1	[1.0]	the first term in the function for variation of kappa with depth
kappa_depth_2	[0.0]	the coefficient of the exponential in the function for variation of kappa with depth
kappa_depth_scale	[150000.0]	the depth scale for the exponential in the function for variation of kappa with depth
ah	0.8×10^7	diffusion coeff for Redi part (cm^2/s)
ah_bolus	0.8×10^7	diffusion coeff for bolus part (cm^2/s)
ah_bkg	0.0	diffusion coeff for bottom cells (cm^2/s)
slm_r	0.01	max slope for Redi terms
slm_b	0.01	max slope for bolus terms
gm_bolus	[.false.]	option for explicit calculation of bolus velocity

3.6.4 Anisotropic viscosity options

The anisotropic viscosity routine computes the viscous terms in the momentum equation as the divergence of a stress tensor, which is linearly related to the rate-of-strain tensor with viscous coefficents `visc_para` and `visc_perp` . These coefficients represent energy dissipation in directions parallel and perpendicular to a specified alignment direction which breaks the isotropy of the dissipation. There are three options for choosing the alignment direction: 1) along the local instantaneous flow direction, 2) along the east direction, and 3) along the coordinate directions (note: the viscous operator is invariant under a rotation of the alignment direction by 90 degrees, so for example, choosing the alignment direction as north, south, east or west are all equivalent.). A functional approach is used to derived the discrete operator, which ensures positive-definite energy dissipation, provided `visc_para` > `visc_perp`.

Parallel and perpendicular viscosities can vary in space by setting the flag `lvariable_hmix_aniso` to true. The spatially-varying viscosities in the parallel and perpendicular directions are read from a file (`var_viscosity_infile`). A specific form of the viscosities useful for CCSM coupled simulations can be internally computed if the input filename is 'ccsm-internal'. In such a case, the six viscosity parameters for the form must also be supplied.

The viscosities may optionally (`lsmag_aniso` = .true.) be evaluated with Smagorinsky-like non-linear dependence on the deformation rate, which is proportional to the norm of the strain tensor. With the Smagorinsky option, the viscosities are evalutated as

$$\begin{aligned} \nu_\parallel &\rightarrow \max(c_\parallel |D| ds^2), u_\parallel ds) \\ \nu_\perp &\rightarrow \max(c_\perp |D| ds^2), u_\perp ds) \end{aligned} \tag{3.3}$$

where $ds = \min(dx, dy)$, $|D| = \sqrt{2}|E|$ is the deformation rate, $|E|$ is the norm of the strain tensor, $c_\parallel$ and $c_\perp$ are dimensionless coefficients of order 1, and $u_\parallel$ and $u_\perp$ are velocities associated with the grid Reynolds number which determine minimum background viscosities in regions where the nonlinear viscosities are too small to control grid-point noise. Typically $u_\parallel$ and $u_\perp$ are order 1 cm/s. Perpendicular Smagorinsky coefficients can be reduced using a latitudinally-dependent Gaussian function. The form of this function is governed by the three `smag_lat` parameters.

Table 3.20: Anisotropic viscosity namelist

&hmix_aniso_nml		**Anisotropic viscosity**
hmix_alignment_choice	['flow'], 'grid', 'east'	choice for alignment of parallel viscosity component (aligned with local flow, grid lines or east-west direction)
lvariable_hmix_aniso	[.false.]	use spatially-varying viscosity
lsmag_aniso	[.false.]	compute viscosities using a Smagorinsky formulation
visc_para	[0.0]	parallel viscosity component(cm^2/s)
visc_perp	[0.0]	perpendicular viscosity component (cm^2/s)
c_para	[0.0]	dimensionless parallel Smagorinsky coeff
c_perp	[0.0]	dimensionless perpendicular Smagorinsky coeff
u_para	[0.0]	velocity (cm/s) for grid Reynolds no. viscous limit in parallel direction
u_perp	[0.0]	velocity (cm/s) for grid Reynolds no. viscous limit in perpendicular direction
var_viscosity_infile	'ccsm-internal', 'file-name'	name of file containing variable viscosity factors or internal generation of a ccsm-specific form
var_viscosity_infile_fmt	'bin','nc'	viscosity input file format
var_viscosity_outfile	'filename'	output file for writing internally-generated variable viscosity
var_viscosity_outfile_fmt	'bin','nc'	viscosity output file format
vconst_1	1.e7	CCSM variable viscosity parameter
vconst_2	24.5	CCSM variable viscosity parameter
vconst_3	0.2	CCSM variable viscosity parameter
vconst_4	1.e-8	CCSM variable viscosity parameter
vconst_5	3	CCSM variable viscosity parameter
vconst_6	1.e7	CCSM variable viscosity parameter
smag_lat	20.0	latitude (degrees) for starting variation in Smagorinsky viscosity
smag_lat_fact	0.98	amplitude of Gaussian function for reducing Smagorinsky coefficients
smag_lat_gauss	98.0	width (degrees squared) of Gaussian function for reducing Smagorinsky coefficients

3.7 Physical process options

3.7.1 Equation of state approximation

Four options for computing the density from salinity and potential temperature are available. The first is an equation of state introduced by McDougall, Wright,

Table 3.21: Equation of state namelist

&state_nml		**equation of state approximation**
state_choice	['mwjf'], 'jmcd', 'polynomial', 'linear'	McDougall et al. eos OR Jackett and McDougall eos OR polynomial fit to UNESCO eos OR linear eos
state_file	['internal'], filename	compute polynomial coefficients internally OR read from file filename
state_range_opt	['ignore'], 'check', 'enforce'	ignore when T,S outside valid polynomial range OR check and report OR compute eos as if T,S were in valid range (but don't alter T,S)
state_range_freq	[1]	frequency (steps) for checking T,S range
/		

Jackett and Feistel (MWJF [16]) which is a faster and more accurate alternative to the UNESCO equation of state. The second is a UNESCO equation of state based on potential temperature from Jackett and McDougall (JMCD [12]). The third is a polynomial fit to the full UNESCO equation of state. The advantage of the polynomial form is that it is faster; the disadvantage is that the polynomial is only valid over a specified temperature and salinity range and exceeding that range will have unpredictable results. This is a particular issue with the KPP vertical mixing scheme which often computes buoyancy by displacing water near the surface to deep water where the EOS range has been restricted. The last option is a linear eos which is supplied for use only in special situations where such an approximation is appropriate.

For the polynomial option, there are two methods for determining the polynomial coefficients and these are determined by the value of `state_file`. If this variable is defined as 'internal', the code will determine the polynomial coefficients internally based on the vertical grid. The internal routines currently use hard-wired profiles for the limits of validity of the polynomial eos; if the user wishes to change these limits, they can be changed in an off-line coefficient generator (in the **tools/eos** directory) and the coefficients can be read in from a file. The value of `state_file` will be the name of the coefficient input file. As mentioned above, the polynomial eos has a certain temperature and salinity range over which the polynomial is valid. The `state_range_opt` variable determines what to do if these limits are exceeded during a simulation. The first option is to simply 'ignore' when these occur; this is generally not as bad as it sounds as the range is valid for nearly all normal cases. The second option is to 'check' whether the range is exceeded and print a warning if such problems are detected. The last option, 'enforce', simply makes sure the polynomial is evaluated within the correct range without changing the values of T or S. For example, if the temperature drops below -2C, the code will compute a density based on a temperature of -2C without actually changing the temperature.

The state_range_freq can be used to perform the checks infrequently to save computational time.

3.7.2 Baroclinic-mode parameters

The reset_to_freezing option exists to make sure the surface temperature does not drop below freezing, a situation that can occur with some types of forcing in stand-alone mode. This option should be disabled if sea ice formation is enabled (see Sec.3.7.3).

Table 3.22: Baroclinic namelist

&baroclinic_nml		**parameters used in baroclinic mode calculations**
reset_to_freezing	[.true.]	if .true. and $T_{surf}(i,j) < T_{freezing}$, $T_{surf}(i,j)$ is reset to $T_{freezing}$
/		

3.7.3 Sea-ice emulation parameters

If ice_freq_opt is not 'never', the code will create ice whenever the ocean temperature drops below freezing at levels higher then kmxice. This ice formation will be computed at frequencies determined by the ice_freq in ice_freq_opt units. If the model is being run in coupled mode, ice_freq_opt should be set to 'coupled' to compute ice formation on coupling timesteps. In coupled mode, the heat and water fluxes associated with ice formation are saved and sent to the flux coupler for use in the ice model.

Table 3.23: Ice formation namelist

&ice_nml		**parameters used to emulate sea-ice**
ice_freq_opt	['never'], 'coupled', 'nyear', 'nmonth', 'nday', 'nhour', 'nsecond', 'nstep'	frequency units for computing ice formation
ice_freq	1	frequency in above units for computing ice formation
kmxice	[1], 1-km	compute ice formation above this vertical level
/		

Table 3.24: Topographic stress namelist

&topostress_nml		**Topographic stress parameters**
ltopostress	[.false.]	true if topographic stress enabled
nsmooth_topo	[0]	number of passes to smooth topography
/		

3.7.4 Topographic stress

If `ltopostress` is .true., then an implementation of a topographic stress parameterization [8] is enabled. In effect, this changes the field acted on by the Laplacian operator from (U,V) to (U-U*,V-V*) where (U*,V*) are derived based on the topography gradient and a specified length scale (note that this *only* works for Laplacian mixing). A smoothed topography can be used to compute this gradient with the number of smoothing passes with a 9-point averaging stencil is governed by `nsmooth_topo`.

3.8 Forcing options (ocean-only mode)

Presently, POP includes routines for specifying the surface forcing for the velocity, pressure, temperature and salinity as well as interior forcing for temperature and salinity. While it is impossible to anticipate every kind of forcing a user might want, the routines have been constructed so that it should be relatively easy to add new types of forcing by using existing types as a template. In the following sections, these abbreviations apply:

ws: wind stress; surface forcing for the horizontal velocity field

shf: surface heat flux; surface forcing for the potential temperature equation

sfwf: surface fresh water flux; surface forcing for the salinity equation

ap: surface pressure forcing due to variations in the atmospheric pressure

pt_interior: *interior* forcing for the potential temperature equation

s_interior: *interior* forcing for the salinity equation

Sometimes an asterisk (*) will be used as a UNIX-like wildcard.

Each forcing category listed above has a namelist, and a set of options must be specified in each namelist. The options are:

- periodicity
- temporal interpolation method and interval
- formulation
- files containing forcing data

- units of forcing variables
- updating the forcing values

Since the options apply to several or all of the forcing categories, the options will be explained first, then the namelists for the categories will be given.

3.8.1 Periodicity of forcing data

First, the user must decide on the periodicity (or 'type') of the data that will force the model, for example, an annual mean climatology or a re-analysis product available every day. This choice is specified by the namelist variables `*_data_type` where * denotes each of `ws, shf, sfwf`,etc. The options for `data_type` are:

'none' no forcing

'analytic' a time-invariant analytic form for the forcing

'annual' a time-invariant annual mean forcing

'monthly-equal' a monthly mean climatology assumed to consist of 12 values that are separated equally in time by $365/12 = 30.4166$ days.This was included mostly for backward compatibility.

'monthly-calendar' a monthly mean climatology whose 12 values correspond to the non-leap-year calendar

'monthly' synonymous with 'monthly-calendar'

'n-hour' forcing is specified every 'n' equally spaced hours. If this is chosen, the user must also specify a value for `*_data_inc` which is the increment (in hours) between consecutive values of the forcing data. For example, `ws_data_inc` =24 denotes daily wind stress forcing. The value of `*_data_inc` is disregarded for all of the other `*_data_type` options.

3.8.2 Temporal interpolation of forcing data

Next, the user must decide how to temporally interpolate the forcing data to the appropriate point in time for the model to use. If the data type of the forcing is either 'none', 'analytic', or 'annual', then the interpolation options are disregarded since the forcing is invariant in time. The type of interpolation is specified by the value of `*_interp_type` and the options are (envelope,please):

'nearest' use the forcing from the time that is closest to the current model time

'linear' use a linear interpolation to the current model time using the two nearest forcing times

'4point' use a 3rd order polynomial fit using the 4 nearest forcing times and evaluate it at the current model time

How often interpolation is done is specified using the namelist variables `*_interp_freq` and the options are:

'never' never perform any temporal interpolation

'n-hour' perform temporal interpolation every 'n' hours. If this is chosen, the user must also specify a value for `*_interp_inc` which is the increment (in hours) between interpolation calculations. Note that it is assumed that this value is less than or equal to the data increment.

'every-timestep' perform temporal interpolation every timestep

3.8.3 Forcing formulation

For those model forcing terms that typically depend explicitly on the model state (`shf, sfwf, pt_interior, s_interior`) there are various ways of formulating the forcing which can be specified using the `*_formulation` namelist variables for each case.

`shf_formulation` **options:**

'restoring' a simple restoring of the top layer potential temperature in the model to a data value, $dT/dt = (T_{data} - T_{model})/\tau$, where τ is a constant time scale. T_{data} represents a space- and possibly time-dependent array of values of sea-surface temperature (SST) and is the only necessary forcing field. If this option is chosen, then a value for the space- and time-*in*dependent restoring time-scale variable `shf_restore_tau` (in days) also needs to be specified.

'Barnier-restoring' use the ECMWF heat flux analysis of [2] arranged in restoring form. Necessary forcing fields consist of (in order) an effective SST, spatially varying restoring time scale, sea ice mask, and net downward short-wave radiation. If this option is chosen, then it is also necessary to specify the namelist variable `jerlov_water_type` to calculate the depth of short-wave penetration.

'bulk-NCEP' calculate fluxes based on atmospheric state variables and radiation similar to a fully-coupled model (and using bulk flux formulations extracted from the NCAR flux coupler). Necessary forcing fields consist of (in order) SST, air temperature, air humidity, downward short-wave radiation, cloud fraction, and wind speed. If this option is chosen, then it is also necessary to specify the namelist variable `jerlov_water_type` to calculate the depth of short-wave penetration. This option also expects the name of a file used to define different regions of the ocean (including marginal seas) specified by the namelist variable `region_mask_filename` in the grid namelist (see Sec.3.2).

`sfwf_formulation` **options:**

'restoring' a simple restoring of the top layer salinity in the model to a data value, $dS/dt = (S_{data} - S_{model})/\tau$, where τ is a constant time scale. S_{data} represents a space- and possibly time-dependent array of values of sea-surface salinity (SSS) and is the only necessary forcing field. If this option is chosen, then a value for the space- and time-*in*dependent restoring time-scale variable `sfwf_restore_tau` (in days) also needs to be specified.

'bulk-NCEP' calculate fluxes based on atmospheric state variables similar to coupled mode (and using bulk flux formulations extracted from the NCAR flux coupler). Necessary forcing fields consist of (in order) SSS and precipitation.If this option is chosen, it is necessary to also choose 'bulk-NCEP' for `shf_formulation` since the evaporation rate (part of the `sfwf-formulation`) must be proportional to the latent heat flux (part of the `shf-formulation`).

`pt,s_interior_formulation` **options:**

'restoring' a simple restoring of the potential temperature or salinity below the top level in the model to a data value, $d(T,S)/dt = (T,S)_{data} - (T,S)_{model}/\tau$, where τ is a constant time scale. Values of the potential temperature or salinity in the entire volume of the ocean ($(T,S)_{data}$) are the only necessary forcing fields. If this option is chosen, then a value for the restoring time scale namelist variable `pt,s_interior_restore_tau` (in days) also needs to be specified. In addition, a value for the namelist variable which specifies the maximum level for which interior restoring is performed (`pt,s_interior_restore_max_level`) is necessary. For example, if `s_interior_restore_tau=365` and `s_interior_restore_max_level=17`, then salinity will be restored to data with a time scale of one year for model levels 2 through 17 everywhere in the ocean.

Having interior restoring occur everywhere in the ocean as described above is more relevant to data-assimilation than to prognostic simulations, so there is support for variable interior restoring specified by
`pt,s_variable_interior_restore = .true.`.
If this option is selected, the user must supply a file
`pt,s_interior_restore_filename`
that contains the maximum model depth for which interior restoring is performed and the inverse restoring timescale (1/days) for each horizontal grid point. This option can be useful for creating graduated 'buffer zones' at the boundaries of non-global models or to set water mass properties due to outflow from unresolved marginal seas.For example, the maximum level for restoring can be set to zero everywhere except for north of 75N where it takes on a nonzero (though not necessarily constant from point to point) value to help create Arctic water masses. To reduce the direct influence of the buffer zone, the inverse restoring time-scale can be tapered from zero at 75N to a finite value at the

northern edge of the grid. Note that the code expects both fields to be double precision, but converts the maximum depth-level field to integer internally to the nearest integer.

3.8.4 Forcing files

If any of the options for `*_data_type` are chosen to be anything besides 'none' or 'analytic', then the user must supply files that contain the appropriate data via the variables `*_filename`. All data files are currently assumed to be double-precision, direct-access files with each record having the dimensions of the full horizontal grid. The data is also assumed to be in a specific order that varies depending on the forcing formulation to be used. For 'annual' and 'n-hour' forcing, one occurrence of each forcing field should be in the file; for 'monthly-calendar' or 'monthly-equal' forcing, all 12 months of each field should be in the file. For example, if the heat flux is 'monthly-calendar' 'Barnier-restoring', and the horizontal grid is 172x128, then the forcing file should have data in the following order:

1. effective_temperature(1:172,1:128,1:12)
2. time_scale(1:172,1:128,1:12)
3. ice_mask(1:172,1:128,1:12)
4. net_shortwave(1:172,1:128,1:12).

If the forcing is 'n-hour' then there needs to be a different file for each forcing time in the sequence. The files are assumed to be labeled by the date of the *middle* of the forcing period; and are of the form '*root.yyyy.ddd.hh*' where '*root*' is specified using `*_filename`, *yyyy* is the year (0000-9999), *ddd* is the day (001-366), and *hh* is the hour (01-24). Note that the dating convention is relative to year 0000, so results may not be what the user expects. For example, with wind stress forcing every 2 days (`ws_data_inc` = 48.), even number years will expect files dated on even days of the year, and odd days for odd numbered years (in the absence of leap years). Thus, the expected sequence of files at the end of year 1492 is (with `ws_filename` = 'ws'): ... *ws.1492.362.00, ws.1492.364.00, ws.1493.001.00, ws.1493.003.00* ... because ws.1492.364.00 refers to forcing covering days 363 and 364 of year 1492, while ws.1493.001.00 refers to forcing covering day 365 of year 1492 and day 1 of year 1493. Makes perfect sense, doesn't it? It is possible to change the labeling date from the middle of the forcing interval to the beginning by changing a flag in the source code.

3.8.5 Forcing units

The code makes assumptions about the units of the fields read in from the forcing files as follows:

potential temperature:	degrees C
salinity:	g/g
wind stress:	dyne/cm^2
restoring time scale:	days
heat flux:	W/m^2
precipitation:	$kg/m^2/s$
air temperature:	degrees K
humidity:	kg/kg
wind speed:	m/s
cloud fraction:	dimensionless, varying from 0 to 1
ice_mask:	dimensionless, varying from 0 to 1

Any input data that isn't in the correct units can be multiplied by a renormalization factor specified by a component in the namelist variable vector `*_data_renorm`. The components of this vector match up with the order of the fields in the forcing file. For example, salinity data sets are often in psu (ppt), while the model expects msu(g/g) = (0.001)psu, so the user can specify `sfwf_data_renorm(1)` = 0.001 in the namelist if `sfwf_formulation` = 'restoring' or 'bulk-NCEP'.

3.8.6 Updating the forcing values

Forcing values are updated based on the value of `*_interp_inc` *and* whether the value of the forcing term depends explicitly on the ocean state, as detailed below.

Wind stress (`ws`) and atmospheric pressure (`ap`) forcing currently do not depend explicitly on the ocean state so the forcing terms in the equations are updated depending on the value of `[ws,ap]_interp_freq`. For example, with `ws_data_type` = 'monthly-calendar', `ws_interp_type` = 'linear', and `ws_interp_freq` = 24., the code will linearly interpolate monthly wind stress values at the beginning of each day and will use this interpolated value for one model day.

Potential temperature and salinity forcing typically do explicitly depend on the ocean state (for example, restoring to climatology depends on the difference between the climatological value and the current model value) so the forcing terms in the equations are evaluated every timestep. However, just like with the wind stress and atmospheric pressure, the value of the data used in calculating the forcing terms depends on the value of `*_interp_freq`. For example, with `pt_interior_data_type` = 'monthly-calendar', `pt_interior_interp_type` = 'linear', `pt_interior_interp_freq` = 48., and `pt_interior_formulation` = 'restoring', the code will linearly interpolate monthly potential temperature values at the beginning of each 2-day interval and will use this interpolated value for two model days when calculating the forcing. Again, the actual forcing term is evaluated every timestep, but the value that is being restored to stays constant over the 2 day period.

3.8.7 Forcing modules

The files **forcing_**[**ws,shf,sfwf,ap,pt_interior,s_interior**]**.F90** are the modules that initialize and calculate the forcing terms. **forcing.F90** is the driver module and **forcing_tools.F90** contains routines shared by all of the individual forcing modules.

3.8.8 Forcing namelists

Wind stress

Table 3.25: Windstress forcing namelist

&forcing_ws_nml		**surface wind stress forcing**
ws_data_type	[‘analytic’], ‘none’, ‘nhour’, ‘annual’, ‘monthly-calendar’, ‘monthly-equal’	type or periodicity of wind stress forcing
ws_data_inc	[10^{20}]	increment (in hours) between forcing times if ws_data_type=‘n-hour’
ws_interp_freq	[‘never’], ‘n-hour’, ‘every-timestep’	how often to temporally interpolate wind stress data to current time
ws_interp_type	[‘nearest’], ‘linear’, ‘4point’	type of temporal interpolation for wind stress data
ws_interp_inc	[10^{20}]	increment (in hours) between interpolation times if ws_interp_freq = ‘n-hour’
ws_filename	[‘unknown-ws’]	name of file containing wind stress, or root of filenames if ws_data_type=‘n-hour’
ws_file_fmt	[‘bin’], ‘nc’	format of wind stress file
ws_data_renorm(20)	[20*1.]	renormalization constants for the components in the wind stress forcing file
/		

Surface heat flux

Table 3.26: Surface heat flux forcing namelist

&forcing_shf_nml		**surface heat flux (SHF) forcing**
shf_formulation	'bulk-NCEP', 'Barnier-restoring', ['restoring']	surface heat flux formulation
shf_data_type	['analytic'], 'none', 'annual', 'n-hour', 'monthly-calendar', 'monthly-equal'	type or periodicity of surface heat flux forcing
shf_data_inc	[10^{20}]	increment (in hours) between forcing times if shf_data_type='n-hour'
shf_interp_freq	['never'], 'n-hour', 'every-timestep'	how often to temporally interpolate surface heat flux data to current time
shf_interp_type	['nearest'], 'linear', '4point'	type of temporal interpolation for surface heat flux data
shf_interp_inc	[10^{20}]	increment (in hours) between interpolation times if shf_interp_freq = 'n-hour'
shf_restore_tau	[10^{20}]	restoring timescale (days) if type restoring
shf_filename	['unknown-shf']	name of file containing surface heat flux data, or root of filenames if shf_data_type='n-hour'
shf_file_fmt	['bin'], 'nc'	format (binary or netCDF) of shf file
shf_data_renorm(20)	[20*1.]	renormalization constants for the components in the surface heat flux forcing file
jerlov_water_type	[3], 1-5	Jerlov water type for shortwave penetration
shf_weak_restore	[0.0]	restoring flux for weak restoring in bulk-NCEP
shf_strong_restore	[92.64]	restoring flux for strong restoring in bulk-NCEP
/		

Surface fresh water flux

Table 3.27: Surface fresh water flux forcing namelist

&forcing_sfwf_nml		**surface fresh-water flux (SFWF) forcing**
sfwf_formulation	'bulk-NCEP', ['restoring']	surface fresh water flux formulation
sfwf_data_type	['analytic'], 'none', 'n-hour, 'annual', 'monthly-equal', 'monthly-calendar'	type or periodicity of surface fresh water flux forcing
sfwf_data_inc	[10^{20}]	increment (hours) between forcing times if sfwf_data_type='n-hour'
sfwf_interp_freq	['never'], 'n-hour', 'every-timestep'	how often to temporally interpolate surface fresh water flux data to current time
sfwf_interp_type	['nearest'], 'linear', '4point'	type of temporal interpolation for surface fresh water flux data
sfwf_interp_inc	[10^{20}]	increment (hours) between interpolation times if sfwf_interp_freq = 'n-hour'
sfwf_restore_tau	[10^{20}]	restoring timescale (days) if restoring
sfwf_filename	['unknown-sfwf']	name of file containing surface fresh water flux data, or root of filenames if sfwf_data_type='n-hour'
sfwf_file_fmt	['bin'], 'nc'	format (binary or netCDF) for sfwf file
sfwf_data_renorm(20)	[20*1.]	renormalization constants for components in sfwf forcing file
ladjust_precip	[.false.], .true.	adjust precipitation to balance water budget
lfw_as_salt_flx	[.false.], .true.	treat fresh water flux as virtual salt flux when using varthick sfc layer
sfwf_weak_restore	[0.092]	restoring flux for weak restoring in bulk-NCEP
sfwf_strong_restore	[0.6648]	restoring flux for strong restoring in bulk-NCEP
/		

Atmospheric pressure

Table 3.28: Atmospheric pressure forcing namelist

&forcing_ap_nml		**atmospheric pressure (AP) forcing**
ap_data_type	‘none’, [‘analytic’], ‘annual’ , ‘n-hour’, ‘monthly-calendar’, ’monthly-equal’	type or periodicity of atmospheric forcing forcing
ap_data_inc	[10^{20}]	increment (in hours) between forcing times if ap_data_type=’n-hour’
ap_interp_freq	[‘never’], ‘n-hour’, ‘every-timestep’	how often to temporally interpolate atmospheric forcing data to current time
ap_interp_type	[‘nearest’],‘linear’, ‘4point’	type of temporal interpolation for atmospheric forcing data
ap_interp_inc	[10^{20}]	increment (in hours) between interpolation times if ap_interp_freq = ‘n-hour’
ap_filename	[‘unknown-ap’]	name of file containing atmospheric forcing, or root of filenames if ap_data_type=‘n-hour’
ap_file_fmt	[‘bin’], ‘nc’	format (binary or netCDF) for ap file
ap_data_renorm(20)	[20*1.]	renormalization constants for the components in the atmospheric pressure forcing file
/		

Interior potential temperature

Table 3.29: Interior potential temperature forcing namelist

&forcing_pt_interior_nml		**potential temperature (pt) forcing at interior points**
pt_interior_formulation	[‘restoring’]	interior pt formulation
pt_interior_data_type	[‘none’], ‘annual’, ‘monthly-calendar’, ‘monthly-equal’, ‘n-hour’	type or periodicity of interior pt forcing
pt_interior_data_inc	[10^{20}]	increment (hours) between forcing times if data_type ‘n-hour’
pt_interior_interp_freq	[‘never’], ‘n-hour’, ‘every-timestep’	how often to temporally interpolate interior pt data to current time
pt_interior_interp_type	[‘nearest’], ‘linear’, ‘4point’	type of temporal interpolation for interior pt data
pt_interior_interp_inc	[10^{20}]	increment (hours) between interpolation times if interp_freq = ‘n-hour’
pt_interior_restore_tau	[10^{20}]	restoring timescale (days) if restoring
pt_interior_filename	[‘unknown-pt_interior’]	file containing interior pt data, or root of filenames if data_type=‘n-hour’
pt_interior_file_fmt	[‘bin’], ‘nc’	file format (binary or netCDF)
pt_interior_data_renorm(20)	[20*1.]	renormalization constants for components in interior pt forcing file
pt_interior_restore_max_level	[0] 0 km	maximum level for interior pt restoring
pt_interior_variable_restore	[.false.]	enable variable interior pt restoring
pt_interior_restore_filename	[‘unknown-pt_interior_restore’]	name of file containing variable interior pt restoring data
pt_interior_restore_file_fmt	[‘bin’], ‘nc’	file format (binary or netCDF)
/		

Interior salinity

Table 3.30: Interior salinity restoring namelist

&forcing_s_interior_nml		**salinity (S) forcing at interior points**
s_interior_formulation	[‘restoring’]	forcing formulation
s_interior_data_type	[‘none’], ‘annual’, ‘monthly-calendar’, ‘monthly-equal’, ‘n-hour’	type or periodicity of interior salinity forcing
s_interior_data_inc	[10^{20}]	increment (hours) between forcing times if data_type ‘n-hour’
s_interior_interp_freq	[‘never’], ‘n-hour’, ‘every-timestep’	how often to temporally interpolate interior S data to current time
s_interior_interp_type	[‘nearest’], ‘linear’, ‘4point’	type of temporal interpolation for interior S data
s_interior_interp_inc	[10^{20}]	increment (in hours) between interpolation times if interp_freq ‘n-hour’
s_interior_restore_tau	[10^{20}]	restoring timescale (days) if restoring
s_interior_filename	[‘unknown-s_interior’]	name of file containing interior S data, or root of filenames if data_type ‘n-hour’
s_interior_file_fmt	[‘bin’], ‘nc’	format (binary or netCDF) of s interior file
s_interior_data_renorm(20)	[20*1.]	renormalization constants for components in interior S forcing file
s_interior_restore_max_level	[0] 0 km	maximum level for interior S restoring
s_interior_variable_restore	[.false.]	enable variable interior S restoring
s_interior_restore_filename	[‘unknown-s_interior_restore’]	name of file containing variable interior S restoring data
s_interior_restore_file_fmt	[‘bin’], ‘nc’	format (binary or netCDF) of variable interior s file
/		

3.9 Running POP in coupled mode

POP is the ocean component of the Community Climate System Model (CCSM2), a community coupled climate model developed by NCAR and many other collaborators. Other coupled models are adopting POP and CICE for their ocean and sea ice components, including groups at Colorado State University and UCLA. Only the interface to the CCSM2 model will be outlined here and supported in the model.

In the CCSM2 model, POP runs as a separate executable and communicates with other models using messages passed to and from a 'flux coupler'. The routines for passing these messages are included in the forcing_coupled module. In addition, this module takes care of all the manipulations (unit conversions, rotations of vector fields) necessary for the form expected by the flux coupler. POP typically sends current state data to the coupler and receives the usual surface forcing fluxes (windstress, heat flux, water flux, solar short-wave), implying that the fluxes were actually computed using ocean state variables from the previous coupling interval. The flux coupler performs all the computations and necessary averaging and interpolation of these quantities.

To run POP in coupled mode, the coupling interface must be activated at compile-time by specifying `-Dcoupled` in the `makefile` (see Sec.2.4.1). Then the following namelist becomes available. Basically, the only quantities under run-time control are the frequency at which the model communicates with the coupler.

Table 3.31: Coupled namelist

&coupled_nml		**activate interface to coupled model**
coupled_freq_opt	['never'], 'nyear', 'nmonth', 'nday', 'nhour', 'nsecond'	unit of time for coupled_freq
coupled_freq	[1]	frequency in above units for bi-directional communication with 'flux coupler'
/		

3.9.1 Real-time X-window display

A rudimentary X-display can be used to monitor the model's behavior as it is running, which can sometimes be useful in locating when and where (on the model grid) things start to go bad. This capability relies on an unsupported, not-necessarily-portable interface to an X library called **fix**. There are no guarantees that this will work on your system or that this will be supported in future releases. Currently, the fields to be viewed in the x-window are hard-wired and can only be changed by changing the source code. The default version of

Table 3.32: Xdisplay namelist

&xdisplay_nml		**real-time display via x-window**
lxdisplay	[.false.]	if .true., enable x-display
nstep_xdisplay	[1]	frequency (in steps) for updating x-display
/		

this module in the source directory is only a stub version with no executable code. If the user wishes to try this option, the actual code resides in the **input_templates** directory in a file called **xdisplay.F90.unsupported** and this file must be copied into the source directory, overwriting the default xdisplay module.

Chapter 4

Model diagnostics and output

While it is possible to run a POP ocean simulation without generating any output at all, most users desire a means of looking at the results of their simulation. POP offers several types of model output with choices governed by several types of model input. The following sections describe all of the options currently available.

4.1 Output formats

POP now supports both netCDF and binary output formats. The format for a particular type of file is chosen at run time through namelist input for each of the output types. In both cases, most output is in single precision (32-bit). The exception is restart files which are written in full double precision (64-bit) format. There are advantages and disadvantages of using each format which are discussed below.

4.1.1 netCDF

The netCDF output format provides a self-describing output file which is portable across machines. It is also recognized by many graphics and post-processing utilities. These are important and very useful features and are the reasons a netCDF option has been included in POP. However, there are two disadvantages to using this format.

The most serious disadvantage with netCDF format is that netCDF does not currently permit parallel I/O; all netCDF operations are funneled through a single processor. For high resolution simulations, this can present a serious performance bottleneck as the model attempts to pass several Gbytes through a single processor. If netCDF output is desired and is proving to be too slow, the user should switch to binary format and convert the binary files off-line to netCDF.

Currently, the data portion of a netCDF file utilizes IEEE binary format. For portability, if a machine does not use IEEE format for its native binary format, netCDF will be performing a conversion (hidden to the user) to this format. In such a case, loss of precision during this conversion will occur and exact restart can not be guaranteed. To avoid this, binary format should typically be chosen for all restart files.

4.1.2 binary

The binary format option creates files in local machine binary format. These files are written as Fortran direct-access files so there are no record headers or footers and the file can be read by other applications as a pure binary stream. Typically, each record in the file contains a horizontal slice of a particular field (so the record length is the size of a horizontal slice of the global data). On parallel machines, fast parallel I/O is achieved by reading/writing each of these slices from a different processor with the number of processors reading/writing data specified in the I/O namelist (see Sec.3.1.2).

Unlike netCDF, these binary files contain no information about themselves (not self-describing) and no information about the fields in the files. To remedy this, each binary file written by POP is accompanied by a 'header file' with the same name as the binary file and an additional suffix '.hdr'. This header file is an ASCII file which contains all the information you would find in a netCDF file, including file attributes, fields in the file and attributes of those fields. As part of the field attributes, the starting record of the field in the binary file is included. Such a header file provides some of the capability of a self-describing data format and also provides information for easy conversion to netCDF (or other self-describing format).

Because binary formats differ across machines, binary files are not typically portable across machines. To achieve portability, the user is encouraged to convert the binary files to a more portable data format like netCDF.

4.2 File-naming convention

All output file names have the form: `root-filename.runid.time-indicator.output-format`. Here `root-filename` is a name defined by the user through namelist input for each file type, `runid` (see Sec. 3.1.3) is a character identifier for the run sequence, `time-indicator` depends on the output frequency chosen and `output-format` is either 'nc' for netCDF files or 'bin' for binary files.If the output file is written at a frequency of `nday, nmonth` or `nyear`, the `time-indicator` is typically the date in yyyymmdd (where yyyy, mm and dd can be optionally separated by a character defined in the time manager namelist). If the output file is written every `nstep` steps, the `time-indicator` is the step number.

A convention has been developed for naming POP output files in which the root filenames for *restart, time-averaged history, snapshot history,* and *movie*files are simply the one-letter names 'd', 't', 'h', 'm' respectively. These are the default values in the namelists for each of these options, but the user is not required to follow this convention. Come up with your own convention. We don't mind. Really. Have fun with it. The only requirement is that the `root-filename` must be distinct for each type of output file. Because '.' is used to delimit `runid` in file names and '/' is reserved as a separator in Unix path names, neither should be used within `runid`.

4.3 Model diagnostics

The progress of a POP simulation is recorded in a 'log file' that is either written to standard output (stdout) or redirected to an optional log file (see Sec.3.1.2). A new log file is created each time the model is started. Values of the model version number, release date, date and time of the run, input namelist parameters, and initial or restart conditions are documented at the beginning of each log file. After the introductory information, the log file will contain output from model timers and all the scalar diagnostics.

There are three types of scalar diagnostics available. Global diagnostics compute a variety of globally-averaged values of tracers, kinetic energy and several tendencies for checking balances. CFL diagnostics compute the Courant numbers for advection and diffusion terms. Transport diagnostics compute mass, heat and salt transports through various regions defined in an input file.

These diagnostics are chosen by setting the frequency `freq_opt` at which each diagnostic is computed. If diagnostics are chosen, the diagnostics are written both to the log file (or stdout) and to a separate diagnostics output file. Monitoring this output as the model runs is a useful way of making sure the model is behaving reasonably. For example, the Courant numbers reported in the CFL diagnostics should remain small and various tendencies reported in the global diagnostics should remain balanced.

In addition to printing these diagnostics to a log file, the diagnostics are printed to other output files in a format more suitable for various graphics programs. The output files are ASCII files with each line containing `tday` (decimal time in days for the entire simulation), the value of the diagnostic and a name for the diagnostic. The name of these output files can be changed using the variables `diag_outfile, diag_transport_outfile`.

4.3.1 Transport diagnostics

Computing transport diagnostics requires an input file describing the requested transports. A sample transport input file is provided with the distribution. The transport input file must contain the number of transport regions in the first record. Each following record must contain 6 integers `imin, imax, jmin, jmax, kmin, kmax` which are global array indices which define the transport

Table 4.1: Diagnostics namelist

&diagnostics_nml		**generation of model diagnostics**
diag_global_freq_opt	[‘never’], ‘nstep’, ‘nyear’, ‘nmonth’, ‘nday’, ‘nhour’, ‘nsecond’	units of time for ’diag_global_freq’
diag_global_freq	[100000]	how often (in above units) to compute and print global diagnostics
diag_cfl_freq_opt	[‘never’], ‘nstep’, ‘nyear’, ‘nmonth’, ‘nday’, ‘nhour’, ‘nsecond’	units of time for ’diag_cfl_freq’
diag_cfl_freq	[100000]	how often (in above units) to compute and print CFL stability diagnostics
diag_transp_freq_opt	[‘never’], ‘nstep’, ‘nyear’, ‘nmonth’, ‘nday’, ‘nhour’, ‘nsecond’	units of time for ’diag_transp_freq’
diag_transp_freq	[100000]	how often (in above units) to compute and print transport diagnostics
diag_transport_file	‘sample_transport_file’	input filename (with path) describing requested transports
diag_outfile	‘pop_diag’	file to which global and cfl diagnostics are to be written
diag_transport_outfile	‘pop_transp’	file to which transport diagnostics are to be written
diag_all_levels	[.false.]	if true, tracer mean diagnostics at all vertical levels are output
cfl_all_levels	[.false.]	if true, cfl diagnostics at all vertical levels are output
/		

Table 4.2: Transport descriptions

imin	imax	jmin	jmax	kmin	kmax	section	label
64	64	1	128	1	20	merid	sample meridional section
1	192	64	64	1	10	zonal	sample zonal section

region. Note that the region must be a plane so that one of the horizontal dimensions must be fixed.Following these integers must be a 5-character string that specifies 'zonal' or 'merid' transport. Note that this descriptor defines the orientation of the *section* and not the direction of the velocity normal to it. A 'zonal' section implies the transport across that section uses the meridional velocity (velocity perpendicular to the section). The last item in each record is a name for the transport region (e.g. 'Drake Passage').

4.4 Model output files

In this section, we often refer to 'model dates' and 'model times' as corresponding to actual dates and times in the real world. This is only true if the model initial time was set appropriately and only has true meaning if the model is forced by actual observed forcing fields with proper dates associated with them. Otherwise, 'model time' simply refers to the model's internal calendar.

4.4.1 Time-averaged history files

The namelist `tavg_nml` controls the frequency and content of time-averaged history files. These files are constructed by accumulating in memory at each time-step the running sums of selected variables or correlation of variables. Consequently, time averaging can be very memory intensive and may not be feasible on your computer. Snapshot history files (see Sec.4.4.2) provide an alternative, but at the price of having to recall many files from archival storage to compute the sums. The `tavg_freq` determines both the frequency at which the files are written as well as the interval over which the time average is to be performed.

Because the time averages are running averages, `tavg` restart files are written whenever a model restart file is written so that the averaging can continue upon restart. Note that the fields in the output files are normalized by the accumulated time since the start of the time average. The time interval used for this normalization is output as the file attribute `tavg_sum`. When the model restarts from a restart file, the sums are de-normalized before continuing the accumulated sum.

The user may also control when the time averaging will begin. For example, if the time averaging should be started after the model has equilibrated, the user can specify when time averaging should start through the `tavg_start` variables. The choices are similar to the model start options.

Table 4.3: Time-average file namelist

&tavg_nml		**generation of time-average history files**
tavg_freq_opt	['never'], 'nyear', 'nmonth', 'nday', 'nhour', 'nsecond', 'nstep'	units of time for 'tavg_freq'
tavg_freq	[100000]	interval in above units for computation and output of time-average history files
tavg_start_opt	['nstep'], 'nyear', 'nmonth', 'nday', 'date'	units for tavg_start ('date' implies yyyymmdd)
tavg_start	[100000]	time in above units after which to start accumulating time average
tavg_infile	['unknown']	restart file for partial tavg sums if starting from restart (ignored if luse_pointer_files is enabled)
tavg_fmt_in	['bin'],'nc'	format for tavg restart file (binary or netCDF)
tavg_outfile	['unknown']	root filename (with path) for tavg output files (suffixes will be added)
tavg_fmt_out	['bin'],'nc'	format for tavg output files (binary or netCDF)
tavg_contents	'sample_tavg_contents'	file name for input file containing names of fields requested for tavg output
/		

IMPORTANT: Before a new run-sequence is begun, careful thought should be given to the contents of the time-average history files. The same considerations apply to snapshot history and movie files. Although it is possible to redefine the contents at any time during the sequence, this is not recommended. Changing the contents can greatly complicate the process of combining short-interval (e.g., monthly) files into longer-interval files, such as seasonal, annual and multi-year composite files.

For time-averaged output, a `tavg_contents` file is required containing a simple list (one field per line) of accepted short names for all the fields desired for the output file. A `sample_tavg_contents` file is supplied containing a large list of fields available for tavg output. It is meant for the user to use as a template, modifying it for their own needs by deleting entries or adding new ones. If a user wishes to add a field that is currently not available, the user must modify the code to add that field using other available fields as a template.

Table 4.4: Current available tavg fields

Name	**Units**	**Description**
SHF	W/m^2	Surface Heat Flux
SFWF	mm/day	Surface Freshwater Flux (p-e)
SSH	cm	Sea Surface Height
H2	cm^2	SSH^2
H3	unitless	$(\Delta_x(SSH))^2 + (\Delta_y(SSH))^2$
TAUX	$dyne/cm^2$	Zonal windstress
TAUY	$dyne/cm^2$	Meridional windstress
UVEL	cm/s	Zonal Velocity
VVEL	cm/s	Meridional Velocity
KE	cm^2/s^2	Horizontal Kinetic Energy $(U^2 + V^2)/2$
TEMP	$^\circ C$	Potential Temperature
SALT	g/g	Salinity
TEMP2	$^\circ C^2$	Temperature^2
SALT2	$(g/g)^2$	Salinity^2
UET	$^\circ C/s$	East Flux of Heat
VNT	$^\circ C/s$	North Flux of Heat
WTT	$^\circ C/s$	Top Flux of Heat
UES	g/g/s	East Flux of Salt
VNS	g/g/s	North Flux of Salt
WTS	g/g/s	Top Flux of Salt
UEU	cm/s^2	East Flux of Zonal Momentum
VNU	cm/s^2	North Flux of Zonal Momentum
UEV	cm/s^2	East Flux of Meridional Momentum
VNV	cm/s^2	North Flux of Meridional Momentum
PV	1/s	Potential Vorticity
Q	g/cm^4	z-derivative of potential density
PD	g/cm^3	Potential density referenced to surface
UDP	erg	Pressure work
PEC	g/cm^3	Potential energy release due to convection
NCNV	adjustments/s	Convective adjustments per second
WTU	cm/s^2	Top flux of Zonal Momentum
WTV	cm/s^2	Top flux of Meridional Momentum
ST	$^\circ Cg/g$	Temperature*Salinity
RHO	g/cm^3	In-situ density

4.4.2 Snapshot history files

If sufficient memory is not available for run-time accumulation of time-averaged history files or if the user simply needs an instantaneous view of the ocean state, snapshot history files can be written at regular intervals. The interval must be short enough that whatever time-averaging interval may be desired in the future, there will be three or more samples (snapshots) per averaging-interval. For monthly averages, snapshots should be collected at intervals of 10 days or less. Only one time-level of the prognostic variables and selected diagnostic variables needs to be saved, since second-moments and correlations can be computed later from these snapshot files, at the cost of retrieving a potentially large number of snapshot files. To choose which fields are written to the history files, a history contents file must be supplied with a list of fields desired. A sample file is included which contains all the currently available fields; the user may edit this file to select the desired fields. If a field is not currently included in the list of available fields, the user must modify the source code to make that field available (see the Reference Manual).

Table 4.5: History file namelist

&history_nml		**Generation of snapshot history files**
history_freq_opt	['never'],'nstep', 'nyear', 'nmonth', 'nday', 'nhour', 'nsecond'	units of time for history_freq
history_freq	[100000]	number of units between output of snapshot history files
history_outfile	['unknown']	root filename with path of history file output (suffixes will be added)
history_fmt	['bin'],'nc'	format for history files (binary or netCDF)
history_contents	['history_contents']	input file containing names of fields requested to be output
/		

Table 4.6: Currently available history fields

Name	**Units**	**Description**
SHGT	cm	surface height
UBTROP	cm/s	barotropic ‘zonal’ velocity
VBTROP	cm/s	barotropic ‘meridional’ velocity
UVEL	cm/s	‘zonal’ velocity
VVEL	cm/s	‘meridional’ velocity
TEMP	$^{\circ}C$	potential temperature
SALT	g/g (msu)	salinity
SUF	cm^2/s^2	surface velocity flux in U direction
SVF	cm^2/s^2	surface velocity flux in V direction
SHF	W/m^2	surface heat flux
SFWF	m/year	surface fresh water flux
SOLAR	W/m^2	solar short wave flux at surface

4.4.3 Movie files

One of the most exciting aspects of ocean simulation is the opportunity to visualize and animate the evolution of the model variables in time and space. Making movies that progress smoothly requires either output of model variables frequently enough to avoid jerkiness or temporal interpolation of model variables to similarly frequent intervals. Experience has shown that a snapshot every three days (see `movie_freq`) gives satisfactory results. Any variable can be output, but to reduce archiving cost and retrieval time, movie files typically contain only a few two-dimensional arrays, such as sea-surface temperature, salinity and height, and a few sub-surface variables. The choice of fields is made through an input movie contents file containing the names (one per line) of the fields requested. A sample file is included with a list of available fields; the user may modify this list to choose the desired fields. If a field does not appear in the list of available fields, the user may add fields by modifying the source code as described in the Reference Manual.

Table 4.7: Movie file namelist

&movie_nml		**generation of snapshot movie files**
movie_freq_opt	[‘never’], ‘nstep’, ‘nyear’,‘nmonth’,‘nday’, ‘nhour’,‘nsecond’	units of time for movie_freq
movie_freq	[100000]	number of units between output of movie files
movie_outfile	[’unknown’]	root filename with path of movie file output (suffixes will be added)
movie_fmt	[‘bin’],‘nc’	format for movie file output (binary or netCDF)
movie_contents	[‘sample_movie_contents’]	input file containing names of fields requested for movie output
/		

Table 4.8: Currently available movie fields

Name	**Units**	**Description**
SHGT	cm	surface height
UTRANS	cm^2/s	vertically integrated ‘zonal’ transport
VTRANS	cm^2/s	vertically integrated ‘meridional’ transport
TEMP1_2	$^{\circ}C$	potential temperature averaged over levels 1,2
SALT1_2	g/g (msu)	salinity averaged over levels 1,2
TEMP6	$^{\circ}C$	potential temperature at level 6
SALT6	g/g (msu)	salinity at level 6
VORT	1/s	relative vorticity at surface

4.4.4 Current meters, drifters and hydrographic sections.

Versions of these exist for some machines but these are not yet included in the present version of the code. They will be added soon.

Chapter 5

POP tools

This chapter describes various tools available for creating POP input or analyzing POP output.

5.1 Visualizing output

There is currently no standard tool for visualizing POP output. There are two freely-available software packages that can be used.

Ferret is a visualization tool developed by Steve Hankin at NOAAs Pacific Marine Environmental Laboratory (PMEL) in Seattle. It is designed specifically for visualizing ocean model output and data. Ferret is constantly being improved and extended, and there is a very active email-based user group. For more information about Ferret, check the Ferret website

http://ferret.wrc.noaa.gov/Ferret/.

Other free visualization and analysis packages are available from the DOE-sponsored Program for Climate Model Diagnosis and Intercomparison (PCMDI) at Lawrence Livermore National Laboratory (LLNL). Although originally designed for visualizing atmospheric model output from the Atmospheric Model Intercomparison Project (AMIP), many are applicable to ocean model output and future ocean analysis tools are being added. For more information, check the website

http://www-pcmdi.llnl.gov/software/.

5.2 Transformations from general grids

All POP output is on the computational grid. For general grids that are not based on latitude and longitude (e.g. the displaced-pole or tripole grids), analyzing or visualizing data leads to a distorted view of the world and colleagues may begin to question your geography. Transformation to latitude-longitude grids can be performed using the Spherical Remapping and Interpolation Package (SCRIP), available from http://www.acl.lanl.gov/lanlclimate/SCRIP/.

5.3 File format conversion

POP now supports direct netCDF output and such output is recommended unless it creates a performance bottleneck for large grids. In cases where binary output is required, conversion to netCDF can be achieved off-line through a utility called bin2nc. Unfortunately, this utility is being re-worked to handle the new binary output format so is unavailable at this time. It would be relatively easy for a user to create such a code however.

5.4 Generating grid and bottom topography files

5.4.1 Horizontal grid and topography

A graphical tool for generating horizontal grids and creating bottom topography has been developed. A working version is currently in the process of being released and will support almost-global Mercator grids, global displaced-pole grids, global tripole grids and regional grids. Some standard grids and topography used in current production runs are available from the POP website.

5.4.2 Vertical grid

There is a vertical grid generating routine in the tools/grids subdirectory. This code is essentially the same code used to generate vertical grids internally in POP, but can be used to generate a vertical grid file off-line which can be edited to suit your simulation. Note that when changing the vertical grid, you will need to re-generate the bottom topography.

Chapter 6

Trouble-shooting

If you encounter problems gettingPOP to run, a list of known bugs is maintained at the POP website and the user should check there before contacting the developers. In addition, there are other common sources of error listed below with suggestions for avoiding such problems.

6.0.1 Timestep stability criteria

As part of the POP diagnostics, the Courant number for various processes can be output. The Courant number should always be less than one and often should be kept under 0.5 for stability. If such a CFL condition is exceeded, reducing the timestep may be a solution. However, it should be noted that often other underlying problems may be causing these limits to be exceeded (e.g. unrealistically high velocities) and reducing the timestep will only prolong the agony. In such cases, the particular CFL condition that is violated can help pinpoint the source of the problem.

6.0.2 Checking the global energy and work balances

6.0.3 Getting the right combination of mixing options

6.0.4 Getting the right combination of forcing options

6.0.5 Properly normalizing input salinity values

The units for salinity in the model are g/g. Care must be taken to renormalize salinity data, which is often in g/kg.

6.1 Running POP on PCs

POP has been successfully run on a single-processor Intel PC with Windows NT. POP was built with Digital Visual Fortran 5.0 as a Win32 console application (meaning you run it with a command line in a DOS window). The make system used for other platforms was not used on the PC. Instead, a standard Visual Studio project was created.

Source files added to the project were

- All .F90 and .C files in source subdirectory
- All .F90 files in serial subdirectory

In addition to the defaults, these Fortran preprocessor options '/fpp' was needed. To enable POP to run the test problem without stack overflow, reserve stack memory was set to 64 MB. A lower value may be possible, but 48 MB was not enough. The debug version required the link option: /nodefaultlib:libc. The release version (optimized) ran the test problem in about four minutes (262 seconds) on a 450 MHz, 256 MB Pentium II PC.

Chapter 7

References

Bibliography

[1]

[2] B. Barnier, L. Siefridt, and P Marchesiello. Thermal forcing for a global ocean circulation model using a three-year climatology of ecmwf analyses. *J. Mar. Syst.*, 6:363–380, 1995.

[3] J.A. Brown and K.A. Campana. An economical time-differencing system for numerical weather prediction. *Mon. Weather Rev.*, 106:1125–1136, 1978.

[4] K. Brown. A numerical method for the study of the circulation of the world ocean. *J. Comput. Phys.*, 4:347, 1969.

[5] R.M. Chervin and A.J. Semtner Jr. An ocean modeling system for supercomputer architectures of the 1990s. In M. Schlesinger, editor, *Proc. of the NATO Advanced Research Workshop on Climate-Ocean Interaction*. Kluwer, Dordrecht, 1988.

[6] J. K. Dukowicz and R. D. Smith. Implicit free-surface method for the bryan-cox-semtner ocean model. *J. Geophys. Res.*, 99:7991–8014, 1994.

[7] J.K. Dukowicz, R. D. Smith, and R.C. Malone. A reformulation and implementation of the bryan-cox-semtner ocean model on the connection machine. *Atmos. Ocean. Tech.*, 10:195–208, 1993.

[8] Eby and Holloway. Need a reference here. *Journ. Phys. Oceanogr.*, 24:2577–2588, 1994.

[9] L.-L. Fu and R. D. Smith. Global ocean circulation from satellite altimetry and high-resolution computer simulation. *Bull. Amer. Meteor. Soc.*, 11:2625–2636, 1996.

[10] P.R. Gent. The heat budget of the toga-coare domain in an ocean model. *J. Geophys. Res.*, 96:323–333, 1991.

[11] P.R. Gent and J.C. McWilliams. Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.*, 20:150–155, 1990.

[12] D.R. Jackett and T.J. McDougall. Minimal adjustment of hydrographic profiles to achieve static stability. *JTECH*, 12:381–389, 1995.

[13] W.G. Large, J.C. McWilliams, and S.C. Doney. Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Reviews of Geophysics*, 32:363–403, 1994.

[14] B.P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Comp. Meth of Geophysics*, 32:363–403, 1994.

[15] M. E. Maltrud, R.D. Smith, A.J. Semtner, and R.C. Malone. Global eddy resolving ocean simulations driven by 1985-1994 atmospheric winds. *J. Geophys. Res.*, 103:30825–30853, 1998.

[16] Wright Jackett McDougall, T.J. and Feistel. unknown. *JTECH*, 2001.

[17] H. Peters, M.C. Gregg, and J.M. Toole. On the parameterization of equatorial turbulence. *J. Geophys. Res.*, 93:1199–1218, 1988.

[18] A.J. Semtner Jr. Finite-difference formulation of a world ocean model. In J.J. O'Brien, editor, *Advanced Physical Oceanographic Numerical Modeling*. Reidel, Dordrecht, 1986.

[19] R.D. Smith, J.K. Dukowicz, and R.C. Malone. Parallel ocean general circulation modeling. *Physica D*, 60:38–61, 1992.

[20] R.D. Smith, M.E. Maltrud, F.O. Bryan, and M.W. Hecht. Numerical simulation of the north atlantic ocean at 1/10. *J. Phys. Oceanogr.*, page ??, 1999.