Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7320--08-9149

Software Design Description for the Navy Coastal Ocean Model (NCOM) Version 4.0

PAUL MARTIN
CHARLIE N. BARRON
LUCY F. SMEDSTAD
ALAN J. WALLCRAFT
ROBERT C. RHODES
TIMOTHY J. CAMPBELL
CLARK ROWLEY

Ocean Dynamics and Prediction Branch
Oceanography Division

SUZANNE N. CARROLL

Planning Systems, Inc.
Stennis Space Center, Mississippi

December 31, 2008

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
31-12-2008

2. REPORT TYPE
Memorandum Report

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE

Software Design Description for the Navy Coastal Ocean Model (NCOM)
Version 4.0

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0603207N

6. AUTHOR(S)

Paul Martin, Charlie N. Barron, Lucy F. Smedstad, Alan J. Wallcraft, Robert C. Rhodes, Timothy J. Campbell, Clark Rowley, and Suzanne N. Carroll*

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
73-5091-18-5

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Oceanography Division
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/7320--08-9149

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Space & Naval Warfare Systems Command
2451 Crystal Drive
Arlington, VA 22245-5200

10. SPONSOR / MONITOR'S ACRONYM(S)

SPAWAR

11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

*Planning Systems, Inc., Stennis Space Center, MS 39529

14. ABSTRACT

The purpose of this Software Design Description (SDD) is to describe the software design and code of the Navy Coastal Ocean Model Version 4.0 (NCOM). It includes flow charts and descriptions of the NCOM programs, subprograms, and common blocks. This document, along with the User's Manual and two Validation Test Reports forms a comprehensive documentation package for the NCOM 4.0. A User's Manual for the Global Ocean Prediction System (GOPS) is also available.

15. SUBJECT TERMS

NCOM	COAMPS	NOGAPS	Relocatable ocean model
Global ocean model	MODAS	Coupled ocean model	Ocean model

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
Unclassified	Unclassified	Unclassified	UL	149	Lucy F. Smedstad
					19b. TELEPHONE NUMBER *(include area code)* (228) 688-5365

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

SOFTWARE DESIGN DESCRIPTION

FOR THE

NAVY COASTAL OCEAN MODEL (NCOM)

VERSION 4.0

September 2008

Prepared for:
Naval Research Laboratory
Ocean Modeling Division

Prepared by:
Paul Martin
Charlie N. Barron
Lucy F. Smedstad
Alan J. Wallcraft
Robert C. Rhodes
Timothy J. Campbell
Clark Rowley
Naval Research Laboratory

Suzanne N. Carroll
Planning Systems, Incorporated
Stennis Space Center, MS 39529

NRL/MR/7320--08-9149
Approved for public release. Distribution unlimited.

TABLE OF CONTENTS

TABLE OF FIGURES

1.0 SCOPE

1.1 Identification

The Navy Coastal Ocean Model (NCOM) Version 4.0 is based primarily on two existing ocean circulation models, the Princeton Ocean Model (POM) (Blumberg and Mellor 1983; Blumberg and Mellor 1987) and the Sigma/Z-level Model (SZM) (Martin et al., 1998). NCOM Version 4.0 has a free-surface and is based on the primitive equations and the hydrostatic, Boussinesq, and incompressible approximations. The Mellor Yamada Level 2 (MYL2) and MYL2.5 turbulence models are provided for the parameterization of vertical mixing. The vertical mixing enhancement scheme of Large et al. (1994) is also offered for parameterization of unresolved mixing processes occurring at near-critical Richardson numbers. The inclusion of a source term in the model equations allows for the input of river and runoff inflows.

The model uses a staggered Arakawa C grid (as in POM). Spatial finite differences are mostly second-order centered (as in POM), but there are options to use higher-order spatial differences for some terms. The temporal scheme is leapfrog, with an Asselin filter to suppress timesplitting (as in POM). Most terms are treated explicitly in time, but the propagation of surface waves and vertical diffusion are treated implicitly.

The horizontal grid is orthogonal-curvilinear (as in POM). NCOM 4.0 has two choices of vertical grid, which are selected at compile time. One choice is the original vertical grid used by NCOM, which is a hybrid sigma and z-level grid with sigma coordinates used from the surface down to a specified depth and level coordinates used below the specified depth. The switch from sigma to level coordinates can occur at any specified interface between layers, i.e., from just below the uppermost layer (there must be at least one sigma layer at the surface) to the bottom of the lowest layer (in which case the entire grid would be sigma coordinate, as in POM). On the sigma coordinate portion of the grid, each sigma layer is a fixed fraction of the depth from the surface to the bottom of the sigma coordinate grid. This fractional depth may vary for different sigma layers, but cannot change within a particular layer. On the level portion of the grid, each layer's depth and thickness is fixed and the bottom depth is adjusted to match the depth of the nearest layer.

The second, newer, choice of vertical grid is a general vertical coordinate (GVC) grid. The GVC grid consists of a three-tiered vertical grid structure comprised of: (1) a "free" sigma grid near the surface that expands and contracts with the movement of the free surface, (2) a "fixed" sigma grid that does not move with the free surface, and (3) a *z*-level grid that allows for "partial" bottom cells. For both the "free" and "fixed" sigma grids, the fractional layer thickness can be specified independently for each grid cell and the land-sea masking can be different for different sigma layers. The vertical grid structure can consist of just (1), or (1) and (2), or (1) and (3), or (1), (2), and (3). This new vertical grid structure allows for more flexibility on both the sigma and *z*-level portions of the grid. For the sigma grid, the fractional layer thickness can vary both horizontally and vertically (i.e., it can be specified independently at each model grid pt) and masking can be used on the sigma grid to mask land areas and reduce the number of active sigma layers. For the *z*-level grid, grid cells at the bottom can be made "partial" cells so that the *z*-level grid can match the true bottom depth. In addition, a "fixed" sigma grid that does not expand and contract with the movement of the free surface can be used between the "free" sigma grid near the surface and the (fixed) *z*-level grid. However, the increased flexibility of the generalized vertical grid comes at the cost of a 15-20% increase in the required memory storage and CPU time. Also, the use of "partial" *z*-level cells involves increased numerical truncation error because of the abrupt change in grid-layer thickness at a "partial" grid cell. The "classic" sigma grid, where each layer is a fixed fraction of the total depth of the sigma grid, has some numerical advantages over the generalized sigma grid.

The NCOM surface boundary conditions are the surface stress for the momentum equations, the surface heat flux for the temperature equation, and the effective surface salt flux for the salinity equation. The bottom boundary conditions are the bottom drag for the momentum equations, which is parameterized by a quadratic drag law, and zero flux for the temperature and salinity equations.

NCOM provides for an arbitrary number of levels of nesting. This nesting capability is made possible by using dynamic memory allocation with array dimensions specified at run time and by passing model variables to subroutines through subroutine argument lists rather than through common blocks. This allows the same model routines to calculate the different nests.

1.2 Document Overview

The purpose of this Software Design Description (SDD) is to describe the software design and code of the Navy Coastal Ocean Model Version 4.0 (NCOM). It includes flow charts and descriptions of the NCOM programs, subprograms, and common blocks. This document, along with the User's Manual (Martin et al, 2008) and two Validation Test Reports (Barron et al., 2007, 2008) form a comprehensive documentation package for the NCOM 4.0 delivery. A User's Guide for the Global NCOM Nowcast/Forecast model, called the Global Ocean Forecast System (GOFS), is also available (Smedstad et al., 2008).

2.0 REFERENCED DOCUMENTS

2.1 NCOM Software Documentation

Barron, C.N., A.B. Kara, R.C. Rhodes, C. Rowley, and L.F. Smedstad, (2007). "Validation Test Report for the 1/8° Global Navy Coastal Ocean Model Nowcast/Forecast System." *NRL Tech Report*, NRL/MR/7320—07-9019, Naval Research Laboratory, Stennis Space Center, MS.

Barron, C.N., R.W. Helber, T.L. Townsend, L.F. Smedstad, and J.M. Dastugue, (2008). "Validation Test Report: MLD-Modified Synthetics and NCODA Profile Assimilation in Global NCOM." *NRL Tech Report*, submitted, Naval Research Laboratory, Stennis Space Center, MS.

Martin, P.J., (2000). "Description of the Navy Coastal Ocean Model Version 1.0." NRL/FR/7322—00-9962, Naval Research Laboratory, Stennis Space Center, MS.

Martin, P.J., C.N. Barron, L.F. Smedstad, T.J. Campbell, A.J. Wallcraft, R.C. Rhodes, C. Rowley, T.L. Townsend, and S.N. Carroll, (2008). "User's Manual for the Navy Coastal Ocean Model (NCOM) Version 4.0." NRL/MR/7320--08-9151, Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS.

Posey P.G., L.F. Smedstad, R.H. Preller, E.J. Metzger and S.N. Carroll, (2008). "Software Design Description for the Polar Ice Prediction System (PIPS) Version 3.0", NRL/MR/7320--08-9150, Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS.

Smedstad L.F., T.L. Townsend, C.N. Barron, T.J. Campbell, P.J. Martin, P.G. Posey, R.C. Rhodes, and S.N. Carroll, (2008). User's Guide for the Global Ocean Forecast System (GOFS) Version 2.6" NRL/MR/7320--09-????, Ocean Modeling Division, Naval Research Laboratory, Stennis Space Center, MS. In progress.

2.2 General Technical References

Akima, H., (1970). A New Method of Interpolation and Smooth Curve Fitting Based on local Procedures. *J. Ass. For Computing Machinery*, 17(4): 589-602.

Barron, C.N., A.B. Kara, P.J. Martin, R.C. Rhodes, and L.F. Smedstad. (2006): Formulation, implementation and examination of vertical coordinate choices in the Global Navy Coastal Ocean Model (NCOM). *Ocean Modelling*, 11: 347-375.

Barron, C.N., Rhodes, R.C., Smedstad, L.F., Rowley, C.D., Martin, P.J., and Kara, A.B. (2003) Global ocean nowcasts and forecasts with the Navy Coastal Ocean Model (NCOM). *NRL Review*, 175-178.

Blumberg, A. F. and Mellor, G. L., (1983). Diagnostic and prognostic numerical circulation studies of the South Atlantic Bight. *J. Geophys. Res.*, 88: 4579- 4592.

Blumberg, A. F. and Mellor, G. L., (1987). "A description of a three-dimensional coastal ocean circulation model." In: Three-Dimensional Coastal Ocean Models. N. Heaps, ed., American Union, New York, N.Y., p. 208.

Collins-Sussman, B., Fitzpatrick, B.W., and Pilato, C. M. "Version Control with Subversion." [Online]. Copyright © 2002, 2003, 2004, 2005, 2006, 2007 O'Reilly Media Inc, Sebastopol, CA. <http://subversion.tigris.org/>.

Fox, D.N., W.J. Teague, M.R. Carnes, C.M. Lee, and C.N. Barron. (2002). The Modular Ocean Data Assimilation System (MODAS), *J. Atmos. Oceanic Technol.*, 19: 240-252.

Friedrich, H. and Levitus, S., (1972). An approximation to the equation of state for sea water, suitable for numerical ocean models. *J. Phys. Oceanogr.*, 2: 514-517.

Gibson, J.K., P. Kållberg, S. Uppala, A. Hernandez, A. Nomura, and E. Serrano, 1997: ERA description. ECMWF Re-Analysis Project Report Series, No. **1,** 72 pp. [Available from ECMWF, Shinfield Park, Reading RG2 9AX, UK.]

Kara, A.B., P.A. Rochford, H.E. Hurlburt. (2000). Efficient and accurate bulk parameterizations of air-sea fluxes for use in general circulation models. *J. Atmos. Ocean Tech*. 17: 1421-1438.

Kara, A.B., P.A. Rochford, and H.E. Hurlburt, (2002). Air-sea flux estimates and the 1997-1998 ENSO event. *Bound.-Layer Meteor*. 103: 439-458.

Large, W. G., McWilliams, J. C., and Doney, S., (1994). Oceanic vertical mixing: a review and a model with a nonlocal boundary layer parameterization. *Rev. Geophys*., 32: 363-403.

Martin, P. J., Peggion, G., and Yip, K. J., (1998). "A comparison of several coastal ocean models." NRL Report NRL/FR/7322--97-9692. Naval Research Laboratory, Stennis Space Center, MS., 96.

Mellor, G. L., (1991). An equation of state for numerical models of oceans and estuaries. *J. Atmos. and Ocean Tech*., 8: 609-611.

Morel, A., (1988). Optical modeling of the upper ocean in relation to its biogenous matter content (Case I waters). *J. Geophys. Res.*, 93: 10749-10768.

Neumann, G. and W.J. Pierson, Jr. (1966). Principles of Physical Oceanography, Prentice-Hall, Inc. Englewood Cliffs, NJ.

Rosmond, T.E., J. Teixeira, M. Peng, T.F. Hogan, and R. Pauley, (2002). Navy Operational Global Atmospheric Prediction System (NOGAPS): Forcing for ocean models. *Oceanography*, 15: 99-108.

Smith, R. C. and Baker, K. S., (1981). Optical properties of the clearest natural waters (200-800 nm). *Appl. Optics*, 20(2): 177-184.

Urick, R. J., (1975). Principles of Underwater Sound, 1st ed., McGraw-Hill Publishing Co. New York, pp. 99, 102, 105.

Vincenty, T. (1975). Direct and Inverse Solutions of Ellipsoid on the Ellipsoid with Application of Nested Equations, *Survey Review*, XXII (176): 88-93.

2.3 Recommended Reading

Asselin, R. A., (1972). Frequency filter for time integrations. *Mon. Weather Rev*., 100: 487-490.

Barron, C.N. and L.F. Smedstad, (2002). Global River Inflow within the Navy Coastal Ocean Model, Proceedings to Oceans 2002 MTS/IEEE Meeting, 29-31 October 2002.

Bird, R. E., (1984). A simple spectral model for direct normal and diffuse horizontal irradiance. *Solar Energy*, 32: 461-471.

Bleck, R., Rooth, C., Hu, D., and Smith, L. T., (1992). Salinity-driven thermocline transients in a wind-and-thermohaline-forced isopycnic coordinate model of the North Atlantic. *J. Phys. Oceanogr.*, 22: 1486-1505.

Blumberg, A. F., (1992). A Primer for ECOM-si. Technical Report, HydroQual, Inc., Mahwah, N.J., 64 pp.

Bryan, K., (1969). A numerical method for the study of the circulation of the World Ocean. *J. Comput. Phys.*, 4: 347-376.

Buck, A. L., (1981). New equations for computing vapor pressure and enhancement factor. *J. Appl. Meteor.*, 20: 1527-1532)

Casulli, V. and Cattani, E., (1994). Stability, accuracy, and efficiency of a semi-implicit method for three-dimensional shallow water flow. *Comp. and Math. with Appl.*, 27: 99-112.

Casulli, V. and Cheng, R. T., (1994). "Solutions of primitive equations for three-dimensional tidal circulation." In: Estuarine and Coastal Modeling III. *Proc. of the 3rd Int. Conf.*, ASCE, New York, N.Y., pp. 396-406.

Casulli, V. and Stelling, G. S., (1996). "Simulation of three-dimensional, non-hydrostatic free-surface flows for estuaries and coastal seas." In: Estuarine and Coastal Modeling. *Proc. of the 4th Int. Conf.*, M.L. Spaulding and R.T. Cheng, eds., ASCE, New York, N.Y., pp. 1-25.

Craig, P. D. and Banner, M. L., (1994). Modeling wave-enhanced turbulence in the ocean surface layer. *J. Phys. Oceanogr*., 24: 2546-2559.

Craig, P. D., (1996). Velocity profiles and surface roughness under breaking waves. *J. Geophys. Res.*, 101: 1265-1277.

Dietrich, D. E. and Ko, D. S., (1994). A semi-collocated ocean model based on the SOMS approach. *J. Num. Methods in Fluids*, 19: 1103-1113.

Dukowicz, J. K. and Smith, R. D., (1994). Implicit free-surface method for the Bryan-Cox-Semtner ocean model. *J. Geophys. Res*., 99: 7991-8014.

Fofonoff, N. P., (1962). "Physical properties of seawater." In: The Sea: Ideas and observations on progress in the study of the seas. Physical Oceanography. M.N. Hill, ed., Wiley, Interscience, New York, Vol.1 pp. 9.

Garratt, J. R., (1977). Review of Drag Coefficients over Oceans and Continents. *Monthly Weather Review*, 105(7): 915-929.

Gill, A. E., (1982). Atmosphere-Ocean Dynamics. Academic Press, New York, p. 662.

Haney, R. L., (1974). A numerical study of the response of an idealized ocean to large-scale surface heat and momentum flux. *J. Phys. Oceanogr.*, 4: 145-167.

Haney, R. L., (1991). On the pressure gradient force over steep topography in sigma coordinate ocean models. *J. Phys. Oceanogr*., 21: 610-619.

Hodur, R. M., (1997). The Naval Research Laboratory's Coupled Ocean/Atmosphere Mesoscale Prediction System (COAMPS). *Mon. Wea. Rev.*, 125: 1414-1430.

Hurlburt, H. E. and Thompson, J. D., (1980). A numerical study of Loop Current intrusions and eddy shedding. *J. Phys. Oceanogr*., 10: 1611-1651.

Hyland, R.W., (1975). A correlation for the second interaction virial coefficients and enhancement factor for moist air, *J. Res. Natl. Bur. Stand,* (79A):551.

Jerlov, N. G., (1968). Optical Oceanography. Elsevier Publishing Co., New York.

Kantha L. H. and Clayson, C. A., (1994). An improved mixed layer model for geophysical applications. *J. Geophys. Res.*, 99: 25235-25266.

Killworth, P. D., Stainforth, D., Webb, D. J., and Paterson, S. M., (1991). The development of a free-surface Bryan-Cox-Semtner ocean model. *J. Phys. Oceanogr*., 21: 1333-1348.

Large, W. G., and Pond, (1982). Sensible and latent heat flux measurements over the ocean. *J. Phys. Oceanogr.*, 12: 464-482.

Leendertse, J. J., (1989). A new approach to three-dimensional free-surface flow modeling. The RAND Corporation Memorandum R-3712-NETH/RC, Santa Monica, CA. List, R. J., (1951). Smithsonian Meteorological Tables. Washington: Smithsonian Institute. pps. 290-295, 347, 350.

Lumb, F. E., (1964). The influence of cloud on hourly amounts of total solar radiation at the sea surface. *Quarterly Journal of the Royal Met Soc.*, 90: 43-56.

Martin, P. J., (1985). Simulation of the ocean mixed layer at OWS November and Papa with several models. *J. Geophys. Res*., 90: 903-916.

Martin, P. J., (1986). "Testing and Comparison of Several Mixed-Layer Models." NORDA Report 143. Naval Research Laboratory, Stennis Space Center, MS., pp. 30.

Mellor, G. L. and Yamada, T., (1974). A hierarchy of turbulence closure models for planetary boundary layers. *J. Atmos. Sci*., 31: 1791-1806.

Mellor, G. L. and Durbin, P. A., (1975). The structure and dynamics of the ocean surface mixed layer. *J. Phys. Oceanogr.*, 5: 718-728.

Mellor, G. L. and Yamada, T., (1982). Development of a turbulence closure model for geophysical fluid problems. *Geophys. and Space Phys*., 20: 851-875.

Mellor, G. L. and Blumberg, A. F., (1985). Modeling vertical and horizontal diffusivities with the sigma coordinate system. *Mon. Wea. Rev*., 113: 1379-1383.

Mellor, G. L., (1996). User's Guide for a Three-Dimensional, Primitive-Equation, Numerical Ocean Model. Princeton University, Princeton, N.J., pp. 39.

Muellor, J. L. and Lange, R.E., (1989). Bio-optical provinces of the Northeast Pacific Ocean: A provisional analysis. *Limnol. Oceanogr*., 34: 1572-1586.

Orszag, S. A., (1971). Numerical simulation of incompressible flows within simple boundaries: accuracy. *J. Fluid Mech.*, 49: 75-112.

Paul, J. F., (1994). "Observations related to the use of the sigma coordinate transformation for estuarine and coastal modeling studies." In: Estuarine and Coastal Modeling III. *Proc. of the 3rd Int. Conf.*, M. Spaulding, K. Bedford, A. Blumberg, R. Cheng, and C. Swanson, eds., ASCE, New York, N.Y., p. 682.

Pietrzak, J. D., (1995). A comparison of advection schemes for ocean modeling. Report 95-8 of the Danish Meteorological Institute, Copenhagen, Denmark, 45 pp.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., and Metcalf, M., (1997). Numerical Recipes in Fortran 90: The art of parallel scientific computing. 2nd ed. Vol. 2. Numerical Recipes Software, U.S., p. 659.

Rood, R. B., (1987). Numerical advection algorithms and their role in atmospheric transport and chemistry models. *Rev. Geophys*., 25: 71-100.

Smagorinsky, J., (1963). General circulation experiments with the primitive equations, Part I: The basic experiment. *Mon. Wea. Rev*., 91: 99-164.

Troen, I. B. and Mahrt, L., (1986). A simple model of the atmospheric boundary layer; sensitivity to surface evaporation. *Boundary Layer Meteorol*., 37: 129-148.

Wallcraft, A. J., (1991). "The Navy Layered Ocean Model Users' Guide." NOARL Report 35, Naval Research Laboratory, Stennis Space Center, MS.

Wexler, A., (1976). Vapor pressure formulation for water in range 0 to 100 C. A Revision. *J. Res. Natl. Bur. Stand,* 80A: 775.

Wilson, W.D., (1960). Speed of sound in sea water as a function of temperature, pressure, and salinity. *J. Acoust. Soc. Am*., 34: 641.

3.0 MODEL DESIGN DECISION

The goal for initial development of NCOM was to make use of well established ocean modeling techniques and to incorporate improvements and additional capabilities into NCOM as needed. It may not be possible to meet every Navy coastal modeling requirement with a single model, but the approach is to make NCOM as flexible as possible without incurring a significant penalty in terms of efficiency.

NCOM is set up so that the main model program requires little or no alteration to run a particular simulation, as almost everything needed for a model simulation is passed in via input files. A setup program is required to generate the input files for regional domains. It is recommended that the user modify one of the existing setup programs that are available.

4.0 MODEL ARCHITECTURAL DESIGN

4.1 Model Components

a) NCOM can be divided into several software units that include routines for NCOM setup, input files, communication routines, and routines specific to running simulations on different computer platforms. These are briefly described along with commonly used library subroutines and data libraries required for smooth operation of NCOM.

b) RELO_NCOM – A setup program (*RELO_NCOM*) is used to generate the input files for regional simulations. This program is considered to be in the domain of the user, i.e., the setup program must be modified by the user to set up a particular simulation. Most of the model input files are read and written in program ***ncom1rwio.F***. The same subroutine is used to read and write a particular file so that the code for reading and writing the file is in the same place and the read and write instructions can be kept more consistent. All of the subroutines in ***ncom1rwio.F*** have an initial parameter which is either set to 1 (read) or 2 (write). The input/output files are either IEEE binary or ASCII files.

c) GENERAL DIRECTORY STRUCTURE- The model code directory (*ncom_4.0*) contains all of the files needed to generate the NCOM executable. A typical structure of the directory is as follows:

ncom_4.0/

Makefile.ncom	Top-level Makefile for NCOM.
Makefile -	Secondary- level Makefile for NCOM.
README.txt	Compiling and running a simulation.
README.make	NCOM build information.

bin/- Directory for NCOM executable(s). The executables are placed in subdirectories that follow the naming convention described in Section 4.1.2.

config/- Configuration and makefile fragments used for compiling NCOM code. Each makefile fragment is set up for some combination of a (i) specific machine architecture (NCOM_ARCH) (ii) compiler (NCOM_COMP), and (iii) user-specific (NCOM_USER) options.

doc/- Directory of Readme documentation/explanation files.

ncom_changes.txt List of NCOM errors and changes.
ncom_guide.txt User’s guide for NCOM 4.0
README.version- Description of NCOM version number string.
README.<xxx> Symbolic link to specific README on <xxx>.

include/- NCOM include files that are included via cpp (These are now using suffix *.h rather than *.inc).

CAF.h- Co-Array Fortran I/O.
COAMPS.h- Common block to store info about ocean/atm model grid for COAMPS.
COAMPS_parms.h COAMPS parameter include file.
COMMON.h- Common blocks for NCOM.
Dsetnl.h- COAMPS directory path include file.
HEADER_MPI.h- MPI header on generic machine.
HEADER_MPI_AIX.h- MPI header on IBM SP.
HEADER_MPI_T3E.h- MPI header on Cray T3E.
MACROS.h- Macros for customizing NCOM.
NCOMPAR.h- Common blocks for NCOM subroutine OMODEL.
Omnl.h and omnloff.h- COAMPS ocean model namelist include files.
PARAM.h- Compile-time constants for NCOM.
README.include- Help file for includes.
README.macros- Help file for macros in **MACROS.h**.

lib/- Directory of NCOM compiled libraries- Libraries are placed in subdirectories that follow the naming convention described in Section 4.1.2.

sigz.global/-
libncom.a - Compiled library of all NCOM subroutines.
libncom_setup.a - Compiled library of all NCOM setup subroutines.

libsrc/- Directory of all NCOM Fortran subroutine files.

Makefile- Makes compiled libraries containing collections of NCOM Fortran files and puts libraries on lib/ directory.
cdf/- Contains a set of netCDF specific subroutines.

coampslib/- Subroutines for working with COAMPS fields.
Makefile- Makefile to compile local source code.
datar.F- Reads COAMPS-style flat files.
datar_new.F- Reads COAMPS-style flat files.
dataw.F- Writes COAMPS-style flat files.
dataw_new.F- Writes COAMPS-style flat files.

dfalts.F-	Returns information about the specified-input field name, e.g., default contour interval, max/min color shading bar values.
grdcon.F-	Calculates the grid constant for input grid projection and grid parameters.
grdij.F-	Generates real grid index values.
ij2ll.F-	Computes lat/lon from real grid index values for specified grid projection and parameters.
ll2ij.F-	Computes real grid index coordinates from lat/lon values for specified grid projection and parameters.
rdata.F-	Gets information for specified input field.
rotang.F-	Calculates angle of grid with respect to local lat/lon for specified grid.
s2hms.F-	Converts from s to hour, min, sec.
slen.F-	Gives the size of a character string.
uvg2uv.F-	Converts grid u/v to earth-oriented u/v, i.e., with u directed eastward and v directed northward.
wdata.F -	Writes data field to COAMPS-style flat file.

esmf/- Directory of ESMF routines.

Makefile-	Makefile to compile local source code.
ncom1esmf.F-	NCOM ESMF Module.

fnoclib/- Directory of main FNMOC routines and include files.

Makefile-	Makefile to compile local source code.
bessel.F-	General 2D bessel interpolation.
cctop.F-	Converts fields from vector to Polar (magnitude and direction) form.
ch2int.F-	Gets integer numerical value from integer character string.
dfuv.F-	Converts vectors from earth-oriented direction and magnitude to u/v form on a conic grid projection.
differs.F-	Perform operations performed on two input fields depending on value of input flag.
dtgchk.F-	Checks if DTG is valid.
dtgdif.F-	Returns difference in hours of two input DTGs.
dtgmod.F-	Returns new DTG given base DTG and increment in hours.
dtgnum.F-	Given DTG, returns integer values of year, month, day, hour, days into the year, and hours into the year.
dtgops.F-	Returns three types of DTG.
edge.F-	Performs next-to-edge processing for low-pass filter.

fintrp.F-	Interpolates input field values.
gcpnts.F-	Computes evenly spaced lat/lon points along a great circle path between two input lat/lon locations.
gent.F-	Gets a single entry from a HRLS table.
getls.F-	Reads a HRLS table from ISIS or UNIX files.
imaxcv.F-	Computes *imax* from *colcnt* and *rowcnt*.
int2ch.F-	Converts an integer to an left-justified character string.
ioinq.F-	Uses Fortran "Inquire" statement to give info for user in tracking the action of the program I/O.
isint.F-	Tests if a character string contains only digits and a possible sign.
jmaxcv.F-	Computes *jmax* from *colcnt* and *rowcnt*, depending on *stordsc*.
leapyr.F-	Checks to see if input year is a leap year.
lndavg.F-	Computes values for flagged pts in a 2D field as averages of surrounding non-flagged pts.
lpf.F-	Low-pass 2D filter.
niddf.F-	Computes the value of variables, given 1D arrays and independent variables.
ocord.F-	Reads file containing instructions for outputting model fields in flat file format.
pctocc.F-	Converts vector fields from dir and mag to u/v form.
qprint.F-	Quick prints parts of a gridded field.
rlpnts.F-	Computes grid index locations of evenly-spaced x/y pts along a straight line on the grid.
strcmpr.F-	Tests to see that two char strings match, disregarding whether letters are upper or lower case.
strleft.F-	Deletes leading white space from a char string, left-justifying the string.
strlen.F-	Computes the length of an input string.
strnot.F-	Finds the first location in an input string that is not a blank.
strpars.F-	Extracts substrings from a char string.
unstrgr.F-	Unstaggers a staggered gridded field.
uvdf.F-	Converts from u/v on a conic grid to earth-oriented speed and direction.

misc/- Directory of miscellaneous NCOM subroutines.

Makefile-	Makefile to compile local source code.
allocate.F-	Allocates the no. of array elements needed.
cubspl_irr.F-	Cubic spline interp. for irregular output grid.
gc_ellipsoid.F-	Returns distances in m, azimuth angle in deg.

ocubspl_irr.F- Old cubic spline interp. for irreg. output grid.
padarr.F- Embeds model horiz. grid into comp. horiz. grid.
tablk2s.F- Interpolates value from 2D array using linear interp.
timesubs.F- Time subroutines.
w_ncomnc.F- Writes NCOM data into a netCDF file.
w_ncomnc2.F-Writes NCOM data into a netCDF file.
w_rgb.F- Converts real array *f* to an output rgb file.

ncom/- Directory of NCOM main Fortran subroutines.
Makefile- Makefile to compile local source code.
ncom1.F- Routines to set up memory for NCOM and integrate the ocean model in time***(***except for driver module, which is in file ***ncom.F*** **i**n directory *src/ncom/*.
ncom1baro.F- Routines to update free-surface.
ncom1coam.F-Routines to get surface air-sea flux fields from COAMPS atmospheric model flat file output.
ncom1fct_gvc.F- Routines for advection of scalar fields using FCT to avoid advective overshoots- GVC grid.
ncom1fct_sigz.F- Routines for advection of scalar fields using FCT-sig-z grid.
ncom1init_gvc.F-Routines to initialize ocean model-GVC grid.
ncom1init_sigz.F-Routines to initialize ocean model-sig-*z* grid.
ncom1nest2.F-Routines to interpolates boundary conditions for and provide feedback from nested grids.
ncom1obc_gvc.F-Routines to handle OBCs-GVC grid.
ncom1obc_sigz.F- Routines to handle OBCs -sig-*z* grid.
ncom1out_gvc.F-Routines to output model results- GVC grid.
ncom1out_sigz.F- Routines to output model results -sig-*z* grid.
ncom1plib.F- Generic routines from Paul Martin's library *plib*.
ncom1rwio.F- Routines to read/write I/O files.
ncom1sbc.F- Routines to obtain surface forcing.
ncom1tide.F- Routines to provide tidal forcing.
ncom1updt_gvc.F- Main update routines for u, v, T, S- GVC grid.
ncom1updt_sigz.F- Main update routines for u, v, T, S-sig-*z* grid.
ncom1util.F- Utility routines used for testing, etc.
ncom1vmix_gvc.F-Routines to compute vertical mixing-GVC grid.
ncom1vmix_sigz.F- Routines to compute vertical mixing-sig-z grid.

pdum/- Directory for dummy NCOM routines, e.g., plotting.
Makefile – Makefile to compile local source code.
ncom1pdum.F-Dummy plotting routines for NCOM when interactive NCAR graphics are not available.

r10k/- Fortran routines specific to SGI Origin 2000.
Makefile - Makefile to compile local source code.
wtime.c- NCOM routine to calculate wall time on SGIs.

zunder.c- NCOM routine to flush underflows to zero on SGIs.

setup/- General routines to support setting up a simulation and post process output.

Makefile - Makefile to compile local source code.

ncom_setup_plib_gvc.F-General routines for setting up a simulation-GVC grid.

ncom_setup_plib_sigz.F-General routines for setting up a simulation-sig-z grid.

ncom_setup_spln.F- Spline interpolation routines from D. S. Ko.

sunw/- Fortran routines specific to Sun Ultra 2 workstations.

Makefile - Makefile to compile local source code.

wtime.c- NCOM routine to calculate wall time on Sun systems.

util/- Directory of communication routines for shared memory (SM) and multi-processor (MP) computing.

Makefile- Makefile to compile local source code.

README.xmc- Brief descriptions of all communication routines.

README.za- Brief descriptions of machine-specific routines.

xmc.F- Select between ***xmc_mp.F*** and ***xmc_sm.F***.

xmc_mp.F- Communication routines for multiple processors.

xmc_sm.F- Communication routines for shared memory computer.

za.F- Select between ***za_mp.F*** and ***za_sm.F***.

za_mp.F- I/O routines for multiple processors.

za_sm.F- I/O routines for shared memory computer.

mod/- Directory of compiled NCOM Fortran modules. The modules are placed in subdirectories that follow the naming convention described in Section 4.1.2.

sigz.global/- Contains compiled global NCOM Fortran modules.

src/-

Makefile-

esmf/-

ncom.F- ESMF driver for stand-alone NCOM.

ncom/-Directory for NCOM driver and makefile to make the executable.

Makefile - Compile ***ncom.F***, link executable and put on /*bin*.

ncom.F - Main driver routine for NCOM.

test_xca/-

Makefile- Makefile to build program ***test_xca.F***.

test_xca.F- Program to test xctilr.

test_xcl/-

Makefile- Makefile to build program ***test_xcl.F***.

test_xcl.F- Program to test xclget and xclg3d.

4.2 NCOM Build Information

README.make contains essential NCOM build information. GNUmake is required for the NCOM build. Note that on some platforms GNUmake is referenced as "gmake". The build targets include the following:

- ncom: builds NCOM libraries, modules and executables.
- libs: builds NCOM libraries and modules only.
- setup: builds NCOM library and modules only, without halos.
- clean: removes build specific libraries, modules and executables.
- clobber: removes all libraries, modules and executables.
- info: prints information about build settings.
- help: (default) prints help information about build.

For compiling simulations, NCOM_ARCH is set to the appropriate machine type, NCOM_COMP (the compiler). The NCOM_USER variable refers to user specific compile settings that are available in the appropriate **config/$(NCOM_ARCH).$(NCOM_COMP).$(NCOM_USER).mk** makefile fragment.

4.2.1 Required Build Variables

There are some required build variables that must be set either on the compile line or in the user environment:

- *NCOM_ARCH* (platform/architecture):

This variable must be the name as specified by the available platform-specific default configuration: config/$(NCOM_ARCH).$(NCOM_COMP).default.mk. Each platform architecture file found in the /config directory contains compiler options for each machine and each Subversion branch. A directory is then made under /bin with the grid type and Subversion branch name.

- *NCOM_COMP* (compiler set):

This build variable is required only when more than one compiler set is available for the selected platform *NCOM_ARCH*. If only one compiler is available for the selected platform *NCOM_ARCH*, then *NCOM_COMP* is automatically set to '`default`'.

4.2.2 Optional build variables

These optional build variables may be set either on the compile line or in the user environment.

- *NCOM_COMM* (communication protocol):

- Choices are:
  'mpi' = Message Passing Interface (MPI).
  'shmem' = Cray/SGI shared memory programming model (SHMEM) (only available on platforms that support SHMEM).
  'one' = single processor (no external communication library required)
- Default is 'mpi'.

- If build target is setup, then *NCOM_COMM* is overridden and set to 'one'.

- *NCOM_PREC* (floating point precision):

- Choices are:
  'r4' = single precision (4-byte real).
  'r8' = double precision (8-byte real).
- Default is 'r4'.

- *NCOM_BOPT* (optimization):

- Choices are:
  'O' = optimized (optimization settings are defined in the platform/compiler specific makefile fragment.
  'g' = debug.
- Default is 'O'.

- *NCOM_VERT* (vertical coordinate code):

- Choices are:
  'sigz' = enable sigma-z vertical coordinate code.
  'gvc' = enable generalized vertical coordinate code.
- Default is 'sigz'.

- *NCOM_USER* (user specific settings):

- Settings defined in config/$(NCOM_ARCH).$(NCOM_COMP).$(NCOM_USER).mk
- This makefile fragment is included after the default makefile fragment and can be used to override or add to the default settings.

- *NCOM_ESMF* (build with Earth System Modeling Framework, ESMF):

- Variable need only be defined to enable ESMF (for example, NCOM_ESMF=y).
- Requires variable *ESMF_DIR* (location of ESMF install) be set either on command line or in user environment.

- *NCOM_DEV* (enable developer build options):

- Variable need only be defined to enable (for example, NCOM_DEV=y).
- Currently, this only affects the names of the subdirectories where executables, libraries and modules are placed.

The executables, libraries and modules for a build are placed in separate subdirectories that are named according to the optional build variables.

Executables are placed in: `'bin/$(BUILD_ID)'`
Libraries are placed in: `'lib/$(BUILD_ID)'`
Modules are placed in: `'mod/$(BUILD_ID)'`

The default definition of BUILD_ID is:

```
  BUILD_ID = '$(NCOM_VERT).$(NCOM_USER)'
```

When the developer build option is enabled (i.e., NCOM_DEV is defined), then BUILD_ID is defined as:

```
  BUILD_ID =
'$(NCOM_COMP).$(NCOM_COMM).$(NCOM_PREC).$(NCOM_BOPT).$(NCOM_VERT)
.$(NCOM_USER)'
```

Here are some examples of the resulting BUILD_ID for various build options:

```
make ncom NCOM_ARCH=amd64 NCOM_COMP=pgi
                ==>  BUILD_ID = 'sigz.default'
make ncom NCOM_ARCH=amd64 NCOM_COMP=pgi NCOM_VERT=gvc
                ==>  BUILD_ID = 'gvc.default'
make ncom NCOM_ARCH=amd64 NCOM_COMP=pgi NCOM_PREC=r8 NCOM_BOPT=g
NCOM_DEV=y
                ==>  BUILD_ID = 'pgi.mpi.r8.g.sigz.default'
```

The NCOM (non-ESMF) executable is named 'ncom.exe'.
The NCOM-ESMF (stand-alone) executable is named 'ncom_esmf.exe'.

Note: See file *ncom_4.0/doc/README.make* for more discussion.

4.3 Code Modifications

Several code modifications have been made from the original NCOM Version 1.0. For a complete history of all code changes made, refer to **ncom_guide.txt** in the \ncom\4.0\doc folder. The most recent changes are summarized below.

4.3.1 Changes from NCOM 2.6 to NCOM 4.0 (up to 12-26-2007)

- Merged 2.6 (sigma-z) and 3.4 (GVC) versions into single version. This change only affects libsrc/ncom, libsrc/setup and the build system.
- A new C-preprocessor macro called "GVC" is used to select the sigma-*z* code or the GVC code at compile time. The user input build variable NCOM_VERT (=sigz or =gvc) is used to determine the type of build. The default is NCOM_VERT=sigz.
- The name of the subdirectories for executables, libraries and modules is modified to include the NCOM_VERT string.
- Source files particular to the type of vertical coordinate system have either "_sigz" or "_gvc" added to the name of the file.
- Other source files that have subroutines dependent on the coordinate system choice use the GVC C-preprocessor macro to enable the correct subroutines.
- The top level module (libsrc/ncom/***ncom1.F***) uses the GVC C-preprocessor macro to enable the correct array allocation and subroutine calls that are particular to the vertical coordinate system choice.

- There are changes to the build system interface. The build of multiple internal libraries has been changed to a single library named ***libncom.a*** or ***libncom_setup.a*** (depending on which target is selected). A "setup" target has been added (i.e., make setup) for building the setup version of the library and modules.

4.3.2 NCOM Sub-Version Repository

NCOM developers at NRL routinely make improvements, changes and bug fixes to the model, often simultaneously. Therefore, they have created an NCOM Subversion Repository (http://subversion.tigris.org/; Collins-Sussman et al., 2007), whereby different versions of NCOM and the complete developmental history are stored and available for user access. The internet address for the repository is https://www7320.nrlssc.navy.mil/svn/repos/NCOM. For web browser (read-only) viewing, via WebSVN, the repository is available at https://www7320.nrlssc.navy.mil/svn/websvn.

The repository is accessible to NRL-SSC personnel as well as to select DoD IP addresses outside the NRL-SSC system, such as HPCMP MSRC platforms. A user account must be requested from and created by Tim Campbell (tim.campbell@nrlssc.navy.mil). Send Dr. Campbell a digitally signed email request and he will reply with an encrypted email containing a username and initial password. After receiving the initial password, go to https://www7320.nrlssc.navy.mil/svn/websvn and click on the "Change Your SVN Password" link to change the password.

4.4 Concept of Execution

The execution of NCOM consists of three main steps 1) making the NCOM executable, 2) setting up a particular simulation, and 3) running the simulation.

A flow diagram illustrating the basic logic underlying the operation of NCOM is shown in **Figure 4.4-1**.

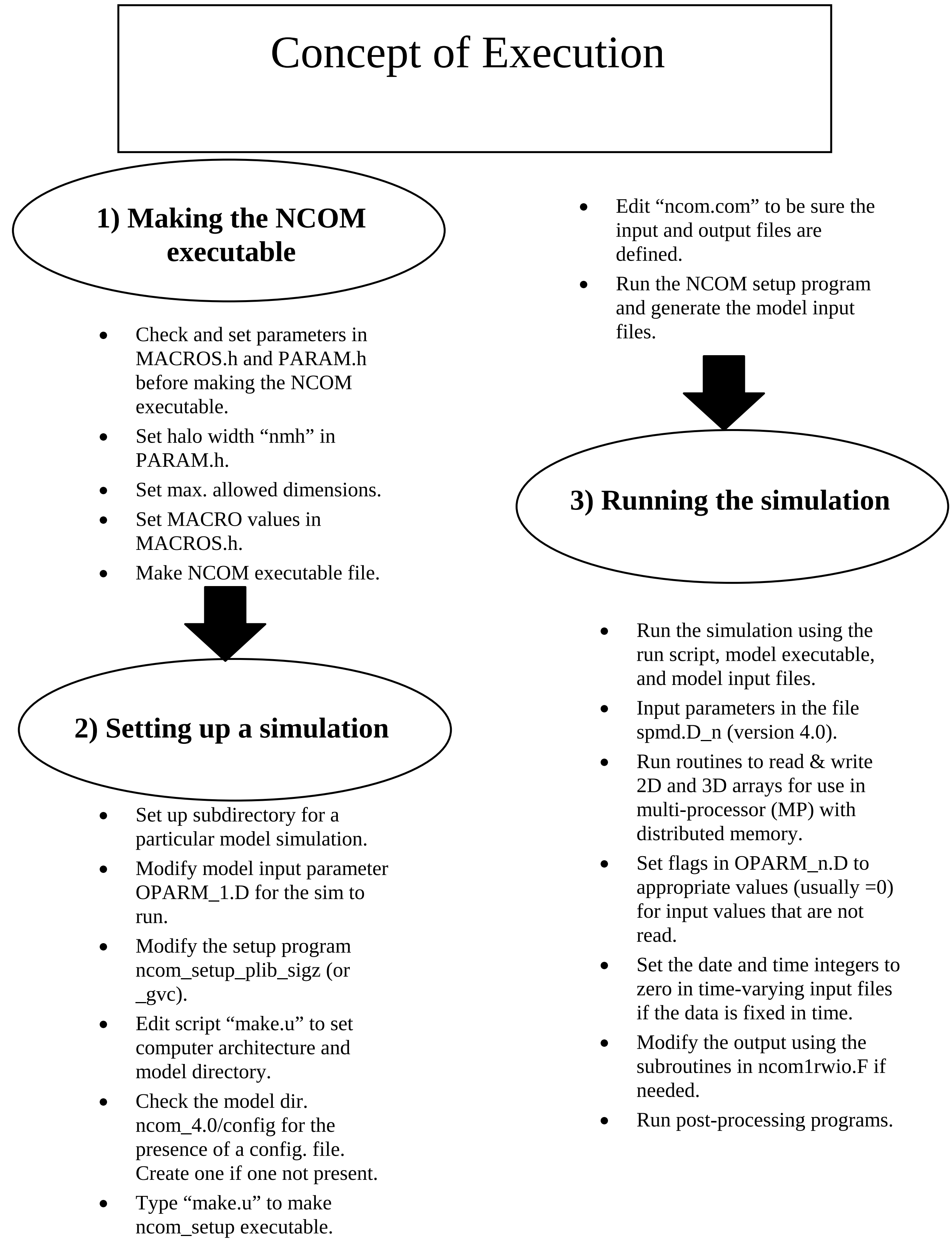

Figure 4.4-1: Flow diagram describing the execution of the NCOM.

4.5 Interface Design

4.5.1 Interface Identification and Diagrams

The only Navy standard NCOM external interfaces are the input and output files. Tables 4.5-1 and 4.5-2 below list the input and output files and give a description of their contents.

Table 4.5-1: List and description of NCOM input files.

File	Description	Unit Number
IOS_tidetbl.D	General tidal constituent info, e.g., tidal frequencies, node factors, phase corrections, etc.	
OPARM_1.D	Input parameters and options.	99+100*nest
odimens.D	Grid and array dimensions for all the grids (nests).	99
oextd_n.A	Array data for solar extinction (chl or K490 values).	99+100*nest
oextd_n.B	Scalar data for solar extinction (chl or K490 values).	
ohgrd_n.A	Array data for horizontal grid.	99+100*nest
ohgrd_n.B	Scalar data for horizontal grid.	
oinit_n.A	Array data for initial conditions.	99+100*nest
oinit_n.B	Scalar data for initial conditions.	
opnbc_n.D	Data for open boundaries.	41+100*nest
orivs_n.D	River inflow data.	42+100*nest
osflx_n.A	Array data for surface forcing fields.	31+100*nest
osflx_n.B	Scalar data for surface forcing fields.	
ossst_n.A	Array data for SST and SSS relaxation.	
ossst_n.B	Scalar data for SST and SSS relaxation.	
ossss_n.A	Array data for SSS relaxation.	
ossss_n.B	Scalar data for SSS relaxation.	
otloc_n.D	List of sections for which transports are to be output.	99+100*nest
otide_n.B	List of constituents for which tidal BC data are supplied.	
otide_n.D	Tidal BC data (tidal constituent elevation and velocity data at the model open boundary points).	99+100*nest
otpcn_n.D	List of tidal constituents for which tidal potential is calculated.	
otscl_n.A	Array data for T-S climatology.	99+100*nest
otscl_n.B	Scalar data for T-S climatology.	
otsf_n.A	Array data to which 3D T and S fields are to be relaxed.	35+100*nest

File	Description	Unit Number
otsf_n.B	Scalar data to which 3D T and S fields are to be relaxed.	
owrlx_n.A	Array data for relaxation timescale (3D).	99+100*nest
owrlx_n.B	Scalar data for relaxation timescale.	
osstf_n.A	Array data for which 2D SST and SSS values are to be relaxed.	33+100*nest
osstf_n.B	Scalar data for which 2D SST and SSS values are to be relaxed.	
otsza_n.A	Array data for horizontally averaged T and S fields.	99+100*nest
otsza_n.B	Scalar data for horizontally averaged T and S fields.	
outpt_n.D	List of grid indices for points at which model results are output.	99+100*nest(.A)
ovgrd_n.A	3D array data for static depth to the top of each grid cell.	
ovgrd_n.B	Scalar data describing the vertical grid.	
ovgrd_n.D	1D array of static interface depths for *z*-level grid.	99+100*nest
owmdf_n.D	List of water mass definitions for which volumes are to be calculated.	99+100*nest
ozout_n.D	List of depths at which fields are to be output.	99+100*nest
stop.D	Stop file, used to pause an interactive run to allow inspection of model fields.	99
spmd.D_n	Parameters describing the processor layout used for running on multiple processors.	99

Table 4.5-2: The output files and their description.

File	Description	Unit Number
out3d_n.A	Array data for 3D output fields.	51+100*nest
out3d_n.B	Scalar data for 3D output fields.	51+100*nest
outsf_n.A	Array data for 2D surface output fields.	52+100*nest
outsf_n.B	Scalar data for 2D surface output fields.	52+100*nest
knrgy_n.D	Volume averaged kinetic energy.	56+100*nest
otran_n.D	Transport through specified sections.	57+100*nest
pt_nn.D	Profiles of model fields at a specified point (pt number *nn*).	61-98+100*nest

5.0 NCOM DETAILED DESIGN

The following sections give a detailed description of the purpose, variables, logic, and constraints for the sigma-z version of NCOM 4.0. The GVC version contains similar subroutines with slight changes in the variables and code for each. Descriptions of the common blocks are found in Appendix A. Argument definitions for some of the most common subroutine variables are found in Appendix B. All routines are written in FORTRAN 90.

5.1 Constraints and Limitations

NCOM Version 4.0 is based on fairly well tested ocean model physics and numerics. However, there are a number of limitations of the model.

1. Since the model is hydrostatic, vertical motions on small horizontal scales may not be properly described. This does not prevent the model from being applied with high horizontal resolution to examine the structure of predominantly horizontal flows. However, non-hydrostatic processes that can occur in these situations will not be correctly simulated.
2. Sigma coordinates can accurately represent the changing bottom depth but can suffer from truncation errors in their horizontal advection, diffusion, and baroclinic pressure gradient terms if steep bottom slopes are not adequately resolved. The solution to this problem is to increase the horizontal grid resolution or artificially decrease the severity of the slope. The problem of numerical truncation error with sigma coordinates can sometimes be reduced using generalized sigma coordinates in which the sigma layers in the upper part of the water column are specified to be nearly level or to have reduced slope. This can be especially helpful if the strongest stratification occurs where the sigma coordinate slopes are small, so that the baroclinic pressure gradient errors are also small.
3. The *z*-level grid does not suffer from these problems but has limitations of its own. Since the *z*-level grid used in the original NCOM grid configuration rounds the bathymetry to the nearest *z*-level, the accuracy of the representation of the bathymetry on this *z*-level grid depends on the vertical grid resolution. The stepwise structure of this *z*-level grid can cause some distortion of flows that cross the steps and does not provide very consistent resolution in the bottom boundary layer unless a large number of levels are used over the depth range at which the bottom boundary layer exists. The bottom z-level grid cells used in NCOM's newer GVC vertical grid configuration can be truncated to match the true bathymetry, so that bottom depths are accurately represented. However, this grid still will not generally provide consistent resolution in the bottom boundary layer.
4. The second-order centered advection scheme provides fairly good accuracy for advection of fields in which the gradients are well resolved, but can generate advective overshoots at sharp fronts. The third-order upwind advection scheme tends to have less overshoot problems than the second-order scheme and generally does a better job of advection. However, in steeply sloping sigma layers these higher-order schemes can have more severe truncation error problems than the second-order schemes. Hence, it is recommended that second-order schemes be used if the bottom slopes are steep and not well resolved. There is an option to use a flux-corrected transport (FCT) advection scheme, which combines first-order upwind advection (which does not overshoot but is highly diffusive) with a user-selectable high-order advection scheme to eliminate overshoots. FCT computes the maximum fraction of the advective flux of the higher-order scheme that can be used without

causing an overshoot. In multi-dimensional applications such as in NCOM, FCT works best if the high-order scheme being used generates smooth solutions that do not overshoot much, so as to minimize the use of the first-order scheme. Hence, the third-order upwind advection scheme is the generally recommended high-order scheme for use with FCT.

5. In setting the timestep for the model, the timestep limitation for the propagation of internal waves and for horizontal and vertical advection must not be exceeded or numerical instability may result.
6. The drying out of a grid cell due to depression of the free surface down to the sea bottom in shallow water or to the bottom of the sigma grid (i.e., where changes in the surface elevation are accommodated), can cause a model simulation to suddenly terminate. Hence, the minimum water depth and the bottom of the sigma grid must be deep enough to contain the maximum expected depression of the sea surface during the model run.

5.2 Logic and Basic Equations

Please refer to Barron et al., (2006) for a complete explanation of the physics and basic equations of NCOM Version 4.0.

5.3 NCOM Setup Routines

The setup program and main routines for the setting up of the NCOM simulation are found in the src/setup/ and libsrc/setup/ subdirectories. There is a separate .F file for GVC setup routines within the same directory.

5.3.1 General Setup Subroutines (ncom_setup_plib_sigz)

This file contains general routines for setting up a simulation for use with the sigma-z vertical coordinate grid.

Subroutine	Description
Adj_topo	Subroutine from Dong Shan Ko to adjust a bathymetry file to reduce steep slopes according to the criteria: abs(h(i) - h(i-1)) * 2/(h(i) + h(i-1)) < slopemax. D.S. Ko works with the value smax = slopemax/2. **Calling Sequence**: adj_topo (slopemax, im, jm, h) **Data Declaration:** Integer im, jm Real slopemax, h
Bicubc3	Subroutine BICUBC3 computes a bicubic interpolation from a 2D grid of data to a specified (different) 2D grid. This routine uses polynomials that are cubic in x and y (not splines). It is assumed that the grid from which the data is being interpolated is regularly spaced in the two coordinate directions in terms of the coordinate use for the interpolation. The constants needed for the interpolation between the two grids are calculated on the first call (and whenever ireset = 1) to save time when doing repeated interpolations between the same two grids. With bicubic interpolation, the weightings for the interpolation depend only on the relative position of the two grids and not on the values being interpolated. BICUBC3 differs from BICUBC2 in that the constants that define the bicubic interpolation (all 2,304 of them) are defined in data statements rather than being read from a file. BICUBC2 differs from BICUBIC in that it can interpolate in the boundary rows of the field being interpolated from. In order to do this, quadratic polynomials are used when interpolating within the outer boundary row of the grid of data being interpolated from (cubic polynomials are used in the interior). This routine will extrapolate values that are just outside the grid being interpolated from. However, if the routine is asked to extrapolate very far outside the grid of data being interpolated from, the program will stop and an error message will be written to unit six. **Calling Sequence:** bicubc3 (ni1, n1, m1, x1a, x1b, y1a, y1b, f1, ni2, n2, m2, x2, y2, f2, ireset, if2, jf2, cf2) **Data Declaration:** Integer ni1, n1, m1, ni2, n2, m2, ireset, if2, jf2 Real x1a, x1b, y1a, y1b, f1, x2, y2, f2, cf2 **Common Block:** BICUBCN **Comments:** Variables if2, jf2, and cf2 must be supplied for storing the constants used for the interpolation, and cannot be overwritten between calls to BICUBC2 unless the

Subroutine	Description
	interpolation constants are recalculated (by setting ireset = 1). If there is a change in the location of either the grid points being interpolated from, or those being interpolated to, the interpolation constants need to be recalculated. However, the grids can be changed without recalculating the interpolation constants, as long as the correct interpolation constants are passed in for the grids being used. Although this subroutine is set up to interpolate to a 2D array of locations, the interpolation does not depend on any regularity in the locations of the points being interpolated to. For example, a 1D array of randomly located points (e.g., from a finite-element grid) can be interpolated to by passing the values of x2, y2, and f2 into this subroutine as 1D arrays with m2 = 1.
Bicublk	Subroutine BICUBLK defines constants needed for bicubic polynomial interpolation. These were derived in program test/***intbicube2.f***. The constants allow for lower order quadratic interpolation near the boundaries of the data being interpolated from where full bicubic is not possible. The nine sets of coefficients correspond to interpolation within nine "zones" of the data being interpolated from: 1. Left-lower corner, 2. Middle-lower edge, 3. Right-lower corner, 4. Left-middle edge, 5. Interior, 6. Right-middle edge, 7. Left-upper corner, 8. Middle-top edge, and 9. Right-upper corner. **Common Block:** BICUBN
Blend2D	Subroutine BLEND2D blends two 2D fields based on minimum distance from the outer open boundary according to weight *w* as: h1 = w*h1 + (1-w)*h2 This routine may give inappropriate blending (too much weight to h2) in interior regions separated from open boundary point interior regions. **Calling Sequence:** blend2d (n, m, nw, w, nobmx, iob, job, h1, h2) **Data Declaration:** Integer n, m, nw, nobmx, iob, job Real w, h1, h2
Bndydepe	Subroutine BNDYDEPE checks if a boundary point is a sea point and sets depth at boundary point = depth at adjacent interior point. **Calling Sequence:** bndydepe (n, m, ibo, indcyc, h) **Data Declaration:** Integer n, m, ibo, indcyc Real h
Bndydepz	Subroutine BNDYDEPZ sets depth at open boundary points less than or equal to the depth at the adjoining interior point on the z-level part of the grid. This is to avoid having the inflow hit a wall as it tries to flow in on the z-level grid. The "rule" used here is: If h_interior > zw(ls), then h_bndy = max[h_bndy, zw(ls)]

Subroutine	Description
	If h_interior < zw(ls), then h_bndy = max[h_bndy, h_interior]. Hence, if the interior point is above the z-level grid, then the boundary point cannot be deeper than zw(ls), and if the interior point is on the z-level grid, then the boundary point cannot be deeper than the interior point. (All depths here are defined + upwards.) This routine can be called before or after the depths have been rounded to z-levels. **Calling Sequence:** bndydepz (n, m, l, ls, indcyc, zw, h) **Data Declaration:** Integer n, m, indcyc Real l, ls, zw, h
Bndyfmc1	This subroutine closes all open boundary points for a refined bathymetry (hr) for a nested grid (Fine Mesh, FM; also known as the “child grid”) that are closed (not open) for the coarse, or parent, grid (CM) in which the nested grid is nested. This is done by comparing values of hr on the FM boundary with values of hc, where hc is a coarse bathymetry for the FM obtained directly from the parent grid. If hr is open and hc is closed, hr is set = hc. It is assumed here that all open boundary points on the FM must be connected to the CM grid. The number of hr pts that are converted from sea to land is printed. This routine should be called before hc and hr are blended, since the blending will be based on the location of open boundary pts for hc. **Calling Sequence:** subroutine bndyfmcl(n,m,hc,hr) **Data Declaration:** Integer n, m Real hc, hr
Bndyorp	Subroutine BNDYORP checks for open boundary points on a grid where the adjoining interior point is a land point. It is best to adjust the grid or the coarse grid in which the grid is nested to avoid this situation. **Calling Sequence:** bndyorp (n, m, h) **Data Declaration:** Integer n, m Real h
Chkdimen	Subroutine CHKDIMEN checks the dimensions set in the main setup program. **Calling Sequence:** chkdimen (ndx, mdx, ldx, nrdx, ntcdx, nobdx, nrivdx, mxgrds, no, mo, lo, lso, nro, ntco, nobmaxo, nrivo) **Data Declaration:** Integer ndx, mdx, ldx, nrdx, ntcdx, nobdx, nrivdx,mxgrds, no, mo, lo, lso, nro, ntco, nobmaxo, nrivo
Cm2fm_grd	Subroutine CM2FM_GRD interpolates grid parameters from CM to FM, or parent to nested grid, respectively. For the z-level grid, the FM depths are set to be the same as the depth on the CM in which the FM point is located. For the sigma grid, the FM depths are directly interpolated from the CM depths. No bathymetry refinement is done here. If bathymetry refinement is desired, this must be done as a separate step. A refined bathymetry can be computed for the FM, and then the refined and unrefined FM bathymetries must be "blended" so that the unrefined FM bathymetry is retained near the FM boundary and matches the CM bathymetry. **Calling Sequence:** cm2fm_grd (nest1, nest2, gr2, is, js, n1, m1, l1, ls1, elon1, alat1, dx1, dy1, h1, ang1, amsk1, x1, y1, zw1, n2, m2, l2, ls2, elon2, alat2, dx2, dy2, h2, ang2, amsk2, x2, y2, zw2, if2, jf2, cf2) **Data Declaration:** Integer nest1, nest2, gr2, is, js, n1, m1, l1, ls1, n2, m2, l2,

Subroutine	Description
	ls2 Real elon1, alat1, dx1, dy1, h1, ang1, amsk1, x1, y1, zw1, elon2,alat2, dx2, dy2, h2, ang2, amsk2, x2, y2, zw2, if2, jf2, cf2
Cm2fm_ic	Subroutine CM2FM_IC interpolates initial conditions from a parent grid to nested grid. **Calling Sequence:** cm2fm_ic (nest1, nest2, gr2, is, js, n1, m1, l1, ls1, nr1, h1,amsk1, x1, y1, zw1, e1, u1, v1, r1, n2, m2, l2, ls2, nr2, h2, amsk2, x2, y2, zw2, e2, u2, v2, r2, if2, jf2, cf2) **Data Declaration:** Integer nest1, nest2,is, js, n1, m1, l1, ls1, nr1, n2, m2, l2, ls2, nr2, if2, jf2 Real gr2, h1, amsk1, x1, y1, zw1, e1, u1, v1, r1, h2, amsk2, x2, y2, zw2, e2, u2, v2, r2, cf2
Cm2fm_ic5	Subroutine CM2FM_IC5 interpolates initial conditions from a CM (“parent” grid) to an FM (“child” grid). Fields are vertically interpolated to z-levels, horizontally filled, then horizontally interpolated on z-levels, and finally vertically interpolated back to sigma layers. **Calling Sequence:** cm2fm_ic5(nest1,nest2,intrpo,intv,gr2,nl,ml,l1,ls1,nr1,zw1,h1, ang1,amsk1,x1,y1,zwt1,e1,u1,v1,r1,n2,m2,l2,ls2,nr2,zw2,h2,ang 2,amsk2,x2,y2,zwt2, e2,u2,v2,r2,if2,jf2,cf2) **Data Declaration:** Integer nest1, nest2,is, js, n1, m1, l1, ls1, nr1, n2, m2, l2, ls2, nr2, if2, jf2 Real gr2, h1, amsk1, x1, y1, zw1, e1, u1, v1, r1, h2, amsk2, x2, y2, zw2, e2, u2, v2, r2, cf2
Cm2fm_sfx	Subroutine CM2FM_SFX interpolates surface forcing fields from a CM to an FM. **Calling Sequence:** cm2fm_sfx (nest1, nest2, indatp, indtau, indsft, indsfs, indsol, n1, m1, nr1,x1, y1, pa1, tx1, ty1, rs1, qr1, n2, m2, nr2, x2, y2, pa2, tx2, ty2, rs2, qr2, if2, jf2, cf2) **Data Declaration:** Integer nest1, nest2, indatp, indtau, indsft, indsfs,indsol, n1, m1,nr1,n2, m2, nr2, if2, jf2 Real x1, y1, pa1, tx1, ty1, rs1, qr1, x2, y2, pa2, tx2, ty2, rs2, qr2,cf2
Conphase	Subroutine CONPHASE converts phase angle from 0 to 360 or from -180 to +180 to try to avoid discontinuity, if it exists. **Calling Sequence:** conphase(n,y) **Data Declaration:** Integer n Real y
Consea	Subroutine CONSEA defines a single contiguous area of ocean within a rectangular region using a 2D array of ocean depths. The largest contiguous ocean area is determined to be the region of interest. The depth values outside the contiguous main ocean basin are set to zero. A (real) land-sea mask is returned for the main contiguous ocean basin with the sea points = 1.0 and all other points = 0.0. **Calling Sequence:** consea (ni, n, m, d, dmsk) **Data Declaration:** Integer ni, n, m

Subroutine	Description
	Real d, dmsk
Creep4	Subroutine CREEP4 extends values where *amsk*=1 into regions where *amsk*=0. The method replaces "bad" pts with an average of the adjoining "good" pts. Only the adjoining "good" points to the E,W,N,S are used, i.e., the adjacent corner pts are not used. When extending for the purpose of interpolation near land-sea boundaries, only a few iterations may be needed (e.g., itermx=10). To fill the entire field, set itermx > *max(n,m)* to be sure all pts will be filled. **Calling Sequence:** creep4(t,amsk,n,m,itermx) **Data Declaration:** Integer n, m, itermx Real t, amsk
Depths_m1	Subroutine DEPTHS_M1 computes an array of mid-layer (static) depths at point (i, j). There is an assumption here that model variables are defined at the layer mid-depth, i.e., for which the vertical grid stretching is not accounted. **Calling Sequence:** depths_m1 (n, m, l, ls, h, zw, i, j, kb, zm1) **Data Declaration:** Integer n, m, l, ls, i, j, kb Real h, zw, zm1
Depths_w1	Subroutine DEPTHS_W1 computes an array of (static) depths to top of layers at point (i, j). **Calling Sequence:** depths_w1 (n, m, l, ls, h, zw, i, j, kb, zw1) **Data Declaration:** Integer n, m, l, ls, i, j, kb Real h, zw, zw1
Depths_w3	Subroutine DEPTHS_W3 calculates 3D arrays of (static) depths at layer interfaces. **Calling Sequence:** depths_w3 (n, m, l, ls, h, zw, zw3) **Data Declaration:** Integer n, m, l, ls Real h, zw, zw3
Gaubmp3	Subroutine GAUBMP3 defines symmetric Gaussian elevation bumps. **Calling Sequence:** gaubmp3 (n, m, amsk, bmax, scal, rem, e) **Data Declaration:** Integer n, m Real amsk, bmax, scal, rem, e
Gaubmpi	Subroutine GAUBMPI defines symmetric Gaussian internal bumps. **Calling Sequence:** gaubmpi (n, m, l, amp, radius, s) **Data Declaration:** Integer n, m, l Real amp, radius, s
Getint	Subroutine GETINT requests integer numbers from standard input. If no value is input, the default value is retained. **Calling Sequence:** getint (query, format, idefalt) **Data Declaration:** Integer idefalt Character query, format
Getlog2	Subroutine GETLOG2 requests a logical value from standard input. If no value is input, the default value is returned. **Calling Sequence:** getlog2 (query, default) **Data Declaration:** Character query Logical default

Subroutine	Description
Getreal	Subroutine GETREAL requests real numbers from standard input. If no value is input, the default value is retained. **Calling Sequence:** getreal (query, format, default) **Data Declaration:** Character query, format Real default
Get_zuw	Subroutine GET_ZUW computes grid fields needed to plot grid cells. **Calling Sequence:** get_zuw(n,m,l,ls,lz,n1,n2,m1,m2,h,z_w,kb,z_uw,z_vw) **Data Declaration**: Integer n,m,l,ls,lz,n1,n2,m1,m2,kb(n,m) Real h,z_w,z_uw,z_vw
Getvc2z	Subroutine GETVC2Z vertically interpolates a 3D array from a general vertical coordinate to a specified z-level grid. **Calling Sequence:** gvc2z(indpt,intv,n,m,l,n1,n2,m1,m2,zwt,amsk,t,lz,z,amskz,tz) **Data Declaration:** Integer indpt,intv,n,m,l,n1,n2,m1,m2,lz Real t,zwt,amsk,z,tz,amskz
Hminmax	Subroutine HMINMAX sets the minimum and maximum depth for bathymetry. All depths are defined + upward, i.e., points with h >= 0 are land points. **Calling Sequence:** hminmax (n, m, hmin, hmax, h, ind) **Data Declaration:** Integer n, m, ind Real hmin, hmax, h
Hor_av2	Subroutine HOR_AV2 calculates horizontally averaged values of a 3D model field (t) at specified depths (z2). Uses Ko's cubic spline routines. **Calling Sequence:** hor_av2 (n, m, l, ls, h, zw, t, l2, z2, t2, k2max) **Data Declaration:** Integer n, m, l, ls, l2, k2max
Hor_avts	Subroutine HOR_AVTS calculates horizontally averaged T and S fields on the model grid. **Calling Sequence:** hor_avts (n, m, l, ls, h, zw, t, s) **Data Declaration:** Integer n, m Real l, ls, h, zw, t, s
Logrid	Subroutine LOGRID calculates the interface depths (zb) for a vertical grid that is linearly spaced (constant) near the surface and logarithmically stretched below a particular depth. **Calling Sequence:** logrid (lp1, ll, dz1, depth, strfac, zb) **Data Declaration:** Integer lp1, l1 Real dz1, depth, strfac, zb
Lsmask2	Subroutine LSMASK2 calculates a 2D land-sea mask based on where the depth (h) is below a "small" value. **Calling Sequence:** lsmask2 (n, m, h, amsk) **Data Declaration:** Integer n, m Real h, amsk
Lsmask3	Subroutine LSMASK3 calculates a 3D land-sea mask. **Calling Sequence:** lsmask3 (n, m, l, ls, h, zw, amsk) **Data Declaration:** Integer n, m, l, ls Real h, zw, amsk

Subroutine	Description
Minmax	Subroutine MINMAX finds the minimum and maximum values of an array t. **Calling Sequence:** minmax (t, n, tmin, tmax) **Data Declaration:** Integer n Real t, tmin, tmax
Minmaxm	Subroutine MINMAXM calculates minimum and maximum of a function f over points where the mask array amsk is set to one. **Calling Sequence:** minmaxm (n, m, l, n1, n2, m1, m2, l1, l2, f, amsk, fmin, fmax) **Data Declaration:** Integer n, m, l, n1, n2, m1, m2, l1, l2 Real f, amsk, fmin, fmax
Orphan	Subroutine ORPHAN removes orphan grid points from bathymetry file, i.e., points that have land on three sides. **Calling Sequence:** orphan (n, m, h, amsk) **Data Declaration:** Integer n, m Real h, amsk
Pause2	Subroutine PAUSE2 pauses the execution of a program that is being run interactively.
Plotuv	Subroutine PLOTUV prints or plots scalar or horizontal vector fields. It does this through the following steps: • Prints/plots contours of u or v (x and y components of vector field). • Prints/plots contours of vector magnitude. • Plots vector arrows. **Calling Sequence:** plotuv (indp, u, nu, mu, lu, v, nv, mv, lv, n1, n2, m1, m2, l1, l2, indgrd, amsk, nm, mm, lm, name, amult, cint, vscale) **Data Declaration:** Integer indp, nu, mu, lu, nv, mv, lv, n1, n2, m1, m2, l1, l2, indgrd, nm, mm, lm, name Real u, v, amsk, amult, cint, vscale **Common Block:** CONRE4
Prnplt1	Subroutine PRNPLT1 prints or plots a scalar or horizontal vector field. **Calling Sequence:** prnplt1 (time, indgrd, n, m, l, am, nam, mam, lam, u, nu, mu, lu, v, nv, mv,lv, name, amult, cint, vscale) **Data Declaration:** Integer indgrd, n, m, l, nam, mam, lam, nu, mu, lu, nv, mv, lvm Real time, am, u, v, amult, cint, vscale Character name
Prnpltic	Subroutine PRNPLTIC prints and/or plots a model grid and initial conditions. **Calling Sequence:** prnpltic (nest, n, m, l, nr, elon, alat, zw3, h, amsk, e, u, v, r) **Data Declaration:** Integer nest, n, m, l, nr Real elon, alat, zw3, h, amsk, e, u, v, r
Read_hgrid	Subroutine READ_HGRID gets NCOM horizontal grid arrays. **Calling Sequence:** read_hgrid(infile, n,m, elon,alat,dx,dy,h,ang) **Data Declaration:** Character infile Integer n, m Real elon,alat,dx,dy,h,ang
Read_out3h	Subroutine READ_OUT3H gets model output fields for time=timed. This is an

Subroutine	Description
	alternative for using RW_OUT3F and is set up to use direct access to skip directly to desired fields at the desired time. Note: This subroutine is for single processor use only and the model arrays do not have halos. Do not use with halos. Note: The flags ind* return the specified field when set =1. This choice is provided since reading output can be accelerated if fields not needed are not requested. Note: This subroutine currently assumes that ALL the fields were written to the output file. If this is not the case, some modifications to this subroutine will be needed to account for the smaller number of fields on the file. **Calling Sequence:** read_out3h(infile,timed,dt,inde,indvb,indv,indw,indt,inds,inda, n,m,l, e,udb,vdb,u,v,w,t,s, patm,usflx,vsflx,tflx,sflx,solar,surruf) **Data Declaration:** Character infile Integer inde,indvb, indw,indt,inds,inda,n,m,l Real timed,dt, e,udb,vdb,u,v,w,t,s, patm, usflx, vsflx, tflx, sflx, solar,surruf
Read_outsfc	Subroutine READ_OUTSFC gets model surface output fields for time=timed. This is an alternative to using rw_outsfa and is set up to use direct access to skip directly to desired fields at the desired time. **Calling Sequence:** read_outsfc(infile,timed,dt,inde,indvb,indv,indt,inds,inda,n,m, e,udb,vdb,u,v,t,s,usflx,vsflx) **Data Declaration:** Character infile Integer inde,indvb,indv,indt, inds,inda,n,m Real timed, dt,e,udb, vdb,u, v,t, s, usflx, vsflx
Read_vgrid	Subroutine READ_VGRID reads input files for the NCOM vertical grid. **Calling Sequence:** read_vgrid(infile,l,ls,zw) **Data Declaration:** Character infile Integer l,ls Real zw
Repeat	Subroutine REPEAT repeats (propagates) an array an integer multiple of times. **Calling Sequence:** repeat (mult, ipos, ivec, n, m, l, n2, m2, f, f2) **Data Declaration:** Integer mult, ipos, ivec, n, m, l, n2, m2 Real f, f2
Rnd_zlev	Subroutine RND_ZLEV rounds off bottom depth (h) to nearest z-level. **Calling Sequence:** rnd_zlev (n, m, l, ls, zw, h) **Data Declaration:** Integer n, m, l, ls Real zw, h
Slope2	Subroutine SLOPE2 examines the rate of change of slopes. **Calling Sequence:** slope2 (n, m, h, amsk) **Data Declaration:** Integer n, m Real h, amsk
Slopmax	Subroutine SLOPMAX calculates the maximum relative slopes in x and y. **Calling Sequence:** slopmax (n, m, h, amsk) **Data Declaration:** Integer n, m

Subroutine	Description
	Real h, amsk
Smth2m	Subroutine SMTH2M applies a Hanning-type box filter to a 2D array f. The array f is only filtered at points where amsk is greater than 0.5. **Calling Sequence:** smth2m (ni, n, m, amsk, f) **Data Declaration:** Integer ni, n, m Real amsk, f
Strlen	Subroutine STRLEN finds the total number of characters in a string not including trailing blanks. **Calling Sequence:** strlen (string, nc) **Data Declaration:** Integer nc Character string
Sz_trans	Subroutine SZ_TRANS inspects the sigma/z-level transition for a FM grid nested in a CM grid. **Calling Sequence:** sz_trans (n1, m1, l1, ls1, h1, zw1, amsk1, n2, m2, l2, ls2, h2, zw2, amsk2) **Data Declaration:** Integer n1, m1, l1, ls, ls1, n2, m2, l2, ls2 Real h1, zw1, amsk1, h2, zw2, amsk2
Tablk2	Subroutine TABLK2 interpolates a value from a 2D array using linear interpolation (i. e., table lookup). The array f varies with both x and y and the spacing of the values of f along the x- and y-axes is assumed to be constant. **Calling Sequence:** tablk2 (ni, n, m, xa, xb, ya, yb, f, x2, y2, f2, indext) **Data Declaration:** Integer ni, n, m, indext Real xa, xb, ya, yb, f, x2, y2, f2
Tablk3	Subroutine TABLK3 interpolates a value from a 3D array f using linear interpolation (i. e. table lookup). The spacing of the x and y arguments of f is assumed to be constant. Spacing in z can be variable. **Calling Sequence:** tablk3 (ni, mj, n, m, l, x, y, z, f, x2, y2, z2, f2, indext) **Data Declaration:** Integer ni, mi, n, m, l, indext Real x, y, z, f, x2, y2, z2, f2
Tablok	Subroutine TABLOK interpolates a value from a 2D array f using linear interpolation (i.e. table lookup). The spacing of the x and y arguments of f is assumed to be constant. **Calling Sequence:** tablok (ni, n, m, x y, f, x2, y2, f2, indext) **Data Declaration:** Integer ni, n, m, indext Real x, y, f, x2, y2, f2
Topave	Subroutine TOPAVE obtains water depth at location (elon, alat) by performing an average over a region of size *dlon x dlat* centered at (elon, alat). The *dlon x dlat* region is subdivided into a 5 x 5 grid of sub-regions, and the bathymetry is obtained for each sub-region and is averaged over the region. If the number of sub-regions that are on land is >= *nlndmin*, the grid point is set to land. **Calling Sequence:** topave (elon, alat, dlon, dlat, nlndmin, h) **Data Declaration:** Integer nlndmin Real elon, alat, dlon, dlat, h
Vgrid_plt	Subroutine VGRID_PLT is a program to plot the layout of grid cells for specified

Subroutine	Description
	vertical sections. Plots of the grid cell layout can be either along x or y coordinates. **Calling Sequence:** vgrid_plt(n,m,l,ls,lz,h,z_w) **Data Declaration:** Integer n,m,l,ls,lz Real h,z_w
Z2gvc	Subroutine Z2GVC interpolates a 3D array in the vertical from z-levels (fixed depths) to a general vertical coordinate at all the sea points on the general vertical grid (as denoted by *amsk*). **Calling Sequence:** z2gvc(indpt,intv,lz,z,amskz,tz, n,m,l, n1,n2,m1,m2,zwt,amsk,t) **Data Declaration:** Integer indpt,intv,lz,n,m,l,n1,n2,m1,m2 Real t,zwt,amsk,z,tz,amskz

5.3.2 Spline Interpolation Subroutines (ncom_setup_spln)

This file contains spline interpolation routines from Dong Shan Ko.

Subroutine	Description
Spak1d	Subroutine SPAK1D is a 1D interpolation using Akima spline Y = f(X) (Akima, 1970). **Calling Sequence:** spak1d (x, y, n, xi, yi, ni) **Data Declaration:** Integer n, ni Real x, y, xi, yi
Spak2d	Subroutine SPAK2D is a 2D interpolation using an Akima spline F = f(x, y) (Akima, 1970). **Calling Sequence:** spak2d (f, x, y, nx, ny, fi, xi, yi, nxi, nyi) **Data Declaration:** Integer nx, ny, nxi, nyi Real f, x, y, fi, xi, yi
Splakm	Subroutine SPLAKM calculates coefficients of an Akima spline (Akima, 1970). Some changes have been made by D. S. Ko. Subroutine SPLAKM should not be separated from the following subroutine SPLDER. This version uses Lagrangian polynomials to extrapolate. The arguments are changed in the subroutine statement for efficiency (s = WORK, slope = WORK2). **Calling Sequence:** splakm (x, y, nx, coef, work, work2) **Data Declaration:** Integer nx Real x, y, coef, work, work2
Splder	**Calling Sequence:** splder (x, y, n, nbr, break, coef) **Data Declaration:** Integer n, nbr Real x, y, break, coef

5.4 Main NCOM Subroutines (libsrc/ ncom/)

This is a directory of main NCOM Fortran routines.

5.4.1 File ncom1

This file contains all of the old ncom1 files except the driver module (found in ***ncom.F*** on directory src/ncom/).

Subroutine	Description
Coamm	Subroutine COAMM coordinates the calculation of the various atmospheric and oceanic model grids. **Calling Sequence:** coamm (nto, mto, iec, no, mo, lo, lso, nro, nqo, ntypo, ntco, nobmaxo,nrvmaxo, ni4s, nl4s, nr4s) **Data Declaration:** Integer nto, mto, iec, no, mo, lo, lso, nro, nqo, ntypo, ntco,nobmaxo, nrvmaxo, ni4s, nl4s, nr4s
Get_nestseq	Subroutine GET_NESTSEQ computes grid calculation sequences. **Calling Sequence:** get_nestseq(nstepsmx,nsteps,nestseq) **Data Declaration:** Integer nstepsmx,nsteps,nestseq
Logico2	Subroutine LOGICO2 computes a grid calculation sequence table "nestseq" on the first calculation pass to define the ocean grid calculation sequence during a single ocean calculation cycle. An ocean calculation cycle consists of the updating of all the ocean grids over a time period corresponding to one timestep of the main grid. The same sequence of calculations is repeated for each ocean calculation cycle. Being in a simple table, the grid calculation sequence can easily be inverted to get the calculation sequence for the ocean model inverse. **Calling Sequence:** logico (ocean, modeocn, surfbco, bndvalo, relaxo, feedbko) **Data Declaration:** Integer modeocn Logical ocean, surfbco, bndvalo, relaxo, feedbko
Memmo	Subroutine MEMMO sets pointers and allocates memory for ocean model forecast grids. **Calling Sequence:** memmo (no, mo, lo, lso, nro, nqo, ntypo, ntco, nobmaxo, nrvmaxo, ni4s,nl4s, nr4s) **Data Declaration:** Integer no, mo, lo, lso, nro, nqo, ntypo, ntco, nobmaxo, nrvmaxo, ni4s, nl4s, nr4s
Memmo2	Subroutine MEMMO2 sets pointers and allocates memory for an ocean model nest. **Calling Sequence:** memmo2 (no, mo, lo, lso, nro, nqo, ntypo, ntco, nobmaxo, nrvmaxo, ni4s, nl4s, nr4s) **Data Declaration:** Integer no, mo, lo, lso, nro, nqo, ntypo, ntco, nobmaxo, nrvmaxo,ni4s, nl4s, nr4s **Common Blocks:** OBLK
Ncom_Init	Initializes NCOM message passing. **Calling Sequence:** NCOM_Init(mpi_comm) **Data Declaration:** Integer mpi_comm
NCOM_Run	Initializes flag for the end of the run. **Calling Sequence:** NCOM_Run(end_time) **Data Declaration:** Real end_time
NCOM_Final	Initializes NCOM message passing. **Calling Sequence:** NCOM_Final(no_stop) **Data Declaration:** Logical no_stop

Subroutine	Description
Omodel	Subroutine for NCOM ocean model. **Calling Sequence:** omodel (modeocn, na, ma, iec, n, m, l, ls, nr, nq, ntyp, ntc, nobmax,nrvmax, il, i2, i3, j1, j2, kb, kbu, kbv, is, ie, ism, iem, isp, iep, js, je, ibo, ke, ilx1, ilx2, iob1, iob2, irv1, irv2, iter, ramp, times, dti2, de, fda, botruf, cbu, cbv, istype, iptype, qrf, ext, elon, alat, ang, dx, dxu, dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, ddx, ddy, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, e, d, du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, r, q, rmean, zkm, zkh, wubot, wvbot, sor, sorb, patm, usflx, vsflx, rsflx, solar, surruf, rlx, wlx, tmlx, nob, neob, nuob, nvob, iob, job, iobi, jobi, ivob, jvob, eob, ubob, vbob, cgwb, uob, vob, rob, tmob, etab, etpb, utab, utpb, vtab, vtpb, nriv, nrriv, lriv, iriv, jriv, isriv, ieriv, wtriv, qriv, rriv, tmriv, w, tl, rho, sos, xk, yk, zkb, wxy, wxz, o) **Data Declaration:** Integer modeocn, na, ma, iec, n, m, l, ls, nr, nq, ntyp, ntc, nobmax,nrvmax, il, i2, i3, j1, j2, kb, kbu, kbv, is, ie, ism, iem, isp, iep, js, je, ibo, ke, ilx1, ilx2, iob1, iob2, irv1, irv2, iter, istype, iptype, nob, neob, nuob, nvob, iob, job, iobi, jobi, ivob, jvob, nriv, nrriv, lriv, iriv, jriv, isriv, ieriv Real ramp, times, dti2, de, fda, botruf, cbu, cbv, qrf, ext, elon,alat, ang, dx, dxu, dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, ddx, ddy, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, e, d,du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, r, q, rmean,zkm, zkh, wubot, wvbot, sor, sorb, patm, usflx, vsflx, rsflx,solar, surruf, rlx, wlx, tmlx, eob, ubob, vbob, cgwb, uob,vob, rob, tmob, etab, etpb, utab, utpb, vtab, vtpb, wtriv,qriv, rriv, tmriv, w, tl, rho, sos, xk, yk, zkb, wxy, wxz, o **Common Blocks :** PAR5O PAR6O PAR7O PAR8O
Padr4add	Subroutine PADR4ADD adds a padding zone to the real*4 allocation array. **Calling Sequence:** padr4add (nr4s, cdesc) **Data Declaration:** Integer nr4s Character cdesc **Common Blocks:** PADR4I

Subroutine	Description
	PADR4C
Padr4set	Subroutine PADR4SET sets all padding zones (defined by PADR4ADD) to PADVAL. **Calling Sequence:** padr4set (o) **Data Declaration:** Real o
Padr4tst	Subroutine PADR4TST tests all padding zones for a nesting nest. Padding zones are defined by PADR4ADD and set by PADR4SET. **Calling Sequence:** padr4tst (o, ctest) **Data Declaration:** Real o Character ctest
Timeset	Subroutine TIMESET sets current time and resets certain parameters that depend on the time (if indicated). **Calling Sequence:** timeset(iter,dtfrac,times) **Data Declaration:** Real dtfrac,times Integer iter
Xcspmd	An interface needed for the compiler to properly resolve subroutines. **Calling Sequence:** xcspmd(mpi_comm_in) **Data Declaration:** Integer mpi_comm_in

5.4.2 Free-Surface Calculation Subroutines (ncom1baro)

Subroutine	Description
Baro1	Subroutine BARO1 calculates new surface elevation and barotropic velocity explicitly with a timestep that is the same as that (dti2) used for the baroclinic calculations. **Calling Sequence:** baro1 (ind, fu, fv, n, m, l, i1, i2, i3, is, ie, ism, iem, js, je, iec, locate, dti2,dxv, dyu, dar, sorb, e, udb, vdb) **Data Declaration:** Integer ind, n, m, l, i1, i2, i3, is, ie, ism, iem, js, je, iec, locate Real fu, fv, dti2, dxv, dyu, dar, sorb, e, udb, vdb
Baro2	Subroutine BARO2 calculates new surface elevation and barotropic velocity implicitly using the same timestep (dti2) used for the baroclinic calculations. The calculation has been split into two parts (called with ind = 1 and ind = 2) to allow the open boundary condition to be set from subroutine UPDATE. **Calling Sequence:** baro2 (ind, fu, fv, aax, aay, na, ma, n, m, l, i1, i2, i3, is, ie, ism, iem, isp, iep, js, je, iec, indbaro, indsolv, indrag, indcyc, indiag, shrnkwp, locate, batch, dti2, eg1, vg1, vg2, vg3, g, cbu, cbv, small, dxv, dxur, dyu, dyvr, da, dar, amsk, umsk, vmsk, sorb, e, du, dv, udb, vdb, u, v, wubot, wvbot, ax, ay, bb, ff, wk1, wk2, wk3, wk4, wk5) **Data Declaration:** Integer ind, na, ma, n, m, l, i1, i2, i3, is, ie, ism, iem, isp, iep, js, je, iec, indbaro, indsolv, indrag, indcyc, indiag, locate

Subroutine	Description
	Real fu, fv, aax, aay, shrnkwp, , batch, dti2, eg1, vg1, vg2, vg3,g, cbu, cbv, small, dxv, dxur, dyu, dyvr, da, dar, amsk, umsk, vmsk, sorb, e, du, dv, udb, vdb, u, v, wubot, wvbot,ax, ay, bb, ff, wk1, wk2, wk3, wk4, wk5
Cgssor	Subroutine CGSSOR conjugates the gradient elliptic solver with red-black SSOR preconditioner. **Calling Sequence:** cgssor (indiag, indcyc, na, ma, n, m, is, ie, js, je, ax, ay, bb, ff, e, zz, rr, pp,qq, rbb) **Data Declaration:** Integer indiag, indcyc, na, ma, n, m, is, ie, js, je Real ax, ay, bb, ff, e, zz, rr, pp, qq, rbb
Cgssorc	Subroutine CGSSORC is a red-black SSOR preconditioner for CGSSOR. **Calling Sequence:** cgssorc (indcyc, na, ma, n, m, is, isr, isb, ie, js, je, ax, ay, zz, rr, rbb) **Data Declaration:** Integer indcyc, na, ma, n, m, is, isr, isb, ie, js, je Real ax, ay, zz, rr, rbb
Sorcyc2	Subroutine SORCYC2 is a SOR solver designed to be used with cyclic BC. **Calling Sequence:** sorcyc2 (batch, indsolv, indiag, indcyc, n, m, is, ie, js, je, ax, ay, bb, gg, e, wk1, wk2) **Data Declaration:** Integer indsolv, indiag, indcyc, n, m, is, ie, js, je Real batch, ax, ay, bb, gg, e, wk1, wk2

5.4.3 COAMPS Specific Subroutines (ncom1coam)

Subroutine	Description
Bulk_ls	Subroutine BULK_LS calculates the latent and sensible heat flux using bulk formulas, the SST from the ocean model, and some atmospheric fields. Net longwave radiation is not calculated since this depends on cloud conditions that are not available. The latent heat flux calculated here is used to provide the evaporation for the surface salt flux if indsfs=4. Variables "*times*" and "*solar*" are passed in only for diagnostics. **Calling Sequence:** bulk_ls(nt,mt,n,m,nr, is,ie,js,je,ico1,ico2, w1co, times, ramp, amsk, t,s, patm2,wspd2,tair2,humd2, rsflx,solar, evap) **Data Declaration:** Integer nt,mt, n,m,nr,is, ie, js, je, ico1, ico2 Real w1co, times, ramp,amsk, t,s,patm2, wspd2, tair2, humd2, rsflx, solar
get_csfx	Subroutine to get COAMPS surface flux fields for the ocean model. It is set up for real-time data only. Fractional hrs (*itmsec*) must be incorporated. **Calling Sequence:** get_csfx(indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr, is,ie,js,je,ico1,ico2,idate,itime,timed,climatp,w1co,elon,alat,an g,amsk, patm2,usflx2,vsflx2,rsflx2,solar2,wspd2,tair2,humd2, tmcoa2, wxy) **Data Declaration:** Integer indatp,indtau,indsft, indsfs, indsol, nt, mt, n,m,nr,ico1,ico2,idate,itime, is,ie,js,je

Subroutine	Description
	Real timed,climatp,w1co,elon, alat, amsk, patm2, usflx2,vsflx2,rsflx2, solar2,wspd2,tair2,humd2,wxy,tmcoa2
Get_csst	Subroutine GET_CSST gets COAMPS SST and/or SSS fields. This is set up for real time data only. **Calling Sequence:** get_csst(indsst,indsss,nt,mt,n,m,is,ie,js,je,ist1,ist2,iss1,iss2, idate,itime,timed,climatp, w1st,w1ss, elon,alat,amsk, sst2,sss2, tmsst2,tmsss2, wxy) **Data Declaration:** Integer indsst, indsss, nt, mt, n, m, ist1, ist2, iss1, iss2, idate, itime, is,ie,js,je Real timed,climatp,w1st,w1ss,elon,alat,amsk, sst2,sss2,tmsst2,tmsss2, wxy
Interp2d	Subroutine INTERP2D performs 2D bilinear interpolation. It interpolates f(x,y) to the m points g where $1 < x < nx$, $1 < y < ny$. **Calling Sequence:** interp2d(f,nx,ny, g,x,y,m) **Data Declaration:** Integer nx, ny, m Real f,g,y,x
Misng_cf	Subroutine MISNG_CF prints out an error message and halts the program when a missing COAMPS field is detected. **Calling Sequence:** misng_cf(istat,sub,field) **Data Declaration:** Integer istat Character sub, field
Ncom_bicubcc	Subroutine NCOM_BICUBCC computes a bicubic interpolation from a 2D grid of data to a specified set of points. This routine uses polynomials that are cubic in x and y (not splines). It is assumed that the grid being interpolated from is regularly spaced in terms of the two coordinates being used for the interpolation. This routine will extrapolate values that are just outside the grid being interpolated from. However, if the routine is asked to extrapolate very far outside the grid from which data is being interpolated, the program will stop and an error message will be written to unit 6. **Calling Sequence:** ncom_bicubcc(f1,n1,m1,x2,y2,f2,n2,m2,irange) **Data Declaration:** Integer n1,m1,n2,m2,irange Real f1,x2,y2,f2
Ncom_biliner	Subroutine NCOM_BILINER performs bilinear interpolation of surface flux fields to the model grid. **Calling Sequence:** ncom_biliner(f,md,nd,x,y,g,n,m) **Data Declaration:** Integer md,nd,n,m Real f,x,y,g
Ncom_creep4	Subroutine NCOM_CREEP4 extends values where *amsk*=1 into regions where amsk=0. The method is to replace "bad" pts with an average of the adjoining "good" pts. Only the adjoining "good" points to the E,W,N,and S are used, i.e., the adjacent corner pts are not used. When extending for the purpose of interpolation near land-sea boundaries, only a few iterations may be needed (e.g., itermx=10). To fill the entire field, set itermx > max(n,m) to be sure all pts will be filled.

Subroutine	Description
	Calling Sequence: ncom_creep4(t,amsk,n,m,itermx) **Data Declaration:** Integer n,m,itermx Real amsk, t
Ncom_rotang2	Subroutine NCOM_ROTANG2 determines the rotation angle for wind vectors when converting from a COAMPS Lambert conformal or polar stereographic grid-relative projection to earth-relative (true) coordinates. **Calling Sequence:** ncom_rotang2(igrid,grdlon,gcon,stdlon,m,n,grdrot) **Data Declaration:** Integer igrid,m,n Real gcon,grdlon,gridrot,stdlon,a
R_coa_dr	Subroutine R_COA_DR gets parameters needed for COAMPS fields. These are stored in common block COAMPS, which is in *COAMPS.h*. **Calling Sequence:** r_coa_dr (nest, idate, itime, batch, indsbc, indatp, indtau, indsft, indsfs,indsol,indsst,indsss) **Data Declaration:** Integer nest,idate,itime,indsbc, indatp, indtau, indsft, indsfs, indsol, indsst, indsss Logical batch
Rcoamps4	Subroutine RCOAMPS4 gets wind stress, heat and moisture fluxes generated by the COAMPS model and interpolates them to the ocean model grid. RCOAMPS4 has been updated to include the option to calculate the latent and sensible heat fluxes via bulk formulas using the current model SST. **Calling Sequence:** rcoamps4(indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr, is,ie,js,je,idate,itime,elon,alat,ang,amsk,curdtg,itmsec,md,nd,pa tm2,usflx2,vsflx2,rsflx2,solar2,wspd2,tair2,humd2) **Data Declaration:** Character curdtg Integer indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr, idate,itime,itmsec,md,nd,is,ie,js,je Real elon, alat, ang, amsk, patm2, usflx2, vsflx2, rsflx2, solar2, wspd2, tair2, humd2
Rcoasst4	Subroutine RCOASST4 reads COAMPS reanalysis SST. **Calling Sequence:** rcoasst4(indsst,indsss,nt,mt,n,m,is,ie,js,je,elon,alat,amsk, curdtg,itmsec, md,nd, sst2,sss2) **Data Declaration:** Character curdtg Integer indsst,indsss,nt, mt,n, m, itmsec, md, nd, is, js, je Real elon,alat, amsk, sst2, sss2
Sigz2z	Subroutine SIGZ2Z interpolates model fields to specified depths. Put a special value at land points or simply set to zero. **Calling Sequence:** sigz2z(n,m,l1,kb1,spval,z1,t1,l2,kb2,z2,t2) **Data Declaration:** Integer n,m,l1,l2,kb1,kb2 Real z1,t1,t2,spval,z2
Write_ff	Subroutine WRITE_FF outputs NCOM fields as COAMPS-style flat files. **Calling Sequence:** write_ff(nt, mt,n,m,l,ls,kb,iter,h,hu,hv,h1,h1u,h1v, z_w,z_t, zm,amsk,umsk,vmsk,e,u,v,w,t,s,patm,usflx,vsflx,tsflx,ssflx,sol ar,surruf,zm3)

Subroutine	Description
	Data Declaration: Integer nt,mt,n,m,l,ls,iter Real h,hu,hv,h1,h1u,h1v,z_w, z_t,zm,amsk, umsk, vmsk,e,u,v,w,t,s, patm, usflx, vsflx, tsflx, ssflx, solar, surruf, zm3

5.4.4 Flux Corrected Transport Subroutines (ncom1fct_sigz)

Subroutine	Description
Advr_fct1	Subroutine ADVR_FCT1 calculates: • the first step for FCT advection of scalar fields: • low-order upwind advective fluxes for scalar fields. • high-order advective fluxes for scalar fields. • anti-diffusive fluxes = (high-order fluxes) - (upwind fluxes). • intermediate values of scalar fields using upwind fluxes. Note: Anti-diffusive fluxes (ADFs) must be saved in 3D arrays. **Calling Sequence:** advr_fct1(j,jf,jb,ua,va,wa,flyr, adxr,adyr,adzr,ro,n,m,l,ls,nr, i1,i3,j1,j2,is,ie,isp,iep,js,je,iec,sigdif,locate,ramp,times,dti2,small,da,dar,sw,sm,dsm,zw,zm,dzm,amsk,sor,d1,r,rmean,xk,yk, flx,flz,dv_i3,dvpr) **Data Declaration:** Integer j,jf,jb,n,m,l,ls,nr,i1,i3,j1,j2, ie, is, isp, iep, js, je, iec Real ua, va,wa, flyr, adxr, adyr, adzr, ro, ramp, times, dti2,small, da, dar, sw, sm, dsm, zw,zm,dzm, amsk,sor,d1,r,rmean,xk,yk
Advr_fct2	Subroutine ADVR_FCT2 limits anti-diffusive fluxes, updates the intermediate scalar values for the anti-diffusive fluxes, and adds some additional source terms (surface flux, solar flux, river inflows). **Calling Sequence:** advr_fct2(j, adxr,adyr,adzr,rp,rn, n,m,l,ls,nr,i1,j1,j2,is,ie, isp,iep,js,je,ke,iec,indriv,indrivr,indbio,locate,idate,itime,iter,ramp,times,dti2,ext,small,da,dar,sw,sm,dsm,zw,zm,dzmr,amsk,sor,d1,r,rsflx,solar,nrvmax,lriv,iriv,jriv,isriv,ieriv,irv1,irv2,rriv, w1riv, rsor, dtdvr, dr) **Data Declaration:** Integer j,n,m,l,ls,nr,i1,j1,j2,is, ie,isp,iep,ke,js,je,iec, indriv, indrivr, indbio idate, itime, iter Real adxr, adyr, adzr, rp, rn, ramp, times, dti2,small, ext, da, dar, sw, sm,dsm,zw, zm, dzmr,amsk, sor, d1, r, rsflx, solar Logical locate
Updatrq_fct	Subroutine UPDATRQ_FCT updates scalar and turbulence fields. Scalar fields are updated using FCT advection. A slab calculation is used for some of the calculations whereby the calculation proceeds through the model domain in x-z sections from the back of the domain to the front. **Calling Sequence:** updatrq_fct(nt,mt,n,m,l,ls,nr,nq,i1,i2,i3,j1,j2,kb,is,ie, isp,iep,js,je,iec,ke,mode,indadvr,indxk,indzk,indtkes,indlxts,in

Subroutine	Description
	driv,indrivr,indbio,indiag,noslip,sigdif,largmix,vector,shrnkwp ,locate,idate,itime,iter,ramp,times,dti2,asf,vg1,vg2,vg3,g,rho0, xkmin,ykmin,xkre,prnxi,zkmmin,zkhmin,zkre,botruf,rlax_ts,rl ax_ds,ext,small,dxur,dxv,dyu,dyvr,da,dar,h1,sw,sm,dsw,dsm,d sm5,dswr,dsmr,zw,zm,dzw,dzm,dzm5,dzwr,dzmr,amsk,umsk,v msk,sor,sorb,e,d,du,dv,d1,d1u,d1v,udb,vdb,u,v,w,r,q,tl,rho,sos, rmean,xk,yk,zkm,zkh,usflx,vsflx,rsflx,solar,surruf,wubot,wvbo t,ilx1,ilx2,rlx,wlx,tmlx,nobmax,nob,iob,job,nrvmax,lriv,iriv,jri v,isriv,ieriv,irv1,irv2,rriv,w1riv,uacr,vacr,wpf,flyr,flyq,qold,ua ,va,wa,rjp1, wxz) **Data Declaration:** Integer nt,mt,n,m,l,ls, nr,np,i1,i2,i3,j1,j2,kb,is,ie, isp,iep,ke,js,j3,iec,mode,indadvr,indxk,indzk,in dtkes,indlxts,indriv,indrivr,indbio,indiag,idate, itime,iter,ilx1,ilx2,nobmax,nob,iob,job,nrvmax,l riv,irv1,irv2,iriv,jriv,isriv,ieriv, Real ramp,times,dti2,asf, vg1,vg2,vg3,g,rho0,xkmin, ykmin,skre,prnxi,zkmmin,zkhmin,zkre,botruf,rl ax_ts,rlax_ds,ext,small,dxur,dxv,dyu,dyvr,da,da r,h1,sw,sm,dsw,dsm,dsm5,dswr,dsmr,zw,zm,dz w,dzm,dzm5,dzwr,dzmr,amsk,umsk,vmsk,sor,s orb,e,d,du,dv,d1,d1u,d1v,udb,vdb,u,v,w,r,q,t1,r ho,sos,rmean,xk,yk,zkm,zkh,usflx,vsflx,rsflx,so lar,surruf,wubot,wvbot rriv,w1riv,rlx,wlx Logical noslip,sigdif,largmix,vector, shrnkwp,locate

5.4.5 Initialization Subroutines (ncom1init_sigz)

Subroutine	Description
Check	Subroutine CHECK checks the model inputs. **Calling Sequence:** check (na, ma, n, m, l, ls, nr, ntyp, i1, i2, i3, j1, j2, times, fda, botruf, cbu,cbv, istype, iptype, qrf, ext, elon, alat, ang, dx, dxu, dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, w, r, rmean) **Data Declaration:** Integer na, ma, n, m, l, ls, nr, ntyp, i1, i2, i3, j1, j2, istype, iptype Real times, fda, botruf, cbu, cbv, qrf, ext, elon, alat, ang, dx, dxu, dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv,

Subroutine	Description
	d1, d1u, d1v, udb, vdb, ub, vb, u, v, w, r, rmean
Chkarr	Subroutine CHKARR checks the range of a real array. **Calling Sequence:** chkarr (name, a, n, m, l, na, ma, ind, amin, amax, ierr, ie) **Data Declaration:** Integer name, n, m, l, na, ma, ind, ierr, ie Real a, amin, amax
Chkint	Subroutine CHKINT checks the range of an integer variable. **Calling Sequence:** chkint (name, iv, ind, imin, imax, ierr, ie) **Data Declaration:** Integer name, iv, ind, imin, imax, ierr, ie
Chklog	Subroutine CHKLOG checks the value of the logical variable. **Calling Sequence:** chklog (name, iv, val, ierr, ie) **Data Declaration:** Integer name, iv, ierr, ie Real val
Chkrel	Subroutine CHKREL checks the range of the real variable. **Calling Sequence:** chkrel (name, a, ind, amin, amax, ierr, ie) **Data Declaration:** Integer name, ind, ierr, ie Real a, amin, amax
Chkrit	Subroutine CHKRIT prints out an error message. **Calling Sequence:** chkrit (string, ierr, ie) **Data Declaration:** Integer ierr, ie Real string
Define	Subroutine DEFINE defines the model parameters. **Calling Sequence:** define (na, ma, n, m, botruf) **Data Declaration:** Integer na, ma, n, m Real botruf
Dragcb	Subroutine DRAGCB calculates the bottom drag coefficients. **Calling Sequence:** dragcb (wetdry, n, m, l, ls, is, ie, ism, iem, js, je, iec, amsk, kb, h1, d1,dsm5, dzm5, botruf, cbmin, cbu, cbv) **Data Declaration:** Integer n, m, l, ls, is, ie, ism, iem, js, je, iec, kb Logical wetdry Real amsk, h1, d1, dsm5, dzm5, botruf, cbmin, cbu, cbv
Initial	Subroutine INITIAL defines initial values for model fields. **Calling Sequence:** initial (na, ma, n, m, l, ls, nr, i1, j1, forward, locate, e, u, v, r) **Data Declaration:** Integer na, ma, n, m, l, ls, nr, i1, j1 Logical forward, locate Real e, u, v, r
Lsmasks	Subroutine LSMASKS calculates land-sea masks. **Calling Sequence:** lsmasks (na, ma, n, m, l, ls, i1, is, ie, js, je, iec, kb, amsk, umsk, vmsk, d,wpf) **Data Declaration:** Integer na, ma, n, m, l, ls, i1, is, ie, js, je, iec, kb Real amsk, umsk, vmsk, d, wpf
Meanr	Subroutine MEANR defines (1) the horizontal mean (horizontally averaged) density field on the model grid and (2) the mean or "climate" scalar (T and S) fields on the

Subroutine	Description
	model grid. **Calling Sequence:** meanr (nt, mt, n, m, l, ls, nr, j1, is, ie, js, je, iec, indden, indcyc, indiag,rho0, g, sm, zm, h1, amsk, r, rmean, ae, be, ce, de, sos) **Data Declaration:** Integer nt, mt, n, m, l, ls, nr, j1, is, ie, js, je, iec, indden, indcyc,indiag Real rho0, g, sm, zm, h1, amsk, r, rmean, ae, be, ce, de, sos
Paramset	Subroutine PARAMSET copies model parameters between the common blocks for all the grids (in ***COMMON.inc***) and the common blocks for the current grid (***NCOMPAR.inc***). ind = flag to denote: =1 get parameters from common blocks for all nests; =2 put parameters into common blocks for all nests. **Calling Sequence:** paramset(ind) **Data Declaration:** Integer ind
Prntpar	Subroutine PRNTPAR prints out model parameters. **Calling Sequence**: prntpar(na,ma,n,m,l,ls,kb,kbu,kbv,is,ie,js,je,iec,fda,botruf,dx, dy,da,dar,h,sw,sm,dsw,dsm,dsm5,dswr,dsmr,zw,zm,dzw,dzm,dzm5,dzwr,dzmr,amsk,umsk,vmsk, wpf) **Data Declaration:** Integer na,ma,n,m,l,ls,kb,kbu,kbv,is,ie,js,je,iec Real fda, botruf,dx,dy,da,dar,h,sw,sm, dsw,dsm, dsm5,dswr,dsmr,zw,zm,dzw,dzm,dzm5,dzwr,dzmr,amsk,umsk,vmsk,wpf
Region	Subroutine REGION defines the model region. **Calling Sequence**: region(na,ma,n,m,indcyc,iec,ibo,elon,alat,dx,dy,h,ang,amsk, wsp) **Data Declaration:** Integer na,ma,n,m,indcyc,iec,ibo Real elon,alat,dx,dy,h,ang,amsk,wsp
Setup1	Subroutine SETUP1 performs some setup calculations. **Calling Sequence**: setup1(na,ma,n,m,l,ls,i1,i2,i3,j1,j2,kb,kbu,kbv,is,ie,ism,iem, isp,iep,js,je,iec,ibo,ke,indcor,indobc,indcyc,shrnkwp,locate,iter times,fda,pi,raddeg,degrad,small,elon,alat,ang,dx,dxr,dxu,dxur, dxv,dxvr,dy,dyr,dyu,dyur,dyv,dyvr,ddx,ddy,da,dar,dau,daur,dav,davr,h,hu,hv,h1,h1u,h1v,sw,sm,dsw,dsm,dsm5,dswr,dsmr,zw zm,dzw,dzm,dzm5,dzwr,dzmr,wpf) **Data Declaration:** Integer na,ma,n,m,l,ls,i1,i2,i3,j1,j2,kb,kbu,kbv,is,ie, ism,iem,isp,iep,ke,js,je,iec,ibo,indcor,indobc,indcyc, iter Real times,fda,pi, raddeg, degrad, small, elon, alat,ang,dx, dxr,dxu,dxur,dxv,dxvr,dy,dyr,dyu,dyur,dyv,dyvr,ddx, ddy,da,dar,dau,daur,dav,davr,h,hu,hv,h1h1u,h1v,sw,dsm5,dswr,dsmr,zw,zm,dzw,dzm,dzm5,dwr,dzmr,wpf
Setup2	Subroutine SETUP2 performs more setup calculations.

Subroutine	Description
	Calling Sequence: setup2(na,ma,n,m,l,ls,nr, i1,i2, j1,j2,kb, is,ie,ism,iem, isp,iep,js,je,iec,indcyc,wetdry,rstart,forward, locate,botruf,cbmin,cbu,cbv,small,h,hu,hv,h1,h1u,h1v,dsm,dsm5,dswr,dsmr,zw,zm,dzw,dzm,dzm5,amsk,umsk,vmsk,e,d,du, dv,d1,d1u,udb,vdb,ub,vb,u,v,r,wpf) **Data Declaration:** Logical rstart,wetdry,forward,locate Integer na,ma,n,m,l,ls,nr, i1,i2,j1,j2,kb,is,ism,iem,isp, iep,js,je,iec, indcyc Real botruf,cbmin,cbu,cbv,smmall,h,hu,h1,h1u,hv,h1v, dsm,dsm5,dzm,dzm5,amsk,umsk,vmsk,e,d,du,dv,d1 d1u,d1v,udb,vdb,ub,vb,u,vbv,r,wpf
Setzero	Subroutine SETZERO initializes some arrays to zero. **Calling Sequence:** setzero(n,m,l,ls,nr,nq,ntyp, kb,kbu,kbv,is,ie,ism,iem,isp,iep,ke, fda,botruf,cbu,cbv,istype,iptype,qrf,ext,elon,alat,ang,dx,dxu,dxv,dxr,dxur,dxvr,dy,dyu,dyv,dyr,dyur,dyvr,da,dau,dav,dar,daur, davr,h,hu,hv,h1,h1u,h1v,sw,sm,dsw,dsm,dsm5,dswr,dsmr, zw,zm,dzw,dzm,dzm5,dzwr,dzmr, amsk,umsk,vmsk, sor,sorb, e,d,du,dv,d1,d1u,d1v,udb,vdb,ub,vb,u,v,w,r,q,tl,rho,sos,rmean, xk,yk,zkm,zkh,wubot,wvbot,patm,usflx,vsflx,rsflx,solar,surruf, nobmax,iob,job,iobi,jobi,ivob,jvob,eob,ubob,vbob,cgwb,uob,vob,rob,ntc,etab,etpb,utab,utpb,vtab,vtpb,nrvmax,iriv,jriv,isriv,ieriv,rriv) **Data Declaration:** Integer n,m,l,ls,nr,nq,ntyp, kb,kbu,kbv,is,ie,ism,iem, isp,iep,ke,istype,iptype,nobmax,iob,job,iobi, jobi,ivob,jvob,ntc, nrvmax,iriv,jriv,isriv,ieriv Real qrf,ext,elon,alat,ang,dx,dxu,dxv,dxr,dxur, dxvr,dy,dyu,dyv,dyr,dyur,dyvr,da,dau,dav,dar,daur,davr,h,hu,hv,h1,h1u,h1v,sw,sm,dsw,dsm,dsm5,dswr,dsmr,zw,zm,dzw,dzm,dzm5,dzwr,dzmr,amsk,umsk,vmsk,sor,sorb,e,d,du,dv,d1,d1u,d1v,udb,vdb,ub,vb,u,v,w,r,q,tl,rho,sos,rmean,xk,yk,zkm,zkh,wubot,wvbot,patm,usflx,vsflx,rsflx,solar,surruf
Vergrid	Subroutine VERGRID defines the vertical grid. **Calling Sequence:** vergrid(l,ls,zw) **Data Declaration:** Integer l,ls Real zw

5.4.6 Nested Grid Boundary Condition Interpolation Subroutines (ncom1nest2)

Subroutine	Description

Subroutine	Description
Feebko	Subroutine FEEBKO feeds back information from FM, or nested grid to a CM, or parent grid. The CM values are replaced with FM values only if there is at least one FM point within the CM grid-cell volume. This calculation is valid for any FM to CM grid-spacing ratio. **Calling Sequence:** feebko(nestf, nestc, nratio, isf, jsf, nc, mc, lc, nrc, nf, mf, lf, nrf, kbc, kbf,j1c, j1f, amskc, rc, amskf, rf) **Data Declaration:** Integer nestf, nestc, nratio, isf, jsf, nc, mc, lc, nrc, nf, mf, lf, nrf,kbc, kbf, j1c, j1f Real amskc, rc, amskf, rf
Intbln2	Subroutine INTBLN2 spatially interpolates from a CM to a point on a nested FM. **Calling Sequence:** intbln2 (r,ncg,id,ec,i,j,a,b,ef) **Data Declaration:** Integer ncg, i,j, id Real r, ec, a, b, ef
Nestbco2	Subroutine NESTBCO2 computes the boundary values needed for calculations on the nest. The grid for which values are being calculated is referred to here as the fine mesh, or nested grid (FM). The grid from which values are being taken is referred to as the coarse mesh (CM), or parent grid, to the nested grid. **Calling Data:** nestbco2 (nest, nestc, nratio, isf, jsf, nct, mct, nc,mc, lc, nrc, nf, mf, lf, nft,mft,nrf, ibof, kbf,kbuf, kbvf, i1, i2, j1, j2, timesc, timesf, amskc, ec, udbc, vdbc, uc, vc, rc, amskf,hf, nobmax, nob, neob, nuob, nvob, iob, job, ivob, jvob, iob1, iob2, eob, ubob, vbob, uob, vob, rob, tmob) **Data Declaration:** Integer nest, nestc, nratio, isf, jsf, nct,mct,nc, mc, lc, nrc, nft,mft, ieecf, nf,mf, lf, nrf, ibof,kbf,kbuf, kbvf, i1, i2, j1, j2, nobmax, nob, neob, nuob, nvob, iob, job, ivob, jvob, iob1, iob2 Real timesc, timesf, amskc, ec, udbc, vdbc, uc, vc, rc, amskf, hf, eob, ubob,vbob, uob, vob, rob, tmob
Nestbwtr2	Subroutine NESTBWTR2 calculates weights needed to interpolate from a coarse mesh to a point on a nested fine mesh at grid cell centers. **Calling Sequence:** nestbwtr2 (nft,mft,ibofg,ncg,icg1,jcg1,amc,r, isf,jsf,ifg,jfg,id,i, j,a,b) **Data Declaration:** Integer nft,mft,ibofg,ncg,icg1,jcg1,isf,jsf,ifg,jfg,id,i,j Real amc, r, a, b
Nestbwtu2	Subroutine NESTBWTU2 calculates weights needed to interpolate from a coarse mesh to a point on a nested fine mesh at a normal velocity point. These normal velocity points lie on the boundary on the FM and also lie along grid-cell boundaries of the CM. **Calling Sequence:** nestbwtu2 (nft,mft,ibofg,ncg,icg1,jcg1,amc,r, isf,jsf,ifg,jfg,id,i, j,a,b) **Data Declaration:** Integer nft,mft,ibofg,ncg,icg1,jcg1,isf,jsf,ifg,jfg,id,i,j Real amc, r, a, b

Subroutine	Description
Nestbwtv2	Subroutine NESTBWTV2 calculates weights needed to interpolate from a coarse mesh to a point on a nested fine mesh at a tangent normal velocity point. **Calling Sequence:** nestbwtv2 (nft,mft,ibofg,ncg,icg1,jcg1,amc,r, isf,jsf,ifg,jfg,id,i,j,a,b) **Data Declaration:** Integer nft,mft,ibofg,ncg,icg1,jcg1,isf,jsf,ifg,jfg,id,i,j Real amc, r, a, b
Nestindx	Subroutine NESTINDX calculates indices for XCLGET calls for all tiles. XCLGET calls are to get CM values for interpolation to the FM, or nested grid. This same calculation is done on each tile. This subroutine also calculates CM mask values along open boundaries of the FM. These are used to calculate indices and weights for interpolation. **Calling Sequence:** nestindx(nest,nestc,gratio,isf,jsf,nct,mct,nc,mc,lc,nft,mft,nf,mf,lf,ibofg,amskc,amskf,hf,ncg,icg1,jcg1,amc, udbc) **Data Declaration:** Integer nest,nestc,isf,jsf,nct,mct,nc,mc,lc,nft,mft,nf,mf,lf,ibofg,ncg,icg1,jcg1 Real gratio,amskf,hf,amskc,amc,udbc
Testxclg	Subroutine TESTXCLG tests calls to XCLGET. **Calling Sequence:** testxclg(ind,ncg,icg1,jcg1,udbc,nc,mc) **Data Declaration:** Integer ind,nc,mc,ncg,icg1,jcg1 Real udbc
Xclget2	Subroutine XCLGET2 acts as an interface to XCLGET to keep aline from being over-written on a call to XCLGET unless the local node=mnflg. **Calling Sequence:** xclget2(aline,nl, a,n,m, i1,j1,ii,ji, mnflg) **Data Declaration:** Integer nl,n,m,i1,j1,ii,ji,mnflg Real aline, a

5.4.7 *Open Boundary Condition Subroutines (ncom1obc_sigz)*

Subroutine	Description
Cycbc	Subroutine CYCBC sets lateral boundary values on cyclic boundaries for problems with cyclic BC. With a C grid and second-order spatial differences, three grid cells at the end of the grid mask overlap in the direction taken to be cyclic. **Calling Sequence:** cycbc (ind, aax, aay, n, m, l, nr, nq, i1, j1, j2, indbaro, indxk, indzk,indcyc, locate, e, udb, vdb, ub, vb, u, v, w, r, q, tl, zkm, zkh, wubot, wvbot) **Data Declaration:** Integer ind, n, m, l, nr, nq, i1, j1, j2, indbaro, indxk, indzk, indcyc Real e, udb, vdb, ub, vb, u, v, w, r, q, tl, zkm, zkh, wubot, wvbot Logical locate
Cycset	Subroutine CYCSET sets cyclic boundary conditions for model variables. **Calling Sequence:** cycseti (indcyc, iloc, iset, n, m, ld, f) **Data Declaration:** Integer indcyc, iloc, iset, n, m, ld

Subroutine	Description
	Real f
Cycseti	Subroutine CYCSETI sets cyclic boundary conditions for model variables. CYCSETI differs from CYCSET in that an integer array, rather than a real array, is set cyclic. **Calling Sequence:** cycseti (indcyc, iloc, iset, n, m, ld, f) **Data Declaration:** Integer indcyc, iloc, iset, n, m, ld Real f
Halo	Subroutine HALO updates halos. **Calling Sequence:** halo (ind, aax, aay, na, ma, n, m, l, nr, nq, i1, j1, j2, indbaro, indxk, indzk,locate, e, udb, vdb, ub, vb, u, v, w, r, q, tl, zkm, zkh, wubot, wvbot) **Data Declaration:** Integer ind, na, ma n, m, l, nr, nq, i1, j1, j2, indbaro, indxk, indzk Real aa., aay, e, udb, vdb, ub, vb, u, v, w, r, q, tl, zkm, zkh,wubot, wvbot Logical locate
Openbc	Subroutine OPENBC defines values at open boundaries. Most of the open boundary conditions in this routine have been consolidated. **Calling Sequence:** openbc (ind, ax, aay, nt, mt, n, m, l, nr, nq, i1, i2, i3, j1, j2, kb, kbu, kbv,is, ie, ism, iem, isp, iep, iec, ibo, ramp, times, dti2, elon, alat, ang, dxr, dxur, dxvr, dyr, dyur, dyvr, h, hu, hv, h1, dsm, dzm, du, dv, amsk, umsk, vmsk, e, d, d1, udb, vdb, ub, vb, u, v, w, r, q, tl, zkm, zkh, wubot, wvbot, nobmax, nob, neob, nuob, nvob, iob, job, iobi, jobi, ivob, jvob, iob1, iob2, eob, ubob, vbob, cgwb, uob, vob, rob, tmob, ntc, etab, etpb, utab, utpb, vtab, vtpb) **Data Declaration:** Integer ind, nt, mt, n, m, l, nr, nq, i1, i2, i3, j1,j2, kb, kbu, kby, is,ie, ism, iem, isp, iep, iec, ibo, nobmax, nob, neob, nuob,nvob, iob, job, iobi, jobi, ivob, jvob, iob1, iob2, ntc Real ax, aay, ramp, times, dti2, elon, alat, ang, dxr, dxur, dxvr,dyr, dyur, dyvr, h, hu, hv, h1, dsm, dzm, du, dy, amsk, umsk, vmsk, e, d, d1, udb, vdb, ub, vb, u, v, w, r, q, tl, zkm, zkh, wubot, wvbot, eob, ubob, vbob, cgwb, uob, vob, rob, tmob, etab, etpb, utab, utpb, vtab, vtpb
Readobc	Subroutine READOBC reads OBC data from an input file and computes the weighting to be used for linear interpolation to the model time. **Calling Sequence:** readobc (nt ,mt, n, m, l, nr, iec, idate, itime, times, nobmax, nob, neob,nuob, nvob, iob, job, ivob, jvob, iob1, iob2, eob, ubob, vbob, uob, vob, rob, tmob, w1) **Data Declaration:** Integer nt, mt, n, m, l, nr, iec, idate, itime, nobmax, nob, neob,nuob, nvob, iob, job, ivob, jvob, iob1, iob2

Subroutine	Description
	Real times, eob, ubob, vbob, uob, vob, rob, tmob, w1

5.4.8 Output Subroutines (ncom1out_sigz)

Subroutine	Description
Bndypro	Subroutine BNDYPRO inspects profiles at open boundary points. This subroutine is for diagnostics only. **Calling Sequence:** bndypro (n, m, l, ls, nr, nobmax, nob, i1, j1, sw, sm, zw, zm, d1,iob, job, rob) **Data Declaration:** Integer n, m, l, ls, nr, nobmax, nob, il, jl, iob, job Real sw, sm, zw, zm, dl, rob
Kinergy	Subroutine KINERGY calculates total kinetic energy. **Calling Sequence:** kinergy (nest,nt,mt,n, m, l, i1, times, rho0, d1, da, dsm, dzm, amsk, u, v, wsp1,wsp2,ake) **Data Declaration:** Integer nest,nt,mt,n,m,l,i1 Real times, rho0, ake,d1, da, dsm, dzm, amsk, u, v, wsp1, wsp2
Ncom_Output	Subroutine NCOM_OUTPUT outputs model results. **Calling Sequence:** ncom_output (nt, mt, n, m, l, ls, nr, nq, i1, i2, i3, j1, j2, kb, iter, times,botruf, cbu, cbv, ext, elon, alat, ang, dx, dxu, dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, w, r, q, tl, rho, sos, rmean, xk, yk, zkm, zkh, patm, usflx, vsflx, rsflx, solar, surruf, zlay, amp2, pha2, vmax, hneg) **Data Declaration:** Integer nt, mt, n, m, l, ls, nr, nq, i1, i2, i3, j1, j2, kb, iter Real times, botruf, cbu, cbv, ext, elon, alat, ang, dx, dxu,dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, w, r, q, tl, rho, sos, rmean, xk, yk, zkm, zkh, patm, usflx, vsflx, rsflx, solar, surruf, zlay, amp2, pha2, vmax, hneg
Outpt	Subroutine OUTPT outputs values and profiles at particular (horizontal) grid points. **Calling Sequence:** outpt(nest, ii, ig,jg, i,j, nt,mt,n,m,l,ls,nr,kb,elon,alat,dx,dy,h,h1, ang,sw,sm,zw,zm,botruf,cbu,cbv,times,e,u,v,w,t,s,rho,rmean,q 2,q2l,tl,zkm,zkh,ext,patm,usflx,vsflx,rsflx,solar,surruf) **Data Declaration:** Integer nest,ii,ig,jg, i,j,nt,mt,n,m,l,ls,nr,kb Real elon,alat,dx,dy,h,h1,ang,sw,sm,zw,zm,botruf,cbu,

Subroutine	Description
	cbv,times,e,u,v,w,t,s,rho,rmean,q2,q2l,t1, zkmzkh,ext,patm,usflx,vsflx,rsflx,solar,arruf
Trans_st	Subroutine TRANS_ST calculates transport through a single point. The location of the strait may involve a single section along the x or y axes, or two sections that meet at right angles in the case that *is1* ne *is2* and *js1* ne *ns2*. **Calling Sequence:** trans_st(is1,js1,is2,js2,idir,n,m,l,kb,dxv,dyu,dsm,dzm,d1u,d1v, u,v,tin,tot) **Data Declaration:** Integer is1,js1,is2,js2,idir,n,m,l,kb Real dxv,dyu,dsm,d1u,d1v,u,v,tin,tot
Transp	Subroutine TRANSP computes transport through a single x-z or y-z section. Calculation below assumes that velocities are zero at land points. **Calling Sequence:** transp(n,m,l,isg,jsg,ns,ii,ji,isgn,dsm,dzm,vs,dxs,d1s,tin,tot) **Data Declaration:** Integer n,m,l,isg,ns,ii,ji,isgn Real vs,dxs,d1s,dsm,dzm,tin,tot
Transprt	Subroutine TRANSPRT calculates transports through straits or other sections. The rules for defining the location of the strait and the direction of flow through it are: (1) Looking downstream (i.e., in the direction defined to be the direction of positive transport) through the strait, pt [*is1*,*js1*] is on the left and pt [*is2*,*js2*] is on the right. The strait can be defined as a single section along the x or y axis, or two sections that form a right angle in the case that the left end of the section has different *i* and *j* indices than the right end of the section. This attempts to accommodate straits that do not lie along either the x or y axes. (2) The direction from pt [*is1*,*js1*] to pt [*is2*,*js2*] or to the corner pt (if there is a corner pt), is indicated by *idir*. The *idir* values are: =1 +x; =2 -x; =3 +y; =4 -y. The indices [*is1*,*js1*] and [*is2*,*js2*] correspond to the velocity points along which the transport is calculated. **Calling Sequence:** transprt(nest,nt,mt,n,m,l,kb,dxv,dyu,dsm,dzm,iter,dti,times, d1u,d1v,u,v) **Data Declaration:** Integer nest,nt,mt,n,m,l,kb,iter Real dxv,dyu,dsm,dti,times,d1u,d1v,u,v
Wm_vol	Subroutine WM_VOL computes the volume of various water masses that are present in the model domain. The water mass T, S, and potential density bounds are defined within an input file that is read in. **Calling Sequence:** wm_vol(nest,nt,mt,n,m,l,ls,j,times, dx,dy,d1,dsm,dzm,amsk,r) **Data Declaration:** Integer nest,nt,mt,n,m,l,ls,j1 Real times,dx,dy,d1,amsk,r,dsm

5.4.9 Generic and Plotting Subroutines (ncom1plib)

File ncom1plib contains generic routines from Paul Martin's library "plib" as well as plotting subroutines.

Subroutine	Description
Akcoll	Subroutine AKCOLL provides attenuation coefficients for pure seawater at wavelength w for range (700-2650 nm). For wavelengths below 800 nm, the data

Subroutine	Description
	from Smith and Baker (1981) in subroutine AKSMITH should be used since these data should have more accuracy and better resolution. **Calling Sequence:** akcoll (w, ak) **Data Declaration:** Real w, ak
Akmorl	Subroutine AKMORL calculates attenuation coefficient (in seawater) for light of wavelengths from 200 to 800 nm (Morel, 1988). Morel's data actually only cover the range 400 to 700 nm. Above and below this range, attenuation coefficients for pure seawater (Smith and Baker, 1981) are used, along with an extrapolation of Morel's chlorophyll parameters. Hence, the effects of chlorophyll on attenuation will not be very accurate outside the range 400-700 nm (but may be better than nothing). **Calling Sequence:** akmorl (c, w, ak) **Data Declaration:** Real c, w, ak
Ce_mel	Subroutine CE_MEL calculates coefficients of thermal expansion for temperature and salinity using the equation of state from POM developed by George Mellor, which is described in Mellor (1991). **Calling Sequence:** ce_mel(t,s,zm,alpha,beta) **Data Declaration:** Real t,s,zm,alpha,beta
Dateadd	Subroutine DATEADD calculates *idate* and *itime,* which is timed days (real) after the reference date idate0, itime0. For time differences of about one year, the calculation of the time difference is accurate to within about one minute for 32-bit integers. **Calling Sequence:** dateadd (idate0, itime0, timed, idate, itime) **Data Declaration:** Integer idate0, itime0, idate, itime Real timed
Dateadd2m	Subroutine DATEADD2M calculates *idate* and *itime*, which is timed days (real) after the reference date *idate0*, *itime0.* For time differences of about one year, the calculation of the time difference is accurate to within about one minute for 32-bit integers. **Calling Sequence:** dateadd2m (idate0, itime0, timed, idate, itime) **Data Declaration:** Integer idate0, itime0, idate, itime Real timed
Datedfc	Subroutine DATEDFC determines the type of date input and (a) for a climate date, calculates elapsed time in days from the beginning of the year, (b) for non-climate (real time) date, calculates elapsed time in days from the reference date. **Calling Sequence:** datedfc (idate, itime, idate0, itime0, indclim, timed) **Data Declaration:** Integer idate, itime, idate0, itime0, indclim Real timed
Datedif	Subroutine DATEDIF calculates elapsed time in days (real) from the reference date. For time differences of about one year, the calculation of the time difference is accurate to within approximately one minute for 32-bit integers. **Calling Sequence:** datedif (idate, itime, idate0, itime0, timed) **Data Declaration:** Integer idate, itime, idat0, itime0 Real timed
Daycen	Subroutine DAYCEN converts a specific date and time to the fractional day-of-the-

Subroutine	Description
	century, which is defined as the number of days since 00Z, January 1, 1900. This is used for temporal interpolation and to compute time differences between dates. **Calling Sequence:** daycen (idate, timed, dcen, dcend) **Data Declaration:** Integer idate Real timed, dcen, dcend
Dayceni	Subroutine DAYCENI converts a fractional day-of-the-century, which is defined as the number of days since 00Z, January 1, 1900, to a date and time. The year must lie between 1901 and 2099. This subroutine is the inverse of subroutine DAYCEN. All years divisible by four are leap years except for century years not divisible by 400, e.g., 1900 was NOT a leap year. Also, use dcend (double precision) or if dcend is zero, use dcen (single precision). **Calling Sequence:** dayceni (dcen, dcend, idate, timed) **Data Declaration:** Integer idate Real dcen, dcend, timed
Dayyr	Subroutine DAYYR converts a specific date and time to the year and the fractional day of the year, which is defined as the (fractional) number of days since 00Z January 1. **Calling Sequence:** dayyr (idate, timed, iyear, dayr) **Data Declaration:** Integer idate, iyear Real timed, dayr
Dena_mel	Subroutine DENA_MEL calculates *in situ* density minus 1000 kg/m^3 (*rho*) using the equation of state from POM developed by George Mellor, which is described in Mellor (1991). **Calling Sequence:** dena_mel (t,s,zm,rho) **Data Declaration:** Real t,s,zm,rho
Ext2bnd	Subroutine EXT2BND provides extinction parameters for 2-band representations of Jerlov's solar extinction profiles (Jerlov, 1968). **Calling Sequence:** ext2bnd (itype, fr1, fr2, ex1, ex2) **Data Declaration:** Integer itype Real fr1, fr2, ex1, ex2
Extmrl5	Subroutine EXTMRL5 computes solar flux attenuance as a function of the chlorophyll-like (mean) pigment concentration (c) according to the chlorophyll-attenuation model of Andre Morel (1988), along with attenuation data from Smith and Baker (1981) and Neumann and Pierson (1966). EXTMRL5 differs from EXTMRL4 in that some simplifications have been made to streamline the calculation. The solar spectrum for the fraction of total solar radiation is defined internally here in a data statement rather than being computed, and the number of wavelength bands is significantly reduced; no distinction is made between direct and diffuse radiation (the effective mean in water angles for these two components was not that much different). Compare with EXTMRL5 to note the simplifications that have been made. **Calling Sequence:** extmrl5 (c, n, z, gam) **Data Declaration:** Integer n

Subroutine	Description
	Real c, z, gam
Idateadd	Subroutine IDATEADD adds elapsed time in the form of days (*iday*), hrs (*ihr*), min (*min*), sec (*isec*), and hundredths of sec (*ihsec*) to a date and time of the form idate1 = YYYYMMDD and itime1 = HHMMSSCC (where CC indicates hundredths of sec) to generate a resultant date and time (*idate2*, *itime2*) of the same form as the input date and time. **Calling Sequence:** idateadd(idate1,itime1,iday,ihr,min,isec,ihsec, idate2,itime2) **Data Declaration:** Integer idate1,itime1,ihr, iday, isec,ihsec,idate2,itime2
Idatedif	Subroutine IDATEDIF computes the temporal difference between two dates, i.e., the temporal difference between the date specified by (idate2,itime2) and the date specified by (idate1,itime1). The temporal difference is returned as an integer number of days (iday), hours (ihr), minutes (min), seconds (isec), and hundredths of seconds (ihsec). **Calling Sequence:** idatedif(idate1,itime1, idate2,itime2, iday,ihr,min,isec,ihsec,) **Data Declaration:** Integer idate1,itime1,ihr, iday, isec,ihsec,idate2,itime2
Idayyr	Subroutine IDAYYR calculates the integer day of the year given the year, month, and day of the month. All years divisible by four are leap years except for century years not divisible by 400, e.g., 1900 and 2100 are NOT leap years. Treat iyear = 0 as a non-leap year. Iyear = 0 is sometimes used when specifying dates for annually varying climatological data, e.g., see subroutine DATEDFC. **Calling Sequence:** idayyr (iyear, month, iday, idayr) **Data Declaration:** Integer iyear, iday, idayr, month
Idcen2idt	Subroutine IDCEN2IDT converts the number of days since 00Z, Jan 1, 1900 to a date of the form (YYYYMMDD). **Calling Sequence:** idcen2idt(idays,idate) **Data Declaration:** Integer idays, idate
Idt2idcen	Subroutine IDT2IDCEN converts a date of the form (YYYYMMDD) to the number of days since 00Z, Jan 1, 1900. **Calling Sequence:** idcen2idt(idate, idays) **Data Declaration:** Integer idays, idate
Idt2ymd	Subroutine IDT2YMD converts an integer of the form YYYYMMDD to year, month, and day. **Calling Sequence:** idt2ymd (idate, iyear, month, iday) **Data Declaration:** Integer idate, iyear, iday, month
Intrpb	Subroutine INTRPB performs linear interpolation. The data t1 at points z1 are interpolated to the points z2. Z1(k) and z2(k) are assumed to be either both increasing or both decreasing with the index k. No extrapolation is used outside the range z1(1) to z1(n1), i.e. for z2 < z1(1) t2 = t1(1), and for z2 > z1(n1) t2 = t1(n1). **Calling Sequence:** intrpb (n1, z1, t1, n2, z2, t2) **Data Declaration:** Integer n1, n2 Real z1, t1, zl, zr, z2, t2
Intrpz2	Subroutine INTRPZ2 computes weights for vertically averaging a field to a particular depth z2. INTRPZ2 differs from INTRPZ in that depths are input as a 3D field, rather

Subroutine	Description
	than as a sigma or z-level grid. This allows the use of more general vertical grids. **Calling Sequence:** intrpz2 (z, nz, mz, lz, amsk, nm, mm, lm, indgrd, nv, indf, iz, i, j, z2, k1, a,b, amsk2) **Data Declaration:** Integer nz, mz, lz, nm, mm, lm, indgrd, nv, indf, iz, i, j, k1 Real z, amsk, z2, a, b, amsk2 **Comments:** If z2 is above the uppermost model level, the uppermost model value is used. If z2 is below the lowest model level, but above the bottom, the lowest model value is used. If z2 is below the bottom, the value of the land-sea mask (amsk2) is set to zero. Set horizontal indices to accommodate offset (staggered) fields. Offset plots are made only when (i) the field is a vector field (nv > 1), (ii) the x or y component of the vector field is plotted (indf = 1 or 2), and (iii) the grid is staggered (indgrd > 1). Zb is the mean depth of the model points used for the interpolation. The following calculation of zb accounts for whether the model field is located at layer interfaces or layer midpoints, and whether or not values are being calculated at staggered grid points: zb = 0.25*(z(ia, ja, 1) + z(ib, jb, 1) + z(ia, ja, 1+kp) + z(ib, jb ,1+kp)) For points (z2) above the shallowest model point, the value at the shallowest model point is used, i.e., a = 1.0 (no extrapolation). For points (z2) below the deepest model point, but above the bottom, one has several choices: (a) use the value at the deepest model point (set a = 0), or (b) use some type of downward extrapolation, e.g., for linear extrapolation set a = (zb-z2)/(zb-za), or (c) mask the value. Either (a) or (b) will result in spurious looking values near a sloping bottom, either too high or too low. Choice (c) avoids spurious looking values on the plot, but truncates the bottom to the midpoint of the bottom layer for fields defined at layer midpoints. The best choice would probably be to interpolate to z-levels and horizontally extend values to fill in areas between the deepest model value and the bottom, but this would require more effort for the user.
Itm2hms	Subroutine ITM2HMS converts an integer of the form HHMMSSCC to hours, minutes, seconds, and hundredths of seconds. **Calling Sequence:** itm2hms (itime, ihr, min, isec, ihsec) **Data Declaration:** Integer itime, ihr, min, isec, ihsec
Itm2tm	Subroutine ITM2TM converts an integer of the form HHMMSSCC to the fractional time of day. **Calling Sequence:** itm2tm (itime, timed) **Data Declaration:** Integer itime Real timed
Mld_tb	Subroutine MLD_TB calculates surface mixed-layer depth (MLD) from the temperature, salinity, or density field. The MLD is calculated as the depth at which the temperature, salinity, or density becomes "*delt*" less than the surface value, or "*delt*" greater than the bottom value if computing the bottom MLD. **Calling Sequence:** mld_tb(iu,indmld,indmld2,delt,t,nt,mt,lt,s,ns,ms,ls,nr,mr,lr, z,nz, mz,lz,amsk,nm,mm,lm,n1,n2,m1,m2,d) **Data Declaration:** Integer iu,indmld,indmld2,nt,

Subroutine	Description
	mt,lt,ns,ms,ls,nr,mr,lr,nz,mz,lz,nm,mm,lm,n1,n2,m1,m2 Real delt,t,s,r,z,amsk,d
Obcpts	Subroutine OBCPTS sets up open boundary points for an ocean model grid. A land-sea mask or the depth can be used to define which points are open boundary points. For most grids, the boundary rows are at the edge of the grid, i.e., n1 = 1, n2 = n, m1 = 1, m2 = m. Some models, such as ECOM-si, use the second row in from the edge of the grid for the open boundary points, in which case the scenario would be n1 = 2, n2 = n-1, m1 = 2, m2 = m-1. **Calling Sequence:** obcpts (indcyc, n, m, iec, n1, n2, m1, m2, h, hmin, hs, nobmax, nob, neob,nuob, nvob, iob, job, iobi, jobi, ivob, jvob) **Data Declaration:** Integer indcyc, n, m, iec, n1, n2, m1, m2, nobmax, nob, neob,nuob, nvob, iob, job, iobi, jobi, ivob, jvob Real h, hmin, hs
Openptt	Subroutine OPENPTT sets up open boundary points for an ocean model grid. A land-sea mask or the depth can be used to define which points are open boundary points. OPENPTT differs from OPENPTS in that it can be used for tiles in a parallel computing environment where some of the edges are interior edges that abut neighboring tiles. For most grids, the boundary rows are at the edge of the grid, i.e., n1 = 1, n2 = n, m1 = 1, m2 = m. Some models, such as ECOM-si, use the second row in from the edge of the grid for the open boundary points, which would be n1 = 2, n2 = n-1, m1 = 2, m2 = m-1. **Calling Sequence:** openptt (indcyc, n, m, iec, n1, n2, m1, m2, h, hmin, hs, nobmax, nob, iob,job, iobi, jobi, kob) **Data Declaration:** Integer indcyc, n, m, iec, n1, n2, m1, m2, nobmax, nob, iob, job,iobi, jobi, kob Real h, hmin, hs
Plot_tsd	Subroutine PLOT_TSD plots a T-S scatter diagram overlaid on potential density. **Calling Sequence:** plot_tsd(nest,t,nt,mt,lt,s,ns,ms,ls,n1,n2,m1,m2,z,nz,mz,lz,amsk,nm,mm,lm) **Data Declaration:** Integer nest,nt,mt,lt,ns,ms,ls,nz,mz,lz,nm,mm,lm,n1,n2,m1,m2 Real t,s,z,amsk
Plotuv5 or Xplotuv5	Subroutine PLOTUV5 prints or plots scalar or horizontal vector fields. It prints/plots contours of u or v (x and y components of vector field) or contours of vector magnitude. It also plots vector arrows. PLOTUV5 differs from PLOTUV4 in that PRNPLT4 has been modified to allow for a halo around the model grid (used in tiling for paralleling). Also, calls to plot routines can be turned off. PLOTUV4 differs from subroutine PLOTUV3 in that arbitrary vertical grids can be accommodated (the input depth array z is changed to 3D). To turn off plotting on computers where plotting software is not available, either comment out calls to or provide dummy routines for the following subroutines: PSETSPV, PSETVFR, PRNTE, PSETAX, PSETLOC, PLTCON, and PLTVEC. **Calling Sequence:** plotuv5 (nest, indp, u, nu, mu, lu, v, nv, mv, lv, n1, n2, m1,

Subroutine	Description
	m2, l1, l2,indgrd, iu, iz, z, nz, mz, lz, e, ne, me, h, nh, mh, amsk, nm, mm, lm, name, amult, cint, vscale) **Data Declaration:** Integer nest, indp, nu, mu, lu, nv, mv, lv, n1, n2, m1, m2, l1, l2,indgrd, iu, iz, nz, mz, lz, ne, me, nh, mh, nm, mm, lm Real u, v, z, e, h, amsk, amult, cont, vscale Character name **Common Blocks:** CONRE4 PRNTEI4 PRNTER4
Prnplt0	Subroutine PRNPLT0 prints or plots a scalar or horizontal vector field. Modified to allow for nests, multi-processors, and halos for use in NCOM. **Calling Sequence:** prnplt0 (nest, time, indgrd, n, m, l, am, nam, mam, lam, u, nu, mu, lu, v,nv, mv, lv, name, amult, cint, vscale) **Data Declaration:** Integer nest, indgrd, n, m, l, nam, mam, lam, nu, mu, lu, nv, mv, lv Real time, am, u, v, amult, cint, vscale Character name
Prntplt10	Subroutine PRNPLT10 prints or plots sections of 2D and 3D model fields. PRNPLT10 differs from subroutine PRNPLT9 in that a number of additional fields have been added. **Calling Sequence:** prnplt10(nest,time,indgrd,iu,n,m,l, x,nx,mx,y,ny,my,dx, ndx,mdx,dy,ndy,mdy,z,nz,mz,lz,h,nh,mh,am,nam,mam,lam,e,ne,me,ue,nue,mue,ve,nve,mve,sorb,nsorb,msorb,sor,nsor,msor,lsor,u,nu,mu,lu,v,nv,mv,lv,w,nw,mw,lw,phi,nphi,mphi,lphi,p,np,mp,lp,t,nt,mt,lt,s,ns,ms,ls,r,nr,mr,lr,ta,nta,mta,lta,sa,nsa,msa,lsa,ra,nra,mra,lra,bn,nbn,mbn,lbn,bp,nbp,mbp,lbp,bz,nbz,mbz,lbz,bd,nbd,mbd,lbd,q,nq,mq,lq,ql,nql,mql,lql,xkm,nxkm,mxkm,lxkm,ykm,nykm,mykm,lykm,xkh,nxkh,mxkh,lxkh,ykh,nykh,mykh,lykh,zkm,nzkm,mzkm,lzkm,zkh,nzkh,mzkh,lzkh,cbfx,ncbfx,mcbfx,cbfy,ncbfy,mcbfy,sr,nsr,msr,br,nbr,mbr,ext,next,mext,lext,pa,npa,mpa,tx,ntx,mtx,ty,nty,mty,qr,nqr,mqr,q0,nq0,mq0,ep,nep,mep,tlx,ntlx,mtlx,ltlx,slx,nslx,mslx,lslx,wlx,nwlx,mwlx,lwlx) **Data Declaration:** Integer nest,indgrd,iu,n,m,lnx,mx,ny,my,ndx,mdx,ndy, mdy,nz,mz,lznh,mh,nam,mam,lam,ne,me,nue,mue,nve,mve, nsorb,msorb,nsor,msor, lsor,nu,mu,lu,nv,mv,lv,nw,mw,lw,nphi,mphi,lphi,np,mp,lpnt,mt,lt,ns,ms,ls,nr,mr,lr,nta,mta,lta, nsa,msa,lsa,nra,mra,lra,nbn,mbn,lbn,nbp,mbp,lbp,nbz,mbz,lbz,nbd,mbd,lbdnq,mq,lq,nql,mql,lql, nxkm,mxkm,lxkm,nykm,mykm,lykm,nxkh,mxkh,lxkh,nykh,mykh,lykh,nzkm,mzkm,lzkm,nzkh, mzkh,lzkh,ncbfx,mcbfx,ncbfy,mcbfy nsr,msr,

Subroutine	Description
	nbr,mbr,next,mext,lext,npa,mpa,ntx,mtx,nty,mty,nqr,mqr,nq0,mq0,nep,mepntlx,mtlx,ltlx,nslx,mslx,lslx,nwlx,mwlx,lwlx Real: x, y, dx,dy,z, h, am, e, ue, ve, sorb, sor,u, v, w, phi, p, t, s, r, ta,sa, ra,bn, bp, bz, bd, q, ql, xkm, ykm, xkh, ykh, zkm, zkh, cbfx, cbfy, sr, br,ext,pa,tx, ty, qr, q0, e, tlx,slx,wlx
Prnte	Subroutine PRNTE prints out all or part of a 2D array with a real or integer format. It is similar to PRNTD but land areas can be masked out. **Calling Sequence:** prnte (fld, n, n1, n2, m1, m2, ncolum, length, ndec, title, amult, ad, iflip) **Data Declaration:** Character title Integer n, n1, n2, m1, m2, ncolum, length, ndec, iflip Real fld, amult, ad
Prnte	Subroutine PRNTE prints out all or part of a 2D array with a real or integer format. It is similar to PRNTD but land areas can be masked out. The variable max is the maximum number of characters that are allowed to be printed across the page. If ncolumn and length are such that lmax is exceeded, the number of columns printed across the page is reduced to the point where lmax is not exceeded. **Calling Sequence:** prnte (fld, n, n1, n2, m1, m2, ncolum, length, ndec, title, amult, ad, iflip) **Data Declaration:** Integer n, n1, n2, m1, m2, ncolum, length, ndec Real fld, amult, ad, iflip Character title
Prntf	Subroutine PRNTF prints out all or part of a 2D array with real or integer format. This is similar to PRNTD but land areas can be masked out in PRNTF. Subroutine PRNTF is set up for arrays that may have halos. **Calling Sequence:** prntf (fld, n, m, n1, n2, m1, m2,ncolum, length, ndec, title, amult, ad,iflip, wsp) **Data Declaration:** Integer n, m, n1, n2, m1, m2, ncolum, length, ndec, iflip Real fld, amult, ad, wsp Character title **Common Blocks:** PRNTFI4 PRNTFR4
Prntf2	Subroutine PRNTF2 prints out all or part of a 2D array with a real or integer format. This is similar to PRNTD but land areas can be masked out in PRNTF2. It is also similar to PRNTF, but the input arrays do not have halos. **Calling Sequence:** prntf2 (fld, n, n1, n2, m1, m2, ncolum, length, ndec, title, amult, ad, iflip) **Data Declaration:** Integer n, n1, n2, m1, m2, ncolum, length, ndec, iflip Real fld, amult, ad Character title

Subroutine	Description
	Common Blocks: PRNTFI4 PRNTFR4
Prntspv	Subroutine PRNTSPV sets special values to mask regions of the table. **Calling Sequence:** prntspv (indspvd, spvalud) **Data Declaration:** Integer indspvd Real spvalud
Prntsv2	Subroutine PRNTSV2 sets special values to mask regions of the table. **Calling Sequence:** prntsv2 (indspvd, spvalud) **Data Declaration:** Integer indspvd Real spvalud **Common Blocks:** PRNTFI4 PRNTFR4
Roll3	Subroutine ROLL3 rolls (switches) three index values. **Calling Sequence:** roll3 (i1, i2, i3) **Data Declaration:** Integer i1, i2, i3
Roll4	Subroutine ROLL4 rolls (switches) four index values. **Calling Sequence:** roll4 (i1, i2, i3, i4) **Data Declaration:** Integer i1, i2, i3, i4
Stat3d	Subroutine STAT3D prints out statistics on fld. **Calling Sequence:** stat3d (fld, n, m, l, na, ma, title) **Data Declaration:** Integer n, m, l, na, ma Real fld Character title **Common Block:** PRNTFR4
Switch	Subroutine SWITCH switches the value of two integers. **Calling Sequence:** switch (i1, i2) **Data Declaration:** Integer i1, i2
Tridd	Subroutine TRIDD solves a tri-diagonal system of linear equations. TRIDD differs from TRID in that TRIDD is double precision. **Calling Sequence:** tridd (n, a, b, c, g) **Data Declaration:** Integer n Real a, b, c, g
Wscurl	Subroutine WSCURL calculates wind stress curl. The grid is assumed to be a regularly spaced spherical or Cartesian grid. The strange wind stress curl units that are output were designed to get wind stress curl values of approximately order one. **Calling Sequence:** wscurl (n, m, tx, ntx, ty, nty, elon, nelon, alat, nalat, wsc) **Data Declaration:** Integer n, m, ntx, nty, nelon, nalat Real tx, ty, elon, alat, wsc
Wtcyc	Subroutine WTCYC determines if a value t lies between t2 and t1, i.e., t2 < t <= t1 for values of t that are periodic with period pd. If t does lie between t2 and t1, a weighting for point t1 is returned that can be used for linear interpolation of function values at points t2 and t1 to point t. The method used here depends on the fact that t2 < t1 unless t2 and t1 span the end of the periodic domain, i.e., the values of t must be

Subroutine	Description
	monotonically increasing over the range of the domain of t. **Calling Sequence:** wtcyc (t, t2, t1, pd, between, wt1) **Data Declaration:** Real t, t2, t1, pd, wt1 Logical between
Ymd2idt	Subroutine YMD2IDT converts the year, month, and day to an integer of the form YYYYMMDD. **Calling Sequence:** ymd2idt (iyear, month, iday, idate) **Data Declaration:** Integer iyear, iday, idate, month

5.4.10 Read/Write Subroutines (ncom1rwio)

Subroutine	Description
Cr_fname2	Subroutine CR_FNAME2 creates COAMPS-style, 64-character filenames. It is adapted from Jim Cummings' subroutine CR_FNAME. **Calling Sequence:** cr_fname2(out_dir,idbms,file_dtg,nest,m,n,file_type,fld_name, fluid,lev1,lev2,lvl_type,itau_hr,itau_mn,itau_sc, file_name,len) **Data Declaration:** Integer idbms,nest,m,n,lev1,lev2,itau_hr, itau_mn,itau_sc,len Character out_dir,file_dtg,file_type,fld_name,fluid, file_name,lvl_type
Ncom_init_io	Subroutine NCOM_INIT_IO initializes *io_unit* offset and *istdo_unit* for NCOM. This should be called immediately after *mpi_init*. It has no arguments.
Rd_out3d	Subroutine RD_OUT3D reads surface elevation, 3D velocity, temperature or salinity, one field at a time. **Calling Sequence:** rd_out3d(ind,indv,nest,nt,mt,n,m,l,field3d,timed) **Data Declaration:** Integer ind,indv,nest,nt,mt,n,m,l,nl Real timed,field3d
Rw_bsfx	Subroutine RW_BSFX reads and writes surface flux fields including air temperature and water vapor mixing ratio to allow internal calculation of latent and sensible heat flux following the Kara (2000) formulation. Fields may be climatological or real time. **Calling Sequence:** rw_bsfx(ind,indatp,indtau,indsft,indsfs,indsol,nest,nt,mt,n, m,nr,patm2,usflx2,vsflx2,rsflx2,solar2,tair2,vapmx2,idate, itime,indclim,close, wxy) **Data Declaration:** Integer ind,indatp,indtau,indsft,indsfs,indsol,nest,nt,mt, n,m,nr,idate,itime,indclim Logical close Real patm,usflx2,vsflx2,rsflx2,solar2,tair2,vapmx2, wxy
Rw_extd	Subroutine RW_EXTD reads and writes solar extinction data. Fields can be climatological or real time. **Calling Sequence:** rw_extd(ind,indextd,nest,nt,mt,n,m,extd,idate,itime,indclim, close) **Data Declaration:** Integer ind, indextd,nest, nt, mt, n, m, idate,itime,

Subroutine	Description
	indclim Logical close Real extd
Rw_fld1	Subroutine RW_FLD1 reads or writes a file for a single 2D or 3D field. **Calling Sequence:** rw_fld1 (ind, nest, nt, mt, n, m, mon, name, t) **Data Declaration:** Integer ind, nest, nt, mt, n, m, mon Character name Real t
Rw_fld1f	Subroutine RW_FLD1F reads or writes a file for a single 2D or 3D field. **Calling Sequence:** rw_fld1f (ind, nest, nt, mt, n, m, mon, cfile, t) **Data Declaration:** Integer ind, nest, nt, mt, n, m, mon Character cfile Real t
Rw_fld2	Subroutine RW_FLD2 reads or writes a file for a pair of fields. **Calling Sequence:** rw_fld2 (ind, nest, nt, mt, n, m, mon, name, t, s) **Data Declaration:** Integer ind, nest, nt, mt, n, m, mon Character name Real t, s
Rw_ic1	Subroutine RW_IC1 reads or writes a file for initial fields. **Calling Sequence:** rw_ic1 (ind, nest, nt, mt, n, m, l, ls, e, u, v, t, s) **Data Declaration:** Integer ind, nest, nt, mt, n, m, l, ls Real e, u, v, t, s
Rw_ic2	Subroutine RW_IC2 reads or writes a file for initial fields. RW_IC2 differs from RW_IC1 in that scalar fields (e.g., temperature and salinity) are handled in a single array r. When only two scalar fields are being used (T and S), the two subroutines should read and write the same file. **Calling Sequence:** rw_ic2 (ind, nest, nt, mt, n, m, l, ls, nrt, nr, j1, e, u, v, r) **Data Declaration:** Integer ind, nest, nt, mt, n, m, l, ls, nrt, nr, j1 Real e, u, v, r
Rw_obc	Subroutine RW_OBC reads and writes open boundary condition data. Data may be climatological or real-time. Data is read for the entire grid, but retained only for a local tile. Indices (*neobx*, *nvobx*) are set to denote the range of the boundary data that lie within the local tile. **Calling Sequence:** rw_obc (ind, nest, nt, mt, n, m, l, nr, iec, idate, itime, nobmax, nob, neob, nuob, nvob, iob, job, ivob, jvob, eob, ubob, vbob, uob, vob, rob, indclim, close) **Data Declaration:** Integer ind, nest, nt, mt, n, m, l, nr, iec, idate, itime, nobmax, nob, neob, nuob, nvob, iob, job, ivob, jvob, indclim Logical close Real eob, ubob, vbob, uob, vob, rob
Rw_out3h	Subroutine RW_OUT3H reads or writes surface elevation, depth-averaged transports (depth-averaged velocity x depth), 3D velocity, temperature, and salinity fields, and

Subroutine	Description
	surface atmospheric forcing fields. **Calling Sequence:** rw_out3d1 (ind, inde, indv, indt, inds, nest, nt, mt, n, m, l, e, u, v, t, s,timed, idate, itime) **Data Declaration:** Integer ind, inde, indv, indt, inds, nest, nt, mt, n, m, l, idate, itime Real e, u, v, t, s, timed
Rw_outpts	Subroutine RW_OUTPTS reads or writes global (i,j) indices for model grid points at which data are to be saved. **Calling Sequence:** rw_outpts(ind,nest,nt,mt,n,m,nsav,isav,jsav) **Data Declaration:** Integer ind, nest, nt, mt, n, m, nsav,isav,jsav
Rw_riv	Subroutine RW_RIV reads and writes river inflow data. River data may be climatological or real-time. When reading river data, data is only retained for the local tile. **Calling Sequence:** rw_riv (ind, nest, nt, mt, n, m, l, nr, nrvmax, nriv, nrriv, lriv, indrivr, idate,itime, iriv, jriv, wtriv, qriv, rriv, indclim, close) **Data Declaration:** Integer ind, nest, nt, mt, n, m, l, nr, nrvmax, nriv, nrriv, lriv,indrivr, idate, itime, iriv, jriv, indclim Logical close Real wtriv, qriv, rriv
Rw_rmean	Subroutine RW_RMEAN reads or writes files for horizontal mean values of scalar fields and density anomaly at the sigma grid points. **Calling Sequence:** rw_rmean (ind, nest, z7, r7, lmax, l7, nr) **Data Declaration:** Integer ind, nest, lmax, l7, nr Real z7, r7
Rw_rmean2	Subroutine RW_RMEAN2 reads or writes files for mean/climate/background fields for scalar variables. **Calling Sequence:** rw_rmean2 (ind, nest, nt,mt,n,m,lm1,nr,rmean) **Data Declaration:** Integer indrw,nest,nt,mt,n,m,lls,,nr,nq,i1,i2,i3,j1, j2,iter, indzk Real e,udb,vdb,u,v,r,q,zkm,zkh,wubot,wvbot,rmean
Rw_rstrt3	Subroutine RW_RSTRT3 reads or writes restart files. **Calling Sequence:** rw_rstrt3(indrw,nest,nt,mt,n,m,l,ls,nr,nq,i1,i2,i3,j1,j2,iter, times,indzk,e,udb,vdb,u,v,r,q,rmean,zkm,zkh, wubot,wvbot) **Data Declaration:** Integer ind, nest, nt,mt,n,m,lm1,nr Real rmean
Rw_sfx	Subroutine RW_SFX reads and writes surface flux fields. Surface flux fields may be climatological or real time. **Calling Sequence:** rw_sfx (ind, indatp, indtau, indsft, indsfs, indsol, nest, nt, mt, n, m, nr, patm2, usflx2, vsflx2, rsflx2, solar2, idate, itime, indclim, close, wxy) **Data Declaration:** Integer ind, indatp, indtau, indsft, indsfs, indsol, nest, nt, mt, n, m,nr, idate, itime, indclim Logical close

Subroutine	Description
	Real patm2, usflx2, vsflx2, rsflx2, solar2, wxy
Rw_sss	Subroutine RW_SSS reads and writes prescribed surface salinity fields. Fields may be climatological or real time. **Calling Sequence:** rw_sss (ind,indsss, nest, nt, mt, n, m, sss2, idate, itime, indclim, close) **Data Declaration:** Integer ind,indsss, nest, nt, mt, n, m, idate, itime, indclim Logical close Real sss2
Rw_sst	Subroutine RW_SST reads and writes prescribed surface temperature and salinity fields. Fields may be climatological or real time. **Calling Sequence:** rw_sst (ind, indsst, indsss, nest, nt, mt, n, m, sst2, sss2, idate, itime,indclim, close) **Data Declaration:** Integer ind, indsst, indsss, nest, nt, mt, n, m, idate, itime, indclim Logical close Real sst2, sss2
Rw_stop	Subroutine RW_STOP reads or writes the stop file. **Calling Sequence:** rw_stop (ind, istop) **Data Declaration:** Integer ind, istop
Rw_tide2	Subroutine RW_TIDE2 reads and writes open boundary tidal data. **Calling Sequence:** rw_tide2 (ind, nest, nt, mt, n, m, ntc, iec, tidecn, nobmax, nob, neob, nuob,nvob, iob, job, ivob, jvob, etab, etpb, utab, utpb, vtab, vtpb) **Data Declaration:** Integer ind, nest, nt, mt, n, m, ntc, iec, nobmax, nob, neob, nuob, nvob, iob, job, ivob, jvob Real tidecn, etab, etpb, utab, utpb, vtab, vtpb
Rw_tpcn	Subroutine RW_TPCN reads and writes names of tidal constituents used for tidal potential forcing. **Calling Sequence:** rw_tpcn (ind, nest,nc,tidecn) **Data Declaration:** Integer ind, nest, nc Character tidecn
Rw_trsec	Subroutine RW_TRSEC reads/writes locations of transport sections to be computed. **Calling Sequence:** rw_trsec(ind,nest,nt,mt,n,m,nstmax,nst,is1,js1,is2,js2,idir, section) **Data Declaration:** Integer ind, nest, nt,mt,n,m,nstmax,nst,is1,js1,is2,js2,idir Character section
Rw_ts	Subroutine RW_TS reads and writes prescribed 3D temperature and salinity fields. Fields may be climatological or real time. **Calling Sequence:** rw_ts (ind, indt, inds, nest, nt, mt, n, m, l, t2, s2, idate, itime, indclim,close) **Data Declaration:** Integer ind, indt, inds, nest, nt, mt, n, m, l, idate, itime, indclim

Subroutine	Description
	Logical close Real t2, s2
Rw_wmdef	Subroutine RW_WMDEF reads water mass definitions. **Calling Sequence:** rw_wmdef(ind,nest,nwmmax,nwm,namewm,twm,swm,dwm) **Data Declaration:** Integer ind, nest, nwmmax,nwm Real twm,swm,dwm Character namewm
Rw_zout	Subroutine RW_ZOUT reads or writes files for depths for output fields. **Calling Sequence:** rw_zout(ind,nest,lzoutmx,lzout,zout) **Data Declaration:** Integer ind, nest, lzoutmx,lzout Real zout
Rwdimen	Subroutine RWDIMEN reads or writes model dimensions to file for all grids. **Calling Sequence:** rwdimen (ind, nto, mto, lo, lso, lzo, nro, nqo, ntypo, ntco, nobmaxo, nrivo) **Data Declaration:** Integer ind, nto, mto, lo, lso, lzo, nro, nqo, ntypo, ntco, nobmaxo, nrivo
Rwhgrid	Subroutine RWHGRID reads or writes files for a horizontal grid. **Calling Sequence:** rwhgrid (ind, nest, nt, mt, n, m, ibo, elon, alat, dx, dy, h, ang) **Data Declaration:** Integer ind, nest, nt, mt, n, m, ibo Real elon, alat, dx, dy, h, ang
Rwspmd	Subroutine RWSPMD reads SPMD processor layout. **Calling Sequence:** rwspmd (iprsum, jprsum) **Data Declaration:** Integer iprsum, jprsum
Rwvgrid	Subroutine RWVGRID reads or writes a file for vertical grid. **Calling Sequence:** rwvgrid (ind, nest, l, ls, zw) **Data Declaration:** Integer ind, nest, l, ls Real zw
Timetag	Subroutine TIMETAG generates date-time tags used for input/output files. **Calling Sequence:** timetag(ind,nest,cdate) **Data Declaration:** Integer ind, nest Character cdate
Wffmp	Subroutine WFFMP reads or writes fields to output files. This is similar to Julie Pullen's WTFF, but scaleable. **Calling Sequence:** wffmp(ind,itimes,nt,mt,n,m,mon,t,out_dir,idbms,file_dtg,nest, file_type, fld_name,fluid, lev1, lev2, lvltyp) **Data Declaration:** Integer ind, nest,itimes,nt,mt,n,m,mon,idbms,lev1,lev2 Character out_dir,file_dtg,file_type,fld_name,fluid,lvltyp Real t

5.4.11 Surface Forcing Subroutines (ncom1sbc)

Subroutine	Description
Atmflux	Subroutine ATMFLUX calculates dummy atmospheric fluxes to check the selection of surface fluxes in OSURFBC. Ocean model input parameters provide for surface

Subroutine	Description
	fluxes to be obtained from the coupled atmospheric model, from an input data file, or to be set to zero. **Calling Sequence:** atmflux (nest, nt, mt, n, m, nr, is, ie, js, je, iat1, iat2, times, elon, alat, ang, amsk, patm2, usflx2, vsflx2, rsflx2, solar2, tmatm2, wxy) **Data Declaration:** Integer nest, nt, mt, n, m, nr, is, ie, js, je, iat1, iat2 Real times, elon, alat, ang, amsk, patm2, usflx2, vsflx2, rsflx2,solar2, tmatm2, wxy **Comments:** All surface fluxes (usflx, vsflx, rsflx, solar) are defined as (+) downward. This means that a (+) value of usflx or vsflx indicates a stress acting to drive the surface current in the x or y direction, and a (+) value of the solar or surface heat flux will act to warm the surface layer of the ocean. This is the reverse of the convention used by POM, where the surface fluxes are defined to be (+) upward. The surface atmospheric pressure (patm) is expressed in terms of meters of water. Only the horizontal gradient of patm drives the ocean, the mean value of patm does not affect the ocean. For a surface pressure (pa) given in mb, it is suggested that patm be calculated as (Note: 1 mb = 100 newtons/m^2 = 100 kg - m/s^2 - m^2): patm = (pa-1000)*100/(g*rho0) where rho0 is the seawater density. From this it is evident that a 10 mb air-pressure differential is equivalent to a sea surface elevation differential of approximately one cm.
Bulk_lsb	Subroutine BULK_LSB calculates the latent and sensible heat flux using the bulk formulas of Kara et al. (2000), the SST from the ocean model, and the input surface atmospheric air temperature and mixing ratio. The existence of ice is checked and if it does exist, then heat flux from the Polar Ice Prediction System (PIPS; Posey et al., 2008) is employed. **Calling Sequence:** bulk_lsb(nt,mt,n,m,nr,is,ie,js,je,ifx1,ifx2,w1fx,times,ramp, amsk,t,s,patm2,wspd2,tair2,humd2,usflx,vsflx,rsflx,solar,evap) **Data Declaration:** Integer nt, mt, n, m, nr, is, js, je, ifx1,ifx2 Real w1fx,times,ramp,amsk,t,s,patm2,wspd2,tair2, humd2, usflx,vsflx,rsflx,solar
Get_bsfx	Subroutine GET_BSFX grabs surface flux fields from the input file. It loads atm pressure, wind stress, scalar fluxes, solar (shortwave) heat flux, air temperature and water vapor mixing ratio. It is set up for data on a single input file. **Calling Sequence:** get_bsfx(indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr, ifx1,ifx2,idate,itime,timed,climatp,w1fx,patm2,usflx2,vsflx2, rsflx2,solar2,tair2,vapmx2,tmsfx2, wxy) **Data Declaration:** Integer indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr,ifx1,ifx2,idate,itime Real timed,climatp,w1fx,patm2,usflx2,vsflx2,rsflx2, solar2,tair2,vapmx2,tmsfx2,wxy
Get_sfx	Subroutine GET_SFX grabs surface flux fields from the input file. It is set up for data on a single input file. **Calling Sequence:** get_sfx(indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr,

Subroutine	Description
	ifx1,ifx2,idate,itime,timed,climatp,w1fx,patm2,usflx2,vsflx2, rsflx2,solar2, tmsfx2, wxy) **Data Declaration:** Integer indatp,indtau,indsft,indsfs,indsol,nt,mt,n,m,nr,ifx1,ifx2,idate,itime Real timed,climatp,w1fx,patm2,usflx2,vsflx2,rsflx2, solar2,tmsfx2,wxy
Get_sss	Subroutine GET_SSS gets surface salinity fields (only SSS, not SST) from the input file. **Calling Sequence:** get_sst(indsss,nt,mt,n,m,iss1,iss2,idate,itime,timed,climatp, w1, sss2,tmsst2) **Data Declaration:** Integer indsss,nt,mt,n,m,iss1,iss2, idate, itime Real timed,climatp,w1,sss2,tmsst2
Get_sst	Subroutine GET_SST gets surface temperature and/or salinity fields from the input file. **Calling Sequence:** get_sst(indsst,indsss,nt,mt,n,m,ist1,ist2,idate,itime,timed, climatp, w1, sst2,sss2,tmsst2, wxy) **Data Declaration:** Integer indsst,indsss,nt,mt,n,m,ist1,ist2, idate, itime Real timed,climatp,w1,sst2,sss2,tmsst2,wxy
Osurfbc	Subroutine OSURFBC defines model surface forcing fields. **Calling Sequence:** osurfbc (nt, mt, iec, n, m, l, nr, is, ie, js, je, ifx1, ifx2, iat1, iat2, iss1, iss2,times, elon, alat, ang, amsk, t, s, patm2, usflx2, vsflx2, rsflx2, solar2,tmsfx2, tmatm2, sst2, sss2, tmsst2, patm, usflx, vsflx, rsflx, solar, surruf,wxy) **Data Declaration:** Integer nt, mt, iec, n, m, l, nr, is, ie, js, je, ifx1, ifx2, iat1, iat2, iss1,iss2 Real times, elon, alat, ang, amsk, t, s, patm2, usflx2, vsflx2,rsflx2, solar2, tmsfx2, tmatm2, sst2, sss2, tmsst2, patm, usflx, vsflx, rsflx, solar, surruf, wxy
Wtset	Subroutine WTSET sets appropriate weighting for temporal interpolation of surface forcing fields. **Calling Sequence:** wtset(ramp,ind, ifx1,iat1,ico1, w1fx,w1at,w1co, i1,i2,w1,w2) **Data Declaration:** Integer ind,ifx1,iat1,ico1,i1,i2 Real ramp,w1fx,w1at,w1co,w1,w2

5.4.12 Tidal Calculation Subroutines (ncom1tide)

Subroutine	Description
Astr	Subroutine ASTR calculates the following five ephermides of the sun and moon: h, pp, s, p, np. Units are cycles for the ephermides and cycles/365 days for their derivatives. **Calling Sequence:** astr (d1, h, pp, s, p, np, dh, dpp, ds, dp, dnp) **Data Declaration:** Integer np Real d1, h, pp, s, p, dh, dpp, ds, dp, dnp

Subroutine	Description
Gday	Given day, month, (each two digits) and year (four digits), subroutine GDAY returns the day number kd based on the Gregorian calendar. GDAY is valid only for Gregorian calendar dates. **Calling Sequence:** gday (idd, imm, iyear, kd) **Data Declaration:** Integer idd, imm, iyear, kd
Opnvuf	Subroutine OPNVUF reads in KONTAB and calls SETVUF. **Calling Sequence:** opnvuf (kh, konk, xlat, fk, vuk, freqk) **Data Declaration:** Integer kh Real xlat, fk, vuk, freqk Character konk **Common Blocks:** VUFC5 VUFI4 VUFR4
Setvuf	Subroutine SETVUF evaluates f and vu for all constituents in KONTAB. **Calling Sequence:** setvuf (kh, konk, xlat, fk, vuk, freqk) **Data Declaration:** Integer kh Real xlat, fk, vuk, freqk Character konk **Common Blocks:** VUFC5 VUFI4 VUFR4 **Comments:** Ntidal is the number of main constituents. Ntotal is the number of constituents (main + shallow water) for the given time kh, the table of f and v+u values is calculated for all the constituents. F is the nodal modulation adjustment factor for amplitude. U is the nodal modulation adjustment factor for phase. V is the astronomical argument adjustment for phase. The astronomical arguments are calculated by linear approximation at the midpoint of the analysis period. Only the fractional part of a solar day needs to be retained for computing the lunar time tau.
Tc_amp	Subroutine TC_AMP returns amplitude for equilibrium tide for a specified tidal constituent. The returned equilibrium tidal amplitudes need to be corrected for the "earth tide" (by multiplying by factor of ~ 0.69) and, if simulating a particular time period, corrected for that particular time period by multiplying by a "node factor" (this is generally a small < 10% correction). These corrections are NOT done here. **Calling Sequence:** tc_amp(tidecn,amp) **Data Declaration:** Real amp Character tidecn
Tidepot	Subroutine TIDEPOT calculates tidal potential (*ep*). It is currently set up only for the M2 tide. **Calling Sequence:** tidepot (times, ramp, n, m, is, ie, j, amsk, elon, alat, ep) **Data Declaration:** Integer n, m, is, ie, j Real times, ramp, amsk, elon, alat, ep
Tide_dat	Subroutine TIDE_DAT gets tidal forcing data for open boundaries. **Calling Sequence:** tide_dat(nt,mt,n,m,iec,ntc,hu,hv,idate,itime,alatave,tidecn,

Subroutine	Description
	tidefq,nobmax,nob,neob,nuob,nvob,iob,job,ivob,jvob,etab,etp, utab,utpb,vtab,vtpb) **Data Declaration:** Integer nt,mt,n,m,iec,tnc,idate,itime,nobmax,nob, neob,nuob,nvob Real hu,hv,alatave,tidefq,etab,utab,etpb,utpb, vtab,vtpb Character tidecn
Tide_fac	Subroutine TIDE_FAC calculates tidal data needed for predicting the tides for a particular time period. A particular tidal constituent can be calculated as: tide = fx2*amp*cos(freqx2*(t - t0) - phase + vud2), where amp and phase are the equilibrium amplitude and phase for the tidal constituent at a particular location, t is the time, and t0 is the time at which the tidal data was calculated (i.e., the input date to *tide_fac*). **Calling Sequence:** tide_fac (ihh, idd, imm, iyear, xlat, ntides, iprint, kon2, freqx2, fx2, vud2) **Data Declaration:** Integer ihh, idd, imm, iyear, ntides, iprint Real xlat, freqx2, fx2, vud2 Character kon2
Vuf	Subroutine VUF finds appropriate f, vu and sig for a specified constituent. **Calling Sequence:** vuf (kh, konk, xlat, fk, vuk, freqk) **Data Declaration:** Integer kh Real xlat, fk, vuk, freqk Character konk **Common Blocks:** VUFC5 VUFI4 VUFR4

5.4.13 Update Subroutines for U, V, T, S (ncom1updt_sigz)

Subroutine	Description
Advq	Subroutine ADVQ calculates advection and horizontal diffusion terms for turbulence fields. Note on slabbing and tiling: advq is called for j = je+1, js, -1. The call for j = je+1 is only to calculate the y-flux at j = je+1. **Calling Sequence:** advq (j, jf, jb, ua, va, wa, flyq, qold, n, m, l, ls, nq, i1, i3, j1, j2, is, ie, isp,iep, js, je, dti2, small, dar, dsw, dsm5, dzm5, dzwr, sor, d1, q, xk, yk, dtdazr, flx, flz) **Data Declaration:** Integer j, jf, jb, n, m, l, ls, nq, i1, i3, j1, j2, is, ie, isp, iep, js, je Real ua, va, wa, flvq, qold, dti2, small, dar, dsw, dsm5, dzm5,dswr, sor, d1, q, xk, yk, dtdazr, flx, flz
Advr	Subroutine ADVR calculates explicit forcing terms for scalar fields. Note on slabbing and tiling: advr is called for j = je+1, js, -1. The call for j = je+1 is only to

Subroutine	Description
	calculate the y-flux at j = je+1. **Calling Sequence:** advr (j, jf, jb, ua, va, wa, flyr, rjp1, n, m, l, ls, nr, i1, i3, j1, j2, is, ie, isp,iep, js, je, iec, ke, indriv, indrivr, indbio, sigdif, locate, idate, itime, iter, ramp, times, dti2, asf, ext, small, da, dar, sw, sm, dsm, zw, zm, dzmr, amsk, sor, sorb, d1, r, rmean, xk, yk, rsflx, solar, nrvmax, lriv, iriv, jriv, isriv, ieriv, irv1, irv2, rriv, w1riv, rsor, dtdazr, flx, flz, dr) **Data Declaration:** Integer j, jf, jb, n, m, l, ls, nr, i1, i3, j1, j2, is, ie, isp, iep, js, je, iec,ke, indriv, indrivr, indbio, idate, itime, iter, nrvmax, lriv,iriv, jriv, isriv, ieriv, irv1, irv2 Real ua, va, wa, flyr, rjp1, ramp, times, dti2, asf, ext, small, da,dar, sw, sm, dsm, zw, zm, dzmr, amsk, sor, sorb, d1, r, rmean, xk, yk, rsflx, solar, rriv, w1riv, rsor, dtdazr, flx, flz, dr Logical sigdif, locate **Comments:** When masking advective and diffusive fluxes and variables, the advective transports need not be masked since the velocities from which these transports are calculated will be zero on land-sea boundaries. This is true for all variables, i.e., the averaged transports calculated at offset grid cells (u, v, and q grid cells) should be correct without any need for masking. Diffusive fluxes, however, must be masked at land-sea boundaries for scalar fields. This can be done by masking the horizontal (xk, yk) and vertical eddy coefficient variables. For offset scalar grid cells, averages of xk and yk that have been masked for the t-grid cells should result in fluxes at land-sea boundaries for offset grid cells that are correct. For momentum, xk and yk should NOT be masked for no-slip boundary conditions but must be masked for free-slip boundary conditions. Vertical eddy coefficients need to be masked for the particular variable when the calculation of vertical diffusion is being made. Calculated velocities and the forcing terms for the barotropic mode (fu, fv) must be masked, since forcing terms at land-sea boundaries may be non-zero. Scalar fields do not need to be masked since transport fluxes are masked. However, be sure surface fluxes, the solar extinction array (ext), and vertical eddy coefficients are masked. Turbulence variables (q) for MYL2.5 model do not need to be masked, since transports will be zero on land-sea boundaries and other forcing terms are either proportional to q or are multiplied by vertical eddy coefficients, which will be zero on land-sea boundaries. With fourth-order advection and diffusion, values of r at row j must be saved (in rjp1) at time j1 and j2 for calculating fourth-order terms for y flux in the next pass. For second-order advection set a = 0 and for fourth-order set a = 2/12. For third-order upwind advection set a = b = 2/12, c = 4/12. When advection is second-order, mixing is Laplacian with a grid-cell Reynolds of six. For Laplacian mixing, make xk = A2*dy*dz/dx and b = 0.0. For biharmonic mixing, make xk = 2*(A4/dx**2)*dy*dz/dx and set b = 0.5. Set values of rjp1 on first pass when j = je + 1. These need to be set when the N boundary is an interior or periodic boundary (i.e.,

Subroutine	Description
	j = je+1 = m+1) and fourth-order differences are used. For an exterior N boundary, they must be set to something for computers like the T3E where a local scratch array may be initialized to be undefined. An alternative for the latter case is to initialize the wxz scratch arrays to zero.
Advuv	Subroutine ADVUV calculates explicit forcing terms for 3D momentum. **Calling Sequence:** advuv (j, jf, jb, jc1, jc2, jc3, jc4, ua, va, wa, pgx, pgy, es, et, fc, fcu, flyu,flyv, fu, fv, na, ma, n, m, l, ls, i1, i2, i3, is, ie, ism, iem, isp, iep, js, je, iec, ibo, indbaro, indatp, curved, tidpot, iterm, ramp, times, dti, dti2, eg1, eg2, eg3, g, fda, small, elon, alat, dx, dy, ddx, ddy, dau, dav, dxur, dyvr, daur, davr, dsm, dzm, dzmr, amsk, umsk, vmsk, sor, du, dv, d1, d1u, d1v, e, u, v, xk, yk, patm, usflx, vsflx, flx, flz, wpf) **Data Declaration:** Integer j, jf, jb, jc1, jc2, jc3, jc4, na, ma, n, m, l, ls, i1, i2, i3, is, ie,ism, iem, isp, iep, js, je, iec, ibo, indbaro, indatp, iterm Real ua, va, wa, pgx, pgy, es, et, fc, fcu, flyu, flyv, fu, fv, ramp,times, dti, dti2, eg1, eg2, eg3, g, fda, small, elon, alat, dx, dy, ddx, ddy, dau, dav, dxur, dyvr, daur, davr, dsm, dzm, dzmr, amsk, umsk, vmsk, sor, du, dv, d1, d1u, d1v, e, u, v, xk, yk, patm, usflx, vsflx, flx, flz, wpf Logical curved, tidpot **Comments:** Note on calculating in x-z slabs ("slabbing") and decomposition into tiles ("tiling"): The procedure to convert from the original 3D loop structure to "slabbing" is to replace j-loops with “if” statements that span the same range (when such an “if” statement is necessary). However, if a flip-flop array (an array in which two adjacent j values are stored) is being evaluated at j-1, the range of j over which the array is calculated must be increased by one at both ends. All of the flip-flop arrays are evaluated at j-1 except *flyu* (*flyu* for u(j) is needed at j and j+1, not at j and j-1). When calculating a flip-flop array at j-1, all the j-indices in the original 3D loops must be replaced by j-1. The procedure used here to convert from a code for a single domain to a parallelizable code that allows for the calculation of subdomain "tiles" with either exterior or interior edges is (1) to put "halos" around all horizontally dimensioned arrays to provide for setting boundary conditions for interior tile edges, and (2) to define the region of calculation of scalar fields on each tile to be in the range "is" to "ie" and "js" to "je". These indices can provide the proper range of calculation of scalar fields for either interior or exterior tile edges. The *is* and *ie* indices can also be used to shrink-wrap calculations in the x direction (though this may not be of much benefit for parallelization with uniformly sized tiles, since the tile with the most calculation will determine the model's execution time). With is, ie, js, je defined as the range of calculation of scalar fields, which are horizontally located at grid cell centers, some additional specification is needed for the range of calculation of velocity fields, which are at staggered grid locations. This is accomplished by adjusting individual calculation loops using an "edge correction"

Subroutine	Description
	variable iec(8). The first four dimensions of iec correspond to the four edges of a tile (W = 1, E = 2, S = 3, N = 4), and each edge is specified with a "0" or a "1" depending on whether it is an interior or exterior edge. Hence, with iec = 0, all the model loops are dimensioned for an interior tile.
Advvel	Subroutine ADVVEL calculates advective transport. This is the advective velocity multiplied by 0.5*(area of the corresponding grid cell face). All the advective transports (ua, va, wa) are multiplied by 0.5 to anticipate the averaging that occurs in calculating the advection terms. **Calling Sequence:** advvel (ind, j, jf, jb, ua, va, wa, uacr, vacr, na, ma, n, m, l, ls, i1, i2, i3, is,ie, isp, iep, js, je, iec, indadv, indiag, shrnkwp, locate, dti2, vg1, vg2, vg3, small, dxv, dyu, da, dar, dsm, dsm5, dzm5, umsk, vmsk, sor, e, du, dv, d1u, d1v, udb, vdb, u, v, w, wpf) **Data Declaration:** Integer ind, j, jf, jb, na, ma, n, m, l, ls, i1, i2, i3, is, ie, isp, iep, js,je, iec, indadv, indiag Real ua, va, wa, uacr, vacr, dti2, vg1, vg2, vg3, small, dxv, dyu,da, dar, dsm, dsm5, dzm5, umsk, vmsk, sor, e, du, dv, d1u, d1v, udb, vdb, u, v, w, wpf Logical shrnkwp, locate
Biology	Subroutine BIOLOGY is for a biological model. BIOLOGY calculates change in biological constituents due to biological interactions within each grid cell. **Calling Sequence:** biology (dr, n, m, l, ls, nr, is, ie, j, ke, i3, j1, dti2, sw, sm, zw, zm, amsk,d1, r) **Data Declaration:** Integer n, m, l, ls, nr, is, ie, j, ke, i3, j1 Real dt, dti2, sw, sm, zw, zm, amsk, d1, r **Comments:** To implement BIOLOGY in the ocean model, set dimension nr to the number of biological constituents +2 (i.e. nr = 6). Initialize biological constituent arrays in subroutine INITIAL. Be sure subroutine SURFBC provides surface fluxes for biological constituents (these are usually just set to zero). Check that subroutine OPENBC provides open BC for biological constituents (the default is usually an Orlanski radiation condition for outflow, with inflow based on the initial values at open boundary locations). Check the treatment of available light in this routine to be sure it is adequate. Modify the OUTPUT subroutine to provide the desired output of biological fields.
Cor_curv	Subroutine COR_CURV calculates combined Coriolis and curvature-correction parameter (fc). COR_CURV also calculates fcu = fc*(u interpolated to t-point). **Calling Sequence:** cor_curv (j, jf, jb, fc, fcu, n, m, l, ls, i1, i2, i3, is, ie, js, je, iec, ibo, curved,fda, ddx, ddy, dsm, dzm, amsk, umsk, d1, u, v) **Data Declaration:** Integer j, jf, jb, n, m, l, ls, i1, i2, i3, is, ie, js, je, iec, ibo Real fc, fcu, fda, ddx, ddy, dsm, dzm, amsk, umsk, d1, u, v Logical curved **Comments:** Put the negative of the curvature correction in the open boundary rows.

Subroutine	Description
	The horizontal averaging will then (approximately) cancel the curvature term for the normal velocities at open boundary points, which is desired (i.e., since no horizontal advection of momentum is currently applied at normal velocity points at open boundaries, a curvature correction should not be applied at these points). If horizontal advection is included at open boundary points, then combine the curvature correction with the Coriolis term (fda) since these two terms are treated the same way. Array "fda" has been stored as: 0.25*(Coriolis_parameter)*(grid_cell_area).
Dens1	Subroutine DENS1 calculates (density - 1000 kg/m^3) using the Fredrich-Levitus equation of state. This equation of state as used here is limited to the range: $-2 < T < 30$, $30 < S < 38$, $Z < 2000$m. **Calling Sequence:** dens1 (ja, jb, n, m, l, ls, is, ie, iec, zm, amsk, t, s, rho, ae, be, ce, de) **Data Declaration:** Integer ja, jb, n, m, l, ls, is, ie, iec Real zm, amsk, t, s, rho, ae, be, ce, de
Dens3	Subroutine DENS3 calculates *in situ* density minus 1000 kg/m^3. The United Nations Educational, Scientific, and Cultural Organization (UNESCO) equation of state is taken from POM. This density calculation includes the effect of pressure and uses the "potential" temperature, NOT the *in situ* temperature. This is an expensive density calculation (over 48 operations per point including a square root and a divide). In many situations, a simpler and more efficient equation of state would be adequate, though one must be sure the equation is approximately valid over the range of T, S, and depth used. **Calling Sequence:** dens3 (ja, jb, n, m, l, ls, is, ie, iec, rho0, g, sm, zm, h1, amsk, t, s, sos, rho) **Data Declaration:** Integer ja, jb, n, m, l, ls, is, ie, iec Real rho0, g, sm, zm, h1, amsk, t, s, sos, rho
Dep_var	Subroutine DEP_VAR calculates depth variables that depend on (and hence change with) the surface elevation. All depth variables that depend on the surface elevation are defined to be (+). Note on tiling: du and d1u at i = 0 are needed at interior tile edges. These cannot be calculated here since this would require e(i = -1) (and also hu and h1u at i = -1). Hence, du and d1u at i = 0 must be set by inter-tile passing after the call to DEP_VAR. During the momentum calculation, du and d1u are needed at time i2 for the calculation of ua and xk, but the values at time i1 are not needed until the correction of momentum by MEANUV at the end of the timestep; likewise for y variables. **Calling Sequence:** dep_var (ii, j, n, m, l, is, ie, isp, iep, js, je, iec, h, hu, hv, h1, h1u, h1v, e, d,du, dv, d1, d1u, d1v) **Data Declaration:** Integer ii, j, n, m, l, is, ie, isp, iep, js, je, iec Real h, hu, hv, h1, h1u, h1v, e, d, du, dv, d1, d1u, d1v
Get_extd	Subroutine GET_EXTD gets solar extinction data from the input file. It is set up for data on a single input file.

Subroutine	Description
	Calling Sequence: get_extd(indextd,nt,mt,n,m,iex1,iex2,idate,itime,timed,climatp, w1, extd, tmext2) **Data Declaration:** Integer indextd,nt,mt,n,m,iex1,iex2,idate,time Real timed,climatp,w1,extd,tmext2
Meanuv	Subroutine MEANUV corrects 3D velocity fields to match barotropic transport. All velocities are corrected, including normal values at open boundary points and values in boundary rows. Correct the baroclinic velocities so that the transport of the baroclinic velocities matches the barotropic transport. Set the baroclinic velocities at land points to zero. If the user is employing an explicit scheme with the same timestep for the baroclinic and barotropic modes, and the baroclinic velocities have all the forcing specified, including the surface pressure gradient, the correction here (for interior points) should be approximately zero. If the Flather open BC is being used, the normal barotropic transport at the open boundary points is calculated based on other criteria and will NOT agree with the baroclinic calculation of the transport. Hence, there will be a non-zero correction to the transport normal to the boundary at open boundary points. Note on tiling: Correct all values including boundary values for both exterior and interior tile edges. Boundary values should have been updated via open, periodic, or tiling BC. **Calling Sequence:** meanuv (ii, jj, j, ucr, vcr, na, ma, n, m, l, ls, is, ie, ism, iem, js, je, iec,indiag, small, dsm, dzm, umsk, vmsk, du, dv, d1u, d1v, udb, vdb, u, v, wpf) **Data Declaration:** Integer ii, jj, j, na, ma, n, m, l, ls, is, ie, ism, iem, js, je, iec, indiag Real ucr, vcr, small, dsm, dzm, umsk, vmsk, du, dv, d1u, d1v,udb, vdb, u, v, wpf
Presgrd	Subroutine PRESGRD calculates horizontal baroclinic pressure terms (pgx and pgy). These are calculated as horizontal pressure "differentials" (not gradients), i.e., the terms are not divided by dx or dy here. The method of calculation of pgx and pgy on the sigma part of the grid is taken from POM. A few things to note: 1) The baroclinic pressure terms are ramped. 2) The horizontally averaged density is subtracted from the density at the sigma layer points when calculating the baroclinic pressure terms. This is to remove the mean vertical gradient from the density, and to reduce truncation error in the calculation of horizontal density gradients on the sigma grid. 3) Invalid values of pgx and pgy will be calculated at land-sea boundaries. However, since the velocity is zero at these points, the invalid values of pgx and pgy will not be used. 4) In relation to slabbing, pgx and pgy are calculated for row j on a single call. No part of the calculation is saved between calls, i.e., each call to

Subroutine	Description
	PRESGRD for row j is independent of other calls. 5) For tiling the user must calculate pgx at all u-points being calculated. Do the same thing for pgy. PRESGRD is called for j = je+1 + iec(4), js -1. No calculation is done for j = je+1 + iec(4). For j = je+iec(4), pgy(j = je+1) is calculated for an exterior open boundary. The range for i-loops can run to ie+iec(2) or ie + 1. Either one will work. 6) Fourth-order interpolations and differences used here implicitly assume that horizontal grid stretching is very weak; otherwise there will be significant spatial truncation error. **Calling Sequence:** presgrd (j, jc1, jc2, jc3, jc4, rho_a, pgx, pgy, n, m, l, ls, nr, i2, is, ie, js, je,iec, bclinic, g, rho0, ramp, sw, sm, dsw, zw, zm, amsk, d1, d1u, d1v, rho, rmean, rsx, rsy, rdx, rdy) **Data Declaration:** Integer j, jc1, jc2, jc3, jc4, n, m, l, ls, nr, i2, is, ie, js, je, iec Real rho_a, pgx, pgy, g, rho0, ramp, sw, sm, dsw, zw, zm, amsk, d1, d1u, d1v, rho, rmean, rsx, rsy, rdx, rdy Logical bclinic
Rlaxdts3	Subroutine RLAXDTS3 relaxes the 3D T and S fields to specified values. This routine allows for the specified T and S fields to be either fixed in time or time-varying. The time-varying T and S relaxation fields can be used to provide a nudging form of data assimilation. **Calling Sequence:** rlaxdts3(j,nt,mt,n,m,l,ls,is,ie,idate,itime,times,indlxts,rlax_ts, rlax_ds,h1,sm,zm,amsk,t,s,tmean,smean,ilx1,ilx2,rlx,wlx,tmlx) **Data Declaration:** Integer j, nt, mt, n, m, l, ls, is, ie, idate, itime, ilx1, ilx2 Real times, rlax_ts, rlax_ds, h1, sm, zm, amsk, t, s, tmean,smean, rlx, wlx,tmlx
Solext	Subroutine SOLEXT calculates solar extinction. Extinction should be set to zero at the bottom of each column. This effectively masks out land points. **Calling Sequence:** solext (j, n, m, l, ls, nr, kb, is, ie, js, ext, h1, sw, zw, amsk, e, d1, r) **Data Declaration:** Integer j, n, m, l, ls, nr, kb, is, ie, js Real ext, h1, sw, zw, amsk, e, d1, r
Source1	Subroutine SOURCE1 defines source flow arrays sor and sorb for river inflows and defines river data arrays used in SOURCE2. The source flow arrays sor and sorb can be used to define various sources/sinks of water including rivers, runoffs, rainfall/evaporation, or other inflows or outflows. **Calling Sequence:** source1 (nt, mt, n, m, l, nr, is, ie, js, je, kb, nrvmax, nriv, nrriv, lriv, indriv,indrivr, locate, idate, itime, times, ramp, dti, irv1, irv2, iriv, jriv, isriv, ieriv, wtriv, qriv, rriv, tmriv, w1riv, sor, sorb) **Data Declaration:** Integer nt, mt, n, m, l, nr, is, ie, js, je, kb, nrvmax, nriv, nrriv, lriv,indriv, indrivr, idate, itime, irv1, irv2, iriv, jriv, isriv, ieriv

Subroutine	Description
	Real times, ramp, dti, wtriv, qriv, rriv, tmriv, w1riv, sor, sorb Logical locate **Comments:** In regards to tiling, sor and sorb need to be defined for 0, n and 0, m for the interior edges because of averaging needed for velocity points. For exterior edges, sor is not needed at normal velocity points. For real-time data associated with temporal interpolation of river data, use model elapsed time in days since start of the model run. For climate data, use elapsed time in days since the beginning of the year.
Source2	Subroutine SOURCE2 defines values of scalar fields for source flows. If the source flow at a grid cell is zero, the value of the scalar field does not need to be defined at that grid cell since it will be multiplied by zero. **Calling Sequence:** source2 (j, ir, n, m, l, nr, is, ie, indriv, indrivr, locate, nrvmax, lriv, irv1,irv2, iriv, jriv, isriv, ieriv, rriv, w1riv, r, rsor) **Data Declaration:** Integer j, ir, n, m, l, nr, is, ie, indriv, indrivr, nrvmax, lriv, irv1,irv2, iriv, jriv, isriv, ieriv Real rriv, w1riv, r, rsor Logical locate
Update	Subroutine UPDATE updates model fields in one timestep. **Calling Sequence:** update (na, ma, n, m, l, ls, nr, nq, ntyp, i1, i2, i3, j1, j2, kb, kbu, kbv, is, ie,ism, iem, isp, iep, js, je, iec, ibo, ke, iter, ramp, times, dti2, de, fda, botruf, cbu, cbv, istype, iptype, qrf, ext, elon, alat, ang, dx, dxu, dxv, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, ddx, ddy, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, w, r, q, tl, rho, sos, rmean, xk, yk, zkm, zkh, patm, usflx, vsflx, rsflx, solar, surruf, wubot, wvbot, ilx1, ilx2, rlx, wlx, tmlx, nobmax, nob, neob, nuob, nvob, iob, job, iobi, jobi, ivob, jvob, iob1, iob2, eob, ubob, vbob, cgwb, uob, vob, rob, tmob, ntc, etab, etpb, utab, utpb, vtab, vtpb, nrvmax, nriv, nrriv, lriv, iriv, jriv, isriv, ieriv, irv1, irv2, wtriv, qriv, rriv, tmriv, fu, fv, aax, aay, ucr1, vcr1, ucr2, vcr2, wxy, wxz, o) **Data Declaration:** Integer na, ma, n, m, l, ls, nr, nq, ntyp, i1, i2, i3, j1, j2, kb, kbu,kbv, is, ie, ism, iem, isp, iep, js, je, iec, ibo, ke, iter, istype,iptype, ilx1, ilx2, nobmax, nob, neob, nuob, nvob, iob, job,iobi, jobi, ivob, jvob, iob1, iob2, ntc, nrvmax, nriv, nrriv,lriv, iriv, jriv, isriv, ieriv, irv1, irv2 Real ramp, times, dti2, de, fda, botruf, cbu, cbv, qrf, ext, elon,alat, ang, dx, dxu, dxy, dxr, dxur, dxvr, dy, dyu, dyv, dyr, dyur, dyvr, ddx, ddy, da, dau, dav, dar, daur, davr, h, hu, hv, h1, h1u, h1v, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw,

Subroutine	Description
	dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, ub, vb, u, v, w, r, q, tl, rho, sos, rmean, xk, yk, zkm, zkh, patm, usflx, vsflx, rsflx, solar, surruf, wubot, wvbot, rlx, wlx, tmlx, eob, ubob, vbob, cgwb, uob, vob, rob, tmob, ntc, etab, etpb, utab, utpb, vtab, vtpb, wtriv, qriv, rriv, tmriv, fu, fv, aax, aay, ucr1, vcr1, ucr2, vcr2, wxy, wxz, o **Comments:** Definition of the timestep: If "forward" is set to true, use a forward difference for the first timestep (i.e., for iter = 1). If a forward timestep is used, values at the old (previous) time level (i3 or j1) should have been set equal to the current (i2 or j2) values. The time level indices are: i3 and j1 = old (n-1) time level i2 and j2 = present (n) time level i1 = new (n+1) time level The scalar fields (e.g., T and S), which are stored in array r, are only stored at two time levels. The old values at j1 are replaced with new values. The momentum calculation can be iterated to correct the advection and bottom drag terms by setting itermom > 1. One reason for doing this is that the advection field for momentum depends on the new surface elevation and the advective transports, which are not known exactly if the free-surface equations are solved implicitly or with a split-explicit scheme. By iterating the solution of the 3D momentum and free-surface equations, the slight error can be removed. This procedure is costly and is usually not necessary since the error tends to be small.
Updatrq	Subroutine UPDATRQ updates scalar and turbulence fields. A slab calculation is used whereby the calculation proceeds through the model domain in x-z sections. The calculation proceeds from the back of the domain to the front. **Calling Sequence:** updatrq (na, ma, n, m, l, ls, nr, nq, i1, i2, i3, j1, j2, kb, is, ie, isp, iep, js, je,iec, ke, mode, indadvr, indxk, indzk, indtkes, indlxts, indriv, indrivr, indbio, indiag, noslip, sigdif, shrnkwp, locate, idate, itime, iter, ramp, times, dti2, asf, vg1, vg2, vg3, g, rho0, xkmin, ykmin, xkre, prnxi, zkmmin, zhmin, botruf, rlax_ts, ext, small, dxur, dxv, dyu, dyvr, da, dar, h1, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, u, v, w, r, q, tl, rho, sos, rmean, xk, yk, zkm, zkh, usflx, vsflx, rsflx, solar, surruf, wubot, wvbot, ilx1, ilx2, rlx, wlx, tmlx, nrvmax, lriv, iriv, jriv, isriv, ieriv, irv1, irv2, rriv, w1riv, uacr, vacr, wpf, flyr, flyq, qold, ua, va, wa, rjp1, wxz) **Data Declaration:** Integer na, ma, n, m, l, ls, nr, nq, i1, i2, i3, j1, j2, kb, is, ie, isp, iep,js, je, iec, ke, mode, indadvr, indxk, indzk, indtkes, indlxts, indriv, indrivr, indbio, indiag, idate, itime, iter, ilx1, ilx2, nrvmax, lriv, iriv, jriv, isriv, ieriv, irv1, irv2

Subroutine	Description
	Real: ramp, times, dti2, asf, vg1, vg2, vg3, g, rho0, xkmin,ykmin, xkre, prnxi, zkmmin, zhmin, botruf, rlax_ts, ext, small, dxur, dxv, dyu, dyvr, da, dar, h1, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, sorb, e, d, du, dv, d1, d1u, d1v, udb, vdb, u, v, w, r, q, tl, rho, sos, rmean, xk, yk, zkm, zkh, usflx, vsflx, rsflx, solar, surruf, wubot, wvbot, rlx, wlx, tmlx, rriv, w1riv, uacr, vacr, wpf, flyr, flyq, qold, ua, va, wa, rjp1, wxz Logical: noslip, sigdif, shrnkwp, locate
Updatuv	Subroutine UPDATUV updates 3D momentum fields. A slab calculation is used whereby the calculation proceeds through the model domain in x-z sections. The calculation proceeds from the back of the domain to the front. **Calling Sequence:** updatuv (fu, fv, na, ma, n, m, l, ls, nr, i1, i2, i3, j1, j2, kb, kbu, kbv, is, ie,ism, iem, isp, iep, js, je, iec, ibo, ke, indbaro, indden, indadv, indxk, indzk, indrag, indatp, indiag, bclinic, curved, noslip, largmix, tidpot, vector, shrnkwp, locate, iter, iterm, ramp, times, dti, dti2, eg1, eg2, eg3, vg1, vg2, vg3, g, rho0, fda, xkmin, ykmin, xkre, prnxi, zkmmin, zkhmin, zkre, botruf, cbu, cbv, small, ae, be, ce, de, cet, ces, elon, alat, dx, dxur, dxv, dy, dyu, dyvr, ddx, ddy, da, dar, dau, daur, dav, davr, h1, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, e, du, dv, d1, d1u, d1v, udb, vdb, u, v, w, r, tl, rho, sos, rmean, xk, yk, zkm, zkh, patm, usflx, vsflx, surruf, wubot, wvbot, uacr, vacr, wpf, ua, va, wa, fc, fcu, flyu, flyv, rho_a, pgx, pgy, wxz) **Data Declaration:** Integer: na, ma, n, m, l, ls, nr, i1, i2, i3, j1, j2, kb, kbu, kbv, is, ie,ism, iem, isp, iep, js, je, iec, ibo, ke, indbaro, indden, indadv, indxk, indzk, indrag, indatp, indiag, iter, iterm Real: fu, fv, ramp, times, dti, dti2, eg1, eg2, eg3, vg1, vg2, vg3,g, rho0, fda, xkmin, ykmin, xkre, prnxi, zkmmin, zkhmin, zkre, botruf, cbu, cbv, small, ae, be, ce, de, cet, ces, elon, alat, dx, dxur, dxv, dy, dyu, dyvr, ddx, ddy, da, dar, dau, daur, dav, davr, h1, sw, sm, dsw, dsm, dsm5, dswr, dsmr, zw, zm, dzw, dzm, dzm5, dzwr, dzmr, amsk, umsk, vmsk, sor, e, du, dv, d1, d1u, d1v, udb, vdb, u, v, w, r, tl, rho, sos, rmean, xk, yk, zkm, zkh, patm, usflx, vsflx, surruf, wubot, wvbot, uacr, vacr, wpf, ua, va, wa, fc, fcu, flyu, flyv, rho_a, pgx, pgy, wxz

Subroutine	Description
	Logical bclinic, curved, noslip, largmix, tidpot, vector, shrnkwp,locate
Xk_re	Subroutine XK_RE calculates horizontal eddy coefficients at the u and v-points. The eddy coefficients are stored as the eddy coefficient times the area of the grid cell face normal to the diffusion direction divided by the grid spacing in the diffusion direction. The magnitude of the eddy coefficients is calculated based on the maximum of the local grid cell Reynolds number and a background value. The horizontal eddy coefficients need to be masked for diffusion of scalars and for momentum for "free-slip" lateral boundaries. They should not be masked for momentum for the "no-slip" lateral boundaries. Consider increasing viscosity near open boundaries to dampen noise (as a last resort). **Calling Sequence:** xk_re (ind, j, n, m, l, ls, i2, isp, iep, js, je, iec, indxk, noslip, locate,xkmin, ykmin, xkre, prnxi, dxv, dxur, dyu, dyvr, dsm, dzm, d1u, d1v, umsk, vmsk, u, v, xk, yk) **Data Declaration:** Integer ind, j, n, m, l, ls, i2, isp, iep, js, je, iec, indxk Real xkmin, ykmin, xkre, prnxi, dxv, dxur, dyu, dyvr, dsm, dzm,d1u, d1v, umsk, vmsk, u, v, xk, yk Logical noslip, locate
Xk_smag2	Subroutine XK_SMAG2 calculates the horizontal eddy coefficients using a modified Smagorinsky scheme. The calculation here differs from that used in POM in that the eddy coefficients, which are calculated at the grid-cell centers, are averaged to the grid-cell boundaries and the cross momentum diffusion terms are not used, i.e., the momentum diffusion is purely Laplacian. In this way, the calculation of horizontal diffusion is the same as what is used for the grid-cell Reynolds diffusion. The momentum diffusion calculation itself does not need to be changed. **Calling Sequence:** xk_smag2 (ind, na, ma, n, m, l, ls, i2, is, ie, isp, iep, js, je, iec, ibo,indxk, indcyc, noslip, locate, xkmin, ykmin, prnxi, smag, dxr, dyr, dxv, dxur, dyu, dyvr, da, dsm, dzm, d1u, d1v, amsk, umsk, vmsk, u, v, xk, yk, aa, bb) **Data Declaration:** Integer ind, na, ma, n, m, l, ls, i2, is, ie, isp, iep, js, je, iec, ibo,indxk, indcyc Real xkmin, ykmin, prnxi, smag, dxr, dyr, dxv, dxur, dyu, dyvr,da, dsm, dzm, d1u, d1v, amsk, umsk, vmsk, u, v, xk, yk, aa, bb Logical noslip, locate **Boundary conditions for diffusion coefficients:** *Land-sea boundaries:* Since the eddy coefficients are averaged from grid-cell centers to grid-cell boundaries, values at land cells adjacent to sea cells are needed for momentum diffusion (for scalar diffusion, eddy coefficients at land-sea boundaries are set to zero). POM sets a constant value at land points to use for this averaging. Here a zero gradient relative to the adjacent sea point is used by (a) first setting values at land points to zero, and then (b) multiplying averages taken at land-sea boundaries by two

Subroutine	Description
	through multiplying by (2-umsk) at u-points and (2-vmsk) at v-points. The gradient normal to the boundary of the flow and tangent to the boundary is underestimated by half in the momentum diffusion term because the zero velocity 1/2 grid cell from the boundary is used rather than taking tangential velocity = 0 right at the boundary (there is an underestimate (33%) in the calculation of the Smagorinsky coefficient in this subroutine for the same reason). This could be accounted for in the momentum diffusion term by multiplying by (2-cmsk) where *cmsk* is a land-sea mask defined at horizontal grid cell corners. *Free slip:* Free slip at land-sea boundaries can be implemented by masking eddy coefficients at land-sea boundaries to zero. This has two problems, however: (1) momentum diffusion at an open corner will not be zero, and (2) the normal velocity grid point from a land-sea boundary will have diffusion reduced by half. It is better to define a "corner" mask that is set to zero at land-sea boundaries and apply it when calculating u diffusion in y or v diffusion in x. *Open boundaries:* Set a zero gradient at open boundaries. This could be done via a call to OPENBC, but it could also just be done within this subroutine. *Periodic/tile boundaries:* Call periodic or halo setting routines either in OPENBC or within this subroutine to set values. *Model calculation procedure for XK_SMAG2:* 1) For diffusion of momentum, call XK_SMAG2 before the x-z slabbing loop, and calculate eddy coefficients for the entire grid on a single call, since boundary and halo values have to be set. Values defined are: xk = (diffusion coefficient in x)*dzm*dyu/dxu at a u-point. yk = (diffusion coefficient in y)*dzm*dxv/dyv at a v-point. 2) For diffusion of scalars, call from within the slabbing loop (just as for the grid-cell-Reynolds scheme) and calculate eddy coefficients for a single slab. Since the eddy coefficients have already been calculated, just multiply by the inverse Prandtl Number and mask the values to zero at land-sea boundaries. Currently, all the eddy coefficients are calculated for scalar diffusion at once rather than slab-by-slab.

5.4.14 Utility Subroutines (ncom1util)

Subroutine	Description
Bc_sym8	Subroutine BC_SYM8 enforces an eight-fold symmetry in the boundary condition. This is used to test nesting since the interpolations used to calculate the nesting boundary conditions are inherently asymmetric. This routine is for single processor use only. **Calling Sequence:** bc_sym8 (l, nr, nob, neob, nuob, nvob, eob, ubob, vbob, uob, vob, rob) **Data Declaration:** Integer l, nr, nob, neob, nuob, nvob

Subroutine	Description
	Real eob, ubob, vbob, uob, vob, rob
Cfl	Subroutine CFL calculates and prints maximum values of CFL parameters for advection and diffusion over the entire 3D grid. **Calling Sequence:** cfl (n, m, l, ls, i2, is, ie, ism, iem, isp, iep, js, je, iec, dti, xkmin, ykmin,zkhmin, small, dxu, dxv, dxur, dyu, dyv, dyvr, dz_t, amsk, umsk, vmsk, d1, u, v, w, xk, yk, zkh, dz5) **Data Declaration:** Integer n, m, l, i2, is, ie, ism, iem, isp, iep, js, je, iec Real dti, xkmin, ykmin, zkhmin, small, dxu, dxv, dxur, dyu, dyv, dyvr, dz_t, amsk, umsk, vmsk, d1, u, v, w, xk, yk,zkh, dz5
Chk_nan	Subroutine CHK_NAN checks an array “a” for bad values (not a number-NaN’s). The program stops execution if bad values are found. **Calling Sequence:** chk_nan(nest,n,m,l,a) **Data Declaration:** Integer nest,n,m,l Real a
Chkolap	Subroutine CHKOLAP checks the Arctic overlap. This is a scalable (multi-tile) version. There is no checking of v-points. **Calling Sequence:** chkolap (name, f, n, m, l, na, ma, ipos, ivec) **Data Declaration:** Integer name, n, m, l, na, ma, ipos, ivec Real f
Chksym4	Subroutine CHKSYM4 checks arrays for four-fold symmetry. This kind of symmetry can be maintained when the Coriolis parameter equals a constant within the domain. A field defined at t-points may be single or paired, and if paired may be vector or not, whereas a field defined at staggered u, v-points must be paired, but may or may not be vector. This routine is for single processor use only. **Calling Sequence:** chksym4 (name, u, v, n, m, l, ipos, pair, ivec, iset) **Data Declaration:** Integer name, n, m, l, ipos, ivec, iset Real u, v, pair
Chksym8	Subroutine CHKSYM8 checks arrays for eight-fold symmetry. A field defined at t-points may be single or paired, and if paired may be vector or not. However a field defined at staggered u, v-points must be paired, but may or may not be vector. This routine is for single processor use only. **Calling Sequence:** chksym8 (name, u, v, n, m, l, ipos, pair, ivec, iset) **Data Declaration:** Integer name, n, m, l, ipost, ivec, iset Real u, v, pair
Conserv	Subroutine CONSERV checks the conservation of volume and scalar fields. This subroutine writes out minimum and maximum values, mean values, initial mean values, and change in mean values. This subroutine is strictly for diagnostics and this version is vectorized. **Calling Sequence:** conserv (na, ma, n, m, l, ls, nr, i1, j1, is, ie, js, je, iter, times, da, dz_t, amsk, e, d1, r, wsp1, wsp2) **Data Declaration:** Integer na, ma, n, m, l, nr, i1, j1, is, ie, js, je, iter

Subroutine	Description
	Real times, da, dz_t, amsk, e, d1, r, wsp1, wsp2
Fcmnmx	Subroutine FCMNMX computes minimum and maximum values of array fc. Array fc is calculated in subroutine COR_CURV. Large accumulations in array fc have caused overflow problems. This has been a sufficient enough problem that this routine was created to compute extreme values of fc and print them out along with their processor and (local) grid-point location. **Calling Sequence:** fcmnmx (j, jf, jb, fc, n, m, l) **Data Declaration:** Integer j, jf, jb, n, m, l Real fc
Out_put	Subroutine OUT_PUT writes 3D model fields to the output file for checking. It is for single processor use only. **Calling Sequence:** out_put(iter,time,nmh,n,m,l,nr,amsk,umsk,vmsk, e,u,v,t,s) **Data Declaration:** Integer iter,nmh,n,m,l,nr Real time,amsk,vmsk,vmsk,e,u,v,t,s
Prnt0	Subroutine PRNT0 prints a 2D field. **Calling Sequence:** prnt0(n,m,f,name,amult) **Data Declaration:** Character name Integer n,m Real f,amult
Prnt3m	Subroutine PRNT3M prints the min, max, and mean value of the input array on each processor. It is used for debugging diagnostics. **Calling Sequence:** prnt3m(message,a,n1,n2,m1,m2,n,m) **Data Declaration:** Character message Integer n1,n2,m1,m2,n,m Real a
Rotcone	Subroutine ROTCONE sets velocity field for solid body rotation. This is used for the rotating cone advection test. **Calling Sequence:** rotcone(ind,n,m,l,h,amsk,e,udb,vdb,ub,vb,u,v,w) **Data Declaration:** Integer ind Real h,amsk,e,udb,vdb,ub,vb,u,v,w
Setscr	Subroutine SETSCR sets scratch arrays to high values for testing. This is done to test the integrity of the model calculations. Since these scratch arrays are reused for different calculations and/or different nests, existing values on entry into subroutine OMODEL should not affect the calculations in OMODEL. The last dimension of scratch arrays wxy and wxz is hardwired in the do loops below, but is subject to change as the ocean model program is modified and updated. Check the space allocated for these two arrays in subroutine MEMMO2. **Calling Sequence:** setscr (n, m, l, tl, rho, sos, xk, yk, zkb, wxy, wxz) **Data Declaration:** Integer n, m, l Real tl, rho, sos, xk, yk, zkb, wxy, wxz
Ssh_0	Subroutine SSH_0 restores global mean sea surface height to zero. **Calling Sequence:** ssh0(na,ma,n,m,da,amsk,e, wsp1,wsp2) **Data Declaration:** Integer na,ma,n, m,

Subroutine	Description
	Real da,amsk,e,wsp1,wsp2

5.4.15 Vertical Mixing Subroutines (ncom1vmix_sigz)

Subroutine	Description
My12tab	Subroutine MY12TAB provides a lookup table for the Richardson Number. **Calling Sequence:** my12tab (ri, sm, sh) **Data Declaration:** Real ri, sm, sh
Profq2	Subroutine PROFQ2 calculates the source and dissipation terms, vertical mixing for turbulence fields, and new values of vertical mixing coefficients. This version of PROFQ (version 2) is modified from the original PROFQ (which is set up like POM's PROFQ) to allow specifying either the surface value of TKE (if indtkes = 1) or the surface flux of TKE (if indtkes = 2). The surface roughness is specified in array surruf. Both the surface and bottom roughness are treated more consistently than in the original version of PROFQ, i.e., they are included in defining the "wall function" in the dissipation term of the Q2L equation. **Calling Sequence:** profq2 (j, qold, n, m, l, ls, nq, i1, i2, j1, j2, kb, is, ie, ke, indtkes, shrnkwp,dti2, asf, g, rho0, zkmmin, botruf, small, sw, sm, dsm, dswr, dsmr, zw, zm, dzm, dzwr, dzmr, amsk, e, d, d1, u, v, q, tl, rho, sos, zkm, zkh, usflx, vsflx, surruf, wubot, wvbot, boygr, bl, aa, bb, cc, ee, gg, gh, sm1, sh1) **Data Declaration:** Integer j, n, m, l, ls, nq, i1, i2, j1, j2, kb, is, ie, ke, indtkes Real dti2, asf, g, rho0, zkmmin, botruf, small, sw, sm, dsm,dswr, dsmr, zw, zm, dzm, dzwr, dzmr, amsk, e, d, d1, u, v, q, tl, rho, sos, zkm, skh, usflx, vsflx, surruf, wubot, wvbot,boygr, bl, aa, bb, cc, ee, gg, gh, sm1, sh1 Logical shrnkwp
Profr	Subroutine PROFR calculates vertical turbulent mixing of scalar fields. **Calling Sequence:** profr (j, n, m, l, ls, nr, i1, j1, j2, is, ie, ke, shrnkwp, dti2, asf, zkhmin,small, dsm, dswr, dsmr, dzm, dzwr, dzmr, amsk, d1, r, zkh, aa, bb, cc, ee, gg) **Data Declaration:** Integer j, n, m, l, ls, nr, i1, j1, j2, is, ie, ke Real dti2, asf, zkhmin, small, dsm, dswr, dsmr, dzm, dzwr,amsk, d1, r, zkh, aa, bb, cc, ee, gg Logical shrnkwp
Profuv	Subroutine PROFUV calculates vertical turbulent mixing of momentum. **Calling Sequence:** profuv (j, fu, fv, n, m, l, ls, i1, i2, i3, kbu, kbv, is, ie, ism, iem, js, je, iec, ke, indrag, dti2, zkmmin, cbu, cbv, small, dsm5, dswr, dsmr, dzm5, dzwr, dzmr, umsk, vmsk, du, dv, d1u, d1v, u, v, zkm, wubot, wvbot, aa, bb, cc, ee, gg)

Subroutine	Description
	Data Declaration: Integer j, n, m, l, ls, i1, i2, i3, kbu, kbv, is, ie, ism, iem, js, je, iec,ke, indrag Real fu, fv, dti2, zkmmin, cbu, cbv, small, dsm5, dswr, dsmr,dzm5, dzwr, dzmr, umsk, vmsk, du, dv, d1u, d1v, u, v, zkm, wubot, wvbot, aa, bb, cc, ee, gg
Trid2	Subroutine TRID2 solves a tri-diagonal set of equations in z over a 2D set of horizontal points. **Calling Sequence:** trid2 (n, m, l, n1, n2, j, l1, l2, aa, bb, cc, dd, ww, gg) **Data Declaration:** Integer n, m, l, n1, n2, j, l1, l2 Real aa, bb, cc, dd, ww, gg
Zkmyl2	Subroutine ZKMYL2 calculates vertical mixing coefficients using a slightly modified version of the MYL2 mixing parameterization. The turbulent length scale (tl) is calculated with a parabolic shape over each turbulent region in which Ri < critical Ri. The eddy coefficients are temporally filtered by averaging the newly calculated values with the values calculated on the previous timestep. The mixing coefficients can be augmented with the Ri-dependent background mixing (Large et al., 1994, also used by Kantha and Clayson, 1994) by setting logical parameter largmix = true. Because of temporal filtering, zkm and zkh need to be saved between timesteps, and need to be in the restart file. **Calling Sequence:** zkmyl2 (j, n, m, l, ls, nr, i1, i2, i3, j1, j2, kb, is, ie, js, je, iec, largmix, iter,g, rho0, zkmmin, zkhmin, zkre, botruf, cet, ces, dsw, dsm, dsm5, dzw, dzm, dzm5, dzwr, amsk, d1, u, v, w, r, tl, zkm, zkh, usflx, vsflx, surruf, wubot, wvbot) **Data Declaration:** Integer j, n, m, l, ls, nr, i1, i2, i3, j1, j2, kb, is, ie, js, je, iec, iter Real g, rho0, zkmmin, skhmin, zkre, botruf, cet, ces, dsw, dsm, dsm5, dzw, dzm, dzm5, dzwr, amsk, d1, u, v, w, r, tl, zkm, zkh, usflx, vsflx, surruf, wubot, wvbot Logical largmix
Zkmyl2v	Subroutine ZKMYL2V differs from ZKMYL2 above in that all the calculations are set up to vectorize on computers like the Cray. On scalar computers, ZKMYL2V may be faster since land points are skipped. **Calling Sequence:** zkmyl2v (j, n, m, l, ls, nr, i1, i2, i3, j1, j2, kb, is, ie, js, je, iec, ibo, largmix, iter, g, rho0, zkmmin, zkhmin, zkre, botruf, cet, ces, small, dsw, dsm, dsm5, dzw, dzm, dzm5, amsk, d1, u, v, w, r, tl, zkm, zkh, usflx, vsflx, surruf, wubot, wvbot, dzw2, dzm2, aa, bb, dr2, du2, ri2) **Data Declaration:** Integer j, n, m, l, ls, nr, i1, i2, i3, j1, j2, kb, is, ie, js, je, iec, ibo, iter Real g, rho0, zkmmin, zkhmin, zkre, botruf, cet, ces, small, dsw,dsm, dsm5, dzw, dzm, dzm5, amsk, d1, u, v, w, r, tl, zkm, zkh, usflx, vsflx, surruf,

Subroutine	Description
	wubot, wvbot, dzw2, dzm2, aa, bb, dr2, du2, ri2 Logical largmix

5.5 NetCDF-Specific Subroutines (libsrc/ cdf/)

Subroutine	Description
Closeds	Subroutine CLOSEDS closes the data set with the identifier *idds*. **Calling Sequence:** closeds(idds,ierrout) **Data Declaration:** Integer idds,ierrout
Closesds	Subroutine CLOSESDS closes the closes the netCDF file with the given ID (*idf*). **Calling Sequence:** closesds(idf,ierrout) **Data Declaration:** Integer idf,ierrout
Convcase	Subroutine CONVCASE converts a character string to all uppercase or all lowercase letters. **Calling Sequence:** convcase(cin,cout,len,upcase) **Data Declaration:** Integer len Character cin,cout Real upcase
Cyclaxis	Subroutine CYCLAXIS checks longitude axis to insure that it is monotonically increasing. If this test is passed, then it determines whether the longitude axis is cyclic. If it is cyclic, then it determines whether the first and last points are at the same longitude, or whether the last point is one grid point to the left of the first grid point. Finally it modifies axis values so that the right end of the axis is greater than zero, but less than or equal to 360. **Calling Sequence:** cyclaxis(rlon,nx,dx,longlobe,ierrout) **Data Declaration:** Integer nx,ierrout,longlobe Real rlon,dx,rlonmin1,rlonmax1
Cyclaxis2	Subroutine CYCLAXIS2 checks longitude axis to insure that it is monotonically increasing. If this test is passed, then it determines whether the longitude axis is cyclic. If it is cyclic, then it determines whether the first and last points are at the same longitude, or whether the last point is one grid point to the left of the first grid point. Finally it modifies axis values so that the right end of the axis is greater than zero, but less than or equal to 360. **Calling Sequence:** cyclaxis2 (rlonmin,rlonmax,nx,longlobe,ierrout) **Data Declaration:** Integer nx,ierrout,longlobe Real rlonmin,rlonmax,dx,rlonmin1,rlonmax1
Decodeidds	**Calling Sequence:** decodeidds(encodedidds,I,O,idf,idds) **Data Declaration:** Integer idf,idds,encodedidds Real rlonmin,rlonmax,dx,rlonmin1,rlonmax1
Encodeidds	**Calling Sequence:** encodeidds(idf,idds,I,O,encodedidds) **Data Declaration:** Integer idf,idds,encodedidds
Fixname	**Calling Sequence:** fixname(name) **Data Declaration:** Character name
Getcattr	Subroutine GETCATTR searches for the character attribute, stored in the character

Subroutine	Description
	attribute array, associated with a given name and loads it into the character variable *cx*. **Calling Sequence:** getcattr(maxattr,maxname,maxannot,name,ncattr,cattr, cattrnam,cx) **Data Declaration:** Character cattrnam,name,cattr,cx,name1c,attr
Getiattr	Subroutine GETIATTR searches for the integer number attribute, stored in integer number attribute array, associated with a given name and loads it into the integer variable *ix*. **Calling Sequence:** getiattr(name,maxattr,maxname,niattr,iattr,iattrnam,ix) **Data Declaration:** Character iattrnam,name,name1c,attr Integer iattr,ix
Getrattr	Subroutine GETRATTR searches for the real*4 number attribute, stored in real number attribute array, associated with a given name and loads it into the real variable *x*. **Calling Sequence:** getrattr(maxattr,maxname,name,nrattr,rattr,rattrnam,x) **Data Declaration:** Character rattrnam,name,name1c,attr,maxattr Real rattr,x
Infods	Given the identification number of a netCDF scientific data file and the consecutive data set index number (which starts at zero), this routine determines the identification code for this data set and whether this is a data grid or a coordinate variable. If it is a data grid, then the name, number of dimensions, the size of each dimension, and the number of attributes for this data set is determined. **Calling Sequence:** infods(idf,maxname,name,encodedidds,index,indexg,indext, ndsattr,irank,ishape,max1d) **Data Declaration:** Integer idds,idf,index,isds,irank,numtype,ndsattr, indexg, indext,ishape encodedidds Character name
Isacoordvar	**Calling Sequence:** isacoordvar(idf,idds,I,O,isds) **Data Declaration:** Integer idds,idf,isds,numtype,irank,ierr,ndsattr Character name
Opensds	Subroutine OPENCDF opens a netCDF scientific data set file for access and then obtains information on the number of scientific data sets it contains and the number of global *sds* attributes for the file. When the file is opened, the file *idis* is retrieved or created, and returned. **Calling Sequence:** opensds(filenm,idf,iaccess,ndatasets,nfileattr,ierr) **Data Declaration:** Integer idf,iaccess,ncopn,nccre,ndatasets,nfileattr,ierr, ndims,irecdim Character filnm
Pack_int2	**Calling Sequence:** pack_int2(npts,grid,work,tmin,tmax,nbits, emax,eavg, erms, ispval,ierr) **Data Declaration:** Integer npts, nbits,ierr,work,ispval, Real grid, tmin,tmax,emax,eavg,erms
Pack_intl	**Calling Sequence:** pack_int1(npts,grid,work,tmin,tmax,nbits, emax, eavg, erms,

Subroutine	Description
	ispval,ierr) **Data Declaration:** Integer npts, nbits,ierr,work,ispval Real grid, tmin,tmax,emax,eavg,erms,evar
Putcattr	Subroutine PUTCATTR searches for the character attribute, stored in character attribute array, associated with a given name and loads it into the character variable *cx*. **Calling Sequence:** putcattr(maxattr,maxname,maxannot,name,ncattr,cattr, cattrnam, cx,ierrout) **Data Declaration:** Character cattrnam,name,cattr,cx,name,attr
Putrattr	Subroutine PUTRATTR searches for the real*4 number attribute, stored in real number attribute array, associated with a given name and loads it into the real variable *x*. **Calling Sequence:** putrattr(maxattr,maxname,name,nrattr,rattr,rattrnam,x,ierrout) **Data Declaration:** Real rattr,x Integer ierrout Character iattrnam,name
Puttiattr	Subroutine PUTTIATTR searches for the integer number attribute, stored in integer number attribute array, associated with a given name and loads it into the integer variable *ix*. **Calling Sequence:** putiattr(maxattr,maxname,name,niattr,iattr,iattrnam,ix,ierrout) **Data Declaration:** Integer iattr,ix,ierrout Character iattrnam,name
Rdglattr	Subroutine RDGLATTR reads the global file attributes in a netCDF scientific data set. This routine should be called only after making a call to OPENSDS. **Calling Sequence:** rdglattr(idf,maxattr,maxname,maxannot,nfileattr,niattr,nrattr, ncattr,iattr,rattr,cattr,iattrnam,rattrnam,cattrnam,ierrout) **Data Declaration:** Integer idf, ierrout,ierr,numtype,icount,niattr,nrattr, ncattr,iattr,nfileattr Real rattr Character iattrnam,rattrnam,cattrnam,cattr, name
Rdsdsa	Subroutine RDSDSA reads everything in an HDF scientific data set, including all associated attributes, except the data grid. The data grid is read by a separate subroutine to allow easy reading of subsets. This routine should be called only after making calls to OPENSDS and then to INFODS, and after allocating space for the array sizes identified from the call to INFODS. **Calling Sequence:** rdsdsa(encodedidds,maxattr,maxname,maxannot,maxrank, irank,ishape,ndsattr,max1d,spval,datamin,datamax,scale,label, unit,fmt,dlabel,dunit,dfmt,coordsys,niattr,nrattr,ncattr,iattr,rattr cattr,iattrnam,rattrnam,cattrnam,ierrout) **Data Declaration:** Integer encodedidds,idim,iddim,idds, indx,ierrout, ierr, max1d,irank,ishape,numtype,icount,niattr,nrattr, iattr,idim_size,idcoordvar,icoordvarstart, icoordvarcounts

Subroutine	Description
	Real validrangera Character label,unit,fmt,dlabel,dunit,dfmt,name,cdata, iattrnamm, rattrnam, cattrnam, cattr,coordsys
Rdsdsd	Subroutine RDSDSD reads a slab (or the entire array) of an HDF scientific data set. The array space for the data set must be allocated before calling this routine. **Calling Sequence:** rdsdsd(encodedidds,maxrank,irank,islab,istart,istride,iedges, ishape,data,ierrout) **Data Declaration:** Integer encodedidds, imap, istart, istride, iedges, ishape, ispval Real data
Rdsdssc	**Calling Sequence:** rdsdssc(encodedidds,maxrank,irank,ishape,max1d,scale,ierrout) **Data Declaration:** Integer encodedidds,idim,iddim, idds,ierr, max1d,irank, ishape, idim_size, idcoordvar,icoordvarstart, icoordvarcounts Real scale Character coordvar_name
Sizeslab	Subroutine SIZESLAB determines scale indices along each dimension which span the subset required from the data set. **Calling Sequence:** sizeslab(maxrank,irankin,istride,xyztmin,xyztmax,irankout, shape,scale,max1d,iaddborder,istart,iedges) **Data Declaration:** Integer ishape,irankin, istart, istride, iedges, iaddborder Real scale,xztmin,xztmax
Unpack_int	**Calling Sequence:** unpack_int(npts,nptsij,datain,dataout,nbits, irank,islab, istart, iedges,istride,tmin,tmax,ispval,spval,ierr) **Data Declaration:** Integer npts, nptsij,nbits, ierr,irank,islab,istart,istride, iedges, ibeg, iinc, iend, jbeg,jinc,jend, kbeg, kinc, kend,datain, ispval Real work,dataout,spval,timin,tmax
Wtglattr	Subroutine WTGLATTR writes the global file attributes to a netCDF file. **Calling Sequence:** wtglattr(idf,maxattr,maxname,maxannot,niattr,nrattr,ncattr, iattr, rattr,cattr,iattrnam,rattrnam,cattrnam,ierrout) **Data Declaration:** Integer idf,ierrout,ierr,niattr,nrattr,ncattr,iattr,lenstr Character iattrnam,rattrnam,cattrnam,cattr Real rattr
Wtsda	Subroutine WTSDA writes an entire data set into a netCDF scientific data set. Associated attributes are also written. After the data set and attributes are written to the file, the access to this data set is terminated. **Calling Sequence:** wtsdsa(idf,idds,maxattr,maxname,maxannot,maxrank, max1d, irank, ishape, spval,scale,label,unit,fmt,dlabel,dunit,dfmt, coordsys,niattr,nrattr,ncattr,iattr,rattr,cattr,iattrnam,rattrnam, cattrnam,ierr) **Data Declaration:** Integer lenstr,idf, idim,iddim,idds,ierrout,ierr, max1d,irank,ishape,numtype,niattr,nrattr,ncattr,

Subroutine	Description
	iattr,istart,istride,imap, ivdimsra,ivarsidsra ivartypera Character label,unit,fmt,dlabel,dunit dfmt, iattrnam, rattrnam,cattrnam,cattr,coordsys Real scale,spval,rattr,datamin,datamax,validrangera
Wtsds	Subroutine WTSDS writes an entire data set into a netCDF scientific data set. Associated attributes are also written. After the data set and attributes are written to the file, the access to this data set is terminated. **Calling Sequence:** wtsds(idf,maxattr,maxname,maxannot,maxrank,max1d,irank, ishape,spval,scale,label,unit,fmt,dlabel,dunit,dfmtcoordsys,niattr,nrattr,ncattr,iattr,rattr,cattr,iattrnam,rattrnam,cattrnam,data, ierr) **Data Declaration:** Integer lenstr,idf, idim,iddim,idds,ierrout,ierr, max1d,irank,ishape,numtype,niattr,nrattr,ncattr, iattr,istart,istride,imap,i,icount Character label,unit,fmt,dlabel,dunit dfmt, iattrnam, rattrnam,cattrnam,cattr,coordsys Real data, scale,spval, rattr,datamin, datamax, validrangera
Wtsds_pack	Subroutine WTSDS_PACK writes an entire data set into a netCDF scientific data set. Associated attributes are also written. After the data set and attributes are written to the file, the access to this data set is terminated. **Calling Sequence:** wtsds_pack(idf,maxattr,maxname,maxannot,maxrank, max1d, irank, ishape,spval,scale,label,unit,fmt,dlabel,dunit, dfmt, coordsys,niattr,nrattr,ncattr,iattr,rattr,cattr,iattrnam,rattrnam,nbits,single,work,cattrnam,data,ierr) **Data Declaration:** Integer lenstr,idf, idim,iddim,idds,ierrout,ierr, max1d,irank,ishape,numtype,niattr,nrattr,ncattr, iattr,istart,istride,imap,i,icount,nbits,work Logical single Character label,unit,fmt,dlabel,dunit dfmt, iattrnam, rattrnam,cattrnam,cattr,coordsys Real data, scale,spval, rattr, datamin,datamax, validrangera
Wtsdsd	Subroutine WTSDSD writes a partial data set into a netCDF scientific data set. **Calling Sequence:** wtsdsd(idf,idds,maxrank,istart,iedges,istride,data,ierrout) **Data Declaration:** Integer idf, idds, ierrout,ierr,iedges,numtype, istart, istride, imap, ivartypera Real data

5.6 COAMPS Related Subroutines (libsrc/ coampslib/)

Subroutine	Description
Coamps_datar	Subroutine COAMPS_DATAR reads flat file fields for COAMPS.

Subroutine	Description
	Calling Sequence: coamps_datar(istdo,d,lend,fldnam,inest,itime,cdtg,cfluid, lvltyp,rlev1,rlev2,dsetux,lsetux,lwritu,istat) **Data Declaration:** Integer istdo,lend,inest,itime,lsetux,istat Real d,rlev1,rlev2, Logical lwritu Character cdtg,cfluid,fldnam,lvltyp,dsetux
Coamps_datar_new	Subroutine COAMPS_DATAR_NEW reads flat file fields for COAMPS. **Calling Sequence:** coamps_datar_new(istdo,d,lend,fldnam,inest,itime,cdtg, cfluid, lvltyp,rlev1,rlev2,dsetux,lsetux,lwritu,istat,outtyp,m,n) **Data Declaration:** Integer istdo,lend,inest,itime,lsetux,istat, m,n Real d,rlev1,rlev2 Logical lwritu Character cdtg,cfluid,fldnam,lvltyp,dsetux,outyp
Coamps_grdcon	Subroutine COAMPS_GRDCON calculates grid constants. igrid: type of grid projection: =1, mercator projection =2, Lambert conformal projection =3, polar stereographic projection =4, Cartesian coordinates =5, spherical projection **Calling Sequence:** coamps_grdcon(igrid,stdlt1,stdlt2,gcon) **Data Declaration:** Integer igrid Real gcon,stdlt1,stdlt2
Coamps_grdij	**Calling Sequence:** coamps_grdij(m,n,grdi,grdj) **Data Declaration:** Integer m,n Real grdi,grdj
Coamps_ij2ll	Subroutine COAMPS_IJ2LL computes latitude and longitude of specified i- and j-points on a grid. All latitudes in this routine start with -90.0 at the south pole and increase northward to +90.0 at the North Pole. The longitudes start with 0.0 at the Greenwich meridian and increase to the east, so that 90.0 refers to 90.0E, 180.0 is the International Dateline and 270.0 is 90.0W. **Calling Sequence:** coamps_ij2ll(igrid,reflat,reflon,iref,jref,stdlt1stdlt2,stdlon, delx,dely,grdi,grdj,npts,grdlat,grdlon) **Data Declaration:** Integer igrid,iref,jref,npts Real delx,dely,grdi,grdj,grdlat,grdlon,reflat,reflon, stdlon,stdlt1,stdlt2
Coamps_ll2ij	Subroutine COAMPS_LL2IJ computes latitude and longitude of specified i- and j-points on a grid. All latitudes in this routine start with -90.0 at the south pole and increase northward to +90.0 at the North Pole. The longitudes start with 0.0 at the Greenwich meridian and increase to the east, so that 90.0 refers to 90.0E, 180.0 is the International Dateline and 270.0 is 90.0W. **Calling Sequence:** coamps_ll2ij(igrid,reflat,reflon,iref,jref,stdlt1stdlt2,stdlon,

Subroutine	Description
	delx,dely,grdi,grdj,npts,grdlat,grdlon) **Data Declaration:** Integer igrid,iref,jref,npts Real delx,dely,grdi,grdj,grdlat,grdlon,reflat,reflon, stdlon,stdlt1,stdlt2
Coamps_rdata	Subroutine COAMPS_RDATA inputs data from either DBMS or a user-selected directory. **Calling Sequence:** coamps_rdata(istdo ,din,len,lvlcnt,parmnm,units,lvltyp, lvlval,cdtg10,itime,cfluid,inest,dsetnm,geomnm,mdltyp,ldbs, dsetux lwritu,errary,istats,isub,idbms,outtyp,m,n) **Data Declaration:** Integer istdo,len,lvlcnt,itime,inest,errary,istats, idbms,m,n Real din,lvlval Logical ldbms,lwritu Character paramnm,units,lvltyp,cdtg10,cfluid, dsetnm,geomnm,mdltyp, dsetux,isub,outtyp
Coamps_rotang	Subroutine COAMPS_ROTANG determines the rotation angle for wind vectors when converting from a COAMPS Lambert conformal or polar stereographic grid-relative projection to earth-relative (true) coordinates. **Calling Sequence:** coamps_rotang(grdlat,grdlon,m,n,grdrot) **Data Declaration:** Integer m,n Real grdlat,grdlon,grdrot
Coamps_s2hms	Subroutine COAMPS_S2HMS converts from seconds to hours, minutes and seconds. **Calling Sequence:** coamps_s2hms(itime,ihour,minute,isec) **Data Declaration:** Integer ihour,isec,itime,minute
Coamps_slen	Subroutine COAMPS_SLEN gets the size of the character string. **Calling Sequence:** coamps_slen (cstr,lenc) **Data Declaration:** Integer lenc Character cstr
Coamps_uvg2uv	Subroutine COAMPS_UVG2UV converts grid u and v to real u and v, assuming grid *u*= real u and grid *v* = real v along the standard longitude and *rot* is the rotation array. **Calling Sequence:** coamps_uvg2uv (u, v, m, n, rot,utru,vtru) **Data Declaration:** Integer m,n Real u,v,rot,utru,vtru
Coamps_wdata	**Calling Sequence:** coamps_wdata (dout,len,lvlcnt,parmnm,units,lvltyp,lvlval, dtg10,itime,cfluid,inest,dsetnm,geomnm,mdltyp,ldbms,dsetu x,lwritu,errary,istats,isub,idbms,outtyp,m,n) **Data Declaration:** Integer len,lvlcnt,itime,inest,errary,istats,idbms,m,n Character parmnm,units,lvltyp,cdtg10,cfluid, dsetnm, geomnm, mdltyp,dsetux,outtyp Real dout,lvlval Logical ldbms,lwritu

Subroutine	Description
Dataw	Subroutine DATARW writes flat file fields for COAMPS. **Calling Sequence:** coamps_dataw(d,lend,fldnam,inest,itime,cdtg,cfluid,lvltyp, rlev1,rlev2, dsetux,lsetux,lwritu,istat) **Data Declaration:** Integer istdo,lend,inest,itime,lsetux,istat Real d,rlev1,rlev2 Logical lwritu Character cdtg,cfluid,fldnam,lvltyp,dsetux
Dataw_new	Subroutine DATARW_NEW writes flat file fields for COAMPS. **Calling Sequence:** coamps_dataw_new(d,lend,fldnam,inest,itime,cdtg,cfluid, lvltyp, rlev1,rlev2, dsetux,lsetux,lwritu,istat,outtyp,m,n)) **Data Declaration:** Integer istdo,lend,inest,itime,lsetux,istat,m,n Real d,rlev1,rlev2 Logical lwritu Character cdtg,cfluid,fldnam,lvltyp,dsetux,outtyp
Dfalts	Subroutine DFALTS returns the default contour interval and maximum and minimum values for color shading bar. It uses the old FNMOC standard field name and units. **Calling Sequence:** coamps_dfalts (parmnm,units,lvltyp,rlvl1,rlvl2,ci,co,dmax, dmin, cunix,istats,cunix_new) **Data Declaration:** Integer istats Real ci,co,dmax,dmin,rlvl1,rlvl2 Character units,parmrm,lvltyp,cunix,cunix_new

5.7 ESMF Related Subroutines (libsrc/ esmf/)

Subroutine	Description
Load_Export	**Calling Sequence:** Load_Export(n, m, t, s, flxp) **Data Declaration:** Integer n,m Type flxp Real t,s
Load_Import	Subroutine LOAD_IMPORT loads ESMF atmospheric surface fluxes into appropriate ocean model arrays. Units and directions of fluxes are assumed to be already set appropriately by the coupler. Data pointers for import data must already be set. **Calling Sequence:** Load_Import(nest,n,m,nr, times, flxp,iat1,iat2,patm2,usflx2, vsflx2,rsflx2,solar2,tmatm2) **Data Declaration:** Integer nest,n,m,nr,iat1,iat2 Real times,patm2usflx2,rsflx2,solar2,tmatm2 Type flxp
NCOM_ESMF_Final	**Calling Sequence:** NCOM_ESMF_Final(gridComp, impState, expState, extClock, rc) **Data Declaration:** Integer rc Type gridComp, impState,expState,extClock
NCOM_ESMF_Init	**Calling Sequence:** NCOM_ESMF_Init(gridComp, impState, expState,

Subroutine	Description
	extClock, rc) **Data Declaration:** Integer rc Type gridComp,impState,expState,extClock
NCOM_ESMF_Run	**Calling Sequence:** NCOM_ESMF_Run(gridComp, impState, expState, extClock, rc) **Data Declaration:** Integer rc Type gridComp, impState,expState,extClock
NCOM_SetServices	**Calling Sequence:** NCOM_SetServices(gridComp, rc) **Data Declaration:** Integer rc Type gridComp
Setup_ESMF	**Calling Sequence:** Setup_ESMF(nest, nt, mt, n, m,elon, alat, ang, dx, dy, amsk,t, s,gridComp, impState, expState, extClock, rc) **Data Declaration:** Integer nest,nt,mt,n,m,rc Real elon,alat,ang,dx,dy,amsk,t,s Type gridComp, impState,expState,extClock

5.8 Primary FNMOC Subroutines (libsrc/ fnoclib/)

The following routines were written by FLENUMOCEANCEN (c) 1993 (FNMOC). Property of the US Government. All rights reserved.

Subroutine	Description
Bessel	Subroutine BESSEL is a general purpose 2D bessel interpolation. **Calling Sequence:** bessel(xi,xj,array,m,n,result,ierror) **Data Declaration:** Integer m,n,ierror Real xi,xj,array,result
Cctopc	Subroutine CCTOPC converts a pair of fields containing vector components from u and v (Cartesian) form to direction (DD) and magnitude (MM) (Polar) form. This routine is vectorizable. Direction is measured clockwise from the positive y-axis and uses the “direction toward” convention. U is the component along the positive x-axis and v is the component along the positive y-axis. **Calling Sequence:** cctopc (fuu, fvv, n, cunits, iflag, fval, fdd, fmm) **Data Declaration:** Integer n,iflag Real fuu,fvv,fval,fdd,fmm Character cunits
Ch2int	Subroutine CH2INT gets the integer number value from an integer string. Leading and trailing white space characters are insignificant (blanks,tabs, lf, cr, nul). **Calling Sequence:** ch2int(str,int,ierr) **Data Declaration:** Integer int,ierr Character str
Dfuv	Subroutine DFUV converts vectors from earth-oriented direction and magnitude to u and v component form on a conic projection. Argument *fdd* is in degrees clockwise

Subroutine	Description
	from the positive y-axis using the 'direction toward' convention. This routine is vectorizable. A transverse projection is one where the pole may not be the geographic pole. **Calling Sequence:** dfuv (fdd, fff, fdx, fdy, n, iflag, fval, fuu, fvv) **Data Declaration:** Integer n,iflag,fval Real fdd,fff,fdx,fdy,fuu,fvv
Differs	Subroutine DIFFERS performs operations on field *fldi1*, depending on the mode specified, *fldi2*. The output is written to *fldo*. An additional mode computes only the mean and standard deviation of a single input field. **Calling Sequence:** differs (fldi1, fldi2, mode, len,mdif, rmsd, fldo ,istat) **Data Declaration:** Integer len,mode,istat Real fldi1,fldi2,fldo,mdif,rmsd
FNOC_dtgdif	Given two DTGs, this subroutine returns the difference in hours (=*mdtg-ndtg*). It handles DTGs in the range 1800 through 2799. **Calling Sequence:** fnoc_dtgdif (ndtg,mdtg,ihrs,istat) **Data Declaration:** Integer ihrs,istat Character mdtg,ndtg
FNOC_dtgmod	Given base DTG and increment (+/- hours), FNOC_DTGMOD returns new DTG (= indtg + idif) and the status value. **Calling Sequence:** fnoc_dtgmod (indtg, idif, newdtg, istat) **Data Declaration:** Integer indtg,idif Character indtg,newdtg
FNOC_dtgyrhr	Given a year and hours of the year, FNOC_DTGYRHR returns a DTG of format YYYYMMDDHH in *newdtg*. **Calling Sequence:** fnoc_dtgyrhr (iyr,ihrs,newdtg,istat) **Data Declaration:** Integer iyr,ihrs,istat Character newdtg
FNOC_dtgnum	Given a DTG (YYYYMMDDHH), FNOC_DTGNUM returns integer values for year, month, day, hour, days into the year, and hours into the year. **Calling Sequence:** fnoc_dtgnum (indtg, iyr,imo,iday,ihour,iyrday,iyrhrs, istat) **Data Declaration:** Integer iyr,imo,iday,ihour,iyrday,iyrhrs,istat Character indtg
Dtgops	Subroutine DTGOPS returns the date-time group (YYYYMMDDHH), which is one of the following: 1) Current operational DTG (NOT YET IMPLEMENTED). 2) + or - offset to current operational DTG. 3) User supplied DTG. **Calling Sequence:** dtgops (cdtg, istat) **Data Declaration:** Integer istat Character cdtg
Edge	This routine performs the next-to-edge processing for a low-pass filter. This routine is vectorizable. **Calling Sequence:** edge (fld, fldwrk, m, n, iedge, jedge, nedge)

Subroutine	Description
	Data Declaration: Integer m,n,nedge,iedge,jedge Real fld,fldwrk
Fintrp	Subroutine FINTRP interpolates values from an input field at a set of x/y coordinates given by two other fields. The input field may be flagged as having missing points or may be continuous. This routine is vectorizable. **Calling Sequence:** fintrp (fx, fy, iflen, fldi, mwrk, min,nin, iflagi, fvali, fvalo, filval, fldo) **Data Declaration:** Integer iflen,min,mwrk,nin,iflagi,ll Real fvali,fvalo,fx,fy,filval,fldi,fldo
Gcpnts	Subroutine GCPNTS computes evenly-spaced latitude/longitude points along a great circle. This routine is scalar. **Calling Sequence:** gcpnts (mo,xla,xlo,dist,istat) **Data Declaration:** Integer mo,istat Real dist,xla,xlo
Gent	Subroutine GENT gets a single entry from a HRLS table. An entry consists of two X values, a start coordinate and a stop coordinate. **Calling Sequence:** gent(tab,y,xseq,x) **Data Declaration:** Integer tab,y,xseq,x
Getls	Subroutine GETLS reads a HRLS table from either an ISIS or a UNIX file. **Calling Sequence:** getls(type,min_res,tab,alen,pathnm,istat) **Data Declaration:** Integer tab,alen,istat Character type,pathnm Real min_res
Int2ch	Subroutine INT2CH converts an integer to a left justified character string. **Calling Sequence:** int2ch(int,chr,ierr) **Data Declaration:** Integer int,ierr Character chr
Ioinq	Subroutine IOINQ employs the Fortran statement "Inquiry" to supply information to a user in taking the action of the program I/O. **Calling Sequence:** ioinq (unitx,locprog,nu) **Data Declaration:** Integer unitx,nu Character loccprog
Lndavg	Subroutine LNDAVG computes values for flagged points in a 2D field. This routine is vectorizable. **Calling Sequence:** lndavg(fld, mwrk, m, n, lasrch, val, lapass, jpnts, istat) **Data Declaration:** Integer mwrk,m,n,lasrch,lapass,jpnts,istat Real fld,val
Lpf	LPF performs a low-pass two-dimensional filter. This routine is vectorizable. **Calling Sequence:** lpf (fld, fldwrk, m, n, mn, ifn, fvalo) **Data Declaration:** Integer m,n,mn,ifn Real fld,fldwrk,fvalo
Niddf	Given: • a 1D array, vi(4), containing values of an independent variable at 4 points,

Subroutine	Description
	• a corresponding array, vd(4), containing values of a dependent variable at the same 4 points, and • a value, *val*, of the independent variable such that (*vd(2) < val <= vd(3)*) or (*vd(3) < val <= vd(2)*), compute the value of vd, vdo, given the independent variable = val. **Calling Sequence:** niddf(vi,vd,val,vdo) **Data Declaration:** Real vd,val,vi,vdo
Ocord	This subroutine reads "MODELNAME_dir.out" flatfiles and fields in accordance with OCARD records. **Calling Sequence:** ocord (lu,actau,ngeom,acogeom,acdset,aclvlt,aclvl,acparm, acnfil, acfilt,nspace,spaces,istat) **Data Declaration:** Integer lu,actau,ngeom,acnfil,nspace,istat Character acogeom,acdset,aclvlt,acparm,acfilt,spaces Real aclvl
Pctocc	This routine converts a pair of fields containing vector components from direction and magnitude (polar) to u and v (Cartesian) form. This routine is vectorizable. Note that direction is measured clockwise from the positive y axis from 0 to 360 degrees or radians using the using the 'direction toward' convention. *U* is the component along the positive x axis and *v* is the component along the positive y axis. **Calling Sequence:** pctocc (fdd, fmm, n, cunits, iflag, fval, fuu, fvv) **Data Declaration:** Integer n,iflag Character cunits Real fdd,fmm,fval,fuu,fvv
Qprint	This routine quick prints portions of a gridded field. **Calling Sequence:** qprint (fld, lbl, mmin, nmin, kmin, mmax, nmax, kmax,minc, ninc, kinc, m, n, k, ndig, scale, stordsc, pcknull, iunit, istat) **Data Declaration:** Integer m,n,k,mmin,nmin,kmin,minc,ninc,kinc, ndig, mmax,nmax, kmax,istat,iunit Character lbl,stordsc Real fld,scale,pcknull
Rlpnts	This routine computes evenly-spaced X/Y grid coordinate points along a straight line on the grid. This routine is scalar. **Calling Sequence:** rlpnts (mo, x, y, istat) **Data Declaration:** Integer mo,istat Real x,y
Strleft	Deletes leading white space (spaces, tabs, carriage returns and line feeds) from a character string, therefore left-justifying the string. **Calling Sequence:** strleft(cstr1, cstr2) **Data Declaration:** Character cstr1,cstr2
Strpars	Extracts substrings from a character string, where the delimiter separating the substrings is defined by the calling routine. Leading spaces are removed from the substrings. **Calling Sequence:** strpars(cline, cdelim, nstr, cstr, nsto, ierr)

Subroutine	Description
	Data Declaration: Character cline,cstr,cdelim Integer nstr,nsto,ierr
Unstgr	This routine unstaggers a staggered gridded field. It is vectorized. **Calling Sequence:** unstgr (fld, mwrk, m, n, istg, iflag, fval) **Data Declaration:** Real fld,fval Integer mwrk,m,n,istg,iflag
Uvdf	Subroutine UVDF converts from u and v vector components on a conic projection to earth-oriented direction and speed form. Direction is measured clockwise from the positive y axis in degrees in the range 0 < fdd < 360, using the 'direction toward' convention. This routine is vectorizable. A transverse projection is one where the 'pole' is not the geographic pole. **Calling Sequence:** uvdf (fuu, fvv, fdx, fdy, n, iflag, fval, fdd, fff) **Data Declaration:** Real fuu,fvv,fdx,fdy,fdd,fff Integer n,iflag,fval

5.9 Miscellaneous NCOM Subroutines (libsrc/ misc/)

5.9.1 Cubic Spline Interpolation Subroutines (cubspl_irr and ocubspl_irr))

Subroutine	Description
Coeff1	Subroutine COEFF1 computes the coefficients for 1D cubic spline interpolation using one of the following boundary conditions at each end of the range: - Second derivative given at boundary. - First derivative given at boundary. - Periodic boundary condition. - First derivative determined by fitting a cubic to the four points nearest to the boundary. **Calling Sequence:** coeff1 (n, x, f, w, iop, int, wk) **Data Declaration:** Integer n, iop, int Real x, f, w, wk
Coeff2	Subroutine COEFF2 computes the coefficients for 2D bicubic spline interpolation with the same choice of boundary conditions as for COEFF1. **Calling Sequence:** coeff2 (nx, x, ny, y, f, fxx, fyy, fxxyy, idm, ibd, wk) **Data Declaration:** Integer nx, ny, idm, ibd Real x, y, f, fxx, fyy, fxxyy, wk
Cubspl_irr	CUBSPL has been modified to accept an irregular output grid. Subroutine CUBSPL_IRR interpolates from the array fldi to the array fld, where fld (i, j) is at coordinates (fx (i, j); fy (i, j)) with respect to the fldi grid (1:nxi, 1:nyi). Cubic spline interpolation is used. The input grid fldi is assumed to be globally uniform. No assumptions are made regarding the output grid regularity. For compatibility with subroutine BESSEL, it is assumed that fx (i, j) lies between 3 and nxi-2 and that fy (i, j) lies between 3 and byi-2. **Calling Sequence:** cubspl_irr (fld, fx, fy, ndx, nx, ny, fldi, ndxi, nxi, nyi, ibd, fxi, fyi, wki,wk)

Subroutine	Description
	Data Declaration: Integer ndx, nx, ny, ndxi, nxi, nyi, ibd Real fld, fx, fy, fldi, fxi, fyi, wki, wk
Interp	Given coefficients provided by COEFF1 and the position of the interpolation point in the independent variable table, subroutine INTERP performs 1D interpolation for the function value, and first and second derivative, as desired. This routine is called by subroutines TERP1 and TERP2. **Calling Sequence:** interp (n, x, f, w, y, i, int, tab, itab) **Data Declaration:** Integer n, i, int, itab Real x, f, w, y, tab
Search	Subroutine SEARCH performs a binary search in a 1D floating point table arranged in ascending order. This routine is called by subroutines TERP1 and TERP2. **Calling Sequence:** search (xbar, x, n, i) **Data Declaration:** Integer n, i Real xbar, x
Terp1	Using the coefficients computed by COEFF1, subroutine TERP1 evaluates the function and/or first and second derivatives at any point where interpolation is required. **Calling Sequence:** terp1 (n, x, f, w, y, int, tab, itab) **Data Declaration:** Integer n, int, itab Real x, f, w, y, tab
Trip	This is a simple, periodic, tridiagonal linear equation solver used by COEFF1 and used to locate entries in array z. **Calling Sequence**: trip (n,a,b,c,y,z,int) **Data Declaration:** Integer n, int Real a, b, c, y, z

5.9.2 Time Conversion Subroutines (timesubs)

Subroutine	Description
Da2jd	Subroutine to calculate an integer Julian day, hour, minute, second and hundredth of a second from a real Julian-type date. Precision problems may cause inaccuracies in the finer time divisions. **Calling Sequence:** da2jd (date, jday, ihour, imin, isec, ihsec) **Data Declaration:** Integer jday, ihour, imin, isec, ihsec Real date
Da2jd1	Subroutine DA2JD1 calculates an integer Julian day from a real Julian-type date. It has integer 1/100 second precision, or full integer precision for coarser time applications. **Calling Sequence:** da2jd1 (date, jday) **Data Declaration:** Integer jday Real date
Dait	Subroutine DAIT calculates a Julian-type date from the year, month, day, hour, and minute. The date is defined as (Julian day - 1) with the hour and minute expressed as

Subroutine	Description
	a fractional part of a day. For example, 00z January 1 is 0.000 and 06z January 14 is 13.250. It has integer second precision. **Calling Sequence:** dait (iyear, month, iday, ihour, imin, isec, date) **Data Declaration:** Integer iyear, iday, ihour, imin, isec, month Real date
Daiti	Subroutine DAITI converts a year and a Julian-type date to month, day, hour, minute, and second. The arguments are defined as dait. Precision problems may cause inaccuracies in the finer time divisions. It has integer second precision. **Calling Sequence:** daiti (iyear, date, month, iday, ihour, imin, isec) **Data Declaration:** Integer iyear, iday, ihour, imin, isec, month Real date
Daywek	Subroutine DAYWEK calculates the day of the week from the year, month, and day. It has integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** daywek (iyear, mon, iday, idow) **Data Declaration:** Integer iyear, iday, idow Real mon
Ddtg	Subroutine DDTG converts a time defined by the year, month, day, hour, minute, and second to a date-time-group. It has integer second precision. **Calling Sequence:** ddtg (iyear, month, iday, ihour, imin, isec, idtg) **Data Declaration:** Integer iyear, iday, ihour, imin, isec, idtg, month
Df2jd	Subroutine DF2JD calculates an integer Julian day, hour, minute, second and hundredth of a second from a real Julian-type date. It was created to reduce roundoff error. It has integer 1/100 second precision, or full integer precision for coarser time applications. **Calling Sequence:** df2jd (idaft, idayfr, iyear, jday, ihour, imin, isec, ihsec) **Data Declaration:** Integer idaft, idayfr, iyear, jday, ihour, imin, isec, ihsec
Df62jd	Subroutine DF62JD calculates an integer Julian day, hour, minute, second and dummy hundredth of a second from a real Julian-type date. It was created to reduce roundoff error. It has integer second precision. **Calling Sequence:** df62jd (idaft, idayfr, iyear, jday, ihour, imin, isec, ihsec) **Data Declaration:** Integer idaft, idayfr, iyear, jday, ihour, imin, isec, ihsec
Dtgadd	Subroutine DTGADD adds (or subtracts) a number of hours from a date-time group. It has integer hour precision. **Calling Sequence:** dtgadd (idtg1, ihrs, idtg2) **Data Declaration:** Integer idtg1, ihrs, idtg2
Dtgd	Subroutine DTGD converts a date-time group to year, month, day, hour, minute and second. It has integer second precision. **Calling Sequence:** dtgd (idtg, iyear, month, iday, ihour, imin, isec) **Data Declaration:** Integer idtg, iyear, iday, ihour, imin, isec, month
Dtgdif	Subroutine DTGDIF calculates the time difference in hours between two date-time groups (idtg2 - idtg1). The minutes and seconds are discarded. Integer hour precision.

Subroutine	Description
	Calling Sequence: dtgdif (idtg1, idtg2, ihrdif) **Data Declaration:** Integer idtg1, idtg2, ihrdif
Dtghc	Subroutine DTGHC converts a date-time-group to hour of the 20^{th} century. Integer hour precision. **Calling Sequence:** dtghc (idtg, ihrcen) **Data Declaration:** Integer idtg, ihrcen
Dtghcr	Subroutine DTGHCR converts a date-time group and minute to hour of the 20^{th} century. Integer second precision. **Calling Sequence:** dtghcr (idtg, hrcen) **Data Declaration:** Integer idtg Real hrcen
Dtgjd	Subroutine DTGJD converts a date-time group to year and Julian-type date. Integer second precision. **Calling Sequence:** dtgjd (idtg, iyear, date) **Data Declaration:** Integer idtg, iyear Real date
Dtglab	Subroutine DTGLAB converts date-time group to a date label, e.g., 19770824, 120000 becomes "12:00:00 GMT May 24, 1977". Integer second precision. **Calling Sequence:** dtglab (idtg, label) **Data Declaration:** Integer idtg Character label
Dtglab2	Subroutine DTGLAB2 converts date-time group to a date label, e.g., 19770824, 120000 becomes "12:00:00 GMT May 24, 1977". Integer second precision. **Calling Sequence:** dtglab2 (idtg, label) **Data Declaration:** Integer idtg Character label
Dtgr2dif	Subroutine DTGR2DIF calculates the time difference in hours between two date-time groups (idtg2 - idtg1). The minutes and seconds are discarded. It has integer second precision. **Calling Sequence:** dtgr2dif (idtg1y, idtg1h, idtg2y, idtg2h, ihrdif) **Data Declaration:** Integer idtg1y, idtg1h, idtg2y, idtg2h, ihrdif
Dtgr2sdif	Subroutine DTGR2SDIF calculates the time difference in integer seconds between two date-time groups (idtg2 - idtg1). It has integer second precision. **Calling Sequence:** dtgr2sdif (idtg1a, idtg1b, idtg2a, idtg2b, isecdif) **Data Declaration:** Integer idtg1a, idtg1b, idtg2a, idtg2b, isecdif
Dtgr3dif	Subroutine DTGR3DIF calculates the time difference in hours between two date-time groups (idtg2 - idtg1). The minutes and seconds are discarded. It has integer second precision. **Calling Sequence:** dtgr3dif (idtg1y, idtg1h, idtg2y, idtg2h, ihrdif) **Data Declaration:** Integer idtg1y, idtg1h, idtg2y, idtg2h, ihrdif
Dtgradd	Subroutine DTGRADD adds (or subtracts) a number of hours from a date-time group. It has integer second precision. **Calling Sequence:** dtgradd (idtg1, hrs, idtg2)

Subroutine	Description
	Data Declaration: Integer idtg1, idtg2 Real hrs
Dtgradds	Subroutine DTGRADDS adds (or subtracts) a number of seconds from a date-time group. Integer second precision. **Calling Sequence:** dtgradds (idtg1, isecadd, idtg2) **Data Declaration:** Integer idtg1, idtg2 Real isecadd
Dtgrdif	Subroutine DTGRDIF calculates the time difference in hours between two date-time groups (idtg2 - idtg1). The minutes and seconds are discarded. It has integer second precision. **Calling Sequence:** dtgrdif (idtg1, idtg2, ihrdif) **Data Declaration:** Integer idtg1, idtg2, ihrdif
Dtgrsdif	Subroutine DTGRSDIF calculates the time difference in integer seconds between two date-time groups (idtg2 - idtg1). It has integer second precision. **Calling Sequence:** dtgrsdif (idtg1, idtg2, isecdif) **Data Declaration:** Integer idtg1, idtg2, isecdif
Dtgrstdif	Subroutine DTGRSTDIF calculates the time difference in integer seconds between two date-time groups (idtg2 - idtg1). If the absolute difference is greater than itol, then isecdif is returned as zero and itol as -1. Itol must be non-negative. It has integer second precision. **Calling Sequence:** dtgrstdif (idtg1, idtg2, isecdif, itol) **Data Declaration:** Integer idtg1, idtg2, isecdif, itol
Hcdtg	Subroutine HCDTG converts the hour of the 20th century to a date-time group. The minutes and seconds are set to zero. It has integer hour precision. **Calling Sequence:** hcdtg (ihrcen, idtg) **Data Declaration:** Integer ihrcen, idtg
Hcrdtg	Subroutine HCRDTG converts the hour of the 20th century to date-time group. It has integer second precision. **Calling Sequence:** hcrdtg (hrcen, idtg) **Data Declaration:** Integer idtg Real hrcen
Hrcen	Subroutine HRCEN calculates the hour of the 20th century from the year, month, day, and hour. It has integer hour precision. **Calling Sequence:** hrcen (iyear, month, iday, ihour, ihrcen) **Data Declaration:** Integer iyear, iday, ihour, ihrcen, month
Hrceni	Subroutine HRCENI calculates the year, month, day, and hour from the hour of the 20th century. It has integer hour precision. **Calling Sequence:** hrceni (ihrcen, iyear, month, iday, ihour) **Data Declaration:** Integer ihrcen, iyear, iday, ihour, month
Hrcenr	Subroutine HRCENR calculates the hour of the 20th century from the year, month, day, hour, minute, and second. It has integer second precision. **Calling Sequence:** hrcenr (iyear, month, iday, ihour, imin, isec, hrcen) **Data Declaration:** Integer iyear, iday, ihour, imin, isec, month

Subroutine	Description
	Real hrcen
Hrcnri	Subroutine HRCNRI calculates the year, month, day, hour and minute from the hour of the 20th century. There is integer hour precision. • 400*365+4*24+1= 146097 days in each 400 year-period. • 100*365+24+1= 36525 days in each 100 year-period if first 00 year is evenly divisible by 400, 36524 days otherwise. • 20*365+4= 7304 days in each 20-year period if it contains a 00 year not evenly divisible by 400, 7305 otherwise. **Calling Sequence:** hrcnri (hrcen, iyear, month, iday, ihour, imin, isec) **Data Declaration:** Integer iyear, iday, ihour, imin, isec, month Real hrcen
Id2jd	Subroutine ID2JD calculates an integer Julian day from an integer year, month, and day. It has integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** id2jd (jday, iyear, month, iday) **Data Declaration:** Integer jday, iyear, iday, month
Jd2da	Subroutine JD2DA calculates a real Julian-type date from integer Julian day, ihour, minute, second, hundredth of a second. Precision problems may cause inaccuracies in the finer time divisions. Integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** jd2da (date, jday, ihour, imin, isec, ihsec) **Data Declaration:** Integer jday, ihour, imin, isec, ihsec Real date
Jd2da1	Subroutine JD2DA1 calculates a real Julian-type date from an integer Julian day. It has integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** da2jd (date, jday) **Data Declaration:** Integer jday Real date
Jd2df	Subroutine JD2DF calculates a real Julian-type date from integer Julian day, ihour, minute, second, and hundredth of a second. It has integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** jd2df (idaft, idayfr, jday, ihour, imin, isec, ihsec) **Data Declaration:** Integer idaft, idayfr, jday, ihour, imin, isec, ihsec
Jd2id	Subroutine JD2ID calculates an integer month and day from an integer Julian day and year. There is integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** jd2id (jday, iyear, month, iday) **Data Declaration:** Integer jday, iyear, iday, month
Jddtg	Subroutine JDDTG converts year and Julian-type date to date-time group and minute. This conversion is not exact, because the seconds are dropped, not rounded to nearest minute. There is integer second precision. **Calling Sequence:** jddtg (iyear, date, idtg, imin, isec)

Subroutine	Description
	Data Declaration: Integer iyear, idtg, imin, isec Real date
Loctime	Subroutine LOCTIME calculates local time of day, given longitude and time at Greenwich (GMT) in days. Integer 1/100 second precision or full integer precision for coarser time applications. **Calling Sequence:** loctime (elong, timegmt, timeloc) **Data Declaration:** Real elong, timegmt, timeloc
Oddtg	Subroutine ODDTG converts a time defined by the year, month, day, hour, minute, and second to a date-time group. **Calling Sequence:** oddtg (iyear, month, iday, ihour, imin, isec, idtg) **Data Declaration:** Integer iyear, iday, ihour, imin, isec, idtg, month
Odtgd	Subroutine ODTGD converts a date-time group to year, month, day, hour, minute and second. **Calling Sequence:** odtgd (idtg, iyear, month, iday, ihour, imin, isec) **Data Declaration:** Integer idtg, iyear, iday, ihour, imin, isec, month
Odtghc	Subroutine ODTGHC converts a date-time group to the hour of the 20th century. **Calling Sequence:** odtghc (idtg, ihrcen) **Data Declaration:** Integer idtg, ihrcen

5.9.3 File Conversion Subroutines (w_ncomnc/ w_ncomnc2)

Subroutine	Description
W_ncomnc/2	Subroutine W_NCOMNC writes NCOM data into a netCDF file. **Calling Sequence:** w_ncomnc (inde, indv, indt, inds, indl, indz, indh, inda, nest, nmax, mmax, lmax, n, m, ll, e, u, v, t, s, wk, timed, run, elon, alat, elonu, alatu, elonv, alatv, dx, dy, h, ang, depth, zm3, idtg, ldefattr, icoordsys, ivcoordsys, outfilnam, axlab, axunit, axfmt, ntypes, dlab, dunit, dfmt, max1d, maxattr, maxname, maxannot, scalee, scalet, scaleu, scalev, rattr, iattr, iattrnam, rattrnam, cattrnam, cattr) **Data Declaration:** Integer inde, indv, indt, inds, indl, indz, indh, inda, nest,nmax, mmax, lmax, n, m, ll, idtg, ldefattr, icoordsys, ivcoordsys, ntypes, max1d, maxattr, maxname, maxannot, iattr, iattrnam Real e, u, v, t, s, wk, timed, elon, alat, elonu, alatu, elonv,alatv, dx, dy, h, ang, depth, zm3, outfilnam, axlab, axunit, axfmt, dlab, dunit, dfmt, scalee, scalet, scaleu, scalev, rattr, rattrnam, cattrnam, cattr Character run

5.9.4 Unit Conversion Subroutines (gc_ellipsoid)

Subroutine	Description
Gc_ellipsoid	Subroutine GC_ELLIPSOID returns distances in m and the azimuth angle in degrees. **Calling Sequence:** subroutine gc_ellipsoid(latd1,latm1,lats1,lond1,lonm1,lons1, latd2,latm2,lats2,lond2,lonm2,lons2,dist,azimuth) **Data Declaration:** Real latd1,latm1,lats1,lond1,lonm1,lons1, latd2,latm2,lats2,lond2,lonm2,lons2, dist,azimuth
Inver1	INVER1 is a solution of the geodetic inverse problem after T. Vincenty modified Rainsford's method with Helmert's elliptical terms effective in any azimuth and at any distance short of antipodal (Vincenty, 1975). Standpoint/forepoint must not be the geographic pole. Variable *a* is the semi-major axis of the reference ellipsoid. The variable f is the flattening (not reciprocal) of the reference ellipsoid. Latitudes and longitudes in radians positive north and east forward azimuths at both points are returned in radians from north. **Calling Sequence:** inver1(glat1,glon1,glat2,glon2,faz,baz,s,a,f,pi,rad) **Data Declaration:** Real glat1, glon1, glat2, glon2, fax, baz, s, a, f, pi,rad
Getrad	Subroutine GETRAD converts deg, min, and sec to radians. **Calling Sequence:** getrad(d,m,s,isign,val,pi,rad) **Data Declaration:** Integer isign Real d, m, s, val, pi, rad
Todmsp	Subroutine TODMSP converts position radians to deg,min,and sec. **Calling Sequence:** todmsp(val,id,im,s,isign,pi,rad) **Data Declaration:** Integer isign, id, im Real s, val, pi, rad

5.9.5 Array Allocation Subroutines (allocate)

Subroutine	Description
Allocate	Subroutine ALLOCATE allocates the number of array elements needed, via pointer variables on the SUNs. This is a hardware dependent routine. **Calling Sequence:** allocate (ipoint,isize) **Data Declaration:** Integer ipoint, isize **Routines called:** malloc

5.9.6 Array Conversion Subroutines (w_rgb)

Subroutine	Description
W_rgb	Subroutine W_RGB converts a real valued array f to an output rgb file in SGI format. Array values f are scaled to the range icolormin to icolormax as fs = am*(f+ad). Values of fs lower than icolormin or higher than icolormax are truncated to these limits. Masked values are returned as 0 (land). It is recommended that 1 is reserved for text/symbols (default black). It is recommended that icolormax+1 is reserved for special text/symbols (default white). When computing a sequence of images, e.g., for an animation, do not change the grid, i.e., the dimensions or the mask, since setup calculations for images will not be changed when num > 1. Equivalent to w_rgb with

Subroutine	Description
	minimum value 1 (zero reserved for land). **Calling Sequence:** w_rgb(ni,n,m,f,amsk,neg,am,ad,sx,sy,num,filnam, iflip, icolormin,icolormax,ncpal,irpal,igpal,ibpal) **Data Declaration:** Integer ni, n, m, neg, num, iflip, ncpal, irpal, igpal, ibpal, icolormin, icolormax Real am, amsk, ad, sx, sy Logical filnam **Common Blocks:** Common/rgbheader/

5.9.7 Table Lookup Subroutines (tablk2s)

Subroutine	Description
Tablk2s	Subroutine TABLK2S interpolates a value from a 2D array f using linear interpolation (i.e. table lookup). F varies with both x and y and the spacing of the values of *f* along the x and y axes is assumed to be constant. **Calling Sequence:** tablk2s(ni,n,m,xa,xb,ya,yb,f,x2,y2,f2,indext,spval) **Data Declaration:** Integer n, ni, m, indext Real spval, xa, xb, ya, yb, f, x2, y2, f2

5.9.8 Horizontal Grid Embedding Subroutine (padarr)

Subroutine	Description
Padarr	This is a subroutine to embed the model horizontal grid into the computational horizontal grid. The model grid is positioned at the 1,1 entry of the comp_array. **Calling Sequence:** padarr(n,m,nibo,mibo,mod_array,comp_array,padval) **Data Declaration:** Integer n, m, nibo, mibo Real mod_array, comp_array, padval

5.10 Dummy Computer-Specific Subroutines (libsrc/ none/)

Subroutine	Description
Nonsuch	Subroutine NONSUCH is a single dummy subroutine that is never invoked. It is used to simplify Makefile logic.

5.11 Dummy NCOM Plotting Subroutines (libsrc/ pdum/)

5.11.1 Plotting Subroutines (ncom1pdum)

File **ncom1pdum** contains dummy plotting routines for NCOM when interactive NCAR graphics are not available.

Subroutine	Description
Paxscal	Subroutine PAXSCAL finds axis limits for plotting values of a function f. **Calling Sequence:** paxscal (n, f, df, fmin, fmax, intf)

Subroutine	Description
	Data Declaration: Integer n, intf Real f, df, fmin, fmax
Pendpg	**Calling Sequence:** pendpg(ind) **Data Declaration:** Integer ind
Pltcon	Subroutine PLTCON creates contour plots using the NCAR routine CONREC. **Calling Sequence:** pltcon (ni, n, m, f, cmin, cmax, cint, xmin, xmax, ymin, ymax, intx, inty,title, lintit, xtit, ytit) **Data Declaration:** Integer ni, n, m, intx, inty, lintit Real f, cmin, cmax, cint, xmin, xmax, ymin, ymax Character title, xtit, ytit
Pltvec	Subroutine PLTVEC creates vector arrow plots. **Calling Sequence:** pltvec (ni, n, m, x, y, vscale, vecmin, vecmax, vecleg, legend, xmin, xmax,ymin, ymax, intx, inty, title, lintit, xtit, ytit) **Data Declaration:** Integer ni, n, m, intx, inty, lintit Real x, y, vscale, vecmin, vecmax, vecleg, smin, smax, ymin,ymax Character title, xtit, ytit, legend
Pltxy	Subroutine PLTXY creates x-y plots. **Calling Sequence:** pltxy (ni, n, m, x, y, xmin, xmax, ymin, ymax, intx, inty, title, lintit, xtit,ytit) **Data Declaration:** Integer ni, n, m, intx, inty, lintit Real x, y, xmin, xmax, ymin, ymax Character title, xtit, ytit
Pseloc	**Calling Sequence:** psetloc (xa, xb, ya, yb) **Data Declaration:** Real xa, xb, ya, yb
Psetax	**Calling Sequence:** psetax (nxtic, nytic, intax, nxdec, nydec, xofset) **Data Declaration:** Integer nxtic, nytic, intax, nxdec, nydec Real xofset
Psetid	**Calling Sequence:** psetid (plotid) **Data Declaration:** Character plotid
Psetlab	**Calling Sequence:** psetlab (siztid, sizled, siznud) **Data Declaration:** Real siztid, sizled, siznud
Psetspv	**Calling Sequence:** psetspv (indspv, spvalu) **Data Declaration:** Integer indspv Real spvalu
Psetvfr	**Calling Sequence:** psetvfr (ifreq, jfreq) **Data Declaration:** Integer ifreq, jfreq
Psymbl	**Calling Sequence:** psymbl(x,y,isym,size) **Data Declaration:** Integer isym Real x,y,ism,size
Xprnte	**Calling Sequence:** xprnte (fld, n, n1, n2, m1, m2, ncolum, length, ndec, title, amult, ad, iflip) **Data Declaration:** Integer n, n1, n2, m1, m2, ncolum, length, ndec, iflip

Subroutine	Description
	Real fld, amult, ad Character title

5.12 Communication Subroutines (libsrc/util/)

The folder /util/ contains files with Alan Wallcraft's message passing routines for shared memory (SM) and multi-processor (MP) computing.

5.12.1 Program xmc

Program XMC selects between programs XMC_MP and XMC_SM.

5.12.2 Communication Subroutines for Shared Memory Computer (xmc_sm)

File ***xmc_sm*** contains communication routines for a shared memory computer.

Subroutine	Description
IEEE_retrospective	Subroutine IEEE_RETROSPECTIVE is a dummy routine to turn off IEEE warning messages on a Sun system.
Xcaget	Subroutine XCAGET converts an entire 2D array from tiled to non-tiled layout. Variable mnflg selects which nodes must return the array: = 0 All nodes. = n Node number n (mnproc = n). **Calling Sequence:** xcaget (aa, na, ma, a, n, m, mnflg) **Data Declaration:** Integer na, ma, n, m, mnflg Real aa, a
Xcaput	Subroutine XCAPUT converts an entire 2D array from non-tiled to tiled layout. **Calling Sequence:** xcaput (aa, na, ma, a, n, m, mnflg) **Data Declaration:** Integer na, ma, n, m, mnflg Real aa, a
Xceget	Subroutine XCEGET finds the value of a(ia, ja) on the non-tiled 2D grid. **Calling Sequence:** xceget (aelem, a, n, m, ia, ja) **Data Declaration:** Integer n, m, ia, ja Real aelem, a
Xceput	Subroutine XCEPUT fills a single element in the non-tiled 2D grid. **Calling Sequence:** xceput (aelem, a, n, m, ia, ja) **Data Declaration:** Integer n, m, ia, ja Real aelem, a
Xchalt	Subroutine XCHALT stops all processes. Only one process needs to call this routine because it is for emergency stops. Use subroutine XCSTOP for ordinary stops called by all processes. **Calling Sequence:** xchalt (cerror) **Data Declaration:** Character cerror
Xciget	Subroutine XCIGET converts (ia, ja) on the non-tiled 2D grid to a local (i, j).

Subroutine	Description
	Calling Sequence: xciget (i, j, n, m, ia, ja) **Data Declaration:** Integer i, j, n, m, ia, ja
Xcigtg	Subroutine XCIGTG converts local (i,j) to global (ia,ja) on the non-tiled 2D grid. **Calling Sequence:** xcigtg(i,j, n,m, ia,ja) **Data Declaration:** Integer i, j, n, m, ia, ja
Xclg3d	Subroutine XCLG3D extracts a vertical slice of elements from the non-tiled 3D grid. **Calling Sequence:** xclg3d(aline,nl, a,n,m,l, i1,j1,ii,ji, mnflg) **Data Declaration:** Integer nl, n, m, l, i1, j1, ii, ji, mnflg Real aline, a
Xclget	Subroutine XCLGET extracts a line of elements from the non-tiled 2D grid. Variable aline(i) = a(i1 + i1*(i-1), j1+j1*(i-1)), for i = 1...nl. Variables ii and ji can each be -1, 0, or +1. Variable mnflg selects which nodes must return the line. = -1 Only nodes owning part of the line. = 0 All nodes. = n Node number n (mnproc = n). **Calling Sequence:** xclget (aline, nl, a, n, m, i1, j1, ii, ji, mnflg) **Data Declaration:** Integer nl, n, m, i1, j1, ii, ji, mnflg Real aline, a
Xclput	Subroutine XCLPUT fills a line of elements in the non-tiled 2D grid. Variable aline(i) = a(i1+i1*(i-1), j1+j1*(i-1)), for i = 1...nl. One of ii and ji must be zero, and the other must be one. **Calling Sequence:** xclput (aline, nl, a, n, m, i1, j1, ii, ji) **Data Declaration:** Integer nl, n, m, i1, j1, ii, ji Real aline, a
Xcmaxr	Subroutine XCMAXR replaces array ‘a’ with its element-wise maximum over all tiles. **Calling Sequence:** xcmaxr (a, n) **Data Declaration:** Integer n Real a
Xcprod	Subroutine XCPROD sums the product of two 2D arrays. Array n, m specifies the local dimensions of the array. The sum is bit for bit reproducible for the same iprsum and jprsum. **Calling Sequence:** xcprod (absum, a, b, n, m) **Data Declaration:** Integer n, m Real absum, a, b **Common Blocks:** PRSUMI
Xcrang	Subroutine XCRANG finds the minimum and/or maximum of part of a 3D array. Variables n and m specify the local 2D dimensions of the array, but n1, n2 and m1, m2 specify which part of the entire array to use. The third dimension is always completely used. Variables a and amask can be the same array. This is legal Fortran 77/Fortran 90 because both a and amask are unchanged on exit. **Calling Sequence:** xcrang (amin, amax, a, n, m, l, n1, n2, m1, m2, amask, itype,

Subroutine	Description
	spval) **Data Declaration:** Integer n, m, l, n1, n2, m1, m1, itype Real amin, amax, a, amask, spval
Xcspmd	Subroutine XCSPMD initializes /cproci/ by identifying the local processor. If jqr is less than ipr*jpr, then sea-less nodes are skipped. A map of which nodes to skip is input. Some node indices in idproc are null, and jdproc replaces the nulls with repeated indices from the same row. Variables jdproc(1,*) and jdproc(ipr,*) contain the identification of the first and last active processor in each row. This simplifies array I/O and some hand coded broadcasts. **Common Blocks:** PRSUMI
Xcspmn	Subroutine XCSPMN identifies local array sizes, no by mo, from total, noa by moa. Boundary flags iec(1:4) are for boundaries W, E, S and N, respectively: = 0 Interior edge. = 1 Exterior edge. The exterior edges may be reset to 0 later if they represent periodic boundaries. Boundary flags iec(5:8) are always defined later. **Calling Sequence:** xcspmn (no, mo, iec, noa, moa) **Data Declaration:** Integer no, mo, iec, noa, moa
Xcstop	Subroutine XCSTOP stops all processes. All processes must call this routine. Use subroutine XCHALT for emergency stops. **Calling Sequence:** xcstop (cerror) **Data Declaration:** Character cerror
Xcsum2	Subroutine XCSUM2 sums part of a 2D array. Array n, m specifies the local dimensions of the array, but n1, n2 and m1, m2 specify the part of the entire array to sum. The sum is bit for bit reproducible for the same iprsum. **Calling Sequence:** xcsum2 (asum, a, n, m, n1, n2, m1, m2) **Data Declaration:** Integer n, m, n1, n2, m1, m2 Real asum, a **Common Blocks:** PRSUMI
Xcsync	Barrier, no processor exits until all arrive. This is a wrapper for the 'BARRIER' macro.
Xctmr0	Subroutine XCTMR0 starts timer n. **Calling Sequence:** xctmr0 (n) **Data Declaration:** Integer n **Common Blocks:** ZCTMRC ZCTMRI ZCTMR8
Xctmr1	Subroutine XCTMR1 adds the time since call to XCTIM0 to timer n. **Calling Sequence:** xctmr1 (n) **Data Declaration:** Integer n **Common Blocks:** ZCTMRC ZCTMRI ZCTMR8

Subroutine	Description
Xctmri	Subroutine XCTMRI initializes timers. It is called by subroutine XCSPMD. • Timers 1:32 are for message passing routines. • Timers 33:80 are for general NCOM routines. • Timers 81:96 are for user selected routines. • Timer 97 is the total time. Call XCTMR0(n) to start timer n. Call XCTMR1(n) to stop timer n and add event to timer sum. Call XCTNRN(n, cname) to register a name for timer n. Call XCTMRP to printout timer statistics (called by XCSTOP). **Common Blocks:** ZCTMRC ZCTMRI ZCTMR8
Xctmrn	Subroutine XCTMRN registers the name of timer n. **Calling Sequence:** xctmrn (n, cname) **Data Declaration:** Integer n Character cname **Common Blocks:** ZCTMRC ZCTMRI ZCTMR8
Xctmrp	Subroutine XCTMRP prints all active timers. Upon exit all timers are reset to zero. **Common Blocks:** ZCTMRC ZCTMRI ZCTMR8

5.12.3 Communication Subroutines for Multiple Processors (xmc_mp)

File ***xmc_mp*** contains communication routines for multiple processors. Many of the subroutines are already documented in **Section 5.12.2**. The following subroutines are either unique to ***xmc_mp*** or contain common blocks not found in the subroutines of ***xmc_sm***.

Subroutine	Description
Shmem32_get	**Calling Sequence:** shmem32_get(target, source, len, pe) **Data Declaration:** Integer len,pe Real target, source
Shmem32_get4	**Calling Sequence:** shmem32_get4(target, source, len, pe) **Data Declaration:** Integer len,pe Real target, source
Xcaget	Subroutine XCAGET converts an entire 2D array from tiled to non-tiled layout. Variable mnflg selects which nodes must return the array: = 0 All nodes. = n Node number n (mnproc = n). **Calling Sequence:** xcaget (aa, na, ma, a, n, m, mnflg) **Data Declaration:** Integer na, ma, n, m, mnflg Real aa, a

Subroutine	Description
	Common Blocks: CPROCN
Xcaput	Subroutine XCAPUT converts an entire 2D array from non-tiled to tiled layout. **Calling Sequence:** xcaput (aa, na, ma, a, n, m, mnflg) **Data Declaration:** Integer na, ma, n, m, mnflg Real aa, a **Common Blocks:** CPROC1D CPROCN
Xceget	Subroutine XCEGET finds the value of a(ia, ja) on the non-tiled 2D grid. **Calling Sequence:** xceget (aelem, a, n, m, ia, ja) **Data Declaration:** Integer n, m, ia, ja Real aelem, a **Common Blocks:** CTILEZ CPROCN
Xceput	Subroutine XCEPUT fills a single element in the non-tiled 2D grid. **Calling Sequence:** xceput (aelem, a, n, m, ia, ja) **Data Declaration:** Integer n, m, ia, ja Real aelem, a **Common Blocks:** CTILEZ CPROCN
Xcgthri	This is an integer all gather subroutine. **Calling Sequence:** xcgthri(a,aa) **Data Declaration:** Real aa, a
Xchalt	Subroutine XCHALT stops all processes. Only one process needs to call this routine because it is for emergency stops. Use subroutine XCSTOP for ordinary stops called by all processes. **Calling Sequence:** xchalt (cerror) **Data Declaration:** Character cerror **Common Blocks:** CPROCN
Xciget	Subroutine XCIGET converts (ia,ja) on the non-tiled 2D grid to a local (i,j). **Calling Sequence:** xciget(i,j, n,m, ia,ja) **Data Declaration:** Integer i,j,n,m,ia,ja **Common Blocks:** CPROCN
Xcigtg	This subroutine converts local (i,j) to global (ia,ja) on the non-tiled 2D grid. **Calling Sequence:** xcigtg(i,j, n,m, ia,ja) **Data Declaration:** Integer i,j,n,m,ia,ja **Common Blocks:** CPROCN
Xclg3d	Subroutine XCLG3D extracts a vertical slice of elements from the non-tiled 3D grid. **Calling Sequence:** xclg3d(aline,nl, a,n,m,l, i1,j1,ii,ji, mnflg) **Data Declaration:** Integer nl, n, m,l, i1, j1, ii, ji, mnflg Real aline, a **Common Blocks:** CPROC1D CTILEZ CPROCN

Subroutine	Description
Xclg3d1	Subroutine XCLG3D1 extracts a vertical slice of elements from the non-tiled 3D grid. **Calling Sequence:** xclg3d1(aline,nl, a,n,m,l, i1,j1,ii,ji, mnflg) **Data Declaration:** Integer nl, n, m,l, i1, j1, ii, ji, mnflg Real aline, a **Common Blocks:** CPROC1D CPROCN CTILEZ
Xclget	Subroutine XCLGET extracts a line of elements from the non-tiled 2D grid. Variable aline(i) = a(i1 + i1*(i-1), j1+j1*(i-1)), for i = 1...nl. Variables ii and ji can each be -1, 0, or +1. Variable mnflg selects which nodes must return the line. = -1 Only nodes owning part of the line. = 0 All nodes. = n Node number n (mnproc = n). **Calling Sequence:** xclget (aline, nl, a, n, m, i1, j1, ii, ji, mnflg) **Data Declaration:** Integer nl, n, m, i1, j1, ii, ji, mnflg Real aline, a **Common Blocks:** CPROC1D CTILEZ CPROCD
Xclget1	Subroutine XCLGET1 extracts a line of elements from the non-tiled 2D grid. **Calling Sequence:** xclget1(aline,nl, a,n,m, i1,j1,ii,ji, mnflg) **Data Declaration:** Integer nl, n, m, i1, j1, ii, ji, mnflg Real aline, a **Common Blocks:** CPROC1D CTILEZ CPROCN
Xclput	Subroutine XCLPUT fills a line of elements in the non-tiled 2D grid. Variable aline(i) = a(i1+i1*(i-1), j1+j1*(i-1)), for i = 1...nl. One of ii and ji must be zero, and the other must be one. **Calling Sequence:** xclput (aline, nl, a, n, m, i1, j1, ii, ji) **Data Declaration:** Integer nl, n, m, i1, j1, ii, ji Real aline, a **Common Blocks:** CPROCN
Xcmaxr	Subroutine XCMAXR replaces array ‘a’ with its element-wise maximum over all tiles. **Calling Sequence:** xcmaxr (a, n) **Data Declaration:** Integer n Real a **Common Blocks:** CPROC1D CPROCN
Xcprod	Subroutine XCPROD sums the product of two 2D arrays. Array n,m specifies the

Subroutine	Description
	local dimensions of the array. The sum is bit for bit reproducible for the same iprsum and jprsum. **Calling Sequence:** xcprod (absum, a, b, n, m) **Data Declaration:** Integer n, m Real absum, a, b **Common Blocks:** CPROC1D CPROCN PRSUMI
Xcrang	Subroutine XCRANG finds the minimum and/or maximum of part of a 3D array. Array n, m specifies the local 2D dimensions of the array, but n1, n2 and m1, m2 specify the part of the entire array to use. The third dimension is always completely used. Variables a and amask can be the same array. This is legal Fortran 77/Fortran 90 because both a and amask are unchanged on exit. **Calling Sequence:** xcrang (amin, amax, a, n, m, l, n1, n2, m1, m2, amask, itype, spval) **Data Declaration:** Integer n, m, l, n1, n2, m1, m1, itype Real amin, amax, a, amask, spval **Common Blocks:** CPROC1D CPROCN
Xcspmd	Subroutine XCSPMD initializes /cproci/, by identifying the local processor. If jqr is less than ipr*jpr, then sea-less nodes are skipped. A map of which nodes to skip is input. Some node indices in idproc are null, and jdproc replaces the nulls with repeated indices from the same row. Variables jdproc(1,*) and jdproc(ipr,*) contain the identification of the first and last active processor in each row. This simplifies array I/O and some hand coded broadcasts. **Common Blocks:** CPROC1D CTILEZ CPROCN PRSUMI
Xcspmn	Subroutine XCSPMN identifies local array sizes, no by mo, from total, noa by moa. Boundary flags iec(1:4) are for boundaries W, E, S and N, respectively: = 0 Interior edge. = 1 Exterior edge. The exterior edges may be reset later to 0 if they represent periodic boundaries. Boundary flags iec(5:8) are always defined later. **Calling Sequence:** xcspmn (no, mo, iec, noa, moa) **Data Declaration:** Integer no, mo, iec, noa, moa **Common Blocks:** CPROCN PRSUMI
Xcstop	Subroutine XCSTOP stops all processes. All processes must call this routine. Use subroutine XCHALT for emergency stops. **Calling Sequence:** xcstop (cerror) **Data Declaration:** Character cerror

Subroutine	Description
	Common Blocks: CPROCN
Xcsum2	Subroutine XCSUM2 sums part of a 2D array. Array n, m specifies the local dimensions of the array, but n1, n2 and m1, m2 specify the part of the entire array to sum. The sum is bit for bit reproducible for the same iprsum. **Calling Sequence:** xcsum2 (asum, a, n, m, n1, n2, m1, m2) **Data Declaration:** Integer n, m, n1, n2, m1, m2 Real asum, a **Common Blocks:** PRSUMI CPROC1D CPROCN
Xctbar	Subroutine XCTBAR is a global collective operation, and the calls on ipe1 and ipe2 must list the processor as one of the two targets. This is used in place of a global barrier in halo operations, but it only provides synchronization of two processors with the local processor. Variables *ipe1* and/or *ipe2* can be -1, to indicate no processor. **Calling Sequence:** xctbar (ipe1, ipe2) **Data Declaration:** Integer ipe1, ipe2 **Common Blocks:** HALOBP

5.12.4 Program za

Program za selects between programs za_mp and za_sm.

5.12.5 I/O Subroutines for Shared Memory Computer (za_sm)

File za_sm contains I/O routines for shared memory computer.

Subroutine	Description
Getenv	Subroutine GETENV provides GETENV functionality on the T3E, using PXFGETENV. **Calling Sequence:** getenv (cname, cvalue) **Data Declaration:** Character cname, cvalue
Zaiocl	Subroutine ZAIOCL is a machine specific routine for array I/O file closing. The user must call ZAIOPN for this array unit before calling ZAIOCL. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. **Calling Sequence:** zaiocl (iaunit) **Data Declaration:** Integer iaunit **Common Block:** CZIOXX
Zaiofl	Subroutine ZAIOFL is a machine specific routine for array I/O buffer flushing. The user must call ZAIOPN for this array unit before calling ZAIOCL. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit+1000. **Calling Sequence:** zaiofl (iaunit) **Data Declaration:** Integer iaunit **Common Block:** CZIOXX

Subroutine	Description
Zaiopd	This is a machine specific routine for opening a file for array I/O. The user must call ZAIOST before the first call to ZAIOPD. See subroutines ZAIOPN, ZAIOPE and ZAIOPF. This version is for the Sun under Sun Fortran. The filename is taken from environment variable 'cenv'. The filename is then modified to reflect the data, date and time. It can be 'scratch', 'old', or 'new'. All I/O to iaunit must be performed by ZAIORD and ZAIOWR. Arrays passed to these routines must conform to 'h'. The file should be closed using ZAIOCL. **Calling Sequence:** zaiopd (cenv, cstat, h, n, m, iaunit, idate, itime) **Data Declaration:** Integer n, m, iaunit, idate, itime Real h Character cenv, cstat **Common Block:** CZIOXX
Zaiope	Subroutine ZAIOPE is a machine specific routine for opening a file for array I/O. ZAIOST must be called before the first call to ZAIOPE. See subroutines ZAIOPN and ZAIOPF. This version is for the Sun under Sun Fortran. **Calling Sequence:** zaiope (cenv, cstat, h, n, m, iaunit) **Data Declaration:** Integer n, m, iaunit Real h Character cenv, cstat **Common Block:** CZIOXX
Zaiopf	Subroutine ZAIOPF is a machine specific routine for opening a file for array I/O. The user must call ZAIOST before the first call to ZAIOPF. See subroutines ZAIOPN and ZAIOPE. This version is for the Sun under Sun Fortran. **Calling Sequence:** zaiopf (cfile, cstat, h, n, m, iaunit) **Data Declaration:** Integer n, m, iaunit Real h Character cfile, cstat **Common Block:** CZIOXX
Zaiopn	Subroutine ZAIOPN is a machine specific routine for opening a file for array I/O. The user must call ZAIOST before first call to ZAIOPN. See subroutines ZAIOPE and ZAIOPF. This version is for the Sun under Sun Fortran. The filename is taken from the environment variable FORxxxA, where xxx = iunit, with default fort.xxxa. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iunit is the nominal Fortran I/O unit (it is not used for array I/O). Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units but, for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. Variable cstat indicates the file type. It can be 'scratch', 'old' or 'new'. All I/O to iaunit must be performed by ZAIORD and ZAIOWR. Arrays passed to these routines must conform to 'h'. The file should be closed using ZAIOCL. **Calling Sequence:** zaiopn (iunit, cstat, h, n, m, iaunit) **Data Declaration:** Integer iunit, n, m, iaunit Real h Character cstat

Subroutine	Description
	Common Block: CZIOXX
Zaiord	Subroutine ZAIORD is a machine specific routine for array reading. The user must call ZAIOPN for this array unit before calling ZAIORD. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units but, for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. The array 'h' must conform to that passed in the associated call to ZAIOPN. **Calling Sequence:** zaiord (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Blocks:** CZIOXX
Zaiorw	Subroutine ZAIORW is a machine specific routine for array I/O file rewinding. The user must call ZAIOPN for this array unit before calling ZAIOCL. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. **Calling Sequence:** zaiorw (iaunit) **Data Declaration:** Integer iaunit **Common Block:** CZIOXX
Zaiosk	Subroutine ZAIOSK is a machine specific routine for skipping an array read. The user must call ZAIOPN for this array unit before calling ZAIOSK. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units but, for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. The array 'h' must conform to that passed in the associated call to ZAIOPN. **Calling Sequence:** zaiosk (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Block:** CZIOXX
Zaiost	Subroutine ZAIOST is a machine specific routine for initializing array I/O. See subroutines ZAIOPN, ZAIORD, ZAIOWR and ZAIOCL. **Calling Sequence:** zaiost (iaoffi) **Data Declaration:** Integer iaoffi **Common Block:** CZIOXX
Zaiowr	Subroutine ZAIOWR is a machine specific routine for array writing. The user must call ZAIOPN for this array unit before calling ZAIORD. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units but, for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. The array 'h' must conform to that passed in the associated call to ZAIOPN. **Calling Sequence:** zaiowr (h, n, m, l, iaunit)

Subroutine	Description
	Data Declaration: Integer n, m, l, iaunit Real h **Common Blocks:** CZIOXX
Zaiowr4	Subroutine ZAIOWR4 is a machine specific routine for array writing. It also converts argument array to real*4, so use ZAIOWR for an unchanged array. **Calling Sequence:** zaiowr4 (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Blocks:** CZIOXX
Zhclos	Subroutine ZHCLOS is a machine specific routine that closes logical unit 'iunit'. This version is for Sun workstations. **Calling Sequence:** zhclos (iunit) **Data Declaration:** Integer iunit
Zhflsh	Subroutine ZHFLSH is a machine specific routine that flushes the output buffers of logical unit 'iunit'. Use ZAIOFL to flush array I/O. This version is for the Sun workstations. It uses the 'flush' Fortran system routine. **Calling Sequence:** zhflsh (iunit) **Data Declaration:** Integer iunit
Zhgeti	Subroutine ZHGETI reads integers from standard input. I/O is called by all nodes but performed by the master node only. **Calling Sequence:** zhgeti (cquery, cformt, iinput) **Data Declaration:** Integer iinput Character cquery, cformt
Zhgetl	Subroutine ZHGETL reads logicals from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgetl (cquery, linput) **Data Declaration:** Integer linput Character cquery
Zhgetr	Subroutine ZHGETR reads real*4 from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgetr (cquery, cformt, rinput) **Data Declaration:** Real rinput Character cquery, cformt
Zhgets	Subroutine ZHGETS reads a string from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgets (cquery, cformt, sinput) **Data Declaration:** Character cquery, cformt, sinput
Zhiodr	Subroutine ZHIODR is direct access and reads a single record. Subroutine ZHIODR is expressed as a subroutine because I/O with implied do loops can be slow on some machines. **Calling Sequence:** zhiodr (a, n, iunit, irec, ios) **Data Declaration:** Integer n, iunit, irec, ios Real a

Subroutine	Description
Zhiodw	Subroutine ZHIODW is direct access and writes a single record. Subroutine ZHIODW is expressed as a subroutine because I/O with implied do loops can be slow on some machines. **Calling Sequence:** zhiodw (a, n, iunit, irec, ios) **Data Declaration:** Integer n, iunit, irec, ios Real a
Zhopen	Subroutine ZHOPEN is a machine specific routine for simple open statements. See subroutine ZHOPNE. This version is for the Sun under Sun Fortran. The filename is taken from the environment variable FORxxx, where xxx = iunit, with default fort.xxx. Variable cstat can be scratch, old, new or unknown. Variable cform can be formatted or 'unformatted'. Variable irlen can be zero (for sequential access) or non-zero (for direct access indicating record length in terms of real variables). If irlen is negative, the output will be in IEEE binary if that capability exists using standard Fortran I/O. This capability is primarily targeted to Crays; on other machines -len and len are likely to do the same thing. On the Sun, len and -len both give IEEE files. Status = 'old' must be invoked on all images, but all other calls must be on image one only. For Fortran 90 compilers, delim = 'quote' is included in the open statement where appropriate. The following call (zhopen(6,'formatted','unknown',0)) is legal and would have the effect of setting delim = 'quote' for stdout. Iunit = 6 is typically treated as a special case. **Calling Sequence:** zhopen (iunit, cform, cstat, irlen) **Data Declaration:** Integer iunit, irlen Character cform, cstat
Zhopnd	Subroutine ZHOPND is a machine specific routine for simple open statements. See subroutines ZHOPNE, and ZHOPEN. This version is for the Sun under Sun Fortran. The filename is taken from environment variable cenv. The filename is then modified to reflect the data date and time. Variable irlen can be zero (for sequential access), or non-zero (for direct access indicating record length in terms of real variables). If irlen is negative, the output will be in IEEE binary if that capability exists using standard Fortran I/O. This capability is primarily targeted to Crays; on other machines -len and len are likely to do the same thing. On the Sun, len and -len both give IEEE files. Status = 'old' must be invoked on all images but all other calls must be on image one only. For Fortran 90 compilers, delim = 'quote' is included in the open statement where appropriate. Additionally, for Fortran 90 compilers: status = 'new' implies action = 'write' status = 'old' implies action = 'read' status = 'scratch' implies action = 'readwrite' **Calling Status:** zhopnd (iunit, cenv, cform, cstat, irlen, idate, itime) **Data Declaration:** Integer iunit, irlen, idate, itime Character cenv, cform, cstat
Zhopne	Subroutine ZHOPNE is a machine specific routine for simple open statements. See subroutine ZHOPEN. This version is for the Sun under Sun Fortran. The filename is taken from environment variable 'cenv'. Variable irlen can be zero (for sequential

Subroutine	Description
	access) or non-zero (for direct access indicating record length in terms of real variables). If irlen is negative, the output will be in IEEE binary, if that capability exists using standard Fortran I/O. This capability is primarily targeted to Crays; on other machines -len and len are likely to do the same thing. On the Sun, len and -len both give IEEE files. Status = 'old' must be invoked on all images, but all other calls must be on image one only. For Fortran 90 compilers, delim = 'quote' is included in the open statement where appropriate. Additionally, for Fortran 90 compilers: status = 'new' implies action = 'write' status = 'old' implies action = 'read' status = 'scratch' implies action = 'readwrite' **Calling Sequence:** zhopne (iunit, cenv, cform, cstat, irlen) **Data Declaration:** Integer iunit, irlen Character cenv, cform, cstat
Zhopnf	Subroutine ZHOPNF is a machine specific routine for simple open statements. See subroutine ZHOPEN. This version is for the Sun for Sun Fortran. The filename is taken from 'cfile'. Variable irlen can be zero (for sequential access) or non-zero (for direct access indicating record length in terms of real variables). If irlen is negative, the output will be in IEEE binary, if that capability exists using standard Fortran I/O. This capability is primarily targeted to Crays; on other machines -len and len are likely to do the same thing. On the Sun, len and -len both give IEEE files. Status = 'old' must be invoked on all images, but all other calls must be on image one only. **Calling Sequence:** zhopnf (iunit, cfile, cform, cstat, irlen) **Data Declaration:** Integer iunit, irlen Character cfile, cform, cstat
Zhrwnd	Subroutine ZHRWND is a machine specific routine that rewinds logical unit 'iunit'. This version is for Sun workstations. **Calling Sequence:** zhrwnd (iunit) **Data Declaration:** Integer iunit
Zhsec	Subroutine ZHSEC is a machine specific routine for wall time up to this point. This version for the Sun (message passing). **Calling Sequence:** zhsec (sec) **Data Declaration:** Real sec **Common Blocks:** ZHSEC8 ZHSECI

5.12.6 I/O Subroutines for Multiple Processors (za_mp)

File za_mp contains I/O routines for multiple processors. See **Section 5.12.5** for documentation on the majority of za_mp subroutines. File za_mp has additional subroutines ZABSTR, ZHCLOS, and ZHRWND (Co-Array Fortran and Array Fortran). The following subroutines are either unique to za_mp or contain common blocks not found in subroutines of za_sm.

Subroutine	Description
Zabstr	Subroutine ZABSTR broadcasts a string from processor one to all processors. **Calling Sequence:** zabstr (string) **Data Declaration:** Character string **Common Blocks:** CPROC1D CPROCN
Zaiocl	Subroutine ZAIOCL is a machine specific routine for array I/O file closing. ZAIOPN must be called for this array unit before calling ZAIOCL. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. **Calling Sequence:** zaiocl (iaunit) **Data Declaration:** Integer iaunit **Common Blocks:** CZIOXX CPROCN
Zaiofl	Subroutine ZAIOFL is a machine specific routine for array I/O buffer flushing. The user must call ZAIOPN for this array unit before calling ZAIOCL. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. **Calling Sequence:** zaiofl (iaunit) **Data Declaration:** Integer iaunit **Common Block:** CZIOXX CPROCN
Zaiopd	This is a machine specific routine for opening a file for array I/O. It must call ZAIOST before the first call to ZAIOPD. See subroutines ZAIOPN, ZAIOPE and ZAIOPF. This version is for the Sun under Sun Fortran. The filename is taken from environment variable cenv. The filename is then modified to reflect the data date and time. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units, but for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. Variable cstat indicates the file type. It can be scratch, old, or new. All I/O to iaunit must be performed by ZAIORD and ZAIOWR. Arrays passed to these routines must conform to 'h'. The file should be closed using ZAIOCL. **Calling Sequence:** zaiopd (cenv, cstat, h, n, m, iaunit, idate, itime) **Data Declaration:** Integer n, m, iaunit, idate, itime Real h Character cenv, cstat **Common Block:** CZIOXX CPROCN
Zaiope	Subroutine ZAIOPE is a machine specific routine for opening a file for array I/O. It must call ZAIOST before the first call to ZAIOPE. See subroutines ZAIOPN and ZAIOPF. This version is for the Sun under Sun Fortran. The filename is taken from environment variable cenv. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units, but for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. Variable cstat indicates the file type. It can be scratch, old, or new.

Subroutine	Description
	All I/O to iaunit must be performed by ZAIORD and ZAIOWR. Arrays passed to these routines must conform to 'h'. The file should be closed using ZAIOCL. **Calling Sequence:** zaiope (cenv, cstat, h, n, m, iaunit) **Data Declaration:** Integer n, m, iaunit Real h Character cenv, cstat **Common Block:** CZIOXX CPROCN
Zaiopf	Subroutine ZAIOPF is a machine specific routine for opening a file for array I/O. The user must call ZAIOST before the first call to ZAIOPF. See subroutines ZAIOPN and ZAIOPE. This version is for the Sun under Sun Fortran. The filename is taken from 'cfile'. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units, but for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. Variable cstat indicates the file type; it can be scratch, old, or new. All I/O to iaunit must be performed by ZAIORD and ZAIOWR. Arrays passed to these routines must conform to 'h'. The file should be closed using ZAIOCL. **Calling Sequence:** zaiopf (cfile, cstat, h, n, m, iaunit) **Data Declaration:** Integer n, m, iaunit Real h Character cfile, cstat **Common Block:** CZIOXX CPROCN
Zaiopn	Subroutine ZAIOPN is a machine specific routine for opening a file for array I/O. **Calling Sequence:** zaiopn (iunit, cstat, h, n, m, iaunit) **Data Declaration:** Integer iunit, n, m, iaunit Real h Character cstat **Common Block:** CZIOXX CPROCN
Zaiord	Subroutine ZAIORD is a machine specific routine for array reading. The user must call ZAIOPN for this array unit before calling ZAIORD. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units, but for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. The array 'h' must conform to that passed in the associated call to ZAIOPN. **Calling Sequence:** zaiord (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Blocks:** CZIOXX CPROCN
Zaiorw	Subroutine ZAIORW is a machine specific routine for array I/O file rewinding. The

Subroutine	Description
	user must call ZAIOPN for this array unit before calling ZAIOCL. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. **Calling Sequence:** zaiorw (iaunit) **Data Declaration:** Integer iaunit **Common Block:** CZIOXX CPROCN
Zaiosk	Subroutine ZAIOSK is a machine specific routine for skipping an array read. The user must call ZAIOPN for this array unit before calling ZAIOSK. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units. but for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. The array 'h' must conform to that passed in the associated call to ZAIOPN. **Calling Sequence:** zaiosk (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Block:** CZIOXX CPROCN
Zaiost	Subroutine ZAIOST is a machine specific routine for initializing array I/O. See subroutines ZAIOPN, ZAIORD, ZAIOWR and ZAIOCL. **Calling Sequence:** zaiost(iaoffi) **Data Declaration:** Integer iaoffi **Common Block:** CZIOXX CPROCN
Zaiowr	Subroutine ZAIOWR is a machine specific routine for array writing. The user must call ZAIOPN for this array unit before calling ZAIORD. This version is for the Sun under Sun Fortran. Array I/O is Fortran direct access I/O to unit iaunit + 1000. Variable iaunit + 1000 is the I/O unit used for arrays. Array I/O might not use Fortran I/O units, but for compatibility, assume that iaunit + 1000 refers to a Fortran I/O unit anyway. The array 'h' must conform to that passed in the associated call to ZAIOPN. **Calling Sequence:** zaiowr (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Blocks:** CZIOXX CPROCN
Zaiowr4	Subroutine ZAIOWR4 is a machine specific routine for array writing. It also converts argument arrays to real*4. Use ZAIOWR for an unchanged array. **Calling Sequence:** zaiowr4 (h, n, m, l, iaunit) **Data Declaration:** Integer n, m, l, iaunit Real h **Common Blocks:** CZIOXX

Subroutine	Description
	CPROCN
Zhclos	Subroutine ZHCLOS is a machine specific routine that closes logical unit 'iunit'. This version is for the Sun (message passing) platform. **Calling Sequence:** zhclos (iunit) **Data Declaration:** Integer iunit
Zhgeti	Subroutine ZHGETI reads integers from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgeti (cquery, cformt, iinput) **Data Declaration:** Integer iinput Character cquery, cformt **Common Blocks:** CPROC1D ZHGETII
Zhgetl	Subroutine ZHGETL reads logicals from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgetl (cquery, linput) **Data Declaration:** Integer linput Character cquery **Common Blocks:** CPROC1D ZHGETLL
Zhgetr	Subroutine ZHGETR reads real*4 from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgetr (cquery, cformt, rinput) **Data Declaration:** Real rinput Character cquery, cformt **Common Blocks:** CPROC1D ZHGETRR
Zhgets	Subroutine ZHGETS reads string from standard input. I/O is called by all nodes, but performed by the master node only. **Calling Sequence:** zhgets (cquery, cformt, sinput) **Data Declaration:** Character cquery, cformt, sinput **Common Blocks:** CPROC1D ZHGETSI

5.13 ESMF Driver Program (src/esmf)

5.13.1 Program ncom

Program NCOM.F is an ESMF driver for the stand-alone NCOM ocean model.

5.14 NCOM Driver Programs (src/ncom)

5.14.1 Program ncom

This is the non-ESMF driver for the stand-alone NCOM ocean model.

5.15 Test_xca Subroutines (src/test_xca)

5.15.1 Program test_xca

Subroutine	Description		
Test	**Calling Sequence:**	test (aorig, na, ma, l, atile, n, m)	
	Data Declaration:	Integer	na, ma, l, n, m
		Real	aorig, atile
	Common Block:	CTILEZ	
Xcspmd	**Calling Sequence:**	xcspmd(mpi_comm_in)	
	Data Declaration:	Integer	mpi_comm_in
Yyprnt	Subroutine YYPRINT prints arctic boundary values.		
	Calling Sequence:	yyprnt (aorig, na, ma, l, atile, n, m)	
	Data Declaration:	Integer	na, ma, l, n, m
		Real	aorig, atile

5.16 Test_xca Subroutines (src/test_xcl)

5.16.1 Program test_xcl

Subroutine	Description		
Test	**Calling Sequence:**	test (aorig, na, ma, l, atile, n, m)	
	Data Declaration:	Integer	na, ma, l, n, m
		Real	aorig, atile
	Common Block:	CTILEZ	
Xcspmd	**Calling Sequence:**	xcspmd(mpi_comm_in)	
	Data Declaration:	Integer	mpi_comm_in
Xxlget	**Calling Sequence:**	xxlget(aline,nl, a,na,ma, i1,j1,ii,ji)	
	Data Declaration:	Integer	nl,na,ma,i1,j1,ii,ji
		Real	aline,a
Xxlg3d	**Calling Sequence:**	xxlg3d(aline,nl, a,na,ma,l, i1,j1,ii,ji)	
	Data Declaration:	Integer	nl,na,ma,l,i1,j1,ii,ji
		Real	aline,a
Yycomp	**Calling Sequence:**	yycomp(a,b,n)	
	Data Declaration:	Integer	n
		Real	a,b
Yycom3	**Calling Sequence:**	yycom3(a,b,n,l)	
	Data Declaration:	Integer	n,l
		Real	a,b

6.0 NOTES

6.1 Acronyms and Abbreviations

Acronym	Description
ASCII	American Standard Code for Information Interchange
BC	Boundary conditions
CFL	Courant Fredrich Levy scheme
CM	Coarse Mesh, refers to the parent grid of a nested grid.
COAMPS	Coupled Ocean Atmosphere Mesoscale Prediction System
CPU	Central Processing Unit
DBMS	Database Management System
DTG	Date Time Group
ECMWF	European Center for Medium-range Weather Forecast
ECOM-si	Estuarine, Coastal and Ocean Model (semi-implicit)
ESMF	Earth System Modeling Framework
FCT	Flux-corrected transport
FM	Fine Mesh, refers to a nested (child) grid.
FNMOC	Fleet Naval Meteorology and Oceanography Center
GMT	Greenwich Mean Time
GOFS	Global Ocean Forecast System
GVC	General Vertical Coordinate
HRLS	Hierarchical Least Squares algorithm
IC	Initial conditions
IEEE	Institute of Electrical and Electronic Engineers
I/O	Input/Output
lm1	l-1 this is the total number of vertical layers or levels.
m	Meter
mb	milibars
MLD	Mixed layer depth.
MODAS	Modular Ocean Data Assimilation System
MPI	Message Passing Interface
MP	Multi-Processor
MYL2	Mellor-Yamada Level 2
NCAR	National Center for Atmospheric Research
NCODA	Navy Coupled Ocean Data Assimilation
NCOM	Navy Coastal Ocean Model
netCDF	Network Common Data Form
NOGAPS	Navy Operational Global Atmospheric Prediction
NRL	Naval Research Laboratory
OBC	Open Boundary Conditions
POM	Princeton Ocean Model

PSI	Planning Systems Incorporated
RMS	Root-mean-square
S	Salinity
SDD	Software Design Description
SGI	Silicon Graphics Incorporated
SHMEM	Shared Memory
SM	Shared Memory Computer
SPMD	Single Processor Multiple Data
SSC	Stennis Space Center
SSH	Sea Surface Height
SSS	Sea Surface Salinity
SST	Sea Surface Temperature
SVN	Subversion
SZM	Sigma Z-Level Model
T	Temperature
TKE	Turbulent Kinetic Energy
t-point	Temperature grid point
UNESCO	United Nations Educational, Scientific, and Cultural Organization
u-point	U-velocity grid point - located at center of west face of a grid cell.
v-point	V-velocity grid point - located at center of south face of grid cell.

7.0 Appendix A FORTRAN Common Blocks

7.1 COMMON Blocks for General Setup Subroutines

COMMON/ BICUBCN	Type	Description
c(4, 4, 4, 4, 9)	Real	
c1(256)	Real	
c2(256)	Real	
c3(256)	Real	
c4(256)	Real	
c5(256)	Real	
c6(250)	Real	
c7(256)	Real	
c8(256)	Real	
c9(256)	Real	

7.2 COMMON Blocks for File ncom1 Subroutines

COMMON/ OBLK	Type Integer	Description Contains pointer variables for ocean model
COMMON/ PADR4I	**Type**	**Description**
ipad(maxpads, mxgrdso)	Integer	
npad(mxgrdso)	Integer	
COMMON/ PADR4C	**Type**	**Description**
cpad(maxpads, mxgrdso)	Character	

7.3 COMMON Blocks for Printing/Plotting Subroutines

COMMON/ PRNTEI4	Type	Description
indspv	Integer	
COMMON/ PRNTER4	**Type**	**Description**
spvalu	Real	
COMMON/ PRNTFI4	**Type**	**Description**

indspv	Integer	
COMMON/ PRNTFR4	**Type**	**Description**
spvalu	Real	
COMMON/ CONRE4	**Type**	**Description**
sizel	Real	Defines the size of contour line labels.
sizem	Real	Defines the size of high/low labels.
sizep	Real	Defines the size of data point values.
nrep	Integer	Number of repetitions of dash pattern between line labels.
ncrt	Integer	Number of pau's per element in dash pattern.
ilab	Integer	Flag to enable contour line labeling: = 0 No; = 1 Yes.
isizel	Integer	Size of the line labels.
isizem	Integer	Size of the labels for minimums and maximums.
isizep	Integer	Size of labels for data point values.
nulbll	Integer	Number of unlabeled lines between labeled lines.
ioffd	Integer	Flag to control normalization of label numbers: = 0 Include decimal point when possible; = Non-zero Normalize all label numbers and output a scale factor in the message below the graph.
ext	Real	Lengths of the sides of the plot are proportional to M and N.
ioffm	Integer	Flag to control the message below the plot: = 0 If the message is to be plotted; = Non-zero If the message is to be omitted.
isolid	Integer	Dash pattern for non-negative contour lines.
nla	Integer	Approximate number of contour levels when internally generated.
nlm	Integer	Maximum number of contour levels.
xlt	Real	Left hand edge of the plot.
ybt	Real	Bottom edge of the plot.
side	Real	Length of longer edge of the plot.

7.4 COMMON Blocks for Tidal Calculation Subroutines

COMMON/ VUFC5	Type	Description
konco(320)	Character	
kontab(170)	Character	Array containing all the constituent names as they are read in from the data file. It should have the minimum

		dimension mtot.
COMMON/ VUFI4	**Type**	**Description**
ii(50), jj(50), kk(50), ll(50), mm(50), nn(50)	Integer	The six Doodson numbers.
ldel(205), mdel(205), ndel(205)	Integer	The changes in the last three Doodson numbers from those of the main constituent.
ir(205)	Integer	= 1 If the amplitude ratio has to be multiplied by the latitude correction factor for diurnal constituents; = 2 If the amplitude ratio has to be multiplied by the latitude correction factor for semi-diurnal constituents; Otherwise if no correction is required to the amplitude ratio.
nj(170)	Integer	The number of satellites for this constituent.
ntidal	Integer	Number of main constituents.
Ntotal	Integer	The number of constituents for the given time kh.
COMMON/ VUFR4	**Type**	**Description**
freq(170)	Real	Array of frequencies (cycles/hr) corresponding to the constituents contained in kontab.
ee(205)	Real	The amplitude ratio of the satellite tidal potential to that of the main constituent.
ph(205)	Real	Phase correction.
semi(50)	Real	Phase correction.
coef(320)	Real	
f(170)	Real	
vu(170)	Real	

7.5 COMMON Blocks for Communications Subroutines for SM Computers

COMMON/ PRSUMI	**Type**	**Description**
iprsum	Integer	
jprsum	Integer	
COMMON/ ZCTMRC	**Type**	**Description**
cc(97)	Character	
COMMON/ ZCTMRI	**Type**	**Description**
nc(97)	Integer	

COMMON/ ZCTMR8	Type	Description
tc(97)	Real	
t0(97)	Real	
COMMON/ CPROCN	**Type**	**Description**
nstn	Integer	
nstna	Integer	
nstm	Integer	
nstma	Integer	

7.6 COMMON Blocks for Communication Subroutines for Multiple Processors

COMMON/ CPROC1D	Type	Description
idproc1	Integer	
jdproc1	Integer	
COMMON/ CPROCD	**Type**	**Description**
idprc	Integer	
jdprc	Integer	
COMMON/ PRSUMI	**Type**	**Description**
iprsum	Integer	
jprsum	Integer	
COMMON/ XCLGET4	**Type**	**Description**
al	Real	
COMMON/ XCLGETI	**Type**	**Description**
nli1j1	Integer	
COMMON/ CTILEZ	**Type**	**Description**
ztile	Real	
COMMON/ XCEGET4	**Type**	**Description**
elem	Real	
COMMON/ CPROCN	**Type**	**Description**
nstn	Integer	
nstna	Integer	
nstm	Integer	
nstma	Integer	

COMMON/ XCMAXR4	**Type**	**Description**
b	Real	
c	Real	
COMMON/ XCMASS8	**Type**	**Description**
sum8x	Real	
sum8y	Real	
sum8j	Real	
sum8p	Real	
sum8s	Real	
sum8r	Real	
COMMON/ XCRANG4	**Type**	**Description**
b	Real	
c	Real	
COMMON/ HALOBP	**Type**	**Description**
ibp	Integer	
COMMON/ XCTILI4	**Type**	**Description**
iai	Integer	
iaj	Integer	
iak	Integer	
COMMON/ XCTILR4	**Type**	**Description**
ai	Real	
aj	Real	
ak	Real	
COMMON/ XCTILXC	**Type**	**Description**
cpadtest	Character	
COMMON/ XCTIL14	**Type**	**Description**
ai	Real	
aj	Real	
ak	Real	
COMMON/ ZCTMRC	**Type**	**Description**
cc	Character	
COMMON/ ZCTMRI	**Type**	**Description**
nc	Integer	

COMMON/ ZCTMR8	Type	Description
tc	Real	

7.7 COMMON Blocks for I/O Shared Memory Subroutines

COMMON/ CZIOXX	Type	Description
iaoff	Integer	
iarec	Integer	
iiunt	Integer	
COMMON/ CZIOXW	**Type**	**Description**
w(nmx*nmx)	Real	Array I/O buffer
COMMON/ ZHSEC8	**Type**	**Description**
offsec	Real	
offset	Real	
persec	Real	
COMMON/ ZHSECI	**Type**	**Description**
icount	Integer	
iover	Integer	
lcount	Integer	
ncount	Integer	

7.8 COMMON Blocks for I/O Multiple Processor Subroutines

COMMON/ CZIOXX	Type	Description
iaoff	Integer	
iarec	Integer	
iiunt	Integer	
team	Integer	
COMMON/ CPROCN	**Type**	**Description**
nstn	Integer	
nstna	Integer	
nstm	Integer	
nstma	Integer	
COMMON/ CZIOXW	**Type**	**Description**
w	Real	

COMMON/ CPROCD	Type	Description
idprc	Integer	
jdprc	Integer	
COMMON/ CPROC1D	**Type**	**Description**
idproc1	Integer	
jdproc1	Integer	
COMMON/ ZABSTRI	**Type**	**Description**
ibuffer(256)CAF_D	Integer	
COMMON/ ZHGETII	**Type**	**Description**
ibuffer CAF_D	Integer	
COMMON/ ZHGETLL	**Type**	**Description**
ibuffer	Integer	
COMMON/ ZHGETRR	**Type**	**Description**
rbufferCAF_D	Real	
COMMON/ ZHGETSI	**Type**	**Description**
ibuffer(256)CAF_D	Integer	
COMMON/ ZHSEC8	**Type**	**Description**
offsec	Real	
offset	Real	
persec	Real	
COMMON/ ZHSECI	**Type**	**Description**
icount	Integer	
iover	Integer	
lcount	Integer	
ncount	Integer	

7.9 COMMON Blocks for Program test_xca and test_xcl

COMMON/ TESTR4	Type	Description
aorig	Real	
atile	Real	
COMMON/ CTILEZ	**Type**	**Description**

ztile	Real	

7.10 COMMON Blocks for Miscellaneous NCOM Source Code

COMMON/ RGBHEADER	Type	Description
magic	Integer	
storage	Integer	
bpc	Integer	
dimensions	Integer	
xsize	Integer	
ysize	Integer	
zsize	Integer	
pixmin	Integer	
pixmax	Integer	
dummy	Integer	
imagename	Character	
colormapid	Integer	
pad	Character	
COMMON/ CAFALL	**Type**	**Description**
all	Integer	

7.11 COMMON Blocks for Subroutine OMODEL (NCOMPAR.h)

These common blocks must be updated for the appropriate ocean grid when the grid being calculated is changed. The variables here are set for the current grid that is being calculated within subroutine OMODEL. Outside of OMODEL (e.g., within subroutine "COAMM") they will not be defined and the corresponding values within the par*o common blocks (in include file "COMMON.h") must be used.

COMMON/ PAR5O	Type	Description Nest-independent constants
pi	Real	
raddeg	Real	
degrad	Real	
small	Real	
COMMON/ PAR6O	**Type**	**Description** Nest-dependent variables
idate	Integer	
itime	Integer	
idatec	Integer	

itimec	Integer	
inde2	Integer	
indvb2	Integer	
indv2	Integer	
indt2	Integer	
inds2	Integer	
inda2	Integer	
inde3	Integer	
indvb3	Integer	
indv3	Integer	
indw3	Integer	
indt3	Integer	
inds3	Integer	
inda3	Integer	
indfcst	Integer	
idatnow	Integer	
itimnow	Integer	
irs_out	Integer	
irs_date	Integer	
irs_mean	Integer	
irs_fmt	Integer	
irs_rset	Integer	
ioutdate	Integer	
ioutnow	Integer	
irlx2now	Integer	
irlx3now	Integer	
mode	Integer	
indcor	Integer	
indden	Integer	
indadv	Integer	
indadvr	Integer	
indxk	Integer	
indzk	Integer	
indtkes	Integer	
indext	Integer	
indtype	Integer	
indbio	Integer	
indice	Integer	
itermom	Integer	
indbaro	Integer	
indsolv	Integer	
indrag	Integer	
ifdadrh	Integer	

ifdadrv	Integer	
ifdaduh	Integer	
ifdaduv	Integer	
ifdpgrd	Integer	
ifdcor	Integer	
indsbc	Integer	
indatp	Integer	
indtau	Integer	
indsft	Integer	
indsfs	Integer	
indsol	Integer	
indcld	Integer	
indsst	Integer	
indsss	Integer	
indsruf	Integer	
indcyc	Integer	
indtide	Integer	
indobc	Integer	
indobe	Integer	
indobvb	Integer	
indobu	Integer	
indobv	Integer	
indobr	Integer	
indriv	Integer	
indrivr	Integer	
indiag	Integer	
COMMON/ PAR7O	**Type**	**Description** Logical variables.
bclinic	Logical	
curved	Logical	
noslip	Logical	
sigdif	Logical	
largmix	Logical	
wetdry	Logical	
tidpot	Logical	
botrun	Logical	
forward	Logical	
vector	Logical	
shrnkwp	Logical	
locate	Logical	
COMMON/ PAR8O	**Type**	**Description** Real variables.
tothrs	Real	

rho0	Real	
g	Real	
cp	Real	
ramphrs	Real	
skmin	Real	
ykmin	Real	
xkre	Real	
smag	Real	
prnxi	Real	
zkmmin	Real	
zkhmin	Real	
zkre	Real	
cbmin	Real	
botruf1	Real	
rlax_ts	Real	
rlax_ds	Real	
b1_myl2	Real	
dti	Real	
dte	Real	
asf	Real	
eg1	Real	
eg2	Real	
eg3	Real	
vg1	Real	
vg2	Real	
vg3	Real	
cb_filt	Real	
cb_dep	Real	
rlaxsst	Real	
rlaxsss	Real	
charnok	Real	
rlxobvb	Real	
rlxobv	Real	
rlxobr	Real	

7.12 COMMON Blocks for NCOM (COMMON.h)

COMMON/ CPROCI	**Type**	**Description** Indicates which processor the local ocean model grid is on.
mproc	Integer	
nproc	Integer	
ipr	Integer	

jpr	Integer	
jqr	Integer	
COMMON/ PAR1O	**Type**	**Description** Character variables for ocean model parameters.
modelo	Character	
expto	Character	
domaino	Character	
COMMON/ PAR2O	**Type**	**Description** Integer variables.
iruno	Integer	
iouto	Integer	
infso	Integer	
iphyo	Integer	
numo	Integer	
isbco	Integer	
iobco	Integer	
irivo	Integer	
idiago	Integer	
io_unit_offset	Integer	
istdo_unit	Integer	
COMMON/ PAR3O	**Type**	**Description** Logical variables.
lruno	Logical	
louto	Logical	
lphyo	Logical	
lnumo	Logical	
lsbco	Logical	
lobco	Logical	
lrivo	Logical	
ldiago	Logical	
COMMON/ PAR4O	**Type**	**Description** Real variables.
runo	Real	
outo	Real	
phyo	Real	
rnumo	Real	
sbco	Real	
obco	Real	
rivo	Real	
diago	Real	
COMMON/ NEST1O	**Type**	**Description** For ocean model nest information.
nesto	Integer	

nnesto	Integer	
nsto	Integer	

7.13 COMMON Blocks for COAMPS (COAMPS.h)

The following common blocks store information about ocean and atmospheric model grids for running in the COAMPS environment.

COMMON/ COAMPS1	Type	Description
inidtg	Character	Initial DTG for simulation.
coamdir	Real	Directory where COAMPS data files are located.
COMMON/ COAMPS2	**Type**	**Description**
dtcosfx	Real	Frequency of COAMPS atm flux fields (s).
dtcosst		Frequency of COAMPS SST fields (s).
dtcocyc		Length of COAMPS forecast cycle (s).
dtcomin		Minimum forecast time for using COAMPS fields (s).
idbms2		Flag to denote use of sequential or direct flat files.
ifcast2		Flag to denote use of long or short COAMPS forecast tau's.
inesta2		
dataa		Data record for atmospheric grid.
datao		Data record for ocean grid.
COMMON/ COAMPS3	**Type**	**Description**
outff	Logical	
out_dir	Character	
idbms_o	Integer	
offpa	Real	
offtx	Real	
offqr	Real	
offq0	Real	
offep	Real	
offse	Real	
offsv	Real	
offst	Real	
offss	Real	
offmv	Real	
offmt	Real	
offms	Real	
offzv	Real	
offzt	Real	
offzs	Real	

8.0 APPENDIX B Argument Variables

Primary NCOM Variables

These variables are for the sigma-z vertical coordinate grid version of NCOM. No GVC variables are included in this table. Units used within the model are mks (meters, kilograms, seconds).

General prefix naming rules/conventions (mostly followed, but not 100%):

- -r appended to name to indicate reciprocal.
- - (if no designation) indicates centered in grid cell at t-pt.
- -m depth variable centered at grid cell mid-pt.
- -u indicates centered at u-pt.
- -v indicates centered at v-pt.
- -w indicates centered at w-pt.
- -x indicates x-direction.
- -y indicates y-direction.
- -z indicates *z*-direction.

Variable	Description
Main Input Dimensions	Note that these may have an "o" suffixed to them in some of the initial routines to distinguish them from the atmospheric model variables in COAMPS when the models are coupled.
n,m	Horizontal grid dimensions in x and y. These generally refer to the dimensions of the entire model grid. However, if the model is running in a multi-processor environment, n and m revert to being the grid dimensions on the local processor. Currently, the overall horizontal grid dimensions need to be evenly divisible by the number of processors used in each of the grid directions (the parallel processing is done by decomposing the domain into equal sized subdomains with each subdomain running on a single processor). In general, it is useful for multiprocessing if the overall grid dimensions are divisible by some moderately high power of 2; e.g., 16, 32, 64, etc., so that a range of sizes of processor arrays can be accommodated.
l	Total vertical layers (or levels) + 1.
ls	Total number of sigma layers + 1.
nr	Number of scalar model prognostic variables.
nq	Number of prognostic turbulence variables.
ntyp	Number of solar extinction profile types (not much used at this point, but available to facilitate implementation of spatially variable solar extinction).
ntc	Number of tidal constituents being forced at open bndy.
nobmax	Maximum number of open boundary points.
nrvmax	Maximum number of rivers.

Variable	Description
Halo width and maximum dimensions	These are defined in include files (**PARAM.h**).
nmh	Halo width.
nmxa	Maximum horizontal dimension for total grid (n or m).
nmx	Maximum horizontal dimension for single processor (n or m).
lmx	Maximum number of vertical levels (l).
nrmx	Maximum number of scalar variables (nr).
nqmx	Maximum number of turbulence fields (nq).
nobmxt	Maximum number of open boundary points for total grid.
nobmx	Maximum number of open boundary points on a single processor.
ntcmx	Maximum number of tidal constituents.
nrivmx	Maximum number of river inflow points.
mxgrdso	Maximum number of grids (including nested grids).
nsavmx	Maximum number of individual model grid points at which output data can be saved (=40).
Time variables	
iter	Temporal iteration number. On a restart, the model starts where it left off.
iterx	Iteration number for barotropic mode (not currently used).
times	Elapsed time in seconds since the start of the run.
timed	Elapsed time in days since the start of the run.
Grid indexing variables	
kb(n,m)	Index of bottom layer at t-pt.
kbu(n,m)	Index of bottom layer at u-pt.
kbv(n,m)	Index of bottom layer at v-pt.
is(m),ie(m)	I-loop start and stop indices for shrinkwrapping.
ism(m),iem(m)	I-loop start and stop indices at v points (minimum).
isp(m), iep(m)	I-loop start and stop indices at v points (maximum).
js,je	J-loop start and stop indices.
ke(m)	Max value of kb in an i-row.
iec(8)	First four values denote whether the W,E,S,N sides are exterior (=1) or interior (=0) tile edges (needed when running MP). Values 5 to 8 are set to one minus the values for 1 to 4.
i,j,k	Indices used when do-looping in x, y, and z.
ir	Index used when do-looping through different scalar fields.
iq	Index used when do-looping through different turbulence fields.
Time indexing variables	
i1,i2,i3	Temporal indices for 3 saved baroclinic time levels.
ib1,ib2,ib3	Temporal indices for 3 saved barotropic time levels (not used).

Variable	Description
j1,j2	Temporal indices for 2 saved baroclinic time levels.
ifx1,ifx2	Temporal indices for surface fluxes from input file.
iat1,iat2	Temporal indices for surface fluxes from coupled atmospheric model.
iss1,iss2	Temporal indices for specified SST and SSS.
iob1,iob2	Temporal indices for open boundary data.
irv1,irv2	Temporal indices for river inflow data.
ilx1,ilx2	Temporal indices for T and S relaxation fields.
Grid related variables	
d(n,m,3)	Total depth at e-pt (e - h, >=0).
du(n,m,3)	Total depth at u-pt.
dv(n,m,3)	Total depth at v-pt.
d1(n,m,3)	Total depth to bottom of sigma layers at e-pt (e - h1, >=0).
d1u(n,m,3)	Total depth to bottom of sigma layers at u-pt.
d1v(n,m,3)	Total depth to bottom of sigma layers at v-pt.
Input values for vertical grid	
zw(l)	Static depth at w-pts on the *z*-level grid (defined positive upward, i.e., values below *z*=0 are negative). These are used to calculate fractional depths on the sigma coordinate grid.
Input values for horizontal grid	
h(n,m)	Static bottom depth at grid-cell center, i.e., water depth when surface elevation is zero. H is positive upward, i.e., bottom depths below *z*=0 are negative and values above *z*=0 are positive. *Z*=0 is ~ the position of the equilibrium sea surface.
elon(n,m)	Longitude at t-pt (deg E).
alat(n,m)	Latitude at t-pt (deg N).
ang(n,m)	Angle between local latitude line and x-axis at t-pt. For counterclockwise rotation of grid with respect to lat-long, ang > 0.
dx(n,m)	Grid spacing in x at t-pt (+).
dy(n,m)	Grid spacing in y at t-pt (+).
ibo(4)	Offset of boundary of model domain from edge of grid (in grid points). The four values correspond to the W, E, S, and N sides of the domain. A value of zero indicates no offset. The purpose of the offset is to allow the model domain to be smaller than the overall grid size to get around the constraint that the grid dimension must be evenly divisible by the number of processors in that direction.
Main prognostic variables	
e(n,m,3)	Surface elevation.
udb(n,m,3)	Barotropic transport (ub*d) at u-pt.
vdb(n,m,3)	Barotropic transport (vb*d) at v-pt.

Variable	Description
u(n,m,lm1,3)	Velocity in x at u-pt.
v(n,m,lm1,3)	Velocity in y at v-pt.
r(n,m,lm1,2,nr)	Scalar variables (t, s, ...) at t-pt.
q(n,m,l,2,3)	TKE and TKE*(turbulent length scale) at w-pt.
e2(n,m,3)	Depth-averaged e at u-pt for explicit barotropic calc.
ub2(n,m,3)	Depth-averaged u at u-pt for explicit barotropic calc.
vb2(n,m,3)	Depth-averaged v at v-pt for explicit barotropic calc (e2, ub2, and vb2 are not currently used).
Variables used for relaxation of T and S to specified values	
rlx(n,m,l-1,2,2)	Externally provided time-varying 3D fields of T and S to which the internal T and S fields can be relaxed. Two sets of fields are held in memory at any one time.
wlx(n,m,l-1)	Externally provided 3D field containing temporal relaxation timescale defined at each model grid pt used to relax internal T and S fields to values in rlx.
tmlx(2)	Time (since start of model run) associated with the two sets of rlx values that are stored in memory.
ilx1,ilx2	Temporal indices used to denote time of relaxation field.
Surface forcing variables	
patm(n,m)	Surface atmospheric pressure (m).
usflx(n,m)	Surface wind stress in x at e-pt (m^2/s^2).
vsflx(n,m)	Surface wind stress in y at e-pt (m^2/s^2).
rsflx(n,m)	Surface fluxes for scalar variables at e-pt (units-m/s).
solar(n,m)	Solar flux penetrating surface at e-pt (°C-m/s).
surruf(n,m)	Surface roughness (e.g., from waves) (m).
patm2(n,m,2)	Surface atmospheric pressure (m) stored at 2 times.
usflx2(n,m,2)	Surface wind stress in x at e-pt stored at 2 times.
vsflx2(n,m,2)	Surface wind stress in y at e-pt stored at 2 times.
rsflx2(n,m,2)	Surface fluxes for scalar variables at e-pt at 2 times.
solar2(n,m,2)	Solar or cloud data e-pt at 2 times.
Open boundary variables	
nob	Total number of open boundary points.
neob(2,4)	Index limits for elevation points along each (W E S N) bndy.
nuob(2,4)	Index limits for normal velocity points along each bndy.
nvob(2,4)	Index limits for tangent velocity points along each bndy.
iob(nob)	X index of center of bndy pt.
job(nob)	Y index of center of bndy pt.
iobi(nob)	X index of center of interior pt adjoining bndy pt.

Variable	Description
jobi(nob)	Y index of center of interior pt adjoining bndy pt.
ivob(nob)	X index of bndy pt at tangent velocity pt.
jvob(nob)	Y index of bndy pt at tangent velocity pt.
kob(nob)	Z index of midpoint of bottom grid cell at a bndy pt.
eob(nob,2)	Surface elevation at boundary (at two times).
ubob(nob,2)	Normal transport at boundary (depth-ave velocity * depth).
vbob(nob,2)	Tangent transport at boundary (depth-ave velocity * depth).
uob(l-1,nob,2)	Baroclinic normal velocity at bndy.
vob(l-1,nob,2)	Baroclinic tangent velocity at bndy.
rob(l-1,nr,nob,2)	Scalar values (including T and S) at bndy.
cgwb(nob,2)	External and internal (1^{st} mode) gravity wave speed at bndy.
tmob(2)	Time of data (values) at open boundary points.
etab(ntc,nob)	Tidal elevation amplitude at boundary (for each constituent).
etpb(ntc,nob)	Tidal phase at boundary (in radians).
utab(ntc,nob)	Amplitude of tidal normal transport (depth-averaged velocity * depth) at boundary.
utpb(ntc,nob)	Phase of tidal normal velocity at boundary (radians).
vtab(ntc,nob)	Amplitude of tidal tangential transport (depth-averaged velocity * depth) at boundary.
vtpb(ntc,nob)	Phase of tidal tangent velocity at boundary (radians).
tidecn(ntc)	Name of tidal constituent.
tidefq(ntc)	Frequency of tidal constituent.
River inflow variables	
nriv	Number of river inflow points on local processor.
nrriv	Number of scalar fields specified for river inflows.
lriv	Number of depths at which river inflow scalar values are specified.
irv1,irv2	Temporal indices for river data.
iriv(nrvmax)	X gridpoint location of river inflow.
jriv(nrvmax)	Y gridpoint location of river inflow.
isriv(m)	Starting index for river pt locations in a y row.
ieriv(m)	Ending index for river pt locations in a y row.
wtriv(nrvmax,l-1)	Fraction of total river inflow at each vertical pt.
qriv(nrvmax,2)	River inflow rate for each river inflow pt.
rriv(nrvmax,l-1,nr,2)	Values of scalar fields for river inflows.
tmriv(2)	Time of river inflow data.
w1riv	Temporal weighting of river data at most recent time.
Other variables	
nt, mt	Total (global) horizontal grid dimensions.
na, ma	Total horizontal grid dimensions (same as *nt* and *mt*).
ni4s	Counter for memory needed for integer variables.

Variable	Description
nl4s	Counter for memory needed for logical variables.
nr4s	Counter for memory needed for real variables.
dti2	Timestep for leapfrog time differencing (usually 2*dti, but may be dti on 1st iteration).
ramp	Current value of ramp for gradual spinup of ocean forcing (i.e., baroclinic pressure gradients, atmospheric forcing, boundary conditions, etc.).
ub(n,m)	Depth-averaged (barotropic) velocity in x at u-pt.
vb(n,m)	Depth-averaged (barotropic) velocity in y at v-pt.
w(n,m,l)	Velocity in *z* at w-pt (+ upwards).
rho(n,m,lm1)	*In situ* density minus reference density rho0.
sos(n,m,lm1)	Speed of sound. Used to calculate stability with Mellor's equation of state if density includes effect of pressure.
sor(n,m,lm1)	Source volume flux at each grid pt (m^3/s).
sorb(n,m)	Vertical integral of sor.
rmean(n,m,ls-1,nr+1)	Climate or mean values of scalar fields and horizontal mean values of density (density is stored at ir=nr+1).
fu(n,m)	Vertically integrated forcing for barotropic u velocity.
fv(n,m)	Vertically integrated forcing for barotropic v velocity.
aax(n,m)	Coefficient used for implicit free surface solver.
aay(n,m)	Coefficient used for implicit free surface solver.
xk(n,m,lm1)	(Horizontal viscosity or diffusivity in x at u-pt)**dyx*dzm*.
yk(n,m,lm1)	(Horizontal viscosity or diffusivity in y at v-pt)**dxy*dzm.*
zkm(n,m,l)	Vertical turbulent viscosity at w-pt.
zkh(n,m,l)	Vertical turbulent diffusivity at w-pt.
ext(n,m,l)	Solar extinction profiles at each horizontal pt, defined at w-pts.
istype(ntyp)	Index corresponding to solar extinction type *iptype*.
iptype(n,m)	Solar extinction type for each horizontal grid pt.
qrf(l,ntyp)	Solar extinction profiles defined for different water types.
tl(n,m,l)	Turbulence length scale, defined at w-pt.
wubot(n,m)	Bottom stress at u-pt.
wvbot(n,m)	Bottom stress at v-pt.
botruf(n,m)	Bottom roughness at each horizontal grid pt.
o	Large array containing all real variables allocated in subroutine MEMMO.
Temporary variables	
jf	Index denoting values for row j.
jb	Index denoting values for row j+1.
iterm	Current number of iterations of momentum equations. The total number of iterations of the momentum equations is set by *itermom*.
ua(n,lm1)	Advective transport in x at u-pt divided by 2 (u*dyu*dz/2).
va(n,lm1)	Advective transport in y at v-pt divided by 2 (v*dxv*dz/2).
wa(n,l)	Advective transport in *z* at w-pt divided by 2 (w*dx*dy/2).

Variable	Description
xk(n,m,lm1)	(Mixing coefficient in x direction)*dyu*dz.
yk(n,m,lm1)	(Mixing coefficient in y direction)*dxv*dz.
flx	Flux in x-direction.
fly	Flux in y-direction.
flz	Flux in z-direction.
rho_a(n,4,l-1)	Density anomaly.
pgx(n,l-1)	Horizontal baroclinic pressure gradient in x.
pgy(n,l-1)	Horizontal baroclinic pressure gradient in y.
fc(n,4,l-1)	Intermediate calculation of Coriolis term for u equation.
fcu(n,4,l-1)	Intermediate calculation of Coriolis term for v equation.
ax(n,m),aax(n,m)	X coefficients for implicit free surface solver.
ay(n,m),aay(n,m)	Y coefficients for implicit free surface solver.
bb(n,m)	Diagonal coefficients for implicit free surface solver.
ff(n,m)	Forcing terms for implicit free surface solver.
alatave	Mean latitude of model domain.
zlay(n,m,l)	Depth to top of each grid cell.
hneg(n,m)	Bottom depth + downwards.
zkb(n,m,l)	Scratch array.
dtdazr	Scratch array.
uacr	Scratch array used for diagnostics.
vacr	Scratch array used for diagnostics.
ucr	Scratch array used for diagnostics.
vcr	Scratch array used for diagnostics.
ucr1	Scratch array used for diagnostics.
vcr1	Scratch array used for diagnostics.
ucr2	Scratch array used for diagnostics.
vcr2	Scratch array used for diagnostics.
wpf(n,m)	Scratch array.
wxy(n,m,*)	Scratch array.
wxz(n,l,*)	Scratch array.

Constants

Defined and Calculated Constants

Constant	Description
Defined Constants	
pi	3.1415926535
raddeg	Pi/180
degrad	180./pi

Constant	Description
small	A small number = 1.0e-8.
ae(7)	Constants for Friedrich-Levitus equation of state.
be(7)	Constants for Friedrich-Levitus equation of state.
ce(7)	Constants for Friedrich-Levitus equation of state.
Calculated Constants	
amsk(n,m,l)	Land-sea mask at t-pts.
umsk(n,m,l)	Land-sea mask at u-pts.
vmsk(n,m,l)	Land-sea mask at v-pts.
cbu(n,m)	Coefficient of bottom friction at u pt.
cbv(n,m)	Coefficient of bottom friction at v pt.
de(7)	Constants for Friedrich-Levitus equation of state.
cet(5)	Constants for Friedrich-Levitus thermal expansion coefficient.
ces(3)	Constants for Friedrich-Levitus salinity expansion coefficient.
Calculated Grid Related Constants	
dxu(n,m)	Grid spacing in x at u-pt.
dyu(n,m)	Grid spacing in y at u-pt.
dxv(n,m)	Grid spacing in x at v-pt.
dyv(n,m)	Grid spacing in y at v-pt.
dxr(n,m)	1/dx.
dyr(n,m)	1/dy.
dxur(n,m)	1/dxu.
dyur(n,m)	1/dyu.
dxvr(n,m)	1/dxv.
dyvr(n,m)	1/dyv.
da(n,m)	Horizontal area of grid cell at t-pt (dx*dy).
dau(n,m)	Horizontal area of grid cell at u-pt.
dav(n,m)	Horizontal area of grid cell at v-pt.
dar(n,m)	1/da(n,m).
daur(n,m)	1/dau(n,m).
davr(n,m)	1/dav(n,m).
hu(n,m)	Static depth at u-pt (depths below z=0 are neg).
hv(n,m)	Static depth at v-pt (depths below z=0 are neg).
h1(n,m)	Static depth to bottom of sigma levels at t-pt (depths below z=0 are neg).
h1u(n,m)	Static depth to bottom of sigma levels at u-pt.
h1v(n,m)	Static depth to bottom of sigma levels at v-pt.
sw(l)	Fractional sigma depth at w-pt (-).
sm(l)	Fractional sigma depth at t-pt (-).

Constant	Description
dsw(l)	Fractional sigma grid spacing at w-pt (+).
dsm(l)	Fractional sigma grid spacing at t-pt (+).
dswr(l)	1/dsw.
dsmr(l)	1/dsm.
dsm5(l)	Dsm/2.
dzm5(l)	Dzm/2.
zw(l)	Static depth at w-pt for *z*-levels (values below *z*=0 are neg).
zm(lm1)	Static depth at t-pt for *z*-levels (values below *z*=0 are neg).
dzw(l)	Vertical grid spacing at w-pt (+).
dzm(lm1)	Vertical grid spacing at t-pt (+).
dzwr(l)	1/dzw.
dzmr(l)	1/dzm.
ddx(n,m)	Difference in x grid spacing in y direction, dx(i,j+1) -dx(i,j-1).
ddy(n,m)	Difference in y grid spacing in x direction, dy(i+1,j) -dy(i-1,j).
fda(n,m)	"Modified" Coriolis parameter, defined at t-pts (= f*da*0.25).