Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7320--09-9171

Data Assimiliation in the Littoral Zone Part I: Analysis of the Navy Coupled Ocean Data Assimilation System (NCODA)

Timothy R. Keen
Richard Allard

Ocean Dynamics and Prediction Branch
Oceanography Division

February 27, 2009

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 27-02-2009	**2. REPORT TYPE** Memorandum Report	**3. DATES COVERED** *(From - To)*
4. TITLE AND SUBTITLE Data Assimilation in the Littoral Zone Part I: Analysis of the Navy Coupled Ocean Data Assimilation System (NCODA)		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER 0602435N
6. AUTHOR(S) Timothy R. Keen and Richard Allard		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER 73-6774-A9-5
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--09-9171
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research, One Liberty Center, 875 North Randolph St., Arlington, VA 22203-1995		**10. SPONSOR / MONITOR'S ACRONYM(S)** ONR
		11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes the detailed operation of the quality control component of the Navy Coupled Ocean Data Assimilation system for processing two- (2D) and three-dimensional (3D) fields. In addition to the conventional flow charts and tree diagrams used to describe sequential programs, several Unified Modeling Language (UML) diagrams are used to demonstrate the relationship between objects within the system. This Object Oriented (OO) analysis is intended to aid in future development and application of the NCODA system. UML is an abstract model of a system, which can be used to describe/develop systems that can be implemented in different computer languages. The UML model can be transformed to other representations (e.g., FORTRAN) for application. The UML diagrams are used to demonstrate three aspects of the NCODA system: (1) the static structure of the system can be easily cast as OO classes; (2) functional requirements (user-computer interaction) of the system; and (3) the dynamic behavior of the system with respect to file system access.

15. SUBJECT TERMS

NCODA   Model output
Numerical algorithms

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Timothy Keen
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UL	106	**19b. TELEPHONE NUMBER** *(include area code)* (228) 688-4950
Unclassified	Unclassified	Unclassified			

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

Table of Contents

1 Introduction

This report describes the detailed operation of program NCODA_PREP. This file describes the sequence of operations for processing two- (2D) and three-dimensional (3D) fields. The related programs, NCODA and NCODA_POST, are only discussed briefly because of the similarity of many of their functions to NCODA_PREP. Details of the theoretical and numerical algorithms that define the MultiVariate Optimal Interpolation (MVOI) scheme used by NCODA are presented elsewhere (*Cummings*, 2005; *Goerss and Phoebus*, 1992; *Lorenc*, 1981). In order to assist the reader in understanding the numerical code, references to the description of the system in *Cummings* (2005) will be made where possible.

The NCODA system is coded in FORTRAN 77 with some extensions from newer versions of the language (e.g., dynamic memory allocation). Although this language limits the programs to sequential architecture, the overall processing of the input data is naturally object oriented. Thus, the description of the system's structure and operation will include Object Oriented (OO) analysis. This is intended to aid in future modifications and development of the overall method.

The most common way of depicting OO systems is with Unified Modeling Language (UML) diagrams (*Page-Jones*, 2000). UML is an abstract model of a system, which can be used to describe/develop systems that can be implemented in different computer languages. The UML model can be transformed to other representations (e.g., FORTRAN) for application. The UML diagrams can be used to represent three different views of a system model: (1) static structure; (2) functional requirements; and (3) dynamic behavior. The functional requirement view represents the users perspective, whereas the static structural view uses objects, attributes, operations, and relationships to show how a system is built. Finally, the collaborations among objects and changes to internal states of objects are seen through the dynamic behavior view. This report will use examples of each kind of UML diagram to describe the NCODA system within the appropriate section.

Manuscript approved January 15, 2009.

2 Structure

The NCODA system consists of three independent programs written in FORTRAN 77 but with many extensions: (1) NCODA_PREP; (2) NCODA; and (3) NCODA_POST. The NCODA_PREP program is also called OCEAN-QC. This is the component that is discussed most thoroughly in this report and in *Cummings* (2005). It processes available observations and computes the covariances and errors for the MVOI that is performed in the NCODA program. The third component is the NCODA_POST program, which processes the OI fields and updates all of the analysis fields. This section presents the relationships between these components using several software analysis tools.

One goal of this section is to demonstrate the object-oriented relationships inherent in the NCODA system. This will prove useful in future modifications of the system for application to a range of littoral data processing problems. Section 2.1 shows the basic architecture of the system using traditional subroutine structure. Section 2.2 attempts to display the inherent classes that make up the NCODA system. This report will use the following conventions in referring to components of the system: (a) subroutines will be given in UPPER CASE; (b) variables within the programs will use *italics*; and (c) file names will use a `fixed-width font`. Classes will be shown in **bold type**. Where appropriate for clarity, **bold type** will also be used for FORTRAN arrays.

2.1 NCODA system structure

The NCODA system consists of three program units. These programs are run sequentially as shown in Figure 2.1-1. If a full 3D analysis is necessary, they must first be run with *opt* = 2d and then with *opt* = 3d (command line). The details of the program commands are not discussed in this report, however. The purpose here is to show how the subroutines are related in order to facilitate future program development. A full listing of subroutine calls is given in Appendix 1.

There are three categories of subroutines in the NCODA system: (1) main routines to do something only once; (2) algorithms that are used several times; for example, processing different kinds of observations; and (3) utility routines that are used repeatedly within every level of the code. The overall structure of program NCODA_PREP for types (1) and (2) is shown in the tree diagram of Figure 2.1-2. Note that the majority of the algorithms are used by subroutine CODA_PREP, which is the only substantive unit called by COAMOA. There are also three types of subroutine called by CODA_PREP: (a) those calculating grid variables and other constants (e.g., OCN_DEPTH, SET_HGRD, and SET_VGRD). These are shown to the left of Figure 2.1-2a. They can also be called at other points during execution; (b) Processing observations and climatological input for use in the OI (e.g., OCN_OBS and MODAS_GRD); and (c) subroutines for preparing the different ocean variables for the interpolation (e.g., SSH_PREP and ICE_PREP). The MV_PREP unit (Figure 2.1-2f) does a final check of all of the variables prior to running the CODA program.

The NCODA program (Figure 2.1-3) shares the grid and utility routines with NCODA_PREP. However, it also includes subroutine VOLUME_ANL, which completes both the 2D and 3D volume analyses. The observation covariance matrices computed in OBS_COVAR and routine

SPPTRF (from linpac) are used to find the decomposition matrix using Cholesky decomposition (see *Horn and Johnson*, 1985). The analysis coefficients are found from the resulting linear system of equations using the linpack routine SPPTRS.

The third member of the NCODA system is the post-processing module, NCODA_POST. This is the simplest of the three components (Figure 2.1-4). It also utilizes many of the utility functions common to the other modules and uses the subroutine ANL_ERR as its primary algorithm.

2.2 The OO class diagram

The NCODA system is a set of three sequential FORTRAN programs with many shared libraries. This heirarchical structure is shown in the tree diagrams from section 2.1. There are basic problems with this depiction, however, in that program units with similar functionality do not have their inherent commonality explicitly shown. This shortcoming can be addressed by examining the NCODA system from an OO perspective, which focuses on the common purpose of the program units. This description must, however, be superficial since there are no classes or methods represented within the system. The purpose is to indicate potential alterations in future versions of the system as they are developed.

The UML class diagram shows the structural relationships among classes much as the tree diagrams from section 2.1. However, instead of showing linkages in a simple hierarchy, the class diagram lists the methods (subroutines) that are associated with a given class (category of program unit). The class description includes the class name, the variables used by the class, and its methods.

In Figure 2.2-1, the UML class diagram for the NCODA system is used to demonstrate a more detailed classification than is suggested in the tree diagrams. The proposed classes include: (1) **databases**; (2) **grids**; (3) **gridded climate**; (4) **analysis**; (5) **file input/output (IO)**; (6) **transformations**; (7) **raw observations**; and (8) **gridded observations**. This diagram shows the variables in the lower sub-box within each class box. For example, the **database** class uses the MODAS temperature, salinity, and sea surface height (ssh) climatology values whereas the **grids** class uses the variables describing the analysis grid; number of cells, cell size, etc. This view shows how the program variables can be classified by the kind of object they represent. The class diagram also lists the class methods (subroutines) in the lowermost sub-box within each class box. These may be available for use by other program units.

There are several features of the class diagram that should be explained in order to interpret it correctly. The class diagram shows associations between classes with solid lines like those between the **database** and **gridded climate** classes in Figure. 2.2-1. Since NCODA is not an OO program, we have given these associations the name "File IO" since files are used to communicate between all of the classes in the NCODA system. An alternative name for this association might be "Preprocessing" for example. This association is depicted as occurring through two classes (**grids** and **transformations**) by the use of a dashed line in the diagram. This is because the relationship between the databases (e.g., GDEM and MODAS) and the usable gridded result requires the subroutines and variables contained within the associative classes. It is further implied by Figure 2.2-1 that **File IO** is an associative class for all of the other classes; thus, it is listed to one side.

The class **Raw Observations** contains no methods because this is a set of files to be read by the system. However, the "transformation" association (using the class **File IO**) between these data and the **Gridded Observations** class permits the observations to belong to the **Gridded Observations** class or not, as indicated by 0.* at the **Raw Observations** end of the association. Conversely, it is possible to complete an analysis using only climatology (i.e., no available obs), as indicated by 0 at the **Gridded Observations** end of this association. The arrowhead at this end of the association further indicates that the **Gridded Observations** are a composite of the **Raw Observations**; i.e., this class cannot exist unless there are obs to process.

The final point to note about the UML class diagram is the construction of the **Analysis** class as an aggregate of the **Gridded Observations** and the **Gridded Climate** classes. This means that they may both be present or absent within it; it can even be empty. The open diamonds at the **Analysis** end of the association indicate this relationship.

The above discussion shows how useful the class diagram is in understanding the NCODA system. This information would need to be discussed separately if the tree diagrams alone were used to describe its structure.

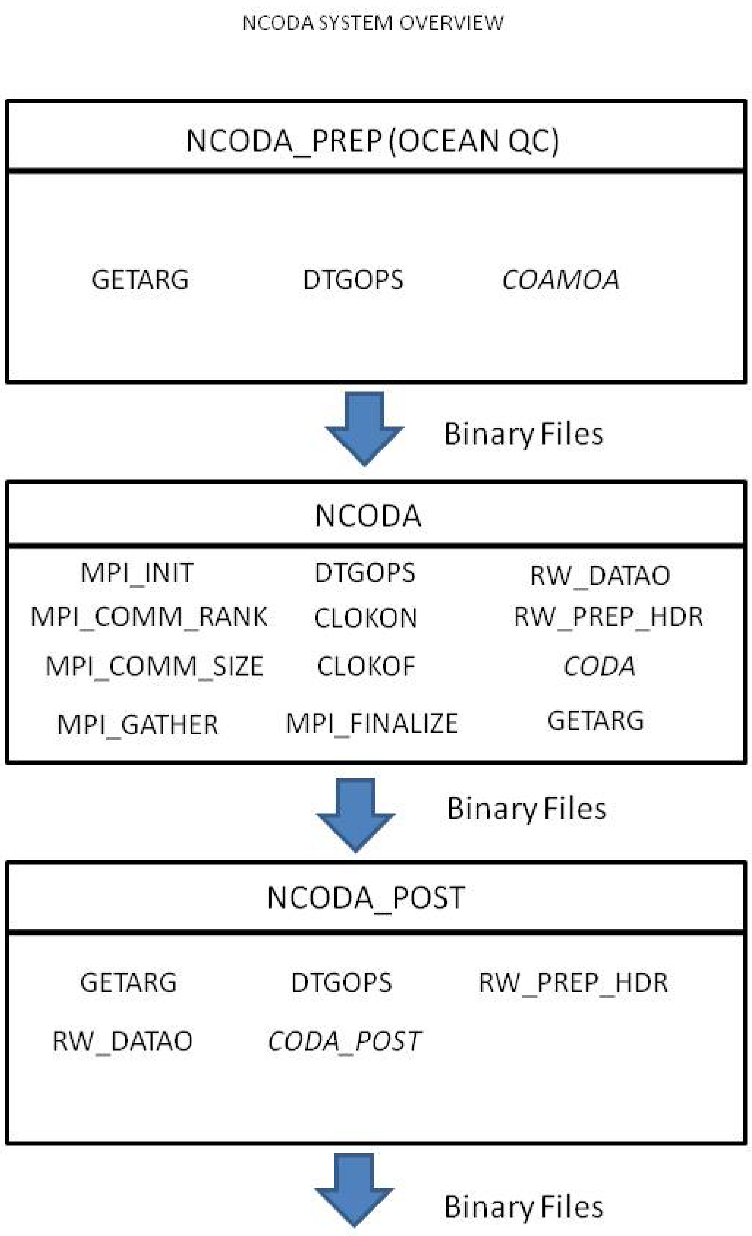

Figure 2.1-1. General structure of the NCODA system, showing subroutines called directly by each program unit. This set of programs is run once for a 2D analysis and a second time to generate the 3D analysis. The binary files are used to communicate between program units and between the 2D and 3D analyses. Names in *ITALICS* are the main computation subroutines. Names in CAPS ONLY are utility routines as described in the text.

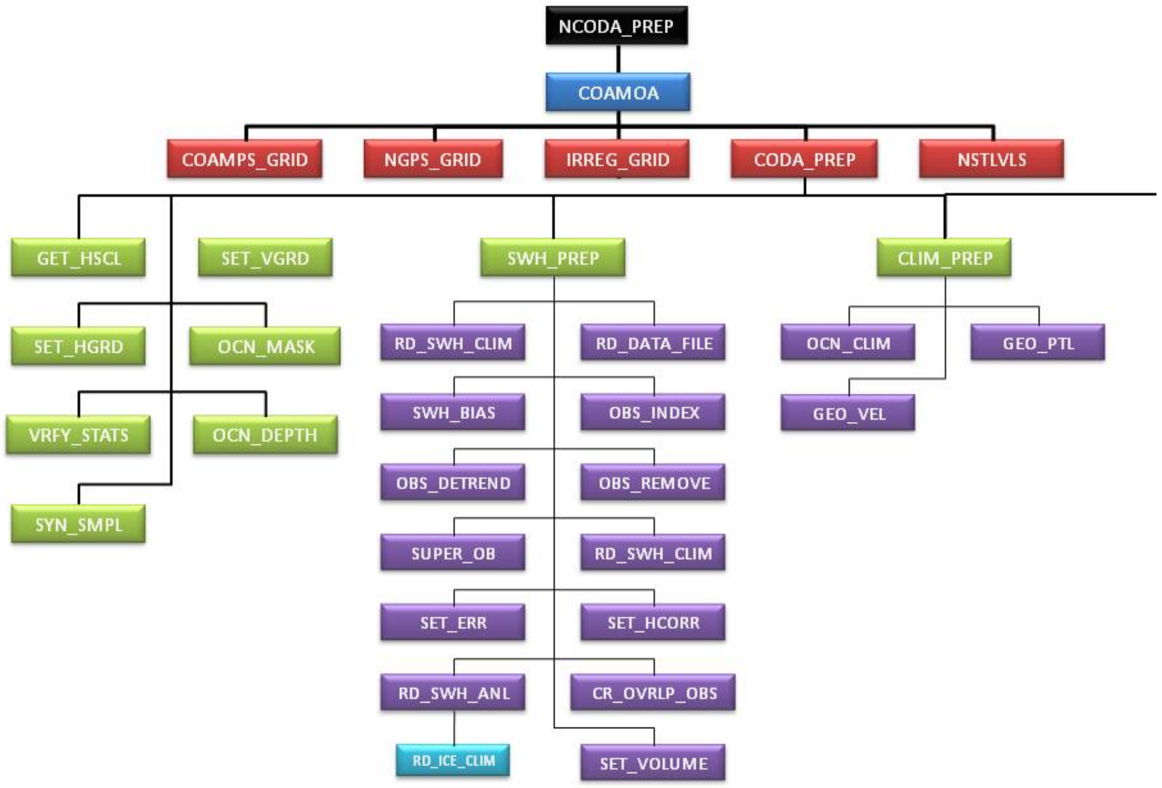

Figure 2.1-2a. Tree diagram of the NCODA_PREP program units. Some utility units are omitted for clarity (Part 1 of 6).

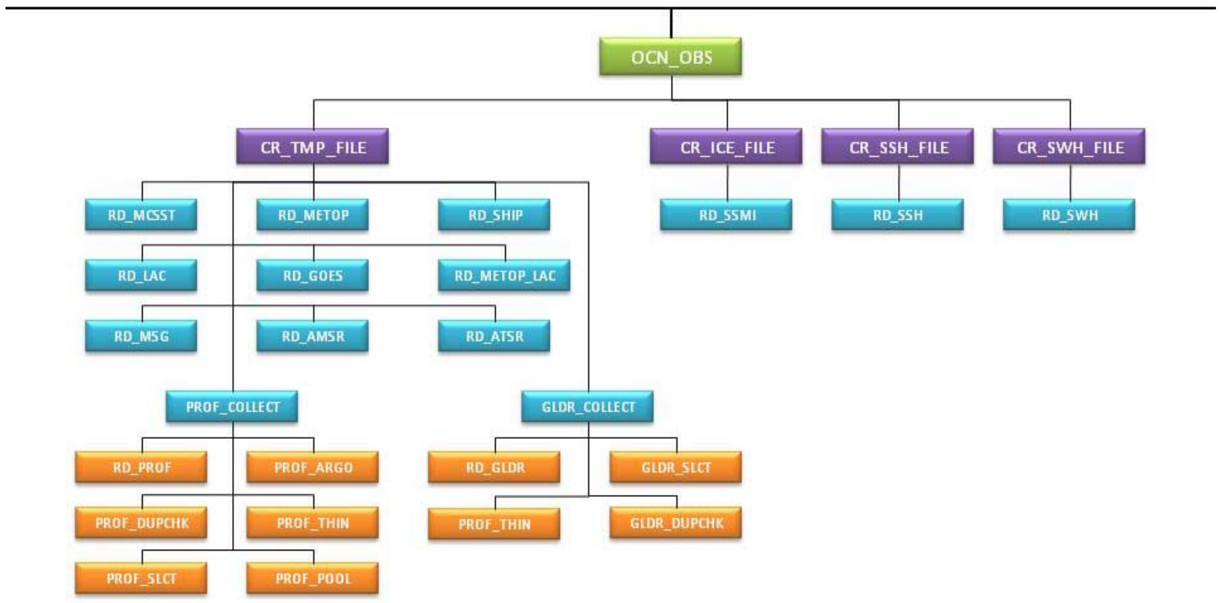

Figure 2.1-2b. Tree diagram of the NCODA_PREP program units. Some utility routines are omitted for clarity (Part 2 of 6).

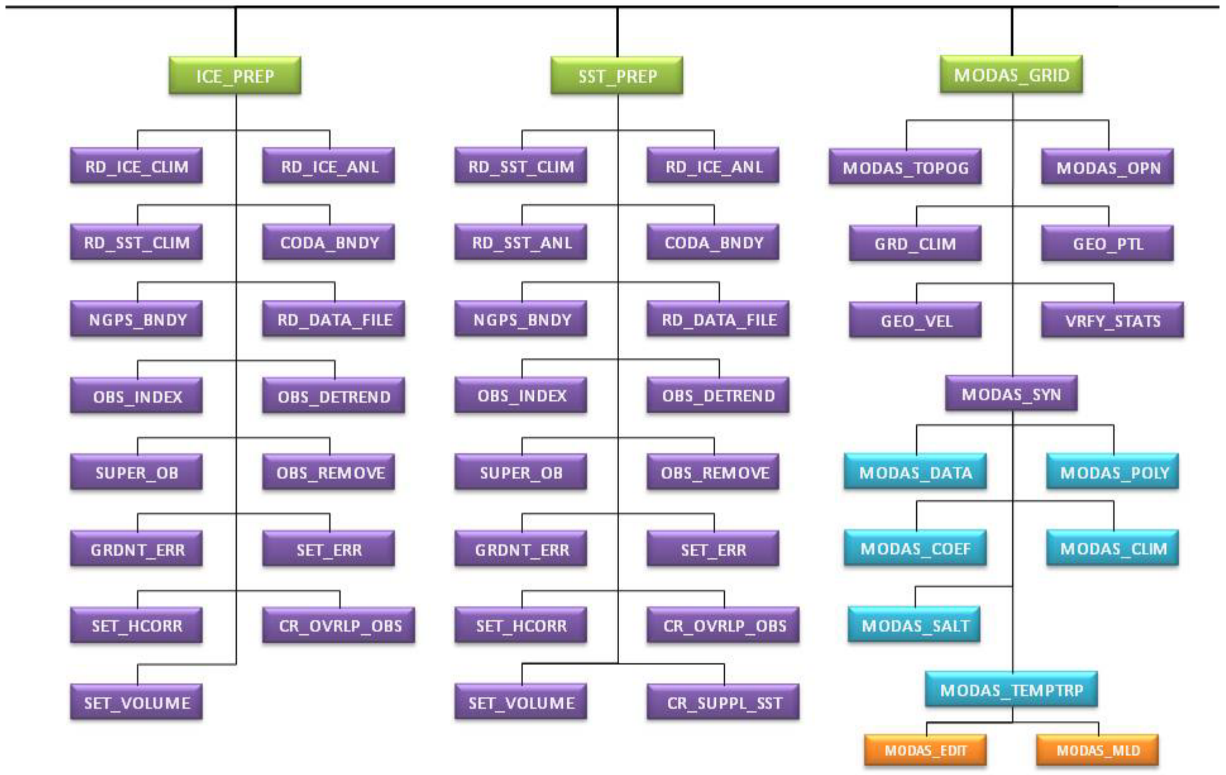

Figure 2.1-2c. Tree diagram of the NCODA_PREP program units. Some utility units are omitted for clarity (Part 3 of 6).

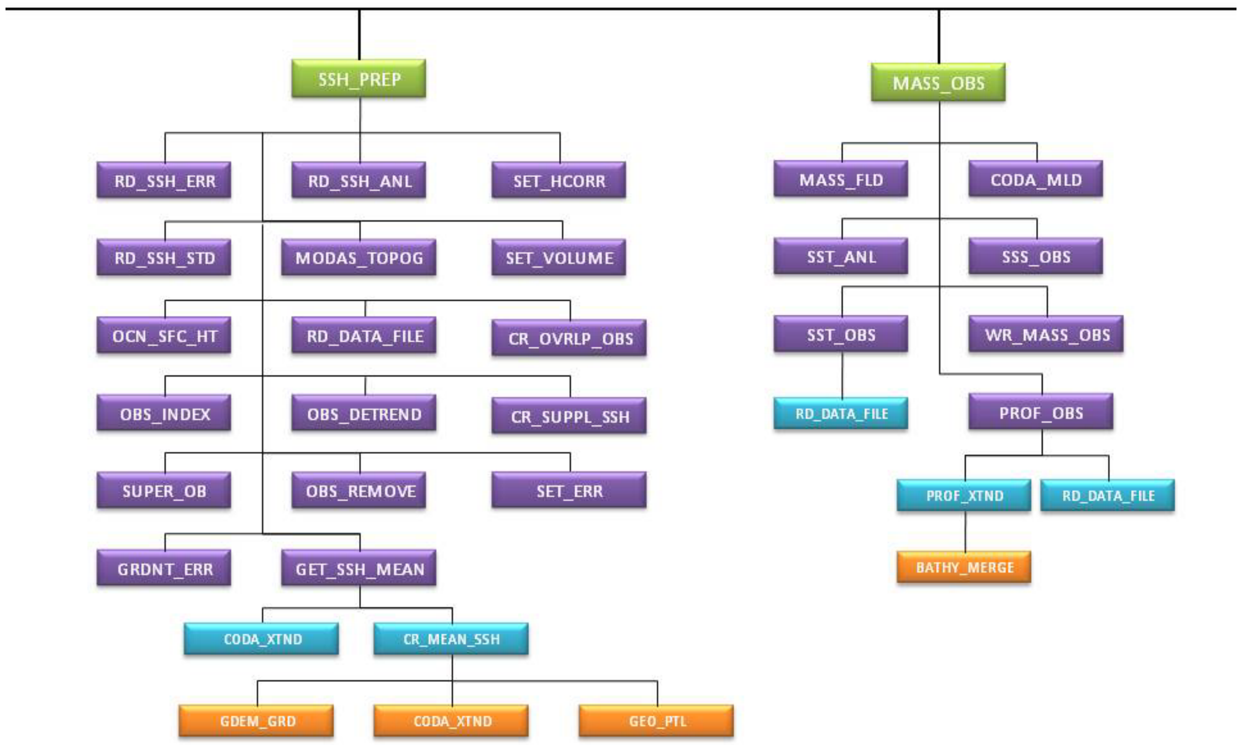

Figure 2.1-2d. Tree diagram of the NCODA_PREP program units. Some utility routines are omitted for clarity (Part 4 of 6).

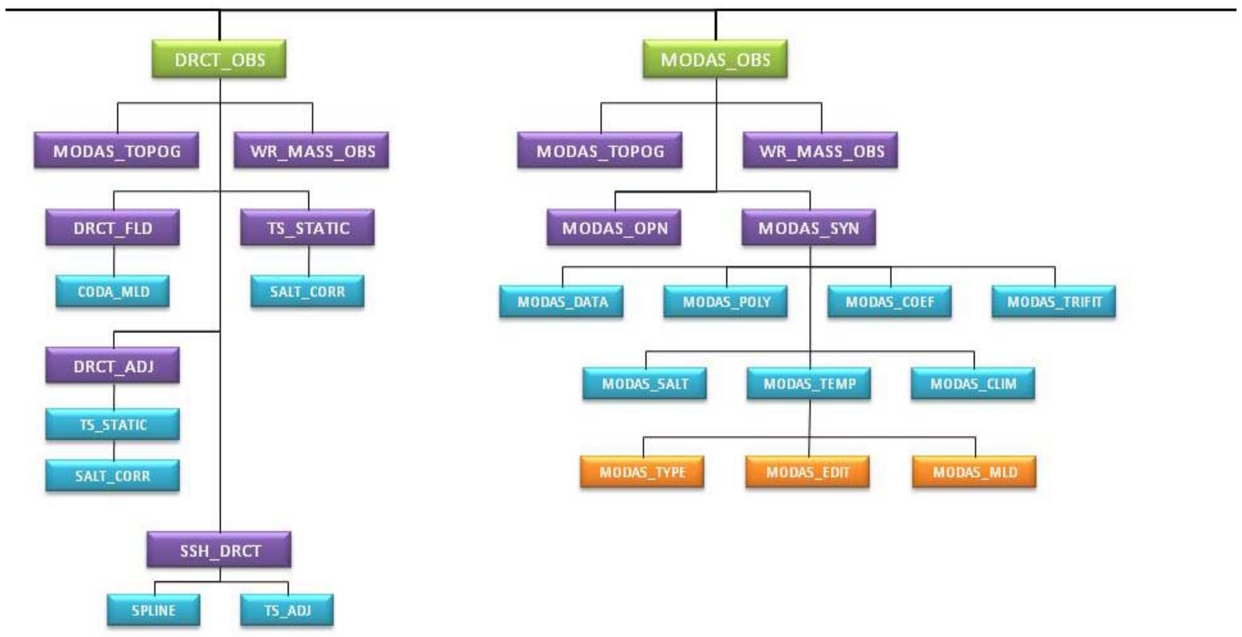

Figure 2.1-2e. Tree diagram of the NCODA_PREP program units. Some utility routines are omitted for clarity (Part 5 of 6).

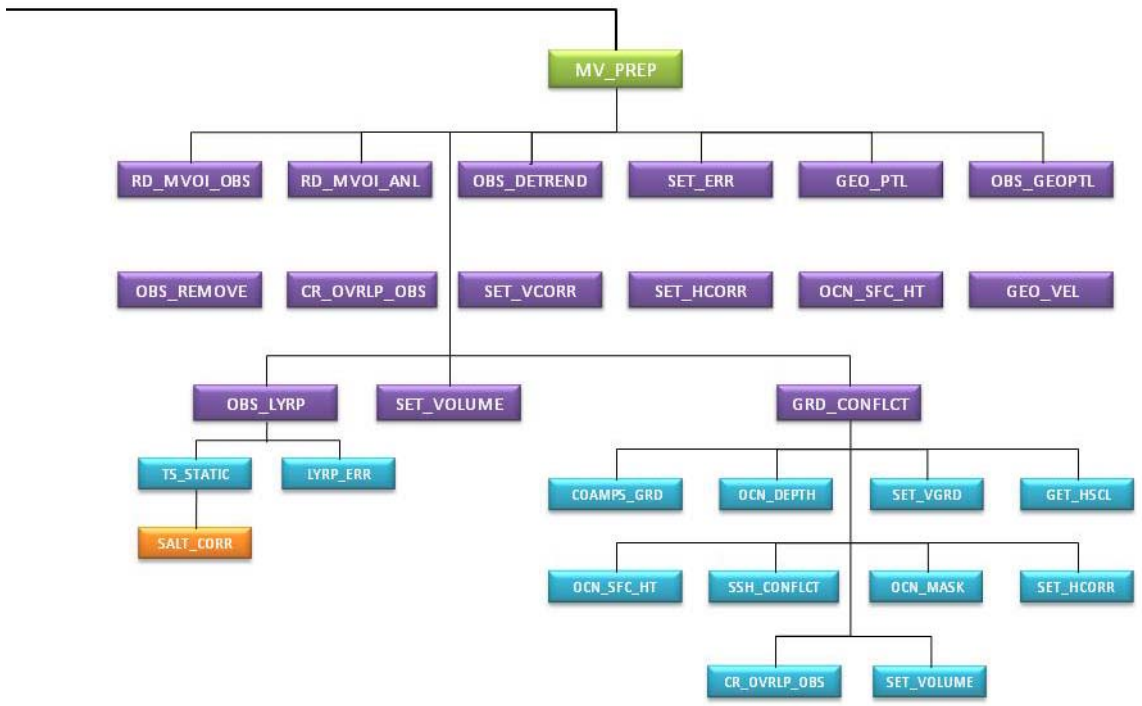

Figure 2.1-2f. Tree diagram of the NCODA_PREP program units. Some utility routines are omitted for clarity (Part 6 of 6).

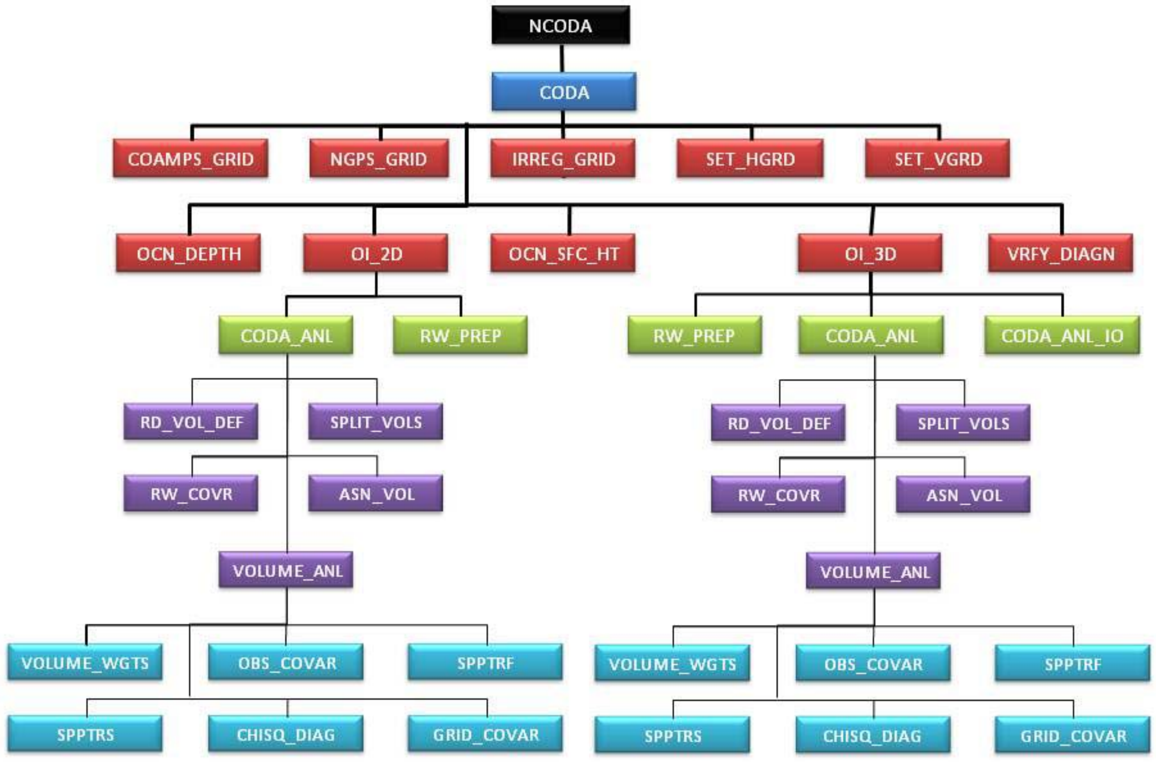

Figure 2.1-3. Tree diagram of the NCODA program units. Note that utility routines like DTGOPS and RW_DATAO, which are called from many locations, are omitted for clarity.

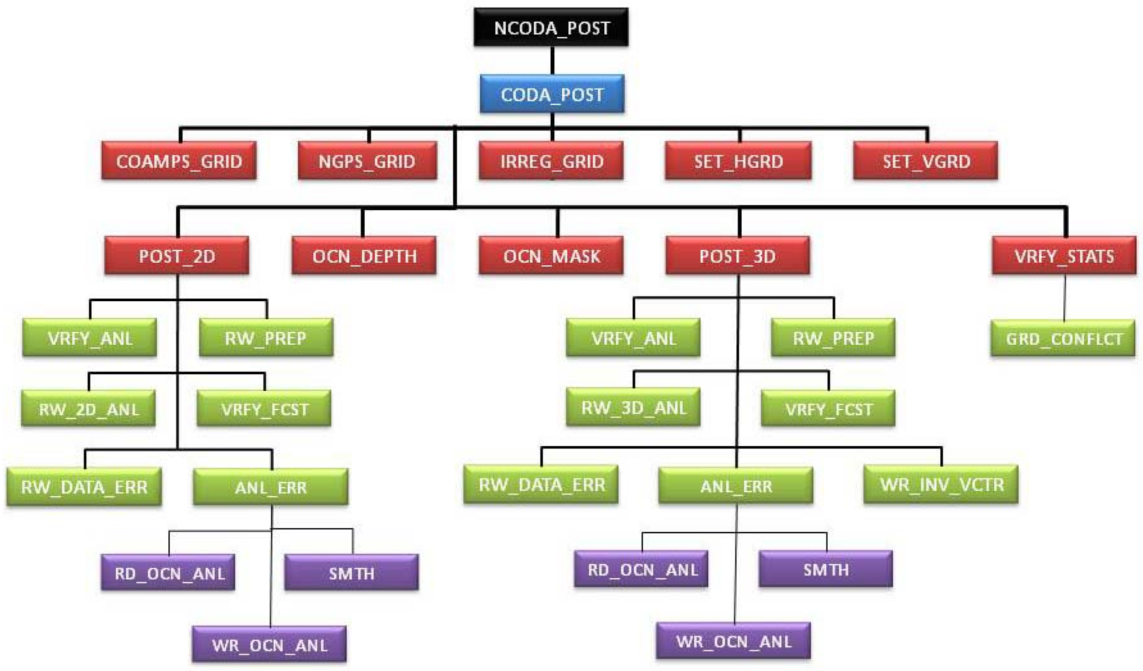

Figure 2.1-4. Tree diagram of the NCODA_POST program units. Note that some utility routines, which are called from many locations, are omitted for clarity.

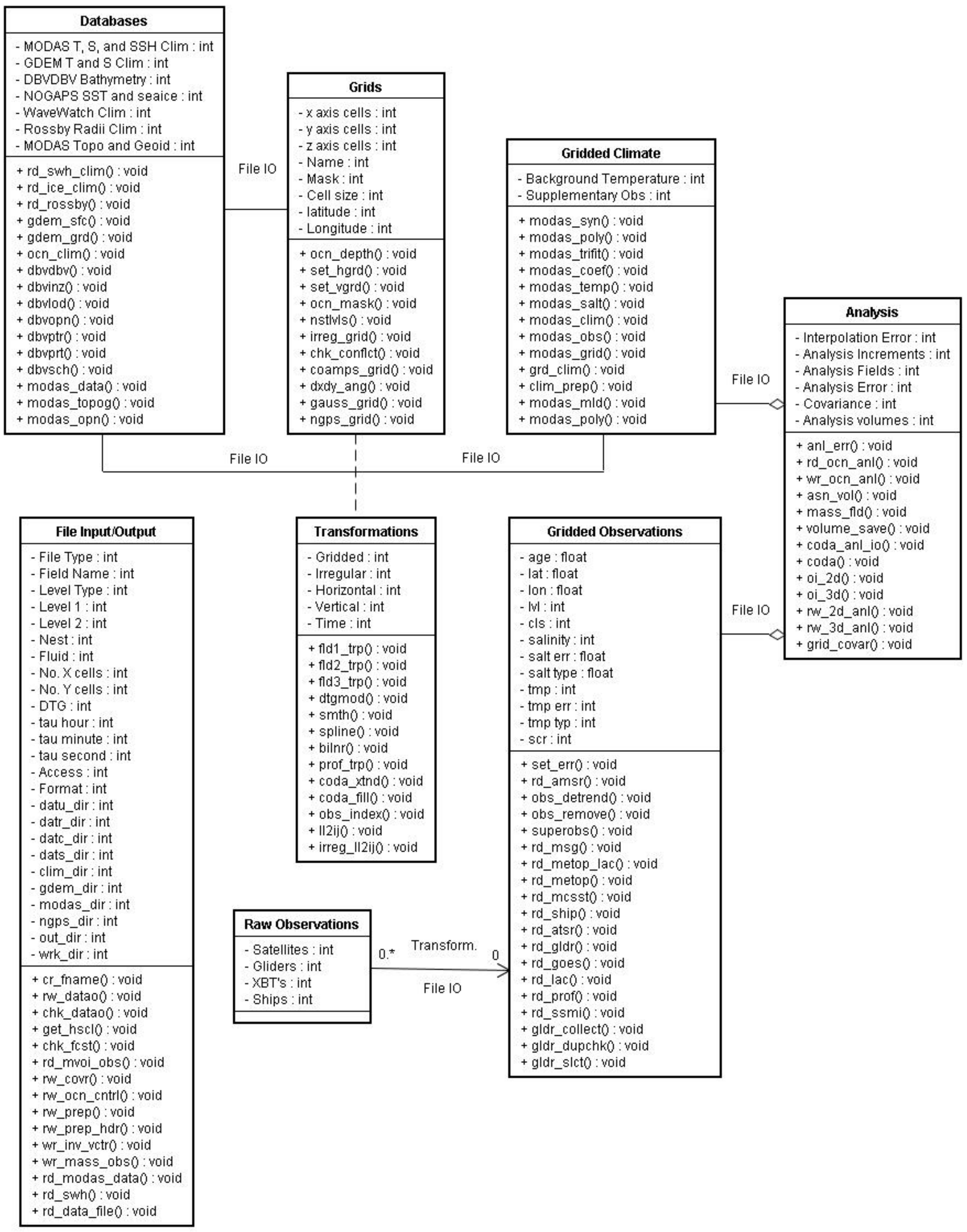

Figure 2.2-1. UML Class Diagram for the NCODA system. The FORTRAN file structure has been interpreted to an object oriented structure. Not all subroutines are shown. This diagram is general only. See the text for an explanation.

3 System input/output

3.1 Directories

There are a maximum of 4 data directories, *datu_dir* (unclassified data), *datr_dir* (restricted data), *datc_dir* (confidential data), *and dats_dir* (secret data), in addition to the climatology directories, *clim_dir*, *gdem_dir*, *modas_dir*, the global atmospheric forcing *ngps_dir*, the output directory, *out_dir*, and the work directory, *wrk_dir*. If none of the input directories is given, execution stops. The number is stored in *n_dir* for later use and the names are kept in array **data_dir(4).**

In addition, the data come from subdirectories named after the observation sources: *mcsst, metop, lac, metop-lac, goes, amsr, atsr, msg, ship, glider, profile.*

3.2 The grid file

This direct access file is a restart file as described in section 3.6 but it has a special content; it contains one record, **datao,** which is a 1D array. It is located in *out_dir*. If this file is absent, restart is set equal to true.

Example: *datahd_sfc_000000_000000_1w0545x0536_2005030100_00240000_infofld*

The input grid data are found in the first 20 entries. Projection-specific grid data are located in entries 30-46. Parent grid, grid nest level and children grid info start with index 47. Analysis vertical grid info starts with index 501. Finally, model vertical grid input from a previous file starting with index 801, otherwise local dummy variables will be present.

3.3 Climatology Files

3.3.1 Wavewatch

The global wave climate comes from a file called "*clim_dir/*Wavewatch.swh.clim," which has 1441 × nodes and 721 *y* nodes. This must be the size of the file if it exits. The base lat/lon is hardwired into the subroutine (-90, 0). The grid is computed from the size.

rec 1:	*nx, ny*	
rec 2:	**wrk (1441 × 721)**	climate error
rec 3:	**wrk (1441 × 721)**	model error
rec 4:	**wrk (1441 × 721)**	climatology

3.3.2 Sea Ice from ECMWF

Sea ice climatology comes the ECMWF database, which is contained in the file, "*clim_dir/*ECMWF.ice_clim," which contains only one record*:*

clm(361 × 181)

3.3.3 GDEM climatology Files

These files are contained in the *clim_dir* directory.

gdem_temp*mon*.short, gdem_tstd*mon*.short
gdem_salt*mon*.short, gdem_sstd*mon*.short

All contain 2 records: 1. *add_offset, scale_factor, missing_value*
2. an array of 1441 × 689 × 78 entries

3.3.4 MODAS

The MODAS climate fields use a smoothed topography and geoid correction; these are contained in files, "*clim_dir/*MODAS.topo" and "*clim_dir/*MODAS.geoid". Each of these contains one record with *nx*=1440 and *ny*=721 entries.

The MODAS data files use a naming convention like "m180+30_12.b" and contain 4 records:

1. *i1, j1, m1, n1, mr1, mi1, nt1*
2. *wrk_beg, wrk_end*, **wrk1** (*nt1*), **wrk2** (*nt1*), **wrk3** (*nt1*), **wrk4** (*nt1*)
3. **iwork** (*mi1*)
4. **work** (*mr1*)

These files are unpacked to produce lat, lon, ssh, deep T and S, and mixed layer dept (MLD).

3.4 Observation Data Files

The observations processed by the NCODA system come from satellites, ships, xbt's, gliders, and some supplemental profiles from MODAS or GDEM can be used. These data are cleaned up and processed to a uniform level of quality and written to temporary files described in section 3.5.

The formats for the files are set within the NCODA_PREP component and are not changeable. The names are constructed from the observation source and date/time group (DTG). The temporal sequence of accessing these files is conveniently displayed using a UML sequence diagram, which shows the interaction (dynamic behavior) of the program objects. This method has been applied to the interaction of NCODA with the file system as shown in Figure 3.4-1. The objects listed across the top of the diagram are: (1) the NCODA_PREP program; (2) the work files for the observations; (3) the restart files (discussed in section 3.6); and (4) the input data files. Execution time is represented downward. Arrows show the direction of information flow. The sequence for each data type is: (i) the work files are opened, (ii) the data files (e.g., MCSST) are accessed and the number of observations counted, (iii) the file is read, and (iv) the quality checked observations are written to the work files after processing. Each object's time line (vertical) is marked with X when it terminates. Each interaction line (horizontal) is labeled with the specific data type (object) on the right and with the activity and subroutine (ALL CAPS) on the left.

The data files are read during processing of the first nest (*nest* = 1). Note that the preprocessed data files are named with "nest00" hard-wired in. This indicates that they are the raw observations with minimal processing. They have not been checked for accuracy. The input parameters (e.g., age, lat, lon) have all been standardized at this stage.

The work files are accessed by the individual data prep routines for analysis (Figure 3.4-1b). The interaction of these data objects with the NCODA system is discussed in section 3.5 (Work Files) and the details of this processing are discussed in section 6 (Operation).

The following observations are contained in files opened and read within subroutines called by CR_TMP_FILE. The file naming convention is YYYYMMDDHH.*var*. Each data observation in the file consists of the following:

Multi-Channel Sea Surface Temperature (*mcsst*)*:*

rec 1: *n_read, n_lvl, (vrsn)*
rec 2: **ob_cls***(n_read)*
rec 3: *ob_glbl*
rec 4: **ob_lat***(n_read)*
rec 5: **ob_lon***(n_read)*
rec 6: **ob_age***(n_read)*
rec 7: *ob_clim*
rec 8: **ob_qc***(n_read)*
rec 9: **ob_typ***(n_read)*
rec 10: *ob_regn*
rec 11: **ob_sst***(n_read)*
rec 12: *ob_aod*
rec 13: **ob_dtg***(n_read)*
`If vrsn > 1:`
rec 14: **ob_err***(n_read)*
rec 15: *ob_wind*
rec 16: *ob_solr*
`if vrsn > 2:`
rec 17: **ob_bias***(n_read)*
rec 18: **ob_dw***(n_read)*

This is the basic format but it is not followed completely by all of the other data types. Differences will be discussed.

Meteorological Operational Satellite (*metop*): The file contents are the same but version is always read and there are no dependencies.

Geostationary Operational Environmental Satellite (*goes*)*:* For the GOES data, the procedure in sub RD_GOES is the same as MCSST through rec 16 (*vrsn* always present) unless *vrsn* > 1, in which case *ob_bias* and *ob_dw* are also read.

NOAA AVHRR Local Area Coverage (*lac*): The algorithm in RD_LAC is the same as RD_GOES.

Meteorological Operational Satellite Local Area Coverage (*metop_lac*): Sub RD_METOP_LAC reads 18 records

Metosat second generation (*msg*): The file contents are the same through rec 11, then differ:

rec 12: **ob_dtg***(n_read)*
rec 13: **ob_err***(n_read)*
rec 14: **ob_bias***(n_read)*
rec 15: **ob_dw***(n_read)*

rec 16: *b_wind*
rec 17: *ob_solr*
rec 18: *ob_aod*

Advanced Microwave Scanning Radiometer (*amsr*): After rec 1 the following sequence is expected:

rec 2: **ob_age***(n_read)*
rec 3: **ob_bias***(n_read)*
rec 4: *ob_clim*
rec 5: **ob_dw***(n_read)*
rec 6: **ob_err***(n_read)*
rec 7: *ob_glbl*
rec 8: **ob_lat***(n_read)*
rec 9: **ob_lon***(n_read)*
rec 10: **ob_qc***(n_read)*
rec 11: *ob_regn*
rec 12: *ob_solr*
rec 13: **ob_sst***(n_read)*
rec 14: **ob_typ***(n_read)*
rec 15: *ob_wind*
rec 16: **ob_cls***(n_read)*
rec 17: **ob_dtg***(n_read)*

Along-Track Scanning Radiometer (*atsr*): After rec 1 the following sequence is read:

rec 2: **ob_age***(n_read)*
rec 3: *ob_aod*
rec 4: **ob_bias***(n_read)*
rec 5: *ob_clim*
rec 6: **ob_dw***(n_read)*
rec 7: **ob_err***(n_read)*
rec 8: *ob_glbl*
rec 9: **ob_lat***(n_read)*
rec 10: **ob_lon***(n_read)*
rec 11: **ob_qc***(n_read)*
rec 12: *ob_regn*
rec 13: *ob_solr*
rec 14: **ob_sst***(n_read)*
rec 15: **ob_typ***(n_read)*
rec 16: *ob_wind*
rec 16: **ob_cls***(n_read)*
rec 17: **ob_dtg***(n_read)*

On-board ship SST (*ship*): The format for the input file reads *vrsn* from rec 1 only if *file_dtg* > '2002100100' (as with MCSST). The following are then read.

rec 2: **ob_cls***(n_read)*
rec 3: *ob_glbl*

rec 4: **ob_lat***(n_read)*
rec 5: **ob_lon***(n_read)*
rec 6: **ob_age***(n_read)*
rec 7: *ob_clim*
rec 8: **ob_qc***(n_read)*
rec 9: *ob_regn*
rec 10: **ob_sst***(n_read)*
rec 11: **ob_typ***(n_read)*
rec 12: **ob_dtg***(n_read)*
rec 13: *ob_rcp*
rec 14: **ob_scr***(n_read)*
rec 15: **ob_sgn***(n_read)*

Expendable bathythermographs (*profile*): The algorithm is similar to the other data bases but some different variables are introduced:

rec 1: *n_read, mx_lvl, version*
rec 2: **ob_btm***(n_read)*
rec 3: **ob_lat***(n_read)*
rec 4: ***ob_lon****(n_read)*
rec 5: *ob_ls*
rec 6: **ob_lt***(n_read)*
rec 7: **ob_ssh***(n_read)*
rec 8: **ob_sst***(n_read)*
rec 9: **ob_sal_typ***(n_read)*
rec 10: **ob_sqc***(n_read)*
rec 11: **ob_tmp_typ***(n_read)*
rec 12: **ob_tqc***(n_read)*
`Read for each record (n_read)`
    rec 13: **ob_lvl(ob_lt***(n_read), n_read)* (first)
    rec 14: **ob_sal(ob_lt***(n_read), n_read)*
    rec 15: **ob_sal_err(ob_lt***(n_read), n_read)*
    rec 16: **ob_sprob(ob_lt***(n_read), n_read)*
    rec 17: **ob_tmp(ob_lt***(n_read), n_read)*
    rec 18: **ob_tmp_err(ob_lt***(n_read), n_read)*
    rec 19 (12 + 7×*n_read*): **ob_tprob(ob_lt***(n_read), n_read)* (last)
rec (12 + 7×*n_read* + 1): **ob_dtg***(n_read)*
rec (12 + 7×*n_read* + 2): **ob_rct***(n_read)*
rec (12 + 7×*n_read* + 3): **ob_scr***(n_read)*
rec (12 + 7×*n_read* + 4): **ob_sgn***(n_read)*
`Read for each record (n_read)`
    rec (16 + 7×*n_read* +1): **ob_clm_sal(ob_lt***(n_read), n_read)* (first)
    rec (16 + 7×*n_read* +2): *ob_clm_tmp*
    rec (16 + 7×*n_read* +3): **ob_cssd(ob_lt***(n_read), n_read)*
    rec (16 + 7×*n_read* +4): **ob_ctsd(ob_lt***(n_read), n_read)*
    rec (16 + 7×*n_read* +5): *ob_glb_sal*
    rec (16 + 7×*n_read* +6): *ob_glb_tmp*

rec (16 + 7×*n_read* +7):	*ob_gssd*
rec (16 + 7×*n_read* +8):	*ob_gtsd*
rec (16 + 7×*n_read* +9):	*ob_mds_sal*
rec (16 + 7×*n_read* +10):	*ob_mds_tmp*
rec (16 + 7×*n_read* +11):	*ob_rgn_sal*
rec (16 + 7×*n_read* +12):	*ob_rgn_tmp*
rec (16 + 7×*n_read* +13):	*ob_rssd*
rec (16 + 7×*n_read* +14**n_read*):	*ob_rtsd* (last)

`If` *`vrsn`* `> 1:`

`Read for each record (`*`n_read`*`)`

rec (16 + 7×*n_read* + 14×*n_read* +1):	*sal_xval*	(first)
rec (16 + 7×*n_read* + 14×*n_read* +2):	*sal_xstd*	
rec (16 + 7×*n_read* + 14×*n_read* +3):	*tmp_xval*	
rec (16 + 7×*n_read* + 14×*n_read* +4×*n_read*):	*tmp_xstd*	(last)

`if` *`vrsn`* `> 2:`

`Read for each record (`*`n_read`*`)`

rec (16 + 7×*n_read* + 14×*n_read* +1/*n_read*): **ob_id***(n_read)*

Autonomous gliders (*glider)*: The data file must already exist. There is only one file structure, which is as follows:

rec 1:	*n_read, mx_lvl, vrsn*
rec 2:	**ob_lt(***n_read)*
rec 3:	**ob_mode(***n_read)*
rec 4:	**ob_ssh(***n_read)*
rec 5:	**ob_sst(***n_read)*
rec 6:	**ob_sal_typ(***n_read)*
rec 7:	**ob_sqc(***n_read)*
rec 8:	**ob_tmp_typ(***n_read)*
rec 9:	**ob_tqc(***n_read***)**
rec 10:	**ob_rct**(*n_read*)
rec 11:	**ob_scr**(*n_read*)
rec 12:	**ob_sgn**(*n_read*)
rec 13:	**ob_id**(*n_read*)

Loop over all *n_read* observations:

rec (13 + 1):	**ob_btm**(**ob_lt**(*n_read)*)	first
rec (13 + 2):	**ob_dtg(ob_lt(***n_read)*)	
rec (13 + 3):	**ob_lat(ob_lt**(*n_read*))	
rec (13 + 4):	**ob_lon(ob_lt**(*n_read*))	
rec (13 + 5):	**ob_lvl(ob_lt**(*n_read*))	
rec (13 + 6):	**ob_sal(ob_lt**(*n_read*))	
rec (13 + 7):	**ob_sal_err(ob_lt**(*n_read)*)	
rec (13 + 8):	**ob_sprob(ob_lt**(*n_read*))	
rec (13 + 9):	**ob_tmp(ob_lt**(*n_read*))	
rec (13 + 10):	**ob_tmp_err(ob_lt**(*n_read)*)	
rec (13 + 11):	**ob_tprob(ob_lt**(*n_read))*	
rec (13 + 12):	**ob_clm_sal(ob_lt**(*n_read*))	

rec (13 + 13): *ob_clm_tmp*
rec (13 + 14): **ob_cssd(ob_lt**(*n_read*))
rec (13 + 15): **ob_ctsd(ob_lt**(*n_read*))
rec (13 + 16): *ob_glb_sal*
rec (13 + 17): *ob_glb_tmp*
rec (13 + 18): *ob_gssd*
rec (13 + 19): *ob_gtsd*
rec (13 + 20): *ob_rgn_sal*
rec (13 + 21): *ob_rgn_tmp*
rec (13 + 22): *ob_rssd*
rec (13 + 23): *ob_rtsd*
rec (13 + 24): *sal_xval*
rec (13 + 25): *tmp_xval*
rec (13 + 26): *sal_xstd*
rec (13 + 27**n_read*): *tmp_xstd* last

Subroutine CR_ICE_FILE reads the sea ice file as follows.

Special Sensor Microwave Imager (*ssmi*): The unformatted input files are of the form are read as follows:

rec 1: *n_read*, *n_lvl*, *vrsn*
rec 2: *ob_glbl*
rec 3: **ob_ice**(*n_read*)
rec 4: **ob_la**t(*n_read*)
rec 5: **ob_lon**(*n_read*)
rec 6: **ob_qc**(*n_read*)
rec 7: **ob_age**(*n_read*)
rec 8: *ob_regn*
rec 9: **ob_sat**(*n_read*)
rec 10: *ob_clim*
rec 11: *ob_dmy*
rec 12: **ob_dtg**(*n_read*)

Subroutine CR_SSH_FILE reads the altimetry observation file.
altim:
rec 1: *n_read*, *n_lvl*, *vrsn* (note that this file uses the *vrsn_dtg* date)
rec 1: *n_read*, *n_lvl*
rec 2: **ob_age**(*n_read*)
rec 3: *ob_clim*
rec 4: *ob_cycle*
rec 5: *ob_glbl*
rec 6: **ob_lat**(*n_read*)
rec 7: **ob_lon**(*n_read*)
rec 8: **ob_qc**(*n_read*)
rec 9: *ob_regn*
rec 10: *ob_smpl*

rec 11: **ob_sat**(*n_read*)
rec 12: *ob_std*
rec 13: **ob_ssh**(*n_read*)
rec 14: *ob_track*
rec 15: **ob_dtg**(*n_read*)
if *vrsn* > 1:
    rec 16: **ob_rcpt**(*n_read*)
else:
    rec (15 + 1,...*n_read*): **ob_dtg**(*n_read*)

Subroutine CR_SWH_FILE reads the significant wave height observations.

swh:

rec 1: *n_read*, *n_lvl*, *vrsn*
rec 2: *ob_glbl*
rec 3: **ob_lat**(*n_read*)
rec 4: **ob_lon**(*n_read*)
rec 5: **ob_age**(*n_read*)
rec 6: *ob_clim*
rec 7: **ob_qc**(*n_read*)
rec 8: **ob_typ**(*n_read*)
rec 9: *ob_regn*
rec 10: **ob_swh**(*n_read*)
rec 11: *ob_wind*
rec 12: *ob_xval*
rec 13: **ob_dtg**(*n_read*)
rec 14: **ob_rcpt**(*n_read)*

3.5 Work files

The NCODA system uses a series of work files to communicate between components. These files are sequential and unformatted. They are written to the *wrk_dir* directory. The work files are also called "prep" files and they all begin with "coda". They contain parameters associated with the preparation of the analysis (e.g., observations, covariances). The names are constructed as follows:

coda. *par_content*.nest*nn*.*dtg*

where *par* = ICE, PRF, SSH, SWH, SST, VEL, HDR, MVOI, SFC, SYN, VELOC, and MASS; *content* = obs, 2D, 3D, cvr, and vol; *nn* = 00 - 09; and *dtg* = YYYYMMDDHH. Not all of the possible combinations are used. Table 3.5 lists the files that are used and the subroutines in which they are accessed. The table also indicates whether the subroutine opens (O), closes (C), reads (R), or writes (W) to the file.

The analysis results are written to the restart files discussed in section 3.6.

These files contain different numbers of records. In addition, there are four files that are hardwired for nest 00 because they contain the original observations after quality control has been completed. These files are written during the first nest loop.

The PRF file contains the temperature and salinity observations from xbt's and gliders (xbt's first), and the SSH file contains the ssh observations from the xbt's.

coda.PRF_obs.nest00.*dtg*: Records = *n_xbt; age; lat; lon; lvl; ndx; sal; sal_err; sal_typ; tmp; tmp_err; tmp_typ; scr*

coda.SSH_obs.nest00.*dtg*: Records = *n_alt; age; lat; lon; lvl; cls; res; dmy; idm; ssh; err; typ; cls*

where there are *n_xbt* /*n_alt* entries for each variable.

The ICE and SST files for nest 00 contain surface ice concentration data (*ssmi*) and sst after quality control.

coda.ICE_obs.nest00.*dtg*: Records = *num, age, lat, lon, lvl, cls, res, dmy, cls, ice, err, typ, cls*

coda.SST_obs.nest00.*dtg*: Records = *num; age, lat, lon, lvl, cls, res, dmy, idm, sst, err, typ, idm*

The SST file is cumulative, so that the records include the sst sources in the following order: *mcsst; metop; goes; lac; metop_lac; msg; amsr; atsr; ship*. Each of these observation sets includes the records listed above.

coda.SFC_obs.nest*nn.dtg*: Records = *n_sst_obs; age; lat; lon; lvl; cls; dmy; dmy; wrk; val; err; typ; scr*

coda.(ICE, SSH, MVOI, SST, SWH)_vol.nest*nn.dtg*: Records = *n_vol; nodes/vol; obs/vol; vx1; vx2; vy1; vy2*

coda. (ICE, SSH, MVOI, SST, SWH)_obs.nest*nn.dtg*: Records = *n_total; age; lat; lon; lvl; msv; ndx; sal; sal_err; sal_typ; tmp; tmp_err; tmp_typ; xi; yj; zk*

coda. (ICE, SSH, MVOI, SST, SWH)_cvr.nest*nn.dtg*: Records = *lon, lat, lvl; hcr; vcr; msk*

The final file is the SYN file, which contains synthetic observations from altimeters (*direct* = T) or MODAS (*modas* = T). There is only an obs file for these data.

coda.SYN_obs.*nn.dtg*: Records = *n_total*; *lat*; *lon*; *lvl*; *surface index*; *profile index*; *salinity*; *salinity error*; *salinity data type*; *temp*; *temp error*; *temp data type*; *x coord*; *y coord*; *z coord*; *security class*.

A UML sequence diagram (Figure 3.5-1) shows the interaction between NCODA_PREP (OCEAN_QC) and the file server in writing and reading these files. The vertical lines to the left of the OCEAN_QC timeline indicate the value of the *swh* parameter (T or F). Most of the operations are writing except for MVOI_PREP, which reads the observations for each nest.

A special kind of work (prep) file is the header file (*par* = HDR and *content* = 2D or 3D), which contains data on the number of observations and analysis:

coda.HDR_*opt*.nest*nn.dtg* contains 1 record: *n_data, n_ice, n_ssh, n_sst, n_swh, n_prf_lvl, n_prf_obs, n_ssh_obs, n_vel_obs, n_vol, file_dtg, n_anl, f3d, fcst, restart*

These files are used for communication between and within components of the NCODA system as shown in a UML sequence diagram (Figure 3.5-2). As seen in the diagram, CODA_PREP writes to these files after both 2D and 3D analyses and the other components read them. This assures continuity in the analysis when mulitiple (independent) program units are used. This will be discussed more in section 6 (Operation).

3.6 Restart Files

The results of the analysis are written to direct access files that are called CODA restart (CR) files. These files contain only one record and all of the necessary parameters for reading them are contained in the file name. The CODA restart file name is built from the variables listed in Table 3.6.

A typical file name looks like this:

out_dir/datahd_sfc_000000_000000_1w0545x0536_2005030100_00240000_infofld

These files typically contain one field with the dimensions given in the file name. Figure 3.6-1 shows interaction between the ocean QC component (NCODA_PREP) and the file system for the 2D option. The sequence for the 3D option is shown in Figure 3.6-2. These files will be used by external applications like data assimilation as well as subsequent analyses. They are also used for system debugging and diagnostics. An exception is the "infofld" file as described in section 3.2.

3.7 NOGAPS boundary files

Surface values of sea ice and surface temperature can be read from one of the following NOGAPS files.

```
seaice_sfc_0000.0_0000.0_glob720x360_YYYYMMDDHH_00000000_fcstfld
seaice_sfc_0000.0_0000.0_glob360x181_YYYYMMDDHH_00000000_fcstfld
seaice_sfc_000000_000000_1a720x360_YYYYMMDDHH_00000000_fcstfld
seaice_sfc_000000_000000_1a360x181_YYYYMMDDHH_00000000_fcstfld
iceca1YYYYMMDDHH0000000000000000sfl
```

Similar files are used for seatmp (*tsea*). The "icecal" file also comes with either resolution fields, which are checked in a loop in subroutine NGPS_BNDY.

3.8 Summary of file IO within CODA

This section has described the different kinds of files used by the NCODA system. The most complex file system access occurs within the NCODA_PREP program because it must read in data from a number of different sources. However, because the prep component does this, file access within the other components is relatively straightforward. There is an exception to this general simplicity; the entire NCODA system must be executed twice (see section 6), first in 2D mode and then in 3D mode. The 2D analysis is restricted to surface fields but it also computes all of the horizontal analysis parameters. A subsequent 3D analysis thus can read these parameters from files and does not need to recalculate them.

A UML sequence diagram (Figure 3.8-1) demonstrates the interaction of the CODA program with the file system. The ocean data record (CR file = datao) is read before the nests are analyzed individually. The prep results comprise the header file, depths, ssh, observations, analysis volumes and covariances required for the objective interpolation (OI). The first ocean variable processed is sea ice, for which analysis increments, time increments, and analysis errors are written to restart files. This sequence of input and output is then repeated for significant wave height, if selected (*swh* = T). If *swh* = T, no other 2D analyses are completed for the current nest and the statistice file (CR file = chisqr) is written. If *swh* = F, however, this sequence of input and output is repeated for sst (CR file = seatmp) and sea surface height (CR file = seahgt) (*opt* = 3D).

On the next execution of the NCODA system, the 3D option must be selected and the CODA program has more interaction with the file system (Figure 3.8-2). The above sequence of input and output is repeated for the analysis increment of 3d temperature (seatmp_pre), salinity (CR file = salint_pre), geopotential anomaly (CR file = geoptl_pre), currents (CR files = uucurr_pre and vvcurr_pre), and the time increment for the observation age (CR file = grdage_pre).

Table 3.5

File	**Subroutines***
coda.ICE_obs.nest00.DTG	OCN_OBS (O, C), RD_SSMI (W), RD_DATA_FILE (R)
coda.PRF_obs.nest00.DTG	OCN_OBS (O, C), PROF_COLLECT (W), RD_DATA_FILE (R)
coda.SSH_obs.nest00.DTG	OCN_OBS (O, C), PROF_COLLECT (W), RD_DATA_FILE (R)
coda.SWH_obs.nest00.DTG	OCN_OBS (O, C), RD_SWH (W), RD_DATA_FILE (R)
coda.SST_obs.nest00.DTG	OCN_OBS (O, C), RD_MCSST (W), RD_METOP (W), RD_GOES (W), RD_METOP_LAC (W), RD_MSG (W), RD_AMSR (W), RD_ATSR (W), RD_SHIP (W), RD_DATA_FILE (R)
coda.VEL_obs.nest00.DTG	Not used at this time.
coda.HDR_2D.nestNN.DTG	RW_PREP_HDR (R, W)
coda.HDR_3D.nestNN.DTG	RW_PREP_HDR (R, W)
coda.ICE_cvr.nestNN.DTG	RW_COVR (R, W)
coda.ICE_obs.nestNN.DTG	RW_PREP (R, W), RD_MVOI_OBS (R), RD_DATA_FILE (R), WR_MASS_OBS (W)
coda.ICE_vol.nestNN.DTG	SET_VOLUME (W), RD_VOL_DEF (R)
coda.MVOI_cvr.nestNN.DTG	RW_COVR (R, W)
coda.MVOI_obs.nestNN.DTG	RW_PREP (R, W), RD_MVOI_OBS (R), RD_DATA_FILE (R), WR_MASS_OBS (W)
coda.MVOI_vol.nestNN.DTG	SET_VOLUME (W) , RD_VOL_DEF (R)
coda.SFC_obs.nestNN.DTG	SST_PREP (W)
coda.SSH_cvr.nestNN.DTG	RW_COVR (R, W)
coda.SSH_obs.nestNN.DTG	RW_PREP (R, W), RD_MVOI_OBS (R), RD_DATA_FILE (R), WR_MASS_OBS (W)
coda.SSH_vol.nestNN.DTG	SET_VOLUME (W) , RD_VOL_DEF (R)

coda.SST_cvr.nestNN.DTG	RW_COVR (R, W)
coda.SST_obs.nestNN.DTG	RW_PREP (R, W), RD_MVOI_OBS (R), RD_DATA_FILE (R), WR_MASS_OBS (R)
coda.SST_vol.nestNN.DTG	SET_VOLUME (W) , RD_VOL_DEF (R)
coda.SWH_obs.nestNN.DTG	RW_PREP (R, W), RD_MVOI_OBS (R)
coda.SWH_cvr.nestNN.DTG	RW_COVR (R, W)
coda.SWH_vol.nestNN.DTG	SET_VOLUME (W) , RD_VOL_DEF (R)
coda.SYN_obs.nestNN.DTG	WR_MASS_OBS (W), RD_MVOI_OBS (R)
coda.VELOC_obs.nestNN.DTG	Not used at this time
coda.MASS_obs.nestNN.DTG	WR_MASS_OBS (W), RD_MVOI_OBS (R)
***O = open, C = close, W = write, R = read**	

Table 3.6

Variable	Value(s)	FORTRAN Format
`fld_name`	datahd, seaice, seatmp, seahgt, salint, geoptl, uucurr, vvcurr, sigwht, cvstat, grdage, lyrprs, chisqr, grdlvl, codaoi, depths, grdlat, grdlon, icec, tsea, movehd, mixlyr, grdscl	A6
`lvl_typ`	sfc, pre	A3
Lev1	NA	I6.6
Lev2	NA	I6.6
Nest	1-9	I1
Fluid	o, w	A1
m	NA	i4.4
n	NA	i4.4
file_dtg	YYYYMMDDHH	a10
tau_hr	e.g. 0-24	i2.2
tau_mn	0	i2.2
tau_sc	0	i2.2
file_typ	infofld, analinc, timeinc, obsdata, datafld, obsfcst, fcstfld, analerr, climfld, climerr, analfld, voldata, dataerr, timefld, fcsterr, modlerr, corrfld, meanfld	a7

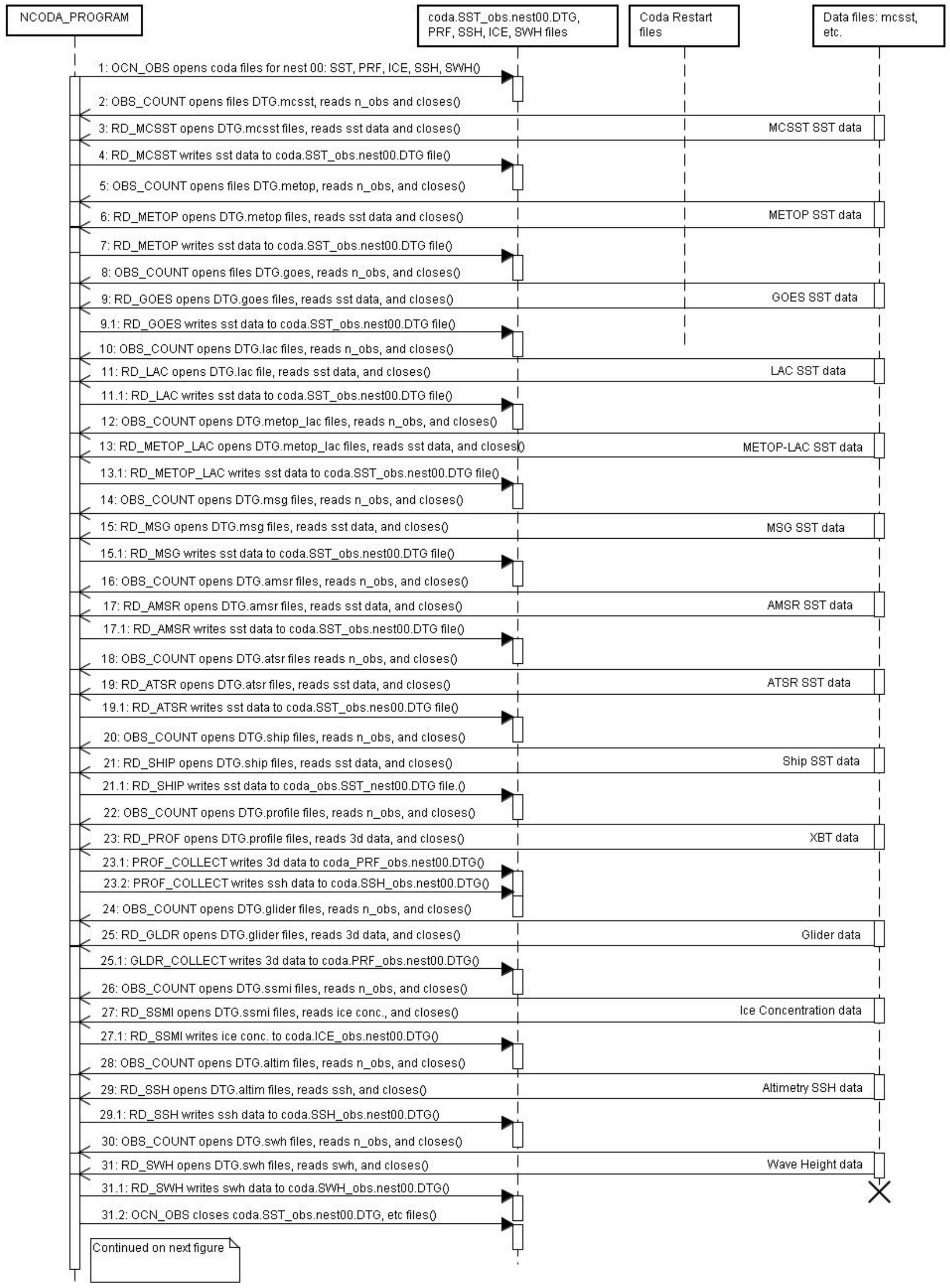
NCODA_PROGRAM
coda.SST_obs.nest00.DTG, PRF, SSH, ICE, SWH files
Coda Restart files
Data files: mcsst, etc.
1: OCN_OBS opens coda files for nest 00: SST, PRF, ICE, SSH, SWH()
2: OBS_COUNT opens files DTG.mcsst, reads n_obs and closes()
3: RD_MCSST opens DTG.mcsst files, reads sst data and closes()
MCSST SST data
4: RD_MCSST writes sst data to coda.SST_obs.nest00.DTG file()
5: OBS_COUNT opens files DTG.metop, reads n_obs, and closes()
6: RD_METOP opens DTG.metop files, reads sst data and closes()
METOP SST data
7: RD_METOP writes sst data to coda.SST_obs.nest00.DTG file()
8: OBS_COUNT opens files DTG.goes, reads n_obs, and closes()
9: RD_GOES opens DTG.goes files, reads sst data, and closes()
GOES SST data
9.1: RD_GOES writes sst data to coda.SST_obs.nest00.DTG file()
10: OBS_COUNT opens DTG.lac files, reads n_obs, and closes()
11: RD_LAC opens DTG.lac file, reads sst data, and closes()
LAC SST data
11.1: RD_LAC writes sst data to coda.SST_obs.nest00.DTG file()
12: OBS_COUNT opens DTG.metop_lac files, reads n_obs, and closes()
13: RD_METOP_LAC opens DTG.metop_lac files, reads sst data, and closes()
METOP-LAC SST data
13.1: RD_METOP_LAC writes sst data to coda.SST_obs.nest00.DTG file()
14: OBS_COUNT opens DTG.msg files, reads n_obs, and closes()
15: RD_MSG opens DTG.msg files, reads sst data, and closes()
MSG SST data
15.1: RD_MSG writes sst data to coda.SST_obs.nest00.DTG file()
16: OBS_COUNT opens DTG.amsr files, reads n_obs, and closes()
17: RD_AMSR opens DTG.amsr files, reads sst data, and closes()
AMSR SST data
17.1: RD_AMSR writes sst data to coda.SST_obs.nest00.DTG file()
18: OBS_COUNT opens DTG.atsr files reads n_obs, and closes()
19: RD_ATSR opens DTG.atsr files, reads sst data, and closes()
ATSR SST data
19.1: RD_ATSR writes sst data to coda.SST_obs.nes00.DTG file()
20: OBS_COUNT opens DTG.ship files, reads n_obs, and closes()
21: RD_SHIP opens DTG.ship files, reads sst data, and closes()
Ship SST data
21.1: RD_SHIP writes sst data to coda_obs.SST_nest00.DTG file.()
22: OBS_COUNT opens DTG.profile files, reads n_obs, and closes()
23: RD_PROF opens DTG.profile files, reads 3d data, and closes()
XBT data
23.1: PROF_COLLECT writes 3d data to coda_PRF_obs.nest00.DTG()
23.2: PROF_COLLECT writes ssh data to coda.SSH_obs.nest00.DTG()
24: OBS_COUNT opens DTG.glider files, reads n_obs, and closes()
25: RD_GLDR opens DTG.glider files, reads 3d data, and closes()
Glider data
25.1: GLDR_COLLECT writes 3d data to coda.PRF_obs.nest00.DTG()
26: OBS_COUNT opens DTG.ssmi files, reads n_obs, and closes()
27: RD_SSMI opens DTG.ssmi files, reads ice conc., and closes()
Ice Concentration data
27.1: RD_SSMI writes ice conc. to coda.ICE_obs.nest00.DTG()
28: OBS_COUNT opens DTG.altim files, reads n_obs, and closes()
29: RD_SSH opens DTG.altim files, reads ssh, and closes()
Altimetry SSH data
29.1: RD_SSH writes ssh data to coda.SSH_obs.nest00.DTG()
30: OBS_COUNT opens DTG.swh files, reads n_obs, and closes()
31: RD_SWH opens DTG.swh files, reads swh, and closes()
Wave Height data
31.1: RD_SWH writes swh data to coda.SWH_obs.nest00.DTG()
31.2: OCN_OBS closes coda.SST_obs.nest00.DTG, etc files()
Continued on next figure

Figure 3.4-1a. UML Sequence diagram of communication between NCODA_PREP and the file system for reading in the observations and writing work files. Note that the leftmost column is labelled NCODA_PROGRAM, which in this case refers to NCODA_PREP.

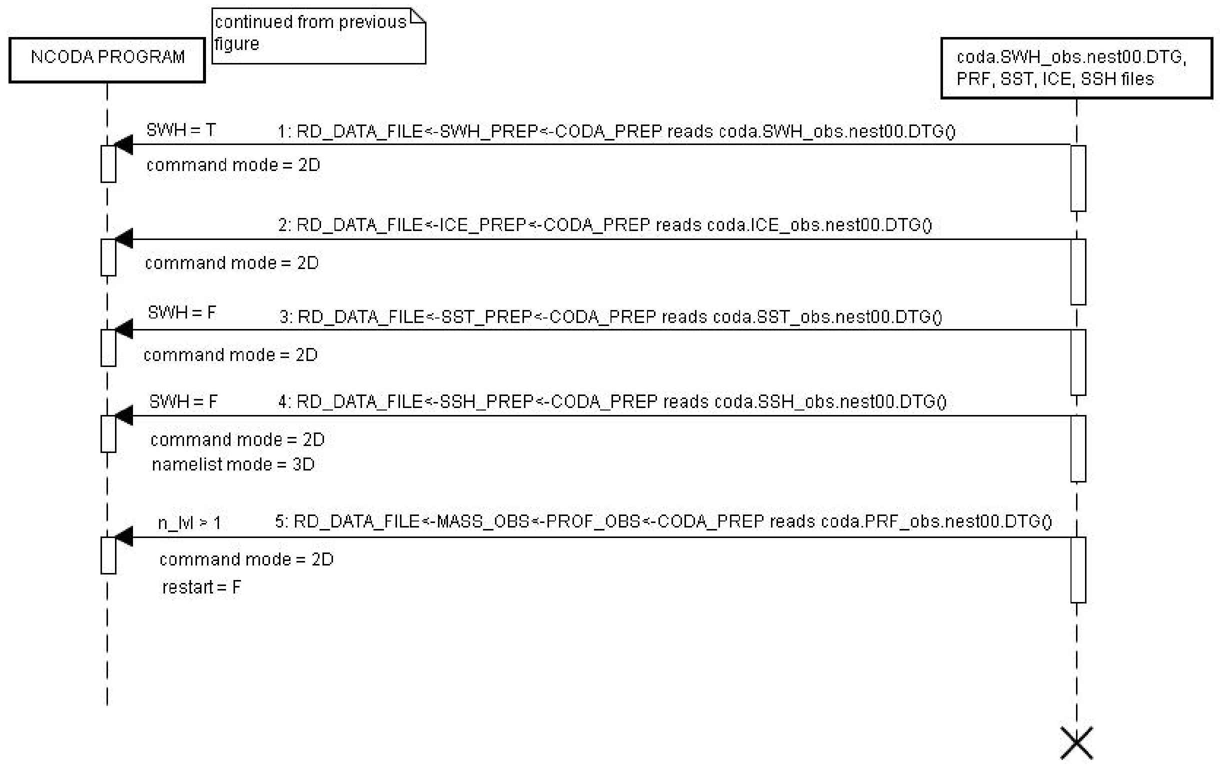
continued from previous figure
NCODA PROGRAM
coda.SWH_obs.nest00.DTG, PRF, SST, ICE, SSH files
SWH = T
1: RD_DATA_FILE<-SWH_PREP<-CODA_PREP reads coda.SWH_obs.nest00.DTG()
command mode = 2D
2: RD_DATA_FILE<-ICE_PREP<-CODA_PREP reads coda.ICE_obs.nest00.DTG()
command mode = 2D
SWH = F
3: RD_DATA_FILE<-SST_PREP<-CODA_PREP reads coda.SST_obs.nest00.DTG()
command mode = 2D
SWH = F
4: RD_DATA_FILE<-SSH_PREP<-CODA_PREP reads coda.SSH_obs.nest00.DTG()
command mode = 2D
namelist mode = 3D
n_lvl > 1
5: RD_DATA_FILE<-MASS_OBS<-PROF_OBS<-CODA_PREP reads coda.PRF_obs.nest00.DTG()
command mode = 2D
restart = F

Figure 3.4-1b. UML Sequence diagram of communication between NCODA and the file system for reading in the observations from work files. Note that the leftmost column represents the NCODA program, which reads in the work files written out in Figure 3.4-1a. The subroutines are listed on the activity line with "←" indicating a calling routine to the right.

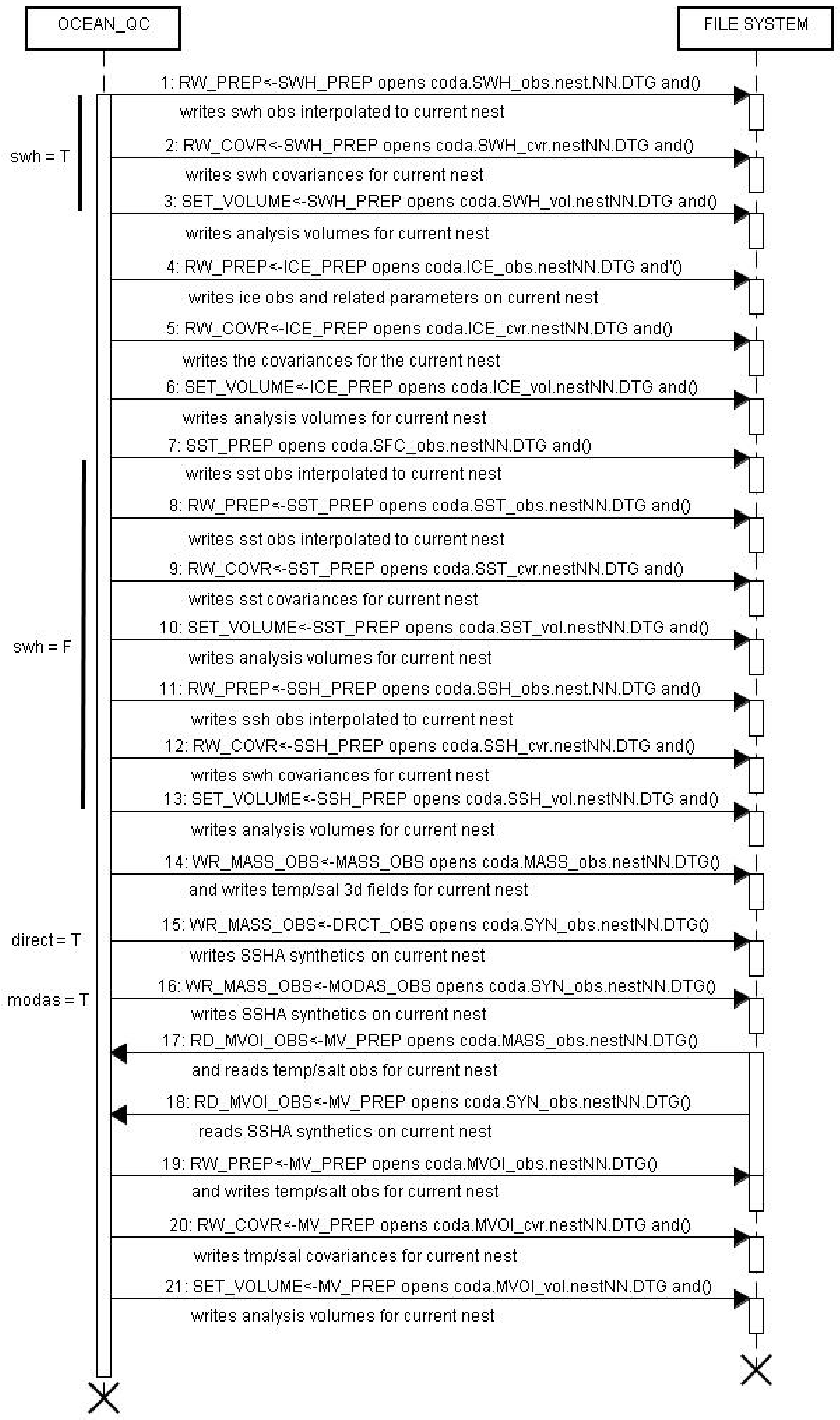
OCEAN_QC
FILE SYSTEM
swh = T
swh = F
direct = T
modas = T
1: RW_PREP<-SWH_PREP opens coda.SWH_obs.nest.NN.DTG and()
writes swh obs interpolated to current nest
2: RW_COVR<-SWH_PREP opens coda.SWH_cvr.nestNN.DTG and()
writes swh covariances for current nest
3: SET_VOLUME<-SWH_PREP opens coda.SWH_vol.nestNN.DTG and()
writes analysis volumes for current nest
4: RW_PREP<-ICE_PREP opens coda.ICE_obs.nestNN.DTG and'()
writes ice obs and related parameters on current nest
5: RW_COVR<-ICE_PREP opens coda.ICE_cvr.nestNN.DTG and()
writes the covariances for the current nest
6: SET_VOLUME<-ICE_PREP opens coda.ICE_vol.nestNN.DTG and()
writes analysis volumes for current nest
7: SST_PREP opens coda.SFC_obs.nestNN.DTG and()
writes sst obs interpolated to current nest
8: RW_PREP<-SST_PREP opens coda.SST_obs.nestNN.DTG and()
writes sst obs interpolated to current nest
9: RW_COVR<-SST_PREP opens coda.SST_cvr.nestNN.DTG and()
writes sst covariances for current nest
10: SET_VOLUME<-SST_PREP opens coda.SST_vol.nestNN.DTG and()
writes analysis volumes for current nest
11: RW_PREP<-SSH_PREP opens coda.SSH_obs.nest.NN.DTG and()
writes ssh obs interpolated to current nest
12: RW_COVR<-SSH_PREP opens coda.SSH_cvr.nestNN.DTG and()
writes swh covariances for current nest
13: SET_VOLUME<-SSH_PREP opens coda.SSH_vol.nestNN.DTG and()
writes analysis volumes for current nest
14: WR_MASS_OBS<-MASS_OBS opens coda.MASS_obs.nestNN.DTG()
and writes temp/sal 3d fields for current nest
15: WR_MASS_OBS<-DRCT_OBS opens coda.SYN_obs.nestNN.DTG()
writes SSHA synthetics on current nest
16: WR_MASS_OBS<-MODAS_OBS opens coda.SYN_obs.nestNN.DTG()
writes SSHA synthetics on current nest
17: RD_MVOI_OBS<-MV_PREP opens coda.MASS_obs.nestNN.DTG()
and reads temp/salt obs for current nest
18: RD_MVOI_OBS<-MV_PREP opens coda.SYN_obs.nestNN.DTG()
reads SSHA synthetics on current nest
19: RW_PREP<-MV_PREP opens coda.MVOI_obs.nestNN.DTG()
and writes temp/salt obs for current nest
20: RW_COVR<-MV_PREP opens coda.MVOI_cvr.nestNN.DTG and()
writes tmp/sal covariances for current nest
21: SET_VOLUME<-MV_PREP opens coda.MVOI_vol.nestNN.DTG and()
writes analysis volumes for current nest

Figure 3.5-1. UML Sequence diagram of work file IO within the ncoda_prep program. NN = nest number (00 - 09) and DTG is current analysis date (YYYYMMDDHH).

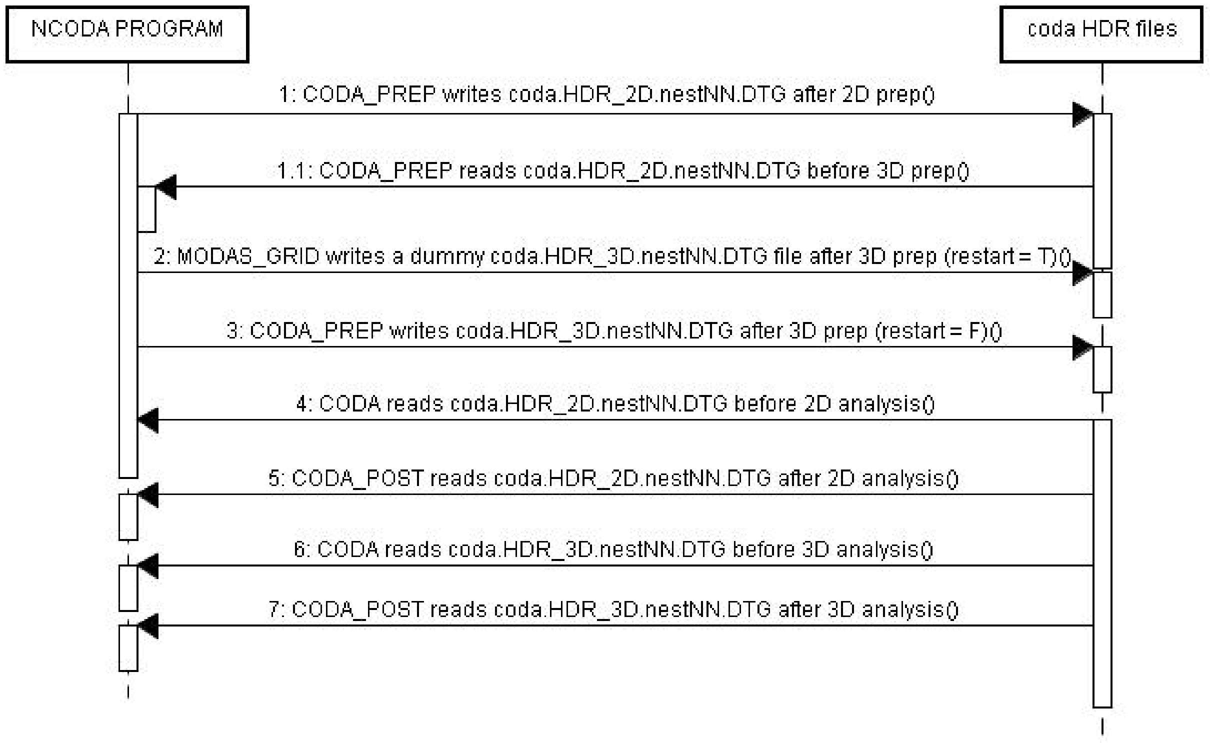
NCODA PROGRAM
coda HDR files
1: CODA_PREP writes coda.HDR_2D.nestNN.DTG after 2D prep()
1.1: CODA_PREP reads coda.HDR_2D.nestNN.DTG before 3D prep()
2: MODAS_GRID writes a dummy coda.HDR_3D.nestNN.DTG file after 3D prep (restart = T)()
3: CODA_PREP writes coda.HDR_3D.nestNN.DTG after 3D prep (restart = F)()
4: CODA reads coda.HDR_2D.nestNN.DTG before 2D analysis()
5: CODA_POST reads coda.HDR_2D.nestNN.DTG after 2D analysis()
6: CODA reads coda.HDR_3D.nestNN.DTG before 3D analysis()
7: CODA_POST reads coda.HDR_3D.nestNN.DTG after 3D analysis()

Figure 3.5-2. UML Sequence diagram of communication between NCODA system components and the file system for writing and reading the coda HDR files. NN = nest number and DTG = date in YYYYMMDDHH format. Note: NCODA PROGRAM in the figure refers to the entire NCODA system.

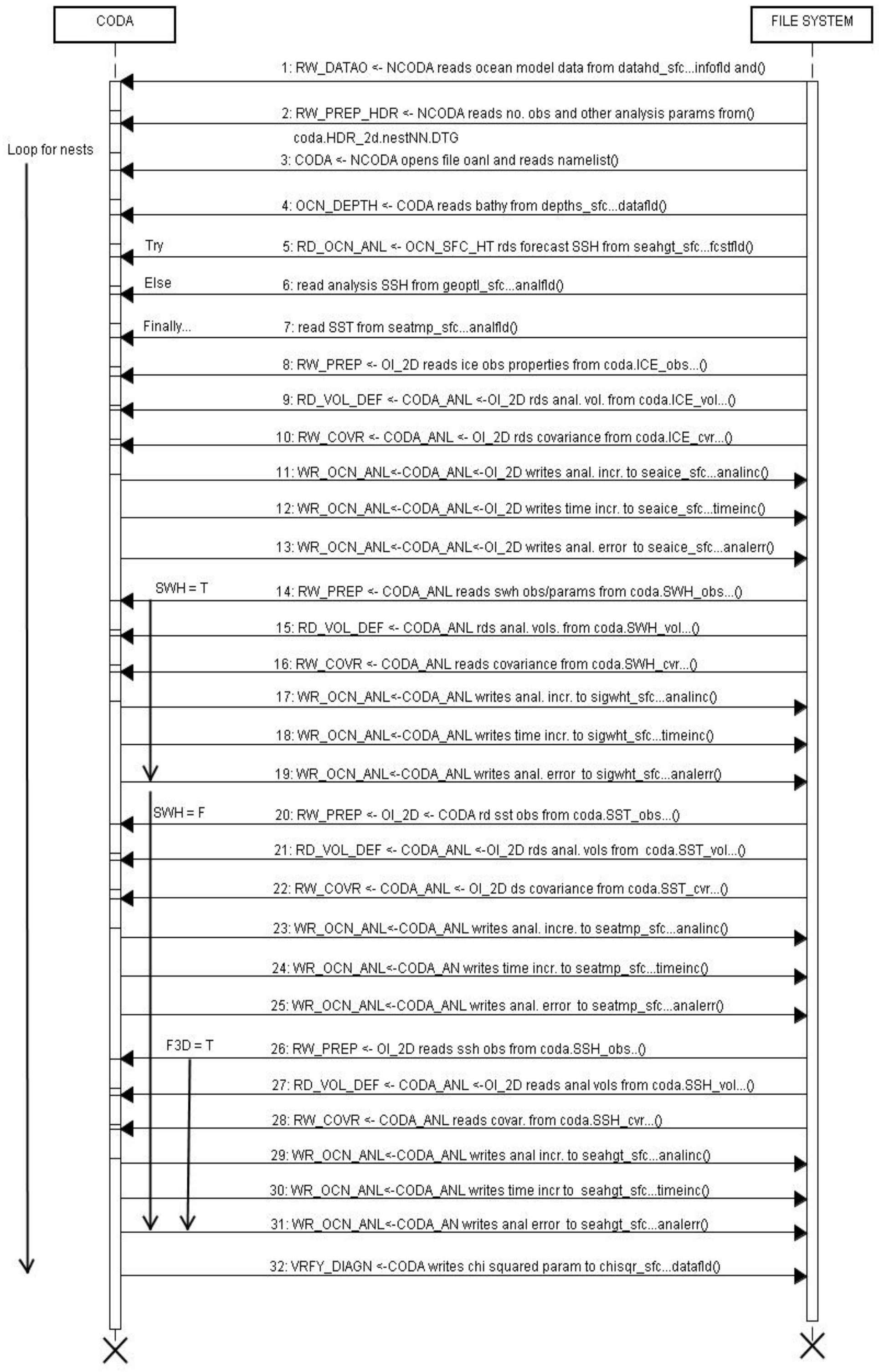
CODA
FILE SYSTEM
Loop for nests
1: RW_DATAO <- NCODA reads ocean model data from datahd_sfc...infofld and()
2: RW_PREP_HDR <- NCODA reads no. obs and other analysis params from()
coda.HDR_2d.nestNN.DTG
3: CODA <- NCODA opens file oanl and reads namelist()
4: OCN_DEPTH <- CODA reads bathy from depths_sfc...datafld()
Try
5: RD_OCN_ANL <- OCN_SFC_HT rds forecast SSH from seahgt_sfc...fcstfld()
Else
6: read analysis SSH from geoptl_sfc...analfld()
Finally...
7: read SST from seatmp_sfc...analfld()
8: RW_PREP <- OI_2D reads ice obs properties from coda.ICE_obs...()
9: RD_VOL_DEF <- CODA_ANL <-OI_2D rds anal. vol. from coda.ICE_vol...()
10: RW_COVR <- CODA_ANL <- OI_2D rds covariance from coda.ICE_cvr...()
11: WR_OCN_ANL<-CODA_ANL<-OI_2D writes anal. incr. to seaice_sfc...analinc()
12: WR_OCN_ANL<-CODA_ANL<-OI_2D writes time incr. to seaice_sfc...timeinc()
13: WR_OCN_ANL<-CODA_ANL<-OI_2D writes anal. error to seaice_sfc...analerr()
SWH = T
14: RW_PREP <- CODA_ANL reads swh obs/params from coda.SWH_obs...()
15: RD_VOL_DEF <- CODA_ANL rds anal. vols. from coda.SWH_vol...()
16: RW_COVR <- CODA_ANL reads covariance from coda.SWH_cvr...()
17: WR_OCN_ANL<-CODA_ANL writes anal. incr. to sigwht_sfc...analinc()
18: WR_OCN_ANL<-CODA_ANL writes time incr. to sigwht_sfc...timeinc()
19: WR_OCN_ANL<-CODA_ANL writes anal. error to sigwht_sfc...analerr()
SWH = F
20: RW_PREP <- OI_2D <- CODA rd sst obs from coda.SST_obs...()
21: RD_VOL_DEF <- CODA_ANL <-OI_2D rds anal. vols from coda.SST_vol...()
22: RW_COVR <- CODA_ANL <- OI_2D ds covariance from coda.SST_cvr...()
23: WR_OCN_ANL<-CODA_ANL writes anal. incre. to seatmp_sfc...analinc()
24: WR_OCN_ANL<-CODA_AN writes time incr. to seatmp_sfc...timeinc()
25: WR_OCN_ANL<-CODA_ANL writes anal. error to seatmp_sfc...analerr()
F3D = T
26: RW_PREP <- OI_2D reads ssh obs from coda.SSH_obs..()
27: RD_VOL_DEF <- CODA_ANL <-OI_2D reads anal vols from coda.SSH_vol...()
28: RW_COVR <- CODA_ANL reads covar. from coda.SSH_cvr...()
29: WR_OCN_ANL<-CODA_ANL writes anal incr. to seahgt_sfc...analinc()
30: WR_OCN_ANL<-CODA_ANL writes time incr to seahgt_sfc...timeinc()
31: WR_OCN_ANL<-CODA_AN writes anal error to seahgt_sfc...analerr()
32: VRFY_DIAGN <-CODA writes chi squared param to chisqr_sfc...datafld()

Figure 3.8-1. UML sequence diagram of messaging between CODA and the File Server for the 2D option.

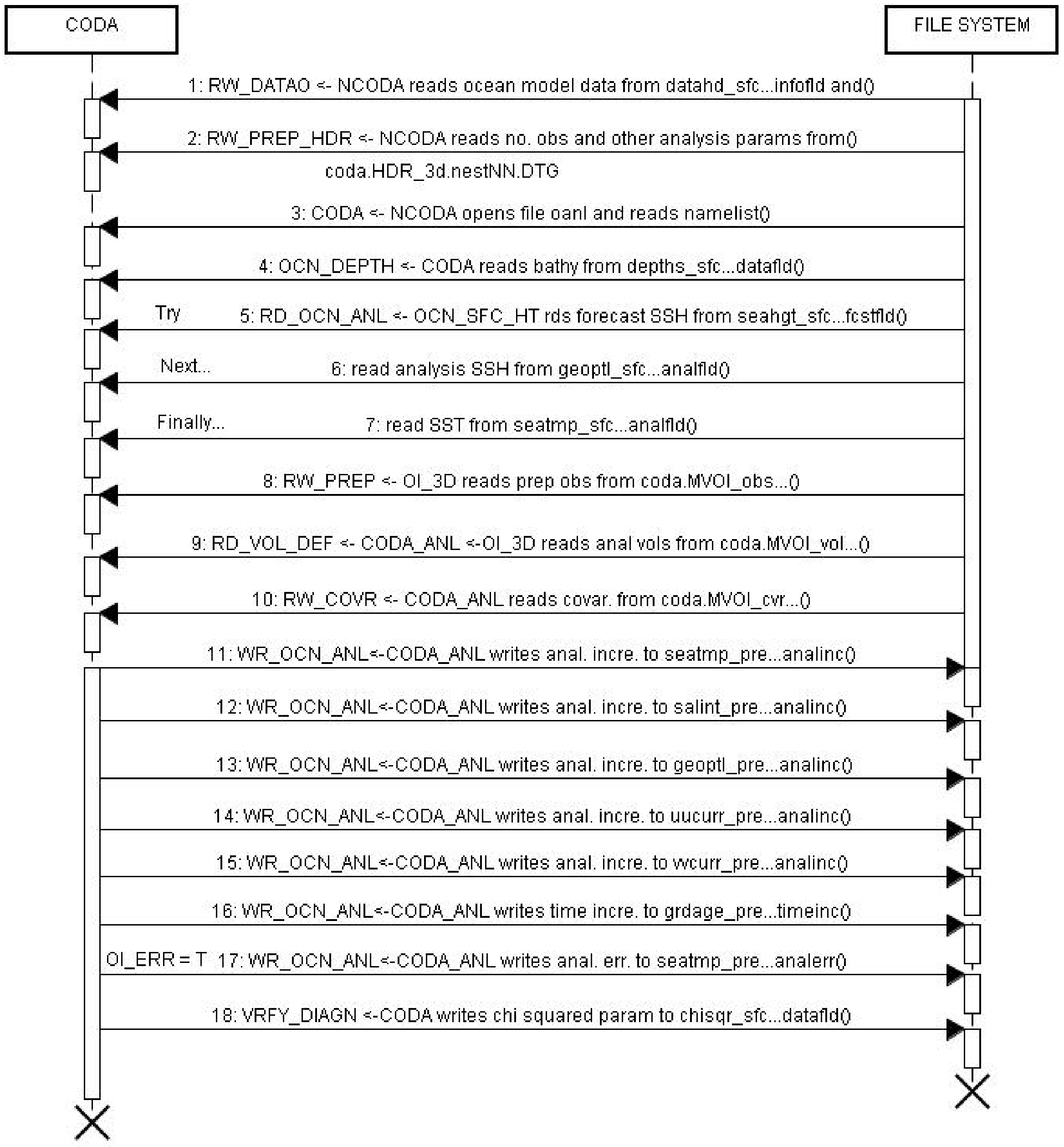
CODA
FILE SYSTEM
1: RW_DATAO <- NCODA reads ocean model data from datahd_sfc...infofld and()
2: RW_PREP_HDR <- NCODA reads no. obs and other analysis params from()
coda.HDR_3d.nestNN.DTG
3: CODA <- NCODA opens file oanl and reads namelist()
4: OCN_DEPTH <- CODA reads bathy from depths_sfc...datafld()
Try
5: RD_OCN_ANL <- OCN_SFC_HT rds forecast SSH from seahgt_sfc...fcstfld()
Next...
6: read analysis SSH from geoptl_sfc...analfld()
Finally...
7: read SST from seatmp_sfc...analfld()
8: RW_PREP <- OI_3D reads prep obs from coda.MVOI_obs...()
9: RD_VOL_DEF <- CODA_ANL <-OI_3D reads anal vols from coda.MVOI_vol...()
10: RW_COVR <- CODA_ANL reads covar. from coda.MVOI_cvr...()
11: WR_OCN_ANL<-CODA_ANL writes anal. incre. to seatmp_pre...analinc()
12: WR_OCN_ANL<-CODA_ANL writes anal. incre. to salint_pre...analinc()
13: WR_OCN_ANL<-CODA_ANL writes anal. incre. to geoptl_pre...analinc()
14: WR_OCN_ANL<-CODA_ANL writes anal. incre. to uucurr_pre...analinc()
15: WR_OCN_ANL<-CODA_ANL writes anal. incre. to vvcurr_pre...analinc()
16: WR_OCN_ANL<-CODA_ANL writes time incre. to grdage_pre...timeinc()
OI_ERR = T
17: WR_OCN_ANL<-CODA_ANL writes anal. err. to seatmp_pre...analerr()
18: VRFY_DIAGN <-CODA writes chi squared param to chisqr_sfc...datafld()

Figure 3.8-2. UML sequence diagram of messaging between CODA and the File Server for the 3D option.

4 Bathymetry and grids

4.1 Specifying a grid.

The desired grid is specified in the "gridnl" file. It consists of a reference point, number of cells along the x and y axes, and cell size. This section will explain how to add a new database file to the bathymetry data and how to use the available input options to specify multiple nests. However, details of the available coordinate systems (e.g., global, mercator, irregular) will not be discussed. This section will focus on the spherical coordinate system only.

4.2 Bathymetry databases and the dictionary

The bathymetry of the grid is found from a set of databases that is listed in a dictionary file, “DBV.dictionary”. This file name is hard-coded in subroutine DBVDBV. It contains info about the available bathymetry databases, which must be located in *clim_dir*.

DBVDBV calls a sequence of subroutines to find depth values for the requested grid using the databases listed in this file. Up to 124 databases are allowed (*nrmax* in DBVLOD) for up to 20 ocean basins (*maxbasn* in DBVLOD). The databases listed in subroutine DBVDBV are:

C	DBVHRBALTIC	Baltic Sea (1-min)	Y
C	DBVMREATL	E Atlantic (1-min)	N
C	DBVMRMED	Med (5-min)	Y
C	DBVHRMED	Med (1-min)	Y
C	DBVHRSCHINA	S China Sea (1-min)	Y
C	DBVHRSEUSA	SE US Coast (.5-min)	Y
C	DBVHRWEST	W US Coast (1-min)	Y
C	DBVMRNATL	N Atlantic Ocean	Y
C	DBVMRSATL	S Atlantic Ocean	Y
C	DBVMRNPAC	N Pacific Ocean	Y
C	DBVMRSPAC	S Pacific Ocean	Y
C	DBVMRIND	Indian Ocean	Y
C	DBVMRARCTIC	Arctic	Y
C	DBVMRANTARC	Antarctic	Y
C	SASHRIND	Indian (2-min)	Y
C	SASHRMED	Med (2-min)	Y
C	SASHRNATL	N Atlantic Ocean (2-min)	Y
C	SASHRNPAC	N Pacific Ocean (2-min)	Y
C	SASHRSATL	S Atlantic Ocean (2-min)	Y
C	SASHRSPAC	S Pacific (2-min)	Y

Comparisons were made between these files (above) to the "DBV.dictionary" file in /net/dynamic/export/data/Rowley/datasets/ncoda/clim to check for consistency. There were three differences: (1) the order is different; (2) DBVHREATL is listed in the dictionary file but not in subroutine DBVDBV; and (3) DBVMREATL is listed in DBVDBV but not in the dictionary file. Point (1) is significant; if the databases overlap, it is necessary to list the preferred one first in the file. Also, if new databases are added, something has to be removed. However, databases can have multiple gridded areas, which can also have different resolution. The databases must reside in the same directory as the dictionary file.

The user has some control of which databases are used. The oanl parameter *dbv_opt* is used to set *dtype* to either 1 (Navy only) or 0 (all). These values are set in DBVDBV and are included in the database files. However, there is a discrepancy; the "oanl" file comments suggest values of “DOD”, “SAS” (i.e. Smith and Sandwell), or “all” as input for *dbv_opt* (source in DBVDBV) but there are only two values in the databases (1 or 0). DBVDBV checks for the value of source in the following string “DoDDODdod”. If it is present (e.g., *dbv_opt* = “DOD”), *dtype* is set to 1; otherwise 0. If *dtype* = 0, all temporary variables for data type are set to 0 to insure that navy ones are also used. Thus, there is no way to select the “SAS” data only and the medium resolution data will be used unless put at the end of the dictionary list. Another user-selected parameter is the minimum resolution (minutes) permitted (*dbv_res* in oanl).

Subroutine DBVSCH identifies the first database that includes each point within the selected grid. Each point on the grid is assigned a basin index from those available or an error flag if none. The lat and lon for a point are converted to minutes and rounded to the nearest minute (~1.85 km). The first data point within a distance of *factor* (= 0.01 minutes) of each cell is selected in subroutine DBVPTR. Subroutine DBVUPK is then called to unpack the requested cell depth. Note that the word size, number of bits per depth field, and number of depth values per word are all dependent on the given database file format. No interpolation of depth to the grid is completed.

4.3 Adding a bathymetry set to the database

It is useful to occasionally add a new or regional database of bathymetry to the dictionary. This must be done with care to assure that it is properly inserted and accessed. The database should be listed at the top of the "DBV.dictionary" file to assure that it is used; at a minimum it must be listed before any other databases that include the selected analysis grid domain:

```
1   779581   2   1   DBVHRMED
2   388800   1   0   SASHRMED
3    62208   1   1   DBVMRMED…
```

The first column should be its index in the list. The second col is record length (words) for the file. Col 3 is the number of records, col 4 is source (0=navy, 1=civilian), and col 5 is the name of the file. Details about the databases are stored in the dictionary file as a table specifying the geographic coverage as a series of rectangles on a latitude/longitude grid. The grid resolution, gridded data file index and parameter group index are given for each rectangle. One line is given for each rectangle (F3.0,4F8.1,F5.0,3X,F5.0):

COLS 1-3	TYPE	Data parameter group index.
COLS 4-11	XLATN	Northernmost latitude, in minutes.
COLS 12-19	XLATS	Southernmost latitude, in minutes.
COLS 20-27	XLONW	Westernmost longitude, in minutes.
COLS 28-35	XLONE	Easternmost longitude, in minutes.

COLS 36-40	RES	Data base resolution, in minutes.
COLS 44-49	XPFN	Gridded data base file index.

The data are stored in direct access files with the names given in column 5 of the dictionary file. The first record in the file should be a real array of length 1009800. This is a header array with the following contents:

Columns	Content
1-4	4-character label
5-8	2 blanks and a 2-character label
9-12	4-character label for basin
13-16	spaces
17-20	1 space and a version number as f3.1
21-28	Date written as D/DDDD/D (I don't know what the values are)
29-36	Time written as T:TTTT:T (ditto)

These entries are written as characters. They are followed by 36 spaces (also a character string). The following entries are all of type real*4. They are written in sequence immediately after the blanks: (1) the latitude (in minutes) from the north (south) pole (i.e., 64 N = 1560 in the file); (2) number of N-S cells; (3) the N-S resolution (minutes); (4) the longitude (in minutes) eastward from prime meridian; (5) number of E-W cells; (6) E-W resolution (minutes); (7) number of 2-value words in array (i.e., product of E-W cells and N-S cells divided by 2); (8) number of N-S blocks (i.e., N-S cells); (9) number of N-S cells per block (i.e., 1); (10) number of E-W blocks (i.e., E-W cells); (11) number of E-W cells per block (i.e., 1). This concludes the header record.

The second record in the database file is the record map, which consists of real*4 variables. These are (1) the number of records for the data set; (3) the record number for the data for the current block; (4) the starting index (*i*) for the current block; and (5) the ending *x* index for the current block. If item (1) is one, (3) should be 3 and the others don't appear to have any effect. Note that item (2) is unused.

The depth data are stored in subsequent records up to the number given in col 3 of the first section of the dictionary file (above). The depth data should be integers of length = 4 (i.e., integer*2). They are read directly by subroutine DBVDBV as follows. The depth values are read as a buffer of fixed length (*maxwds*). Each value in the buffer holds two depths. These are unpacked using the F90 function IBITS. They can be written, however, as integers with length 2. No fractional depths are used. When transforming the 2D depths to a 1D array for writing to the direct access file, you must make the outer loop for the second index like this:

```
nn = 1
do i=1,imax
do j=jmax,1,-1
     int(nn) = int(h1(i,j)
     nn = nn+1
enddo
```

```
            enddo
```

This has been tested on an idealized bathymetry file and compared to the grid calculated from the NPAC database in the original clim directory. For example, using *gridnl* variables of *m* = 20, *n* = 30, *rlat* = 20.0, *rlon* = 240.0, *delx* = 0.5, and *dely* = 0.5, the resultant grid looks like this:

```
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   5 219 147  25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 7711423 336 335 364   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 757 6141758 744 801 584   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 982 983 68910171548 871   0   0   0   0   0   0   0   0   0   0   0   0   0   0
321716291040102116231576 477   0   0   0   0   0   0   0   0   0   0   0   0   0
37683247 975234016061031 964   0   0   0   0  37   0   0   0   0   0   0   0   0
3627361035461640164818271986 225   0   0   0  43 102  30   0   0   0   0   0   0
39833543356829992560200318091704   0   0   0  46 196 100   0   0   0   0   0   0
39423552359235883036202020172742 545   0   0  28 347 160 114   0   0   0   0   0
379435762973317635563011218610781945  97   0   0   0   0 194  39   0   0   0   0
359936153495234833483609378430211791 999  74   0   0 194 308  90   0   0   0   0
379837903849400030243577360839082975 290  26  32   0   0  69 487 148   0   0   0
390836803808382435943688378441804486 196  29  49   0   0   0 786 833 432   0   0
4012393339363650359037883594350636313985 533   0   0   0   0   0101017851383   0
400339943980432339554028367833073574347339451031   0   0   0   0  5117781784 743
398339934014403736123904368636033613359533923573 999  61  25   0   0   015181394
4089403640294020401839993876383837003609350734032962 629 124  77   0   0 4081545
41983800395137973994399639053901398838073605206039951048 205 116   0   0 1021332
403739283727390437883852380539533065364838523596318039411020 198   0   0   0 287
405640633993396838193827351936483766398338043537359436083015 505  12   0   0 185
4039404439313815382537933820378237193802368337293811340133462603 206 252  43   0
40004019395839253925387240094001381037563794377437823775340132002912 512 529 199
39954001400139844029398342004004376738503811379937413578359032572217295721031217
39943939399439833935375439854193399839233575360738163791358933993410334231973001
39903950362540003982399239954196399637813607379237983820342433993395312131803197
38963804399039993819398938114017379934623413393636803633342735843395313833333198
39343959394437783927395039613783392937583382379537943652359129143396328131783229
39944097381439383790335135953973385632763732377837983784341228813365328633693181
42584068418939973964388138893789379137983612379537943634341135863205313032402955
```

That is Baja California to the east. This was written (after being read in DBVDBV) using:

```
do k=30,1,-1
   write (*,'(20i4)') (int(depth(i)), i=(k-1)*20+1,(k-1)*20+20)
enddo
```

The test grid produced from inserted bathymetry using: *m* = 20, *n* = 30, *rlat* = 40, *rlon* = 180, *delx* = 0.25, and *dely* = 0.25, looks like this:

```
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  30  40  40  50  50  60  60  70  70  80  80  90  90 100 100 110 110 120 120 130
  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33
  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33
  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33
  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33  33
```

Note the values of 33 along the souther margin. These were added to row 2 of the original bathymetry as tags to make certain that the N/S orientation was correct.

This was cut from a bathymetry file with a resolution of 30 minutes that deepend eastward to 1000 m and shallowed westward to 10 m. Note the repeated depths, because the algorithm does not interpolate but uses the first depth within the lat/lon cell. The insertion algorithm has only been tested with databases consisting of 1 record and 1 block, which is common to most of the files in the climatology repository. The program used to generate the test file is included in Appendix 3. It must be modified for other datasets.

The dictionary file must be modified to reflect the new database by inserting the appropriate data into the first section as described above. The record length will be the number of cells multiplied by two (the length of the integer). The data included in the header must match the size of the data base, location, and the variable type.

Also note that the character data held in the header are reversed when a little endian system uses -fendian=big (e.g., compiler = G95) because they are read as reals, which are reversed. This occurs with the original database files and is reproducible. It does not happen when the endianess is unchanged. It should be noted that this nonstandard treatment of character data requires -fsloppy-char (g95) as a compiler option. For example:

```
[keen@typhoon test]$ a.out
 Data base file header     PAR  TSET10    basin  CAPN     version  0.1
 Date/time 2/0180/0 3:2100:0
      Latitude grid    55 deg   0 min NORTH    res   30.0 min   cells     30
      Longitude grid  179 deg   0 min EAST     res   30.0 min   cells     20
      Block structure    words     300     LAT   30   1     LON   20   1
Record map       1 entries
```

should be:

```
[keen@typhoon test]$ a.out
 Data base file header     PAR  TEST01    basin  NPAC     version  1.0
 Date/time 10/20/08 12:30:00
```

```
      Latitude grid    55 deg   0 min NORTH    res   30.0 min   cells     30
      Longitude grid  179 deg   0 min EAST     res   30.0 min   cells     20
      Block structure    words     300     LAT   30   1     LON   20   1
Record map       1 entries
```

The grid processing algorithm limits the resolution to 1 minute, which is ~1.85 km. This will need to be modified if higher resolution processing is required.

4.4 Multiple nests

The subnests will always have a cell size ratio of 3:1 to their parent, despite input in the gridnl namelist. It is unnecessary to provide any of the following parameters for subnests: (a) *iref* or *jref*; (b) *delx* or *dely*. The subnests will be calculated using the input values of *ii* and *jj*. These are the offsets from (1, 1) on the parent grid. *Iref* and *jref* are computed using this offset and a fixed grid ratio of 3 in sub COAMOA. *Delx* and *dely* are similarly found with a ratio of 3. The size of the subnests is then found from the input values of **m**(*nest*) and **n**(*nest*). The bathymetry is extracted for each grid nest in the nest loop found in subroutine COAMOA from all available databases.

This step was completed as a test and produced the following results for two nests. Note the blockiness of the inner nest because of the limitations of the algorithm.

Another nest was generated using sufficiently large values of *m* and *n* that the resulting domain exceeded the domain of nest 1. The result (Figure 4.4-2) shows that these are not really nests but simply multiple analysis grids. This is possible because all analysis grids are calculated from the full set of databases. However, this capability may not be useful for most problems because of the increased resolution of the subnests. For more advanced grid generation, see section 5.2.

4.5 Moving nests

It doesn't appear that moving nests are implemented. The parameter *lnmove* in namelist gridnl (*nst_move* in COAMOA and MOVE_DATAO) is not passed any further during prep, analysis, or post processing.

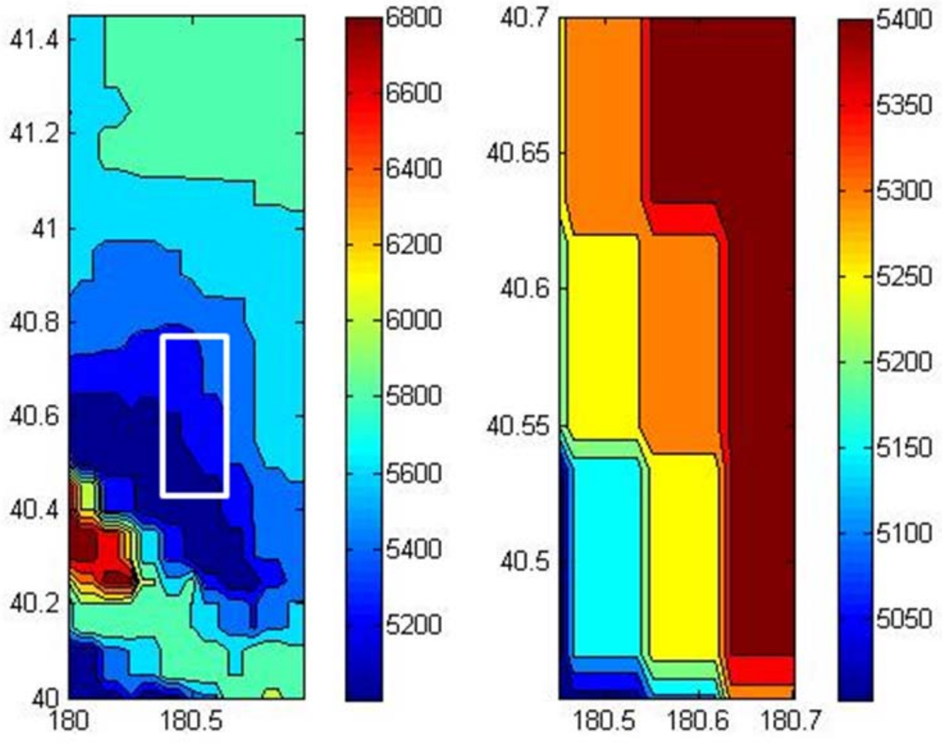

Figure 4.4-1. MATLAB® figures of bathymetry made from the ncoda clim databases for two nests. Note how blocky the smaller nest looks because of no interpolation. The ncoda_prep does not notify the user of which database was used. The inner nest is shown in a white outline on the coarse grid. Note that the contour interval is different for the inner nest because of the limited range of values.

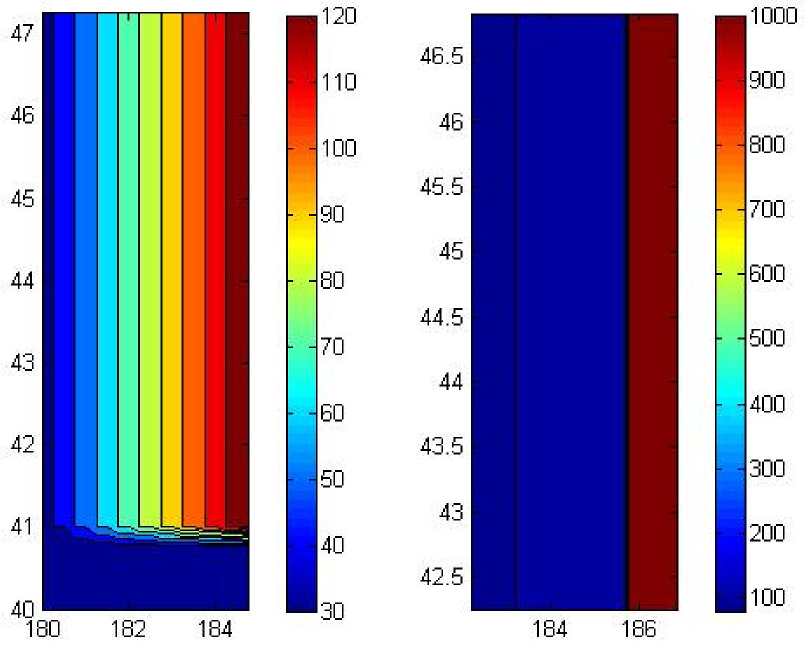

Figure 4.4-2. MATLAB® figures of bathymetry made from the idealized database described in section 4.3. The second nest exceeds the domain of the first because of the large values of *m* and *n,* even though it has a cell ratio of 3:1.

5 System input variables and parameters

The input variables for the NCODA system are available in namelists, which are generally discussed in the user manual. However, because they are placed in namelists, it is highly probable that they will not be listed in any given application of the sytem. This is likely to occur because it is common practice to acquire input files from a previous user. The purpose of this section, therefore, is to discuss the input variables that are more likely to be modified for littoral applications.

5.1 Grid namelist

The grid namelist is read from a file given on the command line, but it contains a namelist called "gridnl." It includes several variables that apply to specific projections. This section will focus on the spherical projection only. The dimensions of all nests are given by *m* (*x* axis) and *n* (*y* axis). As noted in section 4.4, the user must be cautious in defining multiple nests because they can easily exceed their host's domain. This potential problem cannot occur in the vertical dimension, however, because all analysis nests will have the same number of levels (*kko*). The ocean model used for generating forecasts can have a different number (*kkom*) of levels, which is used in COAMOA and INIT_DATAO to process forecast model results. It is placed in array **datao**.

The geographic point given by *rlat*/*rlon* can be anywhere within nest 1 but it must be referenced to the grid cell described by *iref*/*jref*. It is a good idea to select the lower left (SW) corner so that *iref*/*jref* = 1/1. However, only one reference point is used; thus, other nests must be referenced to this point using their indices. These child nests are always three times their host (parent) nest. If the SW corner of a child nest is located at point (10,10) on the parent nest and *rlat*/*rlon* are located at the SW corner of the parent, the values of *iref*/*jref* for the child nest are offset (10-1)×3-1 = 26 cells to the west/south; i.e., *iref*/*jref* for the child nest are -26/-26. The parent indices for the SW corner (10,10) must also be input as *ii/jj* in the namelist. For nest 1, *ii/jj* = 0/0. This pattern continues with all child nests referenced to their parent nest. The values of *iref*/*jref* are recalculated in COAMOA using the parent grid index (*npgrid*) from the gridnl namelist. This is not done for nest 1, however, which must have matching *iref*/*jref* and *rlon*/*rlat* values as input.

The grid cell dimensions (*delx*/*dely*) are also recalculated in COAMOA using a 3:1 ratio between each parent/child nest pair. This approach allows the flexibility of having several child nests within nest 1 (which makes sense), or allowing some of these to have a higher resolution simply by setting their parent to one of the intervening child nests. This would result in a cell size of 1:9 against nest 1, as demonstrated in Figure 5.1-1. Thus, the apparent discrepancy discussed in section 4.4 is actually useful for selecting grids in different locations and varying resolution. Thus, the child nests are not restricted to be subregions of their parents; this is simply a naming convention drawn from more rigid nesting approaches.

5.2 Ocean analysis namelist

The thresholds, tolerances, covariances, etc. for the data as well as forecast model input is read from the Ocean Analysis NameList (oanl), which must be contained within a file called "oanl". These parameters have many complex applications in all three programs within the NCODA system. For this reason, they will not be discussed in detail in this report; instead, Table 5.2-1 lists the parameters and the subroutines in which they are used. The program developer can then go directly to the source code to examine their implementation and potential for modification.

5.3 Directory namelist

The directory namelist must be in a file called "odsetnl." It contains the locations of directories used for general climatology, gdem climatology, MODAS climatology, data, analysis, and output. There are some other directories for classified (etc) output. These directories are discussed in section 3.1.

Table 5.2-1: OANL NAMELIST VARIABLES FINAL USE LOCATIONS

Name	(default) Description	Use Subroutine (dummy name)
amsr_bias	(true) perform bias correction of amsr microwave SST	RD_AMSR (*bias*)
amsr_dw	(true) remove diurnal warming signal and return foundation SST	RD_AMSR (*dw*)
argo_bias	(true) perform bias correction (drift adjustment) of argo salinities	PROF_COLLECT
atsr_bias	(true) perform bias correction of atsr SST	RD_ATSR (*bias*)
atsr_dw	(true) remove diurnal warming signal and return foundation SST	RD_ATSR (*dw*)
blist	black listed call signs	CODA_PREP, RD_GLDR, RD_PROF, RD_SHIP
bv_chk	brunt-vaisala frequency threshold	DRCT_OBS (*cph*)
clm_scl	climate decorr. time scale (hrs): (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) swh	SET_ERR (*mx_age*)
cold_start	(true) force cold start on nest 1	CHK_DATAO
debug	(true) generate diagnostics about: (1) argo salt bias correction, pooling of profile moorings and profile rejections (fort.32); (2) profile inflexion point, standard level data, vertical extension results using background fields (fort.33), (3) geopotential profile observations (fort.34); (4) observation and prediction errors (fort.35); (5) listing of MVOI observations (fort.36); (6) synthetics (direct and MODAS), for MODAS this includes rejections and editing results (fort.37); (7) layer pressure observations (fort.38)	DRCT_OBS, GLDR_SLCT, GLDR_COLLECT, MODAS_CHKSAL, MODAS_CHKTMP, MODAS_SYN, MVOI_OBS, MVOI_ANL, OBS_GEOPTL, PROF_OBS, SET_ERR, OBS_LYRP, PROF_ARGO, PROF_COLLECT, PROF_POOL, SWH_PREP, PROF_XTND, PROF_SLCT, SSH_PREP, RD_MVOI_OBS

dbv_opt	source bathymetry database	DBVDBV (*source*)
dbv_res	minimum resolution of bathymetry to retrieve from variable resolution database (minutes)	DBVSCH (*res*)
del_ssh	minimum change in SSHA to force generation of a synthetic profile	SYN_SMPL
del_sst	minimum change in SST to force sampling of analyzed SST field	SST_ANL
den_ds	change in density def. for MLD	SET_VCORR
deny	data types to deny in analysis (see "coda_types.h" for type codes)	CODA_PREP, CR_SUPPL_SSH, CR_SUPPL_SST, PROF_COLLECT, RD_AMSR, RD_ATSR, RD_GLDR, RD_GOES, RD_LAC, RD_MCSST, RD_METOP, RD_METOP_LAC, RD_MSG, RD_PROF, RD_SHIP, RD_SSH,RD_SSMI, RD_SWH
dh_scl	flow dependent correlation scale - smaller number means more flow dependence. units depend on 2D (deg C) vs 3D run (dyn m). Azero or negative value will disable.	GRD_CONFLICT, OBS_COVAR, GRID_COVAR, OCN_SFC_HT, ICE_PREP, CODA, SSH_PREP, MODAS_GRID, MV_PREP
direct	(true) perform direct assim of SSHA	CHK_FCST, CODA_PREP, SSH_PREP
diurnal	(true) assimilate SST retrievals from diurnal warming events	RD_MCSST, RD_METOP, RD_GOES, RD_LAC, RD_METOP_LAC, RD_MSG, RD_AMSR, RD_ATSR
dv_dz	vertical gradient length scale (units are dependent upon vc_mdl selection)	SET_VCORR
ebkg	background error tuning factors: (1) ice; (2) sst; (3) ssh; (4) temperature; (5) salinity; (6) geopotential; (7) u velocity; (8) v velocity; (9) swh; (10) layer pressure	CODA_PREP (*err_bkg*), SET_ERR (*fct_bkg*)
emdl	background error option: 'simple' = homogeneous errors; 'complx' = non-	SET_ERR (*err_mdl*)

	homogeneous errors: (1) ice; (2) sst; (3) ssh; (4) temperature; (5) salinity; (6) geopotential; (7) velocity; (8) swh; (9) layer pressure	
eobs	obs error tuning factors (see "coda_types.h" for type codes)	SET_ERR (*fct_obs*)
err_scl	error growth time scale (hrs): (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) swh	POST_2D, POST_3D
fcst_off	number of hours to offset the anal. dtg for the start of the forecast - used in HYCOM to correct for the 6-hr IAU that is used	VRFY_FCST (*off*)
fgat	first guess appropriate time update interval (hrs): (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) significant wave height	OBS_DETREND
gc_btm	geostrophic coupling e-fold. bottom depth. Velocity increments at depths shallower than this are scaled to zero	GRID_COVAR, OBS_COVAR
gc_lat	geostr. coupling e-folding latitude. Velocity increments at latitudes less than this are scaled to zero	GRID_COVAR, OBS_COVAR
gldr_slct	glider selection criteria options: (1) acceptable level probability gross error; (2) minimum number of sampling levels; (3) minimum ratio of last sampling depth and bottom depth; (4) minimum sampling depth (if glider has not sampled water column); (5) maximum acceptable distance between adjacent levels; (6) maximum acceptable temperature difference between adjacent levels; (7) maximum acceptable depth difference at level of maximum temperature difference; (8) maximum acceptable temperature difference at level of maximum level difference; (9) maximum depth first sample	RD_GLDR (*lvl_prb*), GLDR_SLCT (*prf_slct*)

global	(true) global cyclic grid	CODA_PREP, CR_MEAN_SSH, RD_CONFLICT, ICE_PREP, MV_PREP, RD_AMSR, RD_ATSR, RD_GLDR, RD_GOES, RD_LAC,RD_MCSST, RD_METOP, RD_METOP_LAC, RD_MSG, RD_PROF, RD_SHIP, RD_SSH, RD_SSMI, RD_SWH, SET_HGRD, SMTH, SSH_PREP, SST_PREP, SWH_MDL, SWH_PREP, VELC_OBS, WR_MASS_OBS
goes_bias	(true) perform bias correction of GOES SST data	RD_GOES (*bias*)
goes_dw	(true) remove diurnal warming sig. and return foundation SST	RD_GOES (*dw*)
hc_mdl	horizontal correlation model options: 'rsby' = rossby radius deformation; 'homo' = homogeneous scales; 'locl' = user defined scales--(1) ice,(2) sst, (3) ssh, (4) multivariate, (5) significant wave height	GET_HSCL, GRD_CONFLICT, SET_HCORR (*opt*)
himem	(true) execute analysis using I/O of analysis volumes rather than mpi_reduce	OI_3D
hst_wt	geom.. series param. for computing background error variances: (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) significant wave height	RW_DATA_ERR
ice_time	ice observation processing option: obst = select obs based on obs time; synt = select obs that are synoptic for analysis update interval	CR_ICE_FILE, RD_SSMI
lac_bias	(true) perform bias correction of AVHRR LAC SST data	RD_LAC (*bias*)
lac_dw	(true) remove diurnal warming sig. and return foundation SST	RD_LAC (*dw*)

linck	(true) perform innovation error check: (1) ice; (2) sst; (3) ssh; (4) temperature; (5) salinity; (6) geopotential; (7) velocity; (8) direct method synthetics; (9) swh; (10) layer pressure	CODA_PREP (*do_invc*), SET_ERR (*inv_chk*)
lndz	minimum bottom depth (m) used to define and points in analysis grid	SET_MASK
locn3d	do 3D analysis on this grid nest	COAMOA, CODA, CODA_POST
mask_opt	Grid mask option: 1D = all water; 2D = land points elliminated; 3D deep points eliminated	SET_MASK (*opt*)
mcsst_bias	perform bias correction of AVHRR GAC SST data	RD_MCSST (*bias*)
mcsst_dw	remove diurnal warming signal and return foundation SST	RD_MCSST (*dw*)
mds_edit	edit MODAS synthetics	CODA_PREP, MODAS_GRID, MODAS_TEMP
mds_grd	generate MODAS synthetic profile initial conditions on a cold start	CODA, CODA_PREP
mds_mld	apply modas mld model	MODAS_GRID, MODAS_TEMP
mds_xtnd	extend MODAS synthetics with Levitus climatology	CODA_PREP, MODAS_GRID, MODAS_TEMP
metop_bias	perform bias correction of METOP AVHRR GAC and LAC SST data	RD_METOP (*bias*)
metop_dw	remove diurnal warming signal and return foundation SST	RD_METOP (*dw*)
modas	perform MODAS assim. SSHA	CODA_PREP
model	forecast model (SWAFS, NCOM, HYCOM) or wavewatch model region name	CODA, CODA_POST, CODA_PREP, MODAS_GRID, OBS_LYRP
msg_bias	perform bias correction of MSG SST data	RD_MSG (*bias*)

msg_dw	remove diurnal warming signal and return foundation SST	RD_MSG (*dw*)
mx_lyr_prs	maximum acceptable layer pressure innovation (a negative value disables the test)	CODA_PREP, OBS_LYRP
n_hst	number days nto the past to use in forming prediction errors from analyzed increments: (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) significant wave height	RW_DATA_ERR
n_pass	number 3x3 smoother passes on full valued analysis output fields	ANL_ERR, CODA_MLD, MASS_OBS, CR_MEAN_SSH, DRCT_FLD, MODAS_GRID, RW_2D_ANL, SMTH, SET_MASK
offset	(i,j) offsets from grid boundary to restrict data selection for the assimilation (only for nest 1): (1) from the north (grid top); (2) from the south (grid bottom); (3) from the east (grid right); (4) from the west (grid left)	CODA_PREP, CR_SUPPL_SSH, ICE_PREP, MODAS_CHKSAL, MODAS_CHKTMP, PROF_OBS, SET_ERR, SSH_PREP, SST_ANL, SST_PRP, SYN_SMPL, VELC_OBS
oi_err	compute interpolation errors: (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) significant wave height	CODA_ANL, POST_2D, POST_3D, VOLUME_ANL, VOLUME_REDUCE, VOLUME_SAVE
pool	pool satellite systems: (1) DMSP F11,F13,F14,F15,F16; (2) NOAA 14,15,16,17, 18 GAC SSTs; (3) TOPEX, ERS, GFO, JASON, ENVISAT; (4) GOES 8,10,11,12 SSTs; (5) NOAA 16,17,18 LAC SSTs; (6) Altimeter / Buoy SWH; (7) AMSRE, AMSR, TRMM;(8) ATSR, AATSR; (9) MSG day/night SSTs; (10) METOP A,B,C GAC SSTs; (11) METOP A,B,C LAC SSTs	CODA_PREP, RD_AMSR, RD_ATSR, RD_GOES, RD_LAC, RD_MCSST, RD_METOP, RD_METOP_LAC, RD_MSG, RD_SSH, RD_SSMI, RD_SWH
prf_hrs	number hours of profile obs	RD_PROF
prf_opt	profile processing option: obsz = assimilation of profiles on observed levels; anlz = assimilation of profiles	CODA_PREP, GLDR_COLLECT, GLDR_SLCT, MODAS_OBS,

	after interpolation to analysis levels	PROF_COLLECT, PROF_SLCT
prf_slct	profile selection criteria options: (1) acceptable level probability gross error; (2) minimum number of sampling levels; (3) minimum ratio of last sampling depth and bottom depth; (4) minimum sampling depth (if profile has not sampled water column); (5) maximum acceptable distance between adjacent levels; (6) maximum acceptable temperature difference between adjacent levels; (7) maximum acceptable depth difference at level of maximum temperature difference; (8) maximum acceptable temperature difference at level of maximum level difference; (9) maximum depth first sample	GLDR_SLCT, PROF_SLCT
prf_time	profile time sampling option: cycl = select obs based on number prf_hrs specified; obst = select profiles based on time profile was observed; rcpt = select profiles based on time profile was received at center; synt = select obs that are synoptic for analysis update interval	CR_TMP_FILE, RD_GLDR, RD_PROF
prf_xtnd	extend inflexion point profiles to bottom using first guess fields	PROF_OBS (*xtnd*)
pt_anl	potential temperature analysis	BV_FREQ, CLIM_PREP, CODA_MLD, CODA_PREP, GLDR_COLLECT, MODAS_CHKSAL, MODAS_GRID, MODAS_OBS, OBS_LYRP, PROF_COLLECT, SALT_CORR, SET_VCORR, TS_STATIC
qc_err	max acceptable probability gross error: (1) DMSP sea ice; (2) AVHRR satellite sst; (3) GOES satellite sst; (4) in situ sst; (5) profile temperatures (integrated over levels); (6) profile salinities (integrated over levels); (7) altimeter	CODA_PREP (*qc_prb*), RD_MCSST (*qc_prob*), RD_AMSR (*qc_prob*), RD_ATSR (*qc_prob*), RD_GOES(*qc_prob*), RD_LAC (*qc_prob*), RD_MCSST (*qc_prob*), RD_METOP (*qc_prob*),

	SSHA; (8) altimeter/buoy SWH; (9) LAC satellite sst; (10) AMSR satellite sst; (11) ATSR satellite sst; (12) MSG satellite sst; (13) METOP GAC satellite sst; (14) METOP LAC satellite sst	RD_METOP_LAC (q*c_prob*), RD_MSG (*qc_prob*), RD_SSMI (*qc_prob*)
rscl	rossby radius scaling factor: (1) ice; (2) sst; (3) ssh; (4) multivariate; (5) swh	CR_SUPPL_SSH, SET_HCORR (*pfct*), SYN_SMPL
run_class	classification level of run: 'U' - unclassified; 'R' - restricted; 'C' - confidential; 'S' - secret; 'T' - top secret	CODA_POST, CODA_PREP
sal_adj	adjust SSS derived from SST obs for density inversions	SSS_OBS
sal_std	max number std dev to scale modas salinity from climatology	MODAS_SALT (*clm_std*)
smpl	synthetic sampling interval	SYN_SMPL (*f*)
spval	missing value (must be <= -999)	ANL_ERR, BILNR, CODA_BNDY, CODA, CODA_FILL, CODA_MLD, CR_MAN_SSH, DRCT_ADJ, DRCT_OBS, GDEM_MOD, GEO_PTL, GLDR_COLLECT, GRD_CLIM, GRDNT_ERR, ICE_PREP, MODAS_CHKSAL, MODAS_CHKTMP, MODAS_CLIM, MODAS_COEF, MODAS_DATA, MODAS_GRID, MODAS_MLD, MODAS_OBS, MODAS_SALT, MODAS_TEMP, MODAS_TRP, MODAS_TYPE, MV_PREP, NGPS_BNDY, OBS_DETREND, OBS_GEOPTL, OBS_LYRP, POST_2D, POST_3D, PROF_COLLECT, PROF_ERR, PROF_OBS, PROF_POOL, PROF_THIN, PROF_XTND, RD_MODAS_DATA, RD_SSH_ANL, RD_SWH_ANL, RD_SWH_CLIM, RW_DATA_ERR,SMTH, SSH_PREP, SSS_OBS, SST_PREP, SWH_ADJ, TS_STATIC, VRFY_DIAGN, VRFY_FCST,

		VRFY_STATS, WR_MASS_OBS
ssh_hrs	number of hours of altimeter ssh obs	CR_SSH_FILE, OCN_OBS
ssh_mean	ssh mean field options: 'modl' - model mean field; 'clim' - climate mean field	CR_MEAN_SSH, GET_SSH_MEAN, CODA_XTND (*fld*)
ssh_time	altimeter ssh processing option: cntr = select obs in data window centered around analysis time; cycl = select obs based on full Topex repeat cycle (10 days); obst = select obs based on time SSHA was observed; rcpt = select obs based on time SSHA was received at center; synt = select obs that are synoptic for analysis update interval	CR_SSH_FILE, RD_SSH
ssh_std	max no. std dev to scale altimeter ssh from climatology	MODAS_TYPE, SSH_PREP
sst_time	sst observation processing option: obst = select obs based on time; synt = select obs that are synoptic for analysis update interval	CR_TMP_FILE, RD_AMSR, RD_ATSR, RD_GOES, RD_LAC, RD_MCSST, RD_METOP, RD_METOP_LAC, RD_MSG, RD_SHIP
st_asm	assimilate SSTs in 3D analysis	SST_PREP
st_chn	generate "thermistor chain" observations to the base of the mixed layer from SST (see also st_grd)	SSS_OBS (*st_chain*), SST_ANL (*st_chain*), SST_ANL (*st_chain*), SST_OBS(*st_chain*)
st_grd	generate SST observations for 3D MVOI from analyzed SST grid	CODA_PREP (*st_grid*), MASS_FLD(*st_grid*), MASS_OBS (*st_grid*), SSS_OBS (*st_grid*), SST_PREP (*st_grid*)
st_ntrvl	SST "thermistor chain" vertical sampling interval	SSS_OBS, SST_ANL, SST_OBS
st_smpl	analyzed SST observation sampling interval	SST_ANL (*fscl*)
swh	perform SWH analysis	CHK_FCST, COAMOA, CODA, CODA_POST, CODA_PREP, OCN_OBS

swh_da	swh assimilation option: bmrc = bmrc method; ncep = ncep method	SWH_ADJ (*opt*)
swh_time	swh observation processing option: obst = select obs based on time swh observed; rcpt = select obs based on time swh received at center; synt = select obs that are synoptic for analysis update interval	CR_SWH_FILE, RD_SWH
tmp_std	max number std dev to scale modas temperature from climatology	MODAS_CHKTMP (*max_std*)
tol_fctr	innovation error check tolerance factor: (1) ice; (2) sst; (3) ssh; (4) temperature; (5) salinity; (6) geopotential; (7) velocity; (8) swh; (9) layer pressure	DRCT_OBS, SET_ERR (*tol*)
topo_mn	minimum depth for ssh obs. (m)	DRCT_OBS, MODAS_TEMP, SSH_PREP
topo_mx	maximum depth for unscaled ssh observations (m). ssh data are scaled from full value at topo_mx to zeroat topo_mn depending on the bottom topography value.	MODAS_TEMP, SSH_PREP
topo_src	source of topography for scaling ssh: 'modas' - modas smoothed topo.; 'model' - forecast model topo.; 'ncoda' - turns off scaling of SSHA	SSH_PREP
upd_cyc	analysis update cycle (hours)	CR_ICE_FILE, CR_TMP_FILE, INIT_DATAO, OBS_DETREND, OCN_OBS, RD_ATSR, RD_GOES, RD_LAC, RD_MCSST, RD_METOP, RD_METOP_LAC, RD_MSG, RW_OCN_CNTRL, VRFY_FCST
vc_bkg	vertical correlation background: 'clim' = climatology; 'fcst' = model forecast	SET_VCORR
vc_mdl	vertical correlation model options: 'mixl' = mixed layer only; 'dens' = density stratification; 'temp' = temperature stratification; 'mono' =	SET_VCORR

	monotonic; 'cons' = constant; 'none' = no vertical correlation	
vol_scl	minimum number of correlationlength scales in an analysis volume: (1) ice; (2) sst; (3) ssh; (4) mvoi; (5) swh	SET_VOLUME (*vscl*)
warm_ice	set sea ice retrievals flagged as too warm ssts to zero ice and use in the analysis	RD_SSMI (*warm*)
wlist	white listed call signs	CODA_PREP, RD_GLDR, RD_PROF, RD_SHIP
z_lvl	analysis vertical grid (meters)	COAMOA, GRD_CONFLCT, SET_VGRD

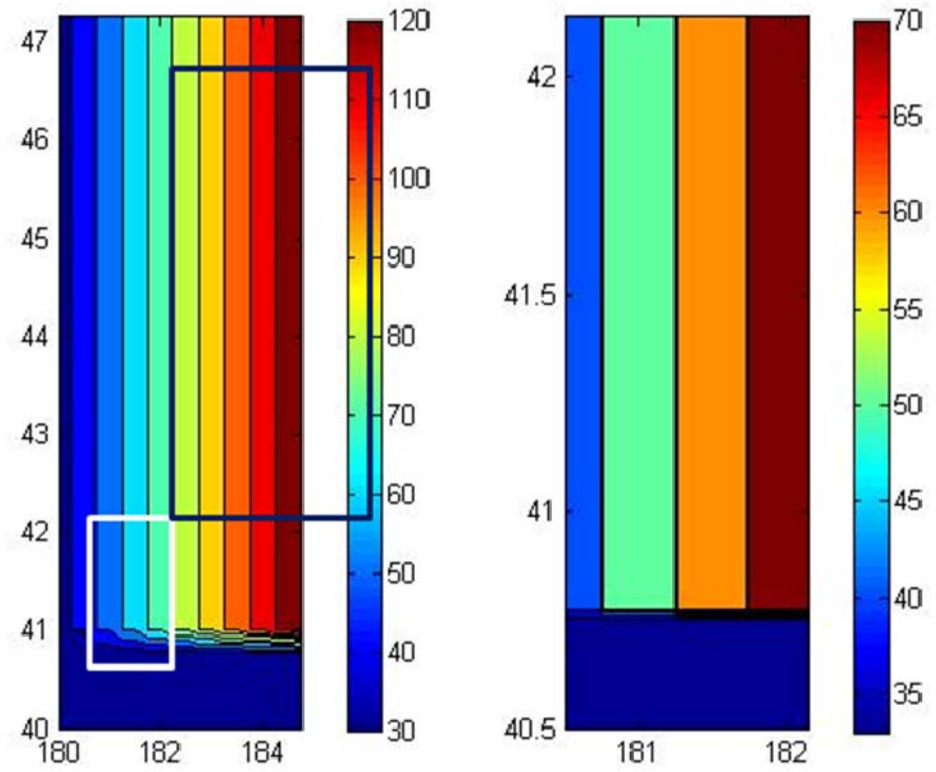

Figure 5.1-1. Plots of bathymetry for nest 1 (left) and nest3 (right), where nest 3 is a child of nest 2 (see Figure 4.4-2 rhs). The black rectangle on the left panel outlines nest 2 and the white is for nest 3. Note that nest 3 does not overlap with nest 2 and that nest 2 exceeds the domain of nest 1. In this example, however, nest 3 has nine times the resolution of nest 1.

6 Operation

The overall operation of the system is conveniently broken into two use cases, the 2D analysis and the 3D analysis. We will treat the analysis of significant wave height as a special case of the 2D analysis use case and discuss differences where they occur. The system operation as described in this section using UML diagrams supplemented with detailed explanations of important operations. An excellent summary of many of the algorithms is also given in *Cummings* (2005).

6.1 Overview

Execution within the NCODA system is partly determined by user action. This type of operation can be demonstrated using the UML use case diagram. The use case diagram for the NCODA system (Figure 6.1-1) shows the dependence of the analysis on user and file system input. Each of the lines between the user and a system component represents starting a separate program or editing a file. There are thus seven user operations required to complete a full 3d analysis. There are even more operations with the file server, as seen in section 3. Furthermore, if analyses are to be completed on a recurring basis, the dates must be changed before each sequence.

This overview will focus on the preparation procedure because it is where most of the user and program developer effort will be concentrated. There are three possible command line arguments to NCODA_PREP: (1) process option, (2) grid namelist file, and (3) analysis date-time group. Options (1) and (2) are required.

Program NCODA_PREP includes header files "coda_parms.h" and "gridnl.h" whereas subroutine COAMOA includes "coda_parms.h", "odsetnl.h" (directory namelist), and "oanl.h" (ocean analysis namelist). Subroutine CODA_PREP includes only "coda_types.h".

The sequence of computations in the NCODA_PREP program depends on the availability of previous analysis files (i.e., the "datao" CR file). If such files are present for the requested analysis date, the system restarts (Figure 6.1-2) and uses these analyses files instead of climatology files. Figure 6.1-2 is a UML Sate Diagram, which shows the states that objects of a class may assume and the transitions the objects may make between states. Thus, all of the objects (classes shown in Figure 2.2-1) may be in a restart or a nonrestart state as shown in Figure 6.1-2. We will refer to this state diagram as the Level 1 state. The circles that are partially filled at the bottom of the diagram indicate expansions of the diagram.

The restart state is expanded in Figure 6.1-3, which shows the circumstances that determine if a forecast analysis is possible or not, as well as the conditions for reading the observation data files. This sequence of states occurs in subroutine COAMOA, which is not listed in the class diagram of Figure 2.2-1. However, these two states are fundamental to the subsequent operations within the entire NCODA system and thus are critical in understanding the analysis cycle. The actual flow of control through subroutine COMOA can be shown using a classic Nassi-Shneiderman flow chart (Figure 6.1-4) (*Nassi and Schneiderman*, 1973). There are two points to be made concerning subroutine COAMOA. First, it contains the loop for the nests and second, it calls subroutine CODA_PREP for each requested nest. Another point that can be seen by comparing Figures 6.1-3 and 6.1-4 is that the restart state is omitted from the flow chart. This state primarily affects operations at a fine scale that does not lend itself well to the flow chart, which is better applied to coarser states like the 2d or 3d analysis. This results in many logical blocks using *fcst* as a variable.

We can expand the CODA_PREP block in Figure 6.1-4 into a more refined flow chart (Figure 6.1-5) that shows the preparation common to both 2D and 3D analyses. The header file, "coda_types.h", that is included in CODA_PREP lists 125 indexes for kinds of observations. These data are described as follows:

data_lbl	data type labels
inst_err	obs instrumentation error
rerr_scl	obs representative scale (km)

These data are not input from a namelist but are instead given in data statements within file "coda_types.h". Note that many of the variables change name within the argument list from COAMOA to CODA_PREP (see Appendix 2).

The grid decorrelation length array, **grd_scl**, is read from a restart file with a field of "grdscl" and a file type of "datafld". If this file does not exist, RD_ROSSBY is called to read the global 1 degree Rossby radius field from a file called "ROSSBY.baroc.radii", which must be located in the climatology directory (*clim_dir*). This field is not calculated in the program. It is interpolated to the analysis grid by FLD2_TRP. The only other option *locl*, is commented out. At this point, further computations depend explicitly on the value of *opt* (2d or 3d).

Before continuing to discuss the 2d and 3d options, it is useful to analyze the evolution of the analysis object between some of the other states defined within the system. For example, Figure 6.1-3 shows separate states for either a forecast (*fcst* = T) or not (*fcst* = F). This condition is common for the beginning of a series of analyses and forecasts. The UML State Diagram (Figure 6.1-6) clearly shows the different states that occur based on the mode (2d or 3d) specified in the "oanl" namelist or on the command line. If both are 2d it is a fully 2d analysis (e.g., swh) but if the namelist value is 3d, a different set of states are entered. If a 3d analysis is selected on the command line, the series of steps is straightforward: (1) generate synthetic xbt's; (2) prepare the intial guess from climatology; (3) process the 3d observations; (4) generate a synthetic sea surface height field; and (5) process all of the variables for the MVOI analysis. This diagram shows the difficulty of understanding the behavior of a sequential program that has a large number of states determined by guards (logical flags).

6.2 Two-dimensional analysis

The first step in completing a full 3D analysis (*ocn3d* = T) is to run all three components with *opt* = 2d on the command line. When this option is selected, execution does not actually split into different pathways. As suggested in the flow chart (Figure 6.1-5), the value of *opt* is used repeatedly to determine specific actions. Call OCN_OBS to read in the data for either a restarted or initial analysis. This section is summarized as well in sections 4a-c from *Cummings*

(2005). The date/time groups for all obs are checked against analysis time. Open output files for the observations. The sequence of reading and writing these observations is described in section 3.4. If *swh* = F, subroutine SST_PREP is called (Figure 6.2-1) to prepare the sst observations; it then calls CR_TMP_FILE to read the observations. These are read in the sequence indicated in Figure 3.4-1.

There is another block, however, that determines action based on the value of *swh* (T or F) as depicted in Figure 6.2-1. The flow chart shows that analyses for wave height or sea surface temperature are mutually exclusive but that sea ice is completed whether desired or not.

6.2.1 Processing Observations

The activity sequence for the *ocn3d* = T box in Figure 6.2-1 can be examined in more detail using a UML Sequence Diagram for SST preparation (Figure 6.2-2) that represents the various components of the system as objects from the classes given in Figure 2.2-1. The advantage of the sequence diagram is that it shows the interaction of objects as well as what tasks are being completed.

SST: All of the MCSST observations are read from the file system whereas the diurnal sst bias correction and pooling of observations are completed by subroutines from the transformation class. Valid sst observations are saved in an array named **mcsst_*parameter*.** NOAA and METOP data are pooled, but pooling is applied to day and night observations only for MSG and ATSR (microwave); AMSR, AMSR-E, and TRMM data are pooled. Ship data are not pooled but are checked for the black list.

PROFILES: Subroutine PROF_COLLECT is called to gather profile data. The first set of data comes from general profiles, which are read by sub RD_PROF: (1) Check white list; (2) Check black list and set error probability (1000 hardwired); (3) Check data denial and set *ep* = 1000; (4) Check space and time window; (5) Find relative age; (6) Go over all input levels (*n_lvl*) and use errors to save a valid profile; (7) Go over this profile in sub PROF_ERR and calculate the temperature error for all levels where *tmp* > -99. This error consists of instrument error (input), climatological variability (.01×T), and vertical gradient (0.5×T). Salinity uses a flag (*sal_typ* = 33) to use input value. Otherwise, calculate like *tmp* except use the input error where *sal_flg* = 12, 14, 22, 24, 36, 46, 48, or 50; (8) Finally, check that all errors are > 0. Back in sub RD_PROF, assign the profiles to permanent variables, *prf_age*,...etc. This procedure is repeated for 1 day earlier data files as with satellite data. Analyze all profiles for argo type codes (*prf_sgn*(2:2) = 9 and *prf_sal_typ* = 37). If *argo_bias* = T in sub PROF_ARGO, find levels below 800m, and add the difference between climatology and the value at that depth. In sub PROF_THIN, interpolate the levels to a fixed grid (*zlvl* with 67 levels) if there are more than 140 levels. Pool profiles in sub PROF_POOL. There are a number of mooring names listed in data statements that are used to locate duplicates. A time-based weight is found. However, if *prf_sal_flg* = 12, 14, 22, or 24, do not average. It looks like these are the only moorings that will be averaged.

PROF_COLLECT then calls sub PROF_SLCT to apply the selection criteria. Find the max separation between observed adjacent levels and (independently) between observed *tmp* at adjacent levels. Reject a profile that meets any of the following criteria: *n_lvl* < a predetermined minimum number of levels; no samples above a specified depth; deepest level too shallow and less than a pre-defined fraction of total water depth; max spacing between adjacent levels more than a predefined value and max temp change greater than preset value; this same is tested using both max temp and level values. The min/max values used in this selection procedure are held in **prf_slct** arrays for each profile. These values are held in array **prf_slct**, which is input from namelist "oanl.h." Sample values are in square brackets. The values in **prf_slct** are: (1) level gross error [0.99]; (2) number of sampling levels [5]; (3) ratio of sample/bottom depth [.5]; (4) min sampling depth [300]; (5) distance between levels [300]; (6) temp between levels [5]; (7) depth difference at max *delt* [100]; (8) temp diff at max *delz* [2.5]; and (9) depth of first sample [50]. There is a debug variable that can be used to look closely at this process.

Duplicate profiles are removed in sub PROF_DUPCHK. The duplicate profiles need to be within 0.5e-6 of the mean grid spacing to be dropped. The one with most samples is retained. Sub PROF_COLLECT then assigns profiles to temporary arrays and writes them to a file (unit 32) if *debug* = T. Place profile data into 1d arrays with all the parameters (e.g., *wrk_age, wrk_lat*) assigned to individual data points (unrolled) for writing to the COAMPS input files. Check for bad ssh data flag (*deny* = 30). Assign a subset of the profile properties (e.g., *prf_age, inst_err*) to work arrays for ssh (e.g., *ssh_cls*) where the depth > 150 m. Write the **wrk_*property*** and **ssh_*property*** arrays to binary files (*unit_xbt, unit_ssh*).

GLIDERS: Glider data are processed by calling a set of analogous subroutines (to those for profiles) from sub CR_TMP_FILE. There are a series of checks for *prf_time* to set the date for the input glider data ('rcpt', 'obst', 'cycl', 'synt'). Count the number of glider observations in sub OBS_COUNT. Then call sub GLDR_COLLECT, which first calls RD_GLDR to input the observations by looking in the same data directories. It starts with the first file date and goes back 1 day on subsequent passes.

After the *n_read* observation sets are read in, loop over them and do the following: (1) Check white list; (2) Check black list and set error probability (*qcs*, *qct* = 1.e3); (3) Check data denial; if *ob_tmp_typ*/*ob_sal_typ* = any deny code, *qct/qcs* = 1.e3; (4) Check space and time window. This is dependent on *prf_time* = 'obst', 'cycl', 'synt', or 'rcpt'. Each glider can have different levels (*ob_lt*). The arrays **ob_tprob**/**ob_sprob** contain QC flags. If **ob_tprob**/**ob_sprob** > 1.1 the probabilities are adjusted; (5) Check white list again for any adjusted flags; (6) Check the adjusted probability error against the limits (*lvl_prb* = **prf_slct**(1) from namelist "oanl"). Only use those with smaller salt and temp errors; (7) Go over this profile in sub PROF_ERR (also used for profiles) and calculate the temperature error for all levels where *tmp* > -99. This error consists of instrument error (input), climatological variability (.01×T), and vertical gradient (0.5×T).

Salinity uses a flag (*sal_typ* = 33) to use an input value. Otherwise, calculate like *tmp* except use the input error where *sal_flg* = 12, 14, 22, 24, 36, 46, 48, or 50. Finally, check that all errors are > 0. After returning to sub RD_GLDR, save the variables in work arrays, **gldr_*variable***. Do this procedure for each day (decrease *dtg* by 24 hrs) and specified directory. Make sure that the glider obs do not exceed *max_lvl* (140) in sub PROF_THIN as with the profiles (described above). The selection criteria for gliders are applied in sub GLDR_SLCT, which is analogous to PROF_SLCT (above). The exact same criteria are used as with profiles (**prf_slct**(7), (5), (8), (6) except it is not modified by a factor of 1.8, (2), (3), (4), and (9)). The main difference between PROF_SLCT and GLDR_SLCT is that arrays **age**, **btm**, **lat,** and **lon** are 2d for gliders. The index for profiles is *n_obs*, but for gliders (*n_depth*, *n_obs*) are used. Finally check for duplicates with sub GLDR_DUPCHK (analogous to PROF_DUPCHK). Save obs to work arrays, unroll, and write to *unit_xbt*. The format is the same as for profile data.

At this point, control leaves CR_TMP_FILE and returns to OCN_OBS. The second option to calling CR_TMP_FILE is when *swh* = T. In this case *dtg* variables are updated from old values and nothing else is done before continuing on to ice. After processing sea ice, sub CR_SWH_FILE is called to collect and count the swh observations. OCN_OBS lastly saves the dtg cuts before returning to CODA_PREP.

SSH: If a 3d analysis is being completed (*f3d* = T), evaluate *ssh_time* as with other variables (e.g., 'rcpt', 'obst'). A typical value is 'obst'. Loop over the directories and times (backwards by 24 hours from current *dtg*) and go through the obs to check the time and space window. Compare the obs probability (*ob_qc*) to the value *qc* = *qc_prb* (in CR_SSH_FILE) = **qc_prob**(7) (in OCN_OBS) = **qc_err**(7) (in namelist "oanl") ~ 1 to determine if checking is necessary. Check for data denial and save the variables to arrays **ssh_*variable***. Pool satellites by setting *ssh_typ* = 81, instead of individual codes (e.g., 12, 13, 14, 53, and 55).

After each kind of observation has been processed within subroutines SST_PREP, SWH_PREP, ICE_PREP, and SSH_PREP, any existing analyses are read by subroutines RD_SST_ANL, RD_SWH_ANL, RD_ICE_ANL, and RD_SSH_ANL, respectively. These subroutines actually call RD_OCN_ANL.

6.2.2 Detrending the Observations

Sub OBS_DETREND is called by the preparation subroutines (ICE, SSH, SST, SWH, and finally MV) to compute the observation innovations from the first guess fields (*obs_val* = *obs_val* and *variable_age* = *guess* in OBS_DETREND) on the analysis grid. The appropriate analysis fields are found as described in section 7.3. The analysis ages (*swh_age* = *guess* in OBS_DETREND) are the first guess. If the current analysis is a restart, or if *fgat* (first-guess appropriate time; see section 7.3) < 0, sub FLD2_TRP (interpolation) is passed the analysis swh age (*guess* = *fld* in FLD2_TRP). This field is located at integer values of the analysis grid indices, whereas the obs are located at real values on this grid. FLD2_TRP will interpolate the analysis field to the observation grid and return it to sub OBS_DETREND as the array **value**(*n_obs*). These interpolated (real-indexed) first-guess fields are then subtracted from the real-indexed observations to produce a first guess of the error, which is called an innovation (see *Cummings*, 2005). The resulting array is **obs_anm**(*n_obs*). For other cases (i.e., *fgat* > 0 or not a restart), the initialization uses available swh age observations as described below.

A temporary dtg is calculated by subtracting the number of hours in the current window from the analysis dtg. Sub RD_OCN_ANL is called to look for an appropriate restart file (e.g., *file_typ* = 'fcstfld', field = 'sigwht'). If no suitable field is found, the analysis (held in *guess*) is used. Interpolate the swh age (analysis) to the real-valued indices of the obs on the analysis grid using FLD2_TRP. Save the interpolated forecast using the index contained in *ndx*, which points to the full observation array. Note that the observations were rolled up in the data file and thus all times and locations are included in **obs_val**. This index scheme, coupled with the loop over available forecast fields, assures that each data point (time and space) now has a unique forecast data point (or a special value) assigned to it.

To recap, compute the first guess innovation using the available analyses by subtracting the unique forecast point from the observation on the real-valued indexed analysis grid. This completes the detrending of the swh age observations.

6.2.3 Form Super-Observations for Water Types

An overview of this procedure is given in section 4b of *Cummings* (2005). Observations are removed that are outside the analysis grid boundary or which failed in the interpolation using sub OBS_REMOVE. Subroutine SUPER_OB is called to form super observations from redundant observations. Super obs are formed within water mass classifications and observation data types at the analysis grid mesh interval. Up to 20 water mass codes are allowed. The parameter *n_spr* = 20 in SWH_PREP. Observation bins and time weights are based on simple grid parameters and age relative to the decorrelation time scale (*tscl* = 48 hrs), which is hardwired in sub SWH_PREP. Count the number of data types (*obs_type* from data file) and water mass classification codes (*obs_cls* from data file). Loop over water masses and super obs data types and do the following: bin super obs data types using grid mesh; sum within water mass and data type; mark obs as used; compute super obs as time-weighted local averages. Clean up by set number super obs, removing super obs data types from data arrays, transfer super obs to data arrays, reset the obs counter. The result is that the data arrays, age, anm, cls, etc, contain point data in the first indexes and super obs in the later indexes. Now call OBS_INDEX to calculate the obs (now including super obs) in real-indexed points on the analysis grid using sub OBS_INDEX. Correct round off errors for obs locations < 1 or > *m*, *n*.

Call sub SET_ERR to compute normalized observation error and perform innovation error check (if requested).

6.2.4 Set the Observation Error Field

This method is described in section 3e of *Cummings* (2005). Instrumentation errors are increased using the age of the observation from the analysis dtg. Observation ages are depth dependent so that older, deeper observations have similar errors as younger, shallower observations. A simple formula is used to approximate an error of representativeness that takes into account grid mesh and background correlation scales. Error of representativeness is defined as the uncertainty of a single ob within a particular space-time interval in representing the mean over that interval, given the expected space-time variability. The error of representativeness is added to the instrumentation errors and the final observation error estimates are normalized by the background prediction errors. For super-obs, observation error is reduced by the number of obs (n) used to form the super-ob by 1/sqrt(n).

Each obs has a default error and offset initialized in the beginning of subroutine SET_ERR (e.g., SWH default=1.; offset= 0.1). The specific error estimation method depends on the variable *err_mdl* (*emdl* in namelist "oanl"), which is often 'complx'. If *err_mdl* = 'simple', the prediction error is found from the background error (*bkg_err*, or *ebk*g in "oanl"), which uses default values unless changed in the namelist. It includes an interpolation factor between obs levels and grid levels. If *err_mdl* = 'complx', the grid error (*grd_err*) is interpolated to the real-valued indices (on analysis grid) of the obs using FLD2_TRP or FLD3_TRP. There is some uncertainty in the execution of this operation, as described in section 7.4.

6.2.5 Forecast/Analysis Fields

The current discusion will focus on swh but the processing is the same for the other analysis variables. The analysis fields are read from restart files (see section 3.6) written by previous analysis cycles. If *fcst* = T (NOTE: sub CHK_FCST already set *fcst* = F if appropriate analysis files are absent), RD_OCN_ANL checks for a forecast file (*file_typ* = 'fcstfld'); otherwise, an analysis file is sought. If the requested file (CR file = 'fcstfld'/'fcsterr' or 'analfld') is present, RD_SWH_ANL sets mask and *clm* =0 for points where swh = a special value. A forecast also requires a forecast error, *file_typ* = 'fcsterr', which is also read by RD_OCN_ANL. If this file is not read, RD_SWH_ANL will exit unless the run is a restart, in which case restart files with *file_typ*='fcsterr' and 'modlerr' are written.

Subroutine RD_SWH_ANL then repeats this procedure (reading either forecast or analysis files) for *file_typ* = 'timefld'. It then goes on to look for forecast or analysis files with field='seaice', *file_typ*='analfld', and *fluid*='o'. However, if these are not present, an ice climatology file can be read by sub RD_ICE_CLIM, which must be read or sub RD_ICE_CLIM exits. These fields are passed back to sub SWH_PREP as **swh_***, which is a little confusing because these names were used as work arrays when writing the observations to the restart files in sub CR_SWH_FILE. The lat/lons of this field are contained in arrays **grd_lat** / **grd_lon**, which are only used for interpolating to the analysis grid for irregular grids (*igrid* < 0).

If the climatology and forecast/analysis fields have been read successfully, the **swh_age** array is increased by the update cycle (*upd*=*upd_cyc*) (see section 7.1). The next step is to call sub RD_DATA_FILE to read a work file (e.g., "SWH" and "obs") that was written by sub OCN_OBS (actually by CR_SWH_FILE using a call to RD_SWH) as described above.

There is an apparent discrepancy here that needs to be examined more closely. The swh observations are read in sub RD_SWH as it loops over directories to look and reads unformatted files (see section 3.4). The output file for these data after combination is a work file in sub OCN_OBS (see section 3.5). Note that the nest number is hard-wired to 00. This file is rewound after opening. In sub RD_DATA_FILE, the nest number is a variable and could well be greater than 0. This is a potential problem.

6.2.6 Calculate Horizontal Correlation Parameters

The processing of error covariances is discussed in section 3 of *Cummings* (2005). Correlation parameters are computed by sub SET_HCORR. They are defined horizontally as a second-order autoregressive (SOAR) function. Horizontal correlation scales are non-homogeneous with location and can be scaled by a factor that is both analysis-variable and grid-mesh dependent. Homogeneous statistics can be specified by setting the *opt* argument to 'homo' and user-defined length scales can be used if the argument is 'locl'. If *opt* = 'rsby', a scale factor is found using *delx* and *dely*, and the input **grd_scl** (previously read from a file or calculated from the Rossby radius in GET_HSCL). Grid cells with water depth > 5 m are used. The input Rossby radius scaling factor, *sfct* (*rscl* in "oanl") is used along with the mean Rossby radius for the grid to calculate a new *hcorr* that is only dependent on grid dimensions and the previously read/computed *hcorr* (in GET_HSCL). If *opt* = 'homo', the result is similar except that the locl grid dx and dy are not used to find *hcorr*. If *opt* = 'locl', the **grd_scl** array is used directly. This is how to get information about the structure of the fields into the algorithm.

6.2.7 Compute Analysis Volumes

This is described in section 2 of *Cummings* (2005). Subroutine SET_VOLUME is called to coordinate calculating the analysis volumes. First, it calls sub VOLUME_INIT, which divides the analysis region into quarters unless *n_obs* <= 1, in which case only 1 volume is used. Consequently, *n_vol* is either 4 or 1. Then sub CR_VOLUMES is called to create the volumes. It computes the average volume size in mesh units. Each obs location is then compared to each volume's limits and assigned. This double loop uses OpenMP (OMP) parallelization. Find the max number of obs in a volume and the mean volume size (*vol_siz*), which must be smaller than the average volume multiplied by the maximum number of correlation length scales in an analysis volume (*vol_scl* in "oanl"). If any are not, they are all halved, resulting in four times the number of original volumes. The parameter *mx_vol* must exceed *n_vol* × 8+*n_vol*, which is *n_vol* × 9; this test is probably written this way because the value 8 is intended to be adjusted by a program developer and thus be adjustable. The intent is that *mx_vol* needs to be at least one greater than the expected number of volumes. A test case for one loop was run and *n_vol* became 16; thus, *mx_vol* > 144 for one time through. This number is a parameter (*mx_volumes* = 4096 × 8+4096 = 36864) in sub SET_VOLUME. SET_VOLUME next calls sub CR_OVRLP_VOL, which creates volumes that overlap the dynamically created analysis volumes. Overlap volumes with identical centers are considered duplicate volumes and are removed. Analysis volumes that have no observations or grid points are also removed. The saved volumes are saved to a work file (e.g., "*wrkdir*/coda.SWH_vol.nest01.*dtg*"). These definitions are also saved to a CR file (e.g., "voldata"; see Table 3.6-1).

6.3 Three-dimensional analysis

The control flow within the NCODA_PREP component is straightforward: (1) check the command line arguments; (2) open the grid definition file and read the namelist; and (3) call subroutine COAMOA to process the individual nests, as shown in Figure 6.1-4. This option continues in subroutine CODA_PREP. The sequence of operations for a 3D analysis are shown in Figure 6.3-1.

6.3.6 Generate Synthetic Fields

Sections 4c and d of *Cummings* (2005) give an excellent overview of this procedure. If the analysis is a restart and modas is used (*mds_grd* = T), subroutine MODAS_GRID computes modas synthetic temperature and salinity profiles on the analysis grid from modas 2.1 data bases. An optional pathway through CODA MVOI analysis is available for use as initial conditions in a cold start (Block [DIRECT ? OR MODAS ?] in Figure 6.3-1). Existing analysis ('analfld') and forecast error ('fcsterr') 2D fields are read from CR files: 'seaice'; 'seatmp'; and 'seahgt'. Synthetic temperature and salinity fields are calculated with sub MODAS_SYN. The MODAS databases are interpolated to the analysis grid and time and temperature, mixed layers, and errors are extracted; the salinity is found from the temperature. If a criterion (*obs_ice* < *ice_cvr*) is met, ice-covered sea values are returned. The calculation of potential temperature depends on temp and salt not failing in the generation of synthetics but there are no contingencies (else...).

GDEM 3.0 climate temperature and salinity fields are read by GRD_CLIM (called by MODAS_GRID) and expanded to the analysis grid. Calculate forecast errors (array **anm**) from the anomaly of GDEM climate fields, **wrk**(*k*,3)and **wrk**(*k*,4), relative to MODAS synthetic fields, **wrk**(*k*,1)and **wrk**(*k*,2). Use these anomalies to find horizontally averaged forecast errors. The difference between the GDEM climatology and MODAS synthetics (using climatology fields) is being called a forecast error for a restarted analysis. The biases are calculated from these values. The resulting temperature ('seatmp') and salinity ('salint') are saved as 'analfld' fields. The RMS errors, **wrk**(*k*,5) and **wrk**(*k*,6) are saved as 'fcsterr' fields. The differences between MODAS and GDEM fields are saved as 'analinc' files. Call GRD_CLIM to retrieve GDEM climate fields. This subroutine reads from the database. The actual reading is done in GDEM_GRD, which is contained within file "gdem_mod.f". This subroutine is passed the GDEM variable name (e.g., *tmp* in CODA = *tstd* in GDEM) to read. If the option passed to GRD_CLIM is 'fld', other GDEM variables are read ('temp' and 'salt'). These 'std' GDEM variables are extrapolated to the analysis grid and written to 'climerr' files by WR_OCN_ANL.

The geopotential fields associated with the MODAS temp/salt, **wrk**(*k*,1) and **wrk**(*k*,2), and GDEM temp/salt, **wrk**(*k*,3) and **wrk**(*k*,4), are calculated and placed in arrays **wrk**(*k*,5) and **wrk**(*k*,6), respectively. The MODAS geopotential is then written to a 'analfld' file and the GDEM geopotential is written to a 'climfld' file. The difference between these geopotentials is the climate error; it is used to find the horizontally averaged bias and rms error, **gpt_bias**(*k*) and **gpt_rms**(*k*), respectively. These are later saved to the **stats** array and written to the 3D restart 'datafld' file for field 'cvstat'. The RMS error between the analysis (MODAS) and climatology (GDEM) is written to a 'fcsterr' file. The difference (named *increment*) is written to an 'analinc' file.

Sub GEO_VEL is called to calculate the geostrophic velocities from the geopotential associated with the analysis (MODAS) and climatology (GDEM). These components are then written to 'analfld' files for 'uucurr' and 'vvcurr' (MODAS) and 'climfld' files, 'uucurr' and 'vvcurr' (GDEM). The horizontally averaged bias (**uuu_bias**(*k*) and **vvv_bias**(*k*)) and rms error (**uuu_rms**(*k*) and **vvv_rms**(*k*)) are then calculated. As with the other variables, the 3D rms errors (**wrk**(*k*,5) and **wrk**(*k*,6) for u and v, respectively) are calculated and written to 'fcsterr' files 'uucurr' and 'vvcurr', respectively. The bias (*increment*) is then computed and written to 'analinc' files 'uucurr' and 'vvcurr'. If *model* = 'HYCOM' or 'hycom', a dummy layer pressure field is saved to a 'fcsterr' file for 'lyrprs'.

Dummy observation age 'timeinc' and 'timefld' files for 'grdage' are written, and the stats array is filled with the layer-averaged biases and rms errors: **tmp_bias**, **sal_bias, gpt_bias**, **uuu_bias**, **vvv_bias**, and **lyp_bias** (similar for rms except that they are squared).

6.3.7 Direct Assimilation Of Modas Synthetics

If a MODAS assimilation is requested (*modas* = T in "oanl"), sub SYN_SMPL is called to generate synthetic BT sampling locations. This subroutine is intended to introduce a subset of the MODAS field as if it were measured. Thus, the locations of the BT's are altered on successive update cycles.

The average Rossby radius is found and an 'analinc' file of 'seahgt' is read for the ssh anomalies. The sampling interval to select BT's from the SSH anomaly field uses the Rossby radius and a user-input factor, *f* (*smpl* in CODA_PREP and elsewhere, from "oanl"), to calculate *ms*, which is the index skip interval. The data type (*sample*) is assigned *syn_typ* (set to 15 in CODA_PREP) for later data processing. The tolerance to include a grid point (*ssh_del*) as a synthetic profile is input from "oanl" as *del_ssh*, but only for cold start analyses or forecasts.

6.3.8 Prepare Climatology Data for MVOI

Sub CLIM_PREP (see Figure 6.3-1) is called to prepare climatology fields for MVOI. First, it tries to read a CODA climatology file using sub OCN_CLIM. This file has *file_typ* = 'climfld' and *fld_name* is either 'salint' or 'seatmp'. If it doesn't exist, sub GRD_CLIM is called as in the GDEM climatology algorithm. Restart files are written for these two variables. Potential temperature is then calculated before moving on to geopotential, 'geoptl'. The potential temperature and salinity fields are used to calculate the geopotential fields only if there were no pre-existing files (from a previous analysis). If temp and salt files do exist (*clm_fail* = F), an ocean analysis file is read for geopotential; if this is inexplicably missing, the program exits. This field (whether read or calculated) is then written to a new file with a different *dtg* value. This procedure is repeated for geostrophic currents using sub GEO_VEL if *clm_fail* = T, or read from an analysis file otherwise.

The final step in prepping the climatology fields is to collect the error fields. Sub OCN_CLIM is now called for *opt* = 'std' and *prm* = 'sal' and 'tmp'. New climate files are written for *file_typ* = 'climerr' and *fld_name* = 'salint' and 'seatmp' by sub WR_OCN_ANL. Note that this procedure is much simpler than for the previous case, in which fcsterr fields were created as well.

6.3.9 Prepare Gridded Obs. (MASS_OBS and VELC_OBS)

The 'seatmp' fields (*file_typ* = 'analfld', 'analinc', 'anlerr', and 'fcsterr') must be present or the program will exit. Sub RD_OCN_ANL is used to read the restart files. Only 'anlerr' will not stop execution. The sst analysis error reduction (**err_red)** is limited to values between 1.0 and 0.2 for good points, and 0.5 for bad points.

3D fields are read if available for either *file_typ* = 'fcstfld' or 'analfld' and *fld_name* = 'seatmp' but for multiple levels. For *restart* = T, a 'climfld' file, which uses GDEM climates, is sought if the file doesn't exist. This procedure is repeated for *fld_name* = 'salint'. For cold starts, execution stops.

The temperature error field, 'fcsterr', is sought next. If it fails, a *file_typ* of 'modlerr' is attempted to be read for *restart* = T; if that fails, a file of type 'climerr' is read; if that fails, execution stops. Execution stops for *restart* = F.

Sub RD_OCN_ANL attempts to read a restart file for *file_typ* = 'fcstfld', *fld_name* = 'mixlyr', and *lvl_typ* = 'sfc'. There are no contingencies. After returning to MASS_OBS, the MLD is calculated by sub CODA_MLD if there was no 'fcstfld' file. This algorithm calculates a potential density from whatever was read by sub MASS_FLD. If these were forecast fields, it will be forecast mld; otherwise, it could even be a climate value. The MLD is based on a density gradient, *den_ds*, which is input from "oanl."

There is a control array, **st_grid** (**st_grd** in "oanl" and COAMOA), that indicates a preference to generate SST observations from analyzed SST grid. If this is on (default), sub SST_ANL is called to generate SST obs by sampling analyzed SST anomalies. The SST grid is sub-sampled as described above but no scaling factor is used (i.e., *ms* = 2). If *st_chain* (from "oanl") = T, the SST are extended to the base of the mixed layer. The other choice is to call sub SST_OBS to read sst super obs, set synthetic BT predictor variables, and remove SSTs outside the grid domain. Sub RD_DATA_FILE is called to read 'SFC' data and real-valued indices for the obs are found on the analysis grid. The observed SST is extended to the base of the mixed layer.

The SSS are calculated from a table for *st_grid* = T (analysis) or found from observations. If *sal_adj* = T (from "oanl"), level 2 is also used to adjust the salt balance. The profile of salinity is found iteratively from SST observations until the density gradient between layers 1 and 2 is 0 (neutral buoyancy). The salinity is extended to the base of the mixed layer and the obs arrays (e.g., **obs_age** (l,*k*))are set.

If *n_prf* > 0, sub PROF_OBS is called to: read observed profiles at inflexion levels; set synthetic BT predictor variables; depending upon the value of *opt*, it will interpolate inflexion levels to analysis levels or leave profiles at observed levels; and remove profiles outside grid domain. This algorithm processes the entire profile simultaneously, rather than treating each depth value as an independent observation; i.e., the arrays are dimensioned as **prf_age** (depth, profile). The data are read from a data file by RD_DATA_FILE, which is passed ‘PRF’ as an option Note that *nest* is hard-wired to *zero* (see section 3.5). The name of these data files is:

“*wrkdir*/coda.PRF_obs.nest*nest.dtg*”

This file can be written by sub PROF_COLLECT as described in section 3.2.2.7 above. The data are returned to PROF_OBS as 1D work arrays. The horizontal grid indices are computed by subs LL2IJ or IRREG_LL2IJ and the input profile data points are rolled into a set of 2D arrays; e.g., **prf_lat** (depth, profile). Note that the 1D arrays used to store the profile observations here are the same as in sub PROF_COLLECT.

PROF_OBS next computes the vertical grid indices for the obs on the grid, interpolates the depth to profile obs locations, extends profiles to bottom, removes obs off grid and failed interpolations, appends to obs arrays, and updates counters. The profile variables are saved on analysis levels if *opt* = ‘anlz’. The salinity type is set to 51 and temperature is set to 50. Control then returns to MASS_OBS, which removes synthetic profiles near profile obs locations by setting **sample**() to zero. A file of mass obs is written by MASS_OBS before it returns to CODA_PREP: “*wrk_dir*/coda.MASS_obs.nest*nest.dtg*”

Subroutine VELC_OBS reads velocity obs, removes observations outside grid domain, rotates to grid orientation, and saves on input file for ocean MVOI. This is very similar to what MASS_OBS does. The steps are: (1) read velocity obs file; (2) compute COAMPS horizontal grid indices; (3) compute vertical indices; (4) interpolate x and y grid positions to obs locations; (5) remove obs off grid and failed interpolations; (6) rotate spherical u,v obs to grid orientation; (7) save obs on MVOI input file. However, (7) is done within sub VELC_OBS instead of using a subroutine.

6.3.10 Generate Direct Assimilation SSH Synthetic Obs

This approach is also discussed in section 4c of *Cummings* (2005). This block in CODA_PREP executes if *direct* = T and *n_simpl* > 0. Subroutine DRCT_OBS computes temperature and salinity at selected grid locations using direct assimilation of changes in observed sea surface height from model forecast height as measured by satellite altimeters. The modified temperature and salinity profiles are then used as supplemental observations in the MVOI. Forecast model errors at the update cycle are used as observation errors, taking into account convergence of the density corrections and prediction errors in the analyzed SSH.

Sub DRCT_FLD tries to read the analyzed SST from a restart file with *file_typ* = 'analfld' and *fld_name* = 'seatmp'; execution fails if the file is missing. It also tries to read the 'seaice' analysis field. There is no dependency on the value of *restart.* The SSH innovation file, 'seahgt' of type 'analinc', must also be present in addition to the forecast error 'fcsterr' file. The analysis error 'analerr' file can be absent. The value of **err_red** (n*m) from the analysis error is set to 0.5 for a missing file, and limited to 0.2 to 1 if it exists.

The forecast (*fcst* = T) or analysis salt/temp fields are then read; execution stops if either is absent. The forecast error ('fcsterr') fields are next read, even if *fcst* = F. If unavailable, model error and climate error are read, respectively. If none are available, a constant of 2 is used for temp and 0.5 for salt. The last required field is the MLD 'mixlyr'. It is calculated if absent.

DRCT_OBS next calls MODAS_TOPOG to get the smoothed MODAS bathymetry. The synthetic profiles are generated from the input fields for the entire grid, except where obs are available (**sample**() = 0 in section 6.3.9). Temperature inversions are corrected in sub TS_STATIC, where the potential temperature (from sub THETA) replaces *tmp* and a new density is found. The comment in sub DRCT_OBS says that TS_STATIC will correct forecast model for temperature inversions. TS_STATIC will use the input *tmp* if *pt_anl* = T (set in "oanl") or the potential temperature otherwise. The default is T. This means that the specific volume (found in function SVAN) uses the forecast temperature. TS_STATIC rechecks the density gradient and sets **stat_lvl**(*k*) = F. Salinity is then adjusted in sub SALT_CORR to achieve static stability.

The direct assimilation has a fixed number of model levels (*nd* = 41 in sub DRCT_OBS). The *sal* and *tmp* variables are interpolated to this vertical grid. Sub SSH_DRCT is called to iteratively solve for temperature and salinity increments from the forecast model state that balance the observed pressure difference at the surface. Computation of the increments is between the depth of the mixed layer at the surface and the level of no motion at depth, which is hard-wired to 2000 m. The forecast model profiles are corrected using the pressure, temperature, and salinity increments (interpolated to the analysis grid) in DRCT_OBS.

Sub DRCT_ADJ is called to adjust the synthetic profiles for temperature inversions that are not present in the forecast and corrects salinity to ensure static stability. This subroutine repeats the previous MLD and inversion analysis from TS_STATIC after removing the inversions. This is slightly different than for the forecast, which made no adjustments before calling TS_STATIC. Note that this function finds the *obs_sal_err*/*obs_tmp_err* by comparing the adjusted *tmp*/*sal* to a Cooper-Haines profile (*Cooper and Haines*, 1996). Thus, it is important to know how far a region is expected to vary from this profile.

If *do_invc* = T, check the innovation error, which was found from the forecast error (*serr*) and the observation error (*obs_sal_err*). The tolerance (*tol* = *tol_fctr* in "oanl") is used to reject an observation. Observations can also be rejected if the residual pressure (*ps_in* – *ps_out*) exceeds a tolerance. The units are m^2/s^2 and *xp_mx* is hardwired in sub DRCT_OBS. This pressure anomaly comes from *ssh_anm* = *ps_in* (from CR *file_typ* = ‘analinc’ and *fld_name* = ‘seahgt’) subtracted from *ps_out*, which is the pressure anomaly after adjusting the temp/salt to balance the observed pressure change at the surface (i.e., *ssh_anm*). The value of *xp_mx* = 100 m^2/s^2. An observation can also be rejected for a lack of stratification.

6.3.11 MODAS Assimilation of SSHA Synthetic Obs

This block is located on line 998 of sub CODA_PREP. It executes if *modas* = T (see Figure 6.3-1). Sub MODAS_OBS is called to compute temperature and salinity at selected grid locations from modas 2.1 climatological data bases for use as supplemental obs in CODA analysis. The modas obs are appended to the in-situ observations. The MODAS levels are predefined.

The ice concentration for an ice-covered sea is predefined to be 55%. This affects the MODAS assimilation. The parameter *n_regns* = 36 × 18, is used to dimension modas arrays. The following fields must be present or execution stops: ‘seaice’ and ‘analfld’; ‘seatmp’ and ‘analfld’; ‘seatmp’ and ‘fcsterr’; ‘seahgt’ and ‘analfld’; ‘seahgt’ and ‘fcsterr’.

Synthetic BT’s are generated where obs are not available (*sample* > 0). Profiles are either assimilated on the observed levels (*prf_opt* = ‘obsz’) or after interpolation to the analysis levels (*prf_opt* = ‘anlz’) using the following code from line 427 in sub MODAS_OBS:

```
m = 0
 if (prf_opt .eq. 'anlz') then          This indicates that interpolation
                                        should occur
   do n = 1, n_lvl
     if (grd_lvl(n) .le. depth(k)) then
       m = m + 1
       obs_age(m,n_data) = 0.
       obs_lat(m,n_data) = grd_lat(k)
       obs_lon(m,n_data) = grd_lon(k)
       obs_lvl(m,n_data) = grd_lvl(n)   Assign grd_lvl to obs_lvl
       obs_xi(m,n_data) = real (i)
       obs_yj(m,n_data) = real (j)
       obs_zk(m,n_data) = real (n)      Assign the grid index to the obs
                                        index
     endif
   enddo
 else if (prf_opt .eq. 'obsz') then     This indicates that the observed
                                        levels should be used
   do n = 1, 37                         Interpolate over the modas levels
     if (modas_lvl(n) .le. depth(k)) then
       m = m + 1
       obs_age(m,n_data) = 0.
       obs_lat(m,n_data) = grd_lat(k)
```

```
          obs_lon(m,n_data) = grd_lon(k)
          obs_lvl(m,n_data) = modas_lvl(n)   Assign the modas index to the obs
          obs_xi(m,n_data) = real (i)
          obs_yj(m,n_data) = real (j)
          do l = 2, n_lvl                    This is an interpolation loop
                                             from modas levels to grid levels
            if (modas_lvl(n) .ge. grd_lvl(l-1) .and.
*                            modas_lvl(n) .le. grd_lvl(l)) then
               obs_zk(m,n_data) = real(l-1) +
*                                 (modas_lvl(n) - grd_lvl(l-1)) /
*                                 (grd_lvl(l)   - grd_lvl(l-1))
            endif
          enddo
        endif
      enddo
    endif
```

This looks like the input fields, which are on the analysis grid, will be interpolated to the analysis grid. This may just be a question of confusing terminology for the different vertical grids used by the analysis, MODAS, and observations. Note that the MODAS synthetics are treated as observations. After completing this preparation for interpolation, we retrieve the MODAS bathymetry again and assign the *n_data* obs to the predefined *n_regns* MODAS regions in a DO LOOP:

Sub MODAS_OPN is called to open the MODAS database files. It tries (success not mandatory) to open bracketing files:

> "*clim_dir*/m*lonhemlat_mon1*.b" and
> "*clim_dir*/m*lonhemlat_mon2*.b"

and returns. The next step is to select the obs that fall within this region and call MODAS_SYN to generate synthetic temperature and salinity. This subroutine was also called by MODAS_GRID as described in section 3.4.1, and a detailed description will not be repeated here.

This completes the *n_regns* loop.

If an analysis of the potential temp is requested (*pt_anl* = T in "oanl"), the obs_tmp is replaced with the potential temp. The MODAS obs are written to a file as described in section 3.4.2.2.4.

6.3.12 Prepare the Observations for MVOI (MV_PREP)

The 'MASS' observations (e.g., *obs_age*, *obs_val*) are read by RD_MVOI_OBS from file "*wrkdir*/coda.MASS_obs.nest*nest.dtg*". This file was written by MASS_OBS (Figure 3.5-1). Next, the 'SYN' file is read by another call to RD_MVOI_OBS, which reads "*wrkdir*/coda.SYN_obs.nest*nest.dtg*"; this file was written by DRCT_OBS. RD_MVOI_OBS, which is called twice with different values of *opt* ('MASS' and 'SYN') but the arguments are the same (e.g., *obs_age*, *obs_err*, etc). This is because the first *n_data* entries are the observations and the entries from *n_strt* to the end of the array (*n_mass_obs*) are the synthetic obs.

The analysis ages are read from a restart file by RD_MVOI_ANL (which actually calls sub RD_OCN_ANL) with *opt* = ‘AGE’. For this option, RD_OCN_ANL is passed *fld_name* = ‘grdage’ and *file_typ* = ‘timefld’. The CR file is not necessary for a restart. The returned ages are passed to OBS_DETREND and processed for *fgat* < 0 or *restart* = T (see section 7.3), in which case a temporary variable is used to force this processing (*fno* = -1 in MV_PREP). This is a little different than the application of this algorithm to other observations because **obs_age** is passed to OBS_DETREND twice in the arg list. The analysis ages (**obs_age**) are passed as the first guess field (*guess*) and interpolated to their locations in real indices. The first-guess innovation (*obs_val – value*) should equal zero for *obs_age* because *obs_age* is already on the analysis grid (*obs_xi*, *obs_yj*) and the interpolated value will be the original. The returned value is *upd*, which is *upd_tau* read from the coda header file by sub RW_OCN_CNTRL. This is for initialization of time only; the other variables pass *upd* instead of the temporary *fno*.

Temperature is processed next by calling RD_MVOI_ANL for ‘TMP’ and ‘TERR’, which will open CR files with *fld_name* = ‘seatmp’, and *file_typ* = ‘analfld’ and ‘fcsterr’, respectively. The returned fields are placed in **wrk**(*i*,4) and **wrk**(*i*,2), respectively, and OBS_DETREND is called. The result is the analysis field for the appropriate update time interpolated to the observation locations, and with the difference between the observations (**obs_val**) and these data (**value**) being placed in the first-guess innovation (**obs_anm**) array. The **value** array is not kept.

Sub SET_ERR is called for *opt* = ‘TMP’ next; the algorithm is described in section 6.2. The array passed is **wrk**(*i*,2), which is the forecast error for temperature that was just read. Sub SET_ERR interpolates the prediction error (*prd_err*) to the obs points. If *err_mdl* (*emdl* in "oanl") = ‘simple’, the background error (*ebkg* in "oanl") is interpolated from the grid levels to the obs levels. However, if *emdl* = ‘complex’, the forecast error, **wrk**(*i*,2), is interpolated to the obs locations as *prd_err*.

The comments in sub SET_ERR state that the prediction error is interpolated from grid average values, referring to **wrk**(*i*,2) as just described. The authors are not certain why these are grid-averaged values, however, because they come from a restart file of *file_typ* = ‘fcsterr’. This file can only be written by one of the CODA programs (coda_prep; coda; or coda_post); Searching for ‘fcsterr’ indicated it showed up in several read subroutines: RD_ICE_ANL; RD_MVOI_ANL; RD_SSH_ANL; and RD_SWH_ANL. It is also used in DRCT_FLD; MASS_FLD; MODAS_GRID; and MODAS_OBS. Subroutines DRCT_FLD and MASS_FLD read the restart file, but MODAS_GRID writes a restart file with *file_typ* = ‘fcsterr’; this file contains forecast errors (rms of MODAS – GDEM fields) and calls WR_OCN_ANL to write them. MODAS_OBS calls RD_OCN_ANL, so this file can only contain the rms difference between MODAS synthetic fields and GDEM climatology. Searching for ‘fcsterr’ in the additional files used by coda_post, resulted in two occurences: RW_2D_ANL; and RW_3D_ANL. These subroutines are called by POST_2D and POST_3D, respectively, to write the prediction error growth to a restart file of *file_typ* = ‘fcsterr’. Consequently, we assume for now that a restart file with *file_typ* = ‘fcsterr’ actually contains errors that are interpolated to the analysis grid rather than grid-averaged values.

A cold start uses the default settings, which are assigned in SET_ERR. For a cold start, the observation error (*obs_err*) is found by interpolating to the obs levels, normalizing for maximum obs age (*mx_age*), corrected for number of obs (e.g., superobs) and normalized by the prediction error (*prd_err*). For restart runs, *obs_err* = 1. Finally, the innovation check is performed if requested (*inv_chk* = T) for a cold start. This procedure, calling OBS_DETREND and SET_ERR, is completed for salinity and geopotential; for geoptl, an analysis file (*file_typ* = ‘analfld’) is written (first guess) and a climate field (*file_typ* = ‘climfld’) is read, before writing the rms difference to a climate error (*file_typ* = ‘climerr’) restart file.

Sub RD_MVOI_OBS is called to read **obs_age**, etc from a file “*wrk_dir*/coda.VELOC_obs.nest*nest.dtg*”. The counter, *n_vel_obs* is reset to 0 immediately after. If the current run is not a forecast (i.e., *fcst* = F), the geopotential velocity is calculated in GEO_VEL and ‘analfld’ CR files are written for ‘uucurr’ and ‘vvcurr’. For a forecast, however, RD_MVOI_ANL is called to fetch ‘UUU’ and ‘VVV’ CR ‘analfld’ files. For all cases, sub RD_MVOI_ANL is called to read ‘UERR’ and ‘VERR’ fields.

The next block is perplexing. The standard sequence, OBS_DETREND and SET_ERR, will only execute if *n_vel_obs* > 0, but this was hard-wired to be zero exactly, and it is surrounded by ****, indicating that it is not supposed to be permanent. It was originally computed from *n_vel_ob*s = *n_data* – *n_mass_obs* before starting the velocity block. As it is, no observation innovations and errors are calculated.

After the skipped block, the rms climate error is found by subtracting the climate (*file_typ* = ‘climfld’) velocities (*fld_name* = ‘uucurr’ and ‘vvcurr’) from the MVOI analysis fields, which are zero for non-forecast runs; however, this restriction is not checked. These arrays are written to CR files of ‘climerr’ type. If innovation error checking (*linck* in "oanl") is requested for temp, salt, geoptl, or velocity, the anomalies (**obs_anm**) and values (**obs_val**) are written to unit 25, which is a formatted ascii file.

For a cold start with *dh_scl* (in "oanl") > 0, OCN_SFC_HT is called to read model surface elevation field (**grd_gpt**) or analysis dynamic height for use in flow dependent covariance analysis. This field is interpolated to the obs locations and renamed **obs_gpt**.

Subs SET_HCORR and SET_VCORR are called again to recalculate the correlation scales as described in section 3.3.1. The water depth, grid dimensions, and correlation scales are interpolated to the obs locations. If the analysis and model grids differ (*conflct* = T), sub GRD_CONFLCT is called to adjust everything to the analysis grid. If there is no conflict, the horizontal and vertical covariances (**grd_hcr** and **grd_vcr**) are written to a file named “*wrk_dir*/coda.MVOI_cvr.nest*nest.dtg*”. Finally, sub SET_VOLUME is called to create analysis volumes as described in section 3.3.1. The analysis volume descriptions are written to a file named “*wrk_dir*/coda.MVOI_vol.nest*nest.dtg*”.

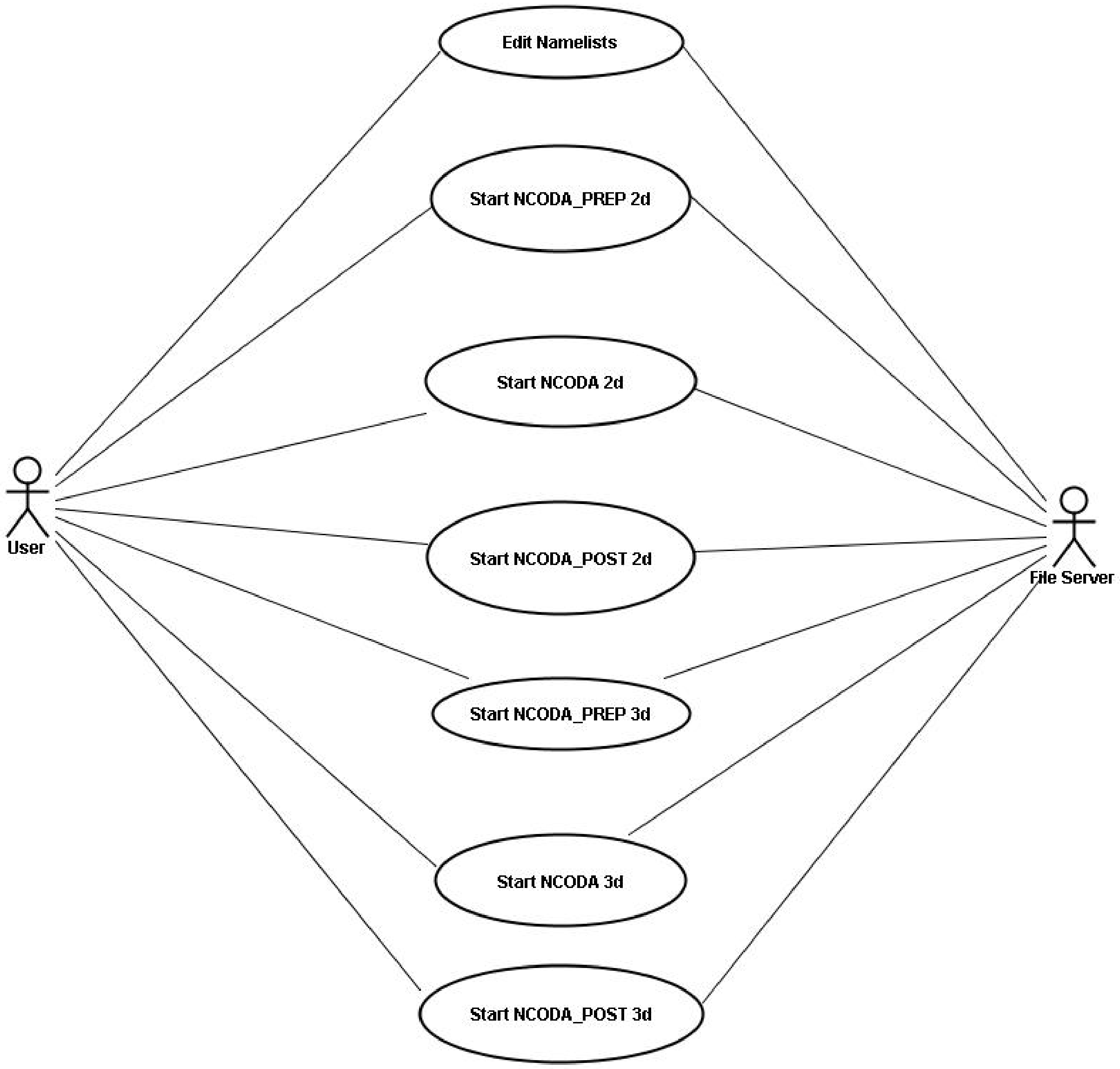

Figure 6.1-1. UML Use Case Diagram for completion of a 3D analysis cycle.

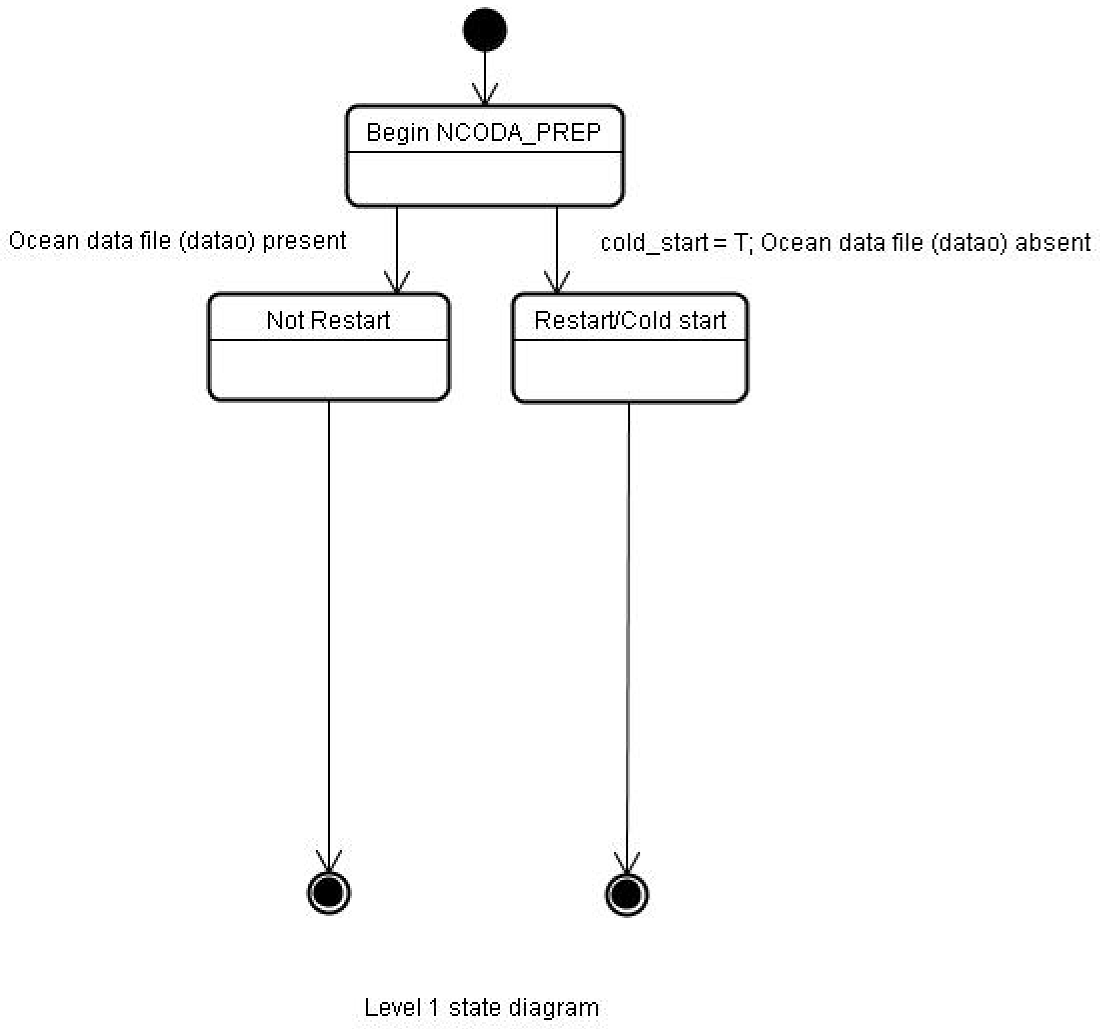
Begin NCODA_PREP
Ocean data file (datao) present
cold_start = T; Ocean data file (datao) absent
Not Restart
Restart/Cold start
Level 1 state diagram

Figure 6.1-2. UML State diagram of the ncoda prep process. The most fundamental states are either for restarted (cold start) or analyses with available forecast and analysis input. This algorithm is contained within subroutine COAMOA.

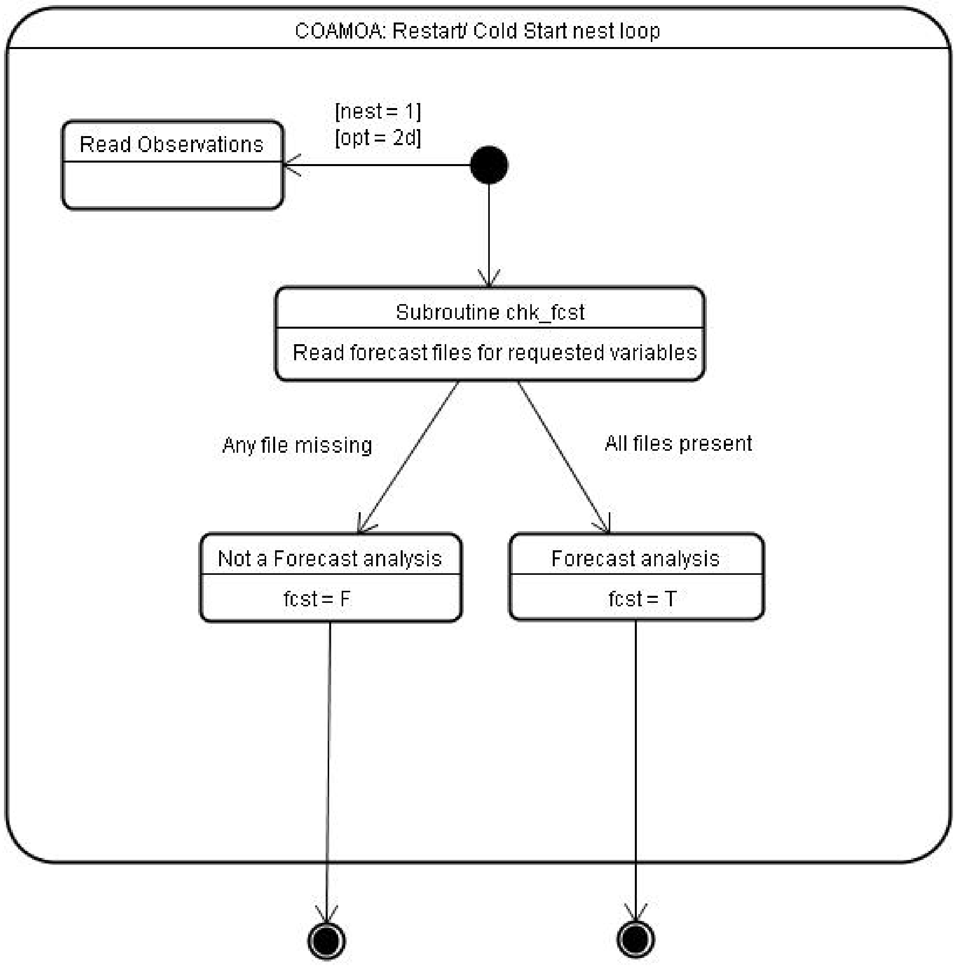

Figure 6.1-3. UML State diagram of the ncoda prep process for restarted analyses. This block represents a loop over all requested nests. This algorithm (shown on the right side of Figure 6.1-2) executes when there is no input data file for the requested time. The observations are read during processing of the first nest.

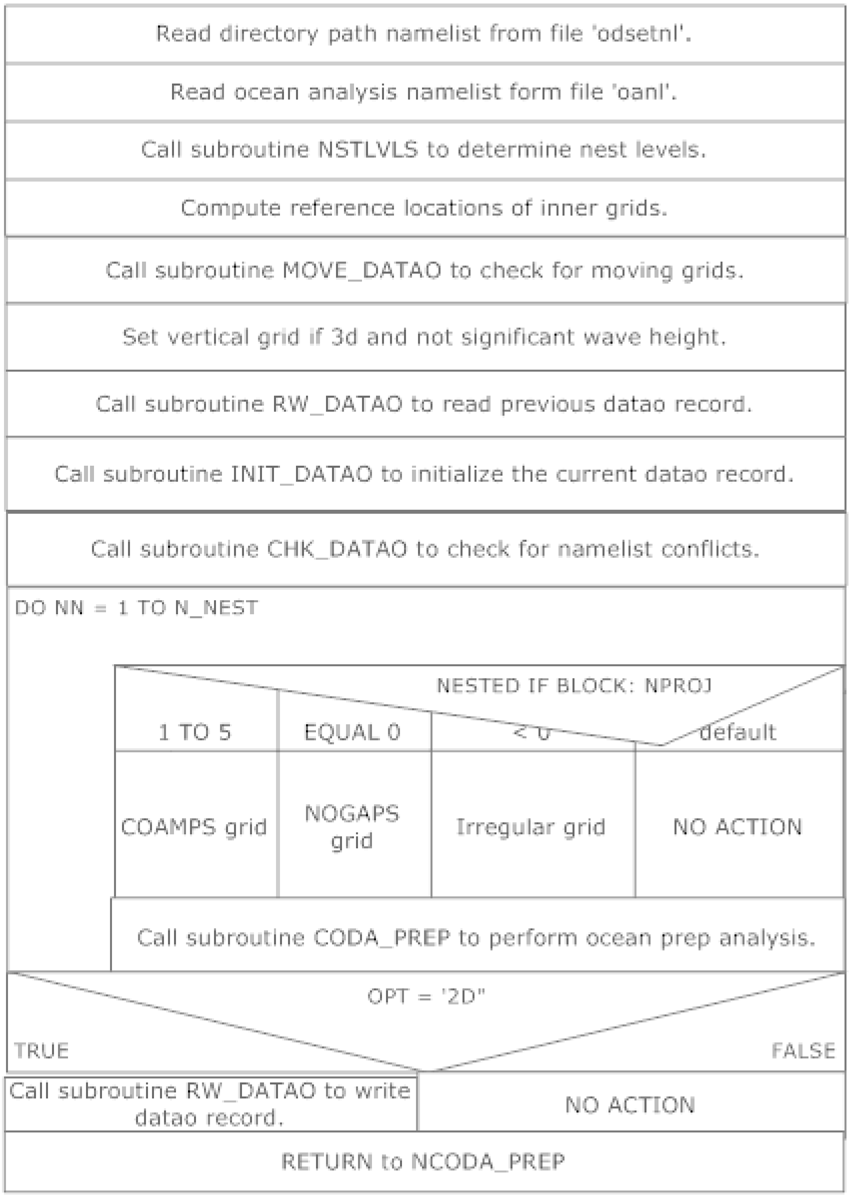
Read directory path namelist from file 'odsetnl'.
Read ocean analysis namelist form file 'oanl'.
Call subroutine NSTLVLS to determine nest levels.
Compute reference locations of inner grids.
Call subroutine MOVE_DATAO to check for moving grids.
Set vertical grid if 3d and not significant wave height.
Call subroutine RW_DATAO to read previous datao record.
Call subroutine INIT_DATAO to initialize the current datao record.
Call subroutine CHK_DATAO to check for namelist conflicts.
DO NN = 1 TO N_NEST
NESTED IF BLOCK: NPROJ
1 TO 5
EQUAL 0
< 0
default
COAMPS grid
NOGAPS grid
Irregular grid
NO ACTION
Call subroutine CODA_PREP to perform ocean prep analysis.
OPT = '2D"
TRUE
FALSE
Call subroutine RW_DATAO to write datao record.
NO ACTION
RETURN to NCODA_PREP

Figure 6.1-4. Nassi-Shneiderman diagram showing the flow of the COAMOA subroutine.

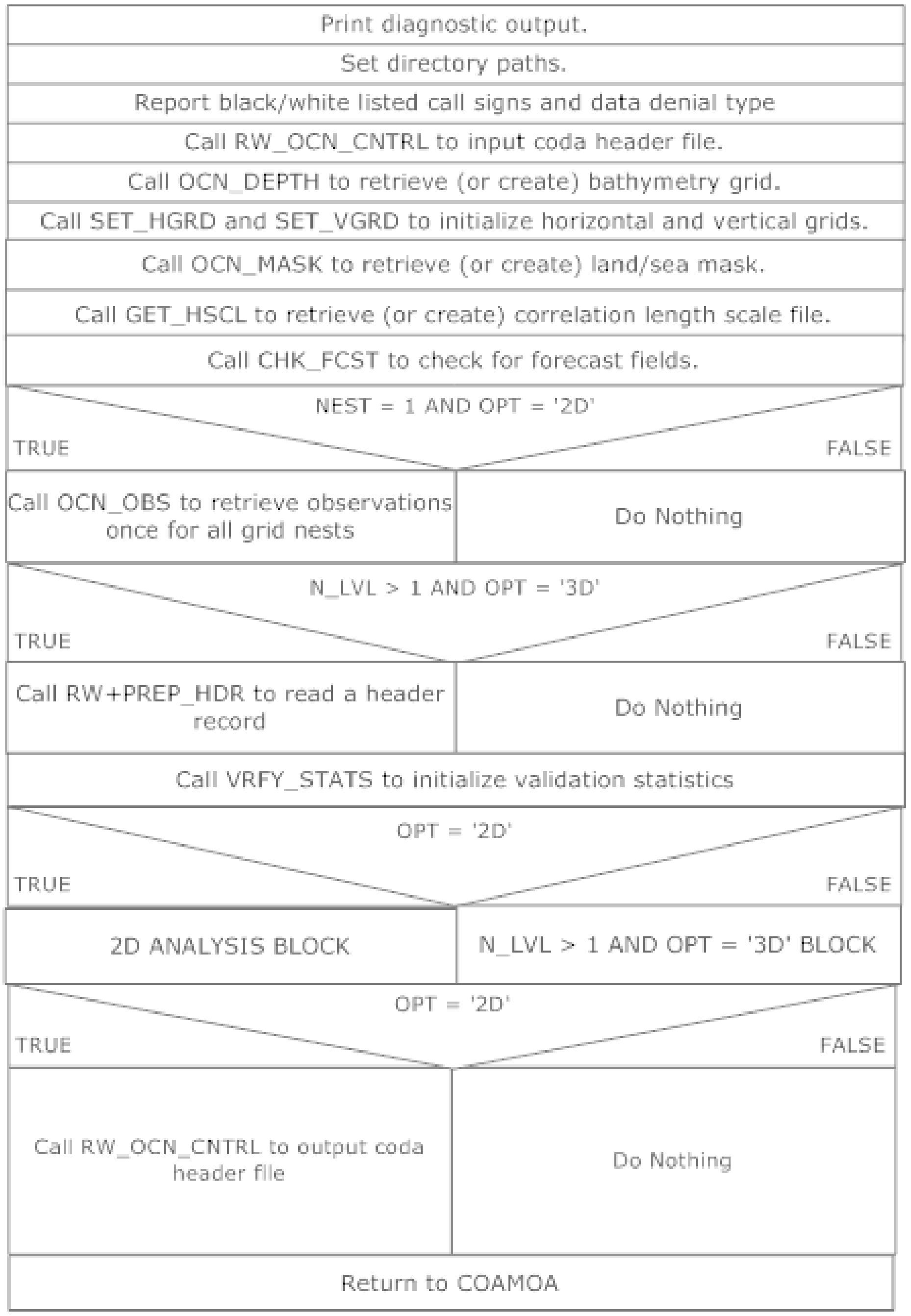
Print diagnostic output.
Set directory paths.
Report black/white listed call signs and data denial type
Call RW_OCN_CNTRL to input coda header file.
Call OCN_DEPTH to retrieve (or create) bathymetry grid.
Call SET_HGRD and SET_VGRD to initialize horizontal and vertical grids.
Call OCN_MASK to retrieve (or create) land/sea mask.
Call GET_HSCL to retrieve (or create) correlation length scale file.
Call CHK_FCST to check for forecast fields.
NEST = 1 AND OPT = '2D'
TRUE
FALSE
Call OCN_OBS to retrieve observations once for all grid nests
Do Nothing
N_LVL > 1 AND OPT = '3D'
TRUE
FALSE
Call RW+PREP_HDR to read a header record
Do Nothing
Call VRFY_STATS to initialize validation statistics
OPT = '2D'
TRUE
FALSE
2D ANALYSIS BLOCK
N_LVL > 1 AND OPT = '3D' BLOCK
OPT = '2D'
TRUE
FALSE
Call RW_OCN_CNTRL to output coda header file
Do Nothing
Return to COAMOA

Figure 6.1-5. Nassi-Shneiderman diagram showing the structure of the CODA_PREP subroutine expanded from Figure 6.1-4. The blocks for 2D analysis and (*n_lvl* > 1 AND 3D analysis) will be discussed in section 6.2 and 6.3, respectively.

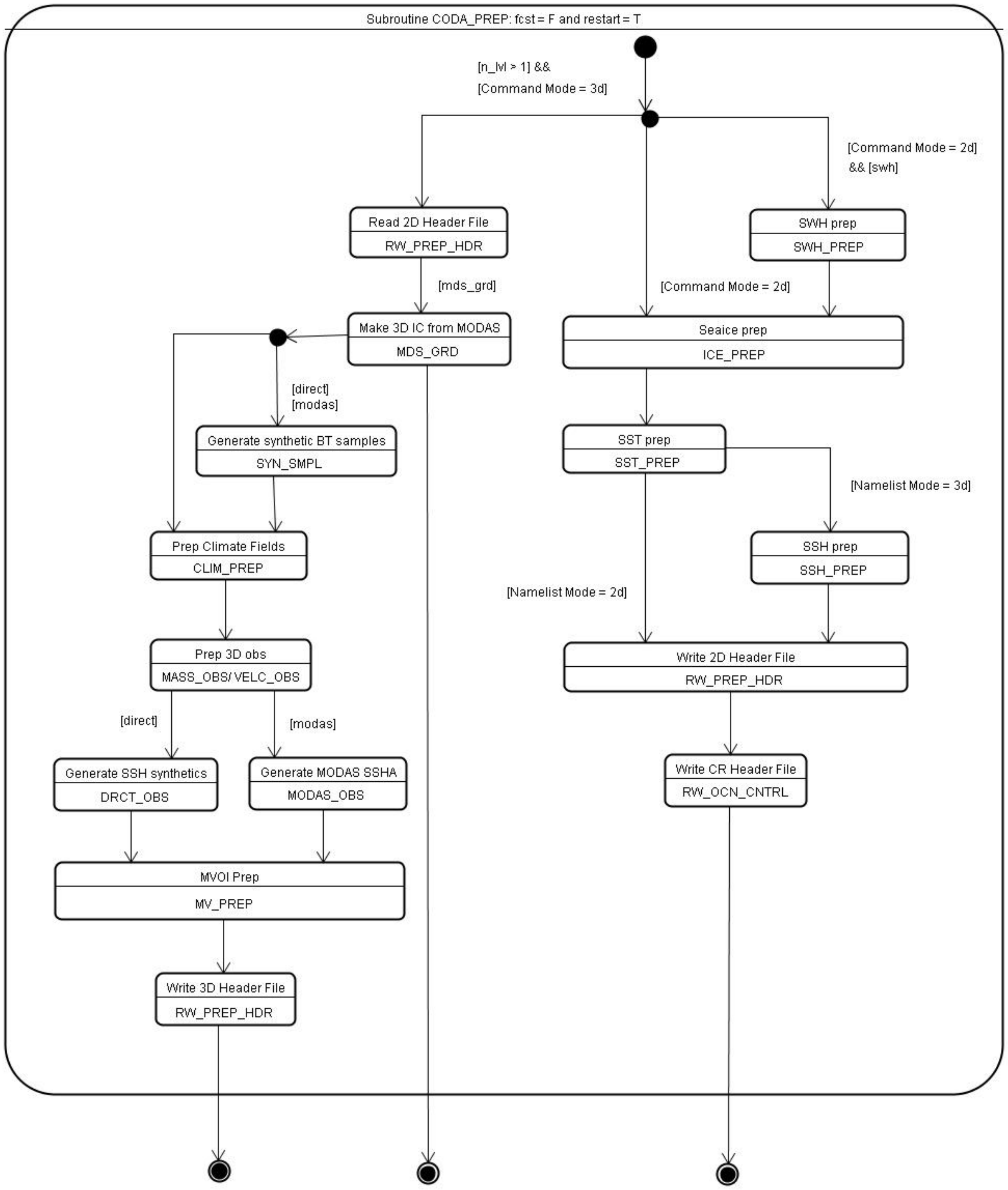

Figure 6.1-6. UML State diagram for the left state of the level 2 diagram (*fcst* = F and *restart* = T) (Figure 6.1-3). The terms in [ ] are guards; if these expressions evaluate to true, the next state is entered. If multiple guards are present, the next state is entered if any of them is true. The names below the lines in the state blocks are the subroutines in which the state occurs. In this program unit, no events are required to move between states.

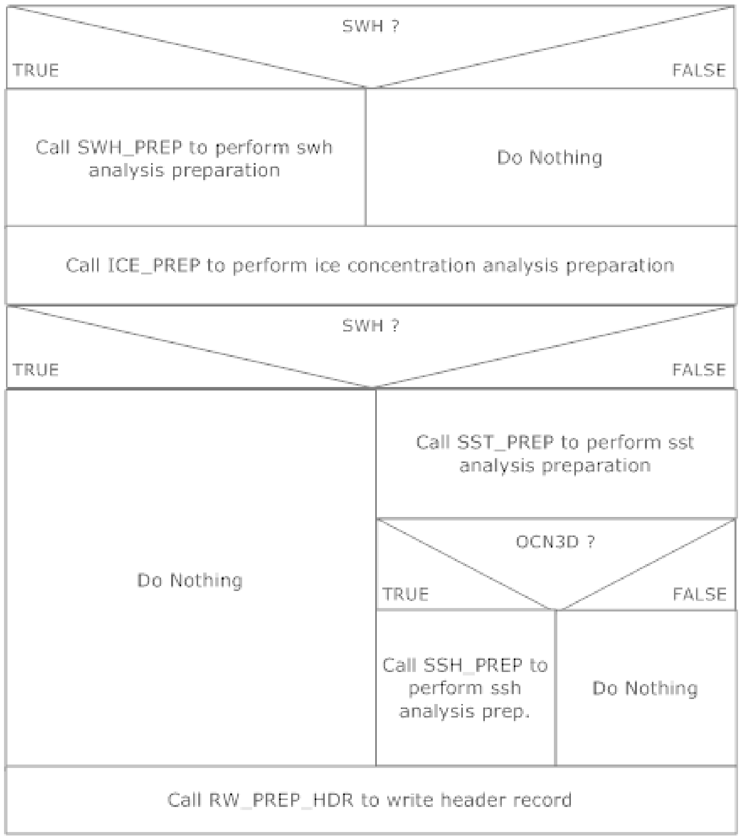
SWH ?
TRUE
FALSE
Call SWH_PREP to perform swh analysis preparation
Do Nothing
Call ICE_PREP to perform ice concentration analysis preparation
SWH ?
TRUE
FALSE
Do Nothing
Call SST_PREP to perform sst analysis preparation
OCN3D ?
TRUE
FALSE
Call SSH_PREP to perform ssh analysis prep.
Do Nothing
Call RW_PREP_HDR to write header record

Figure 6.2-1. Nassi-Shneiderman diagram of the 2D analysis block from Figure 6.1-5.

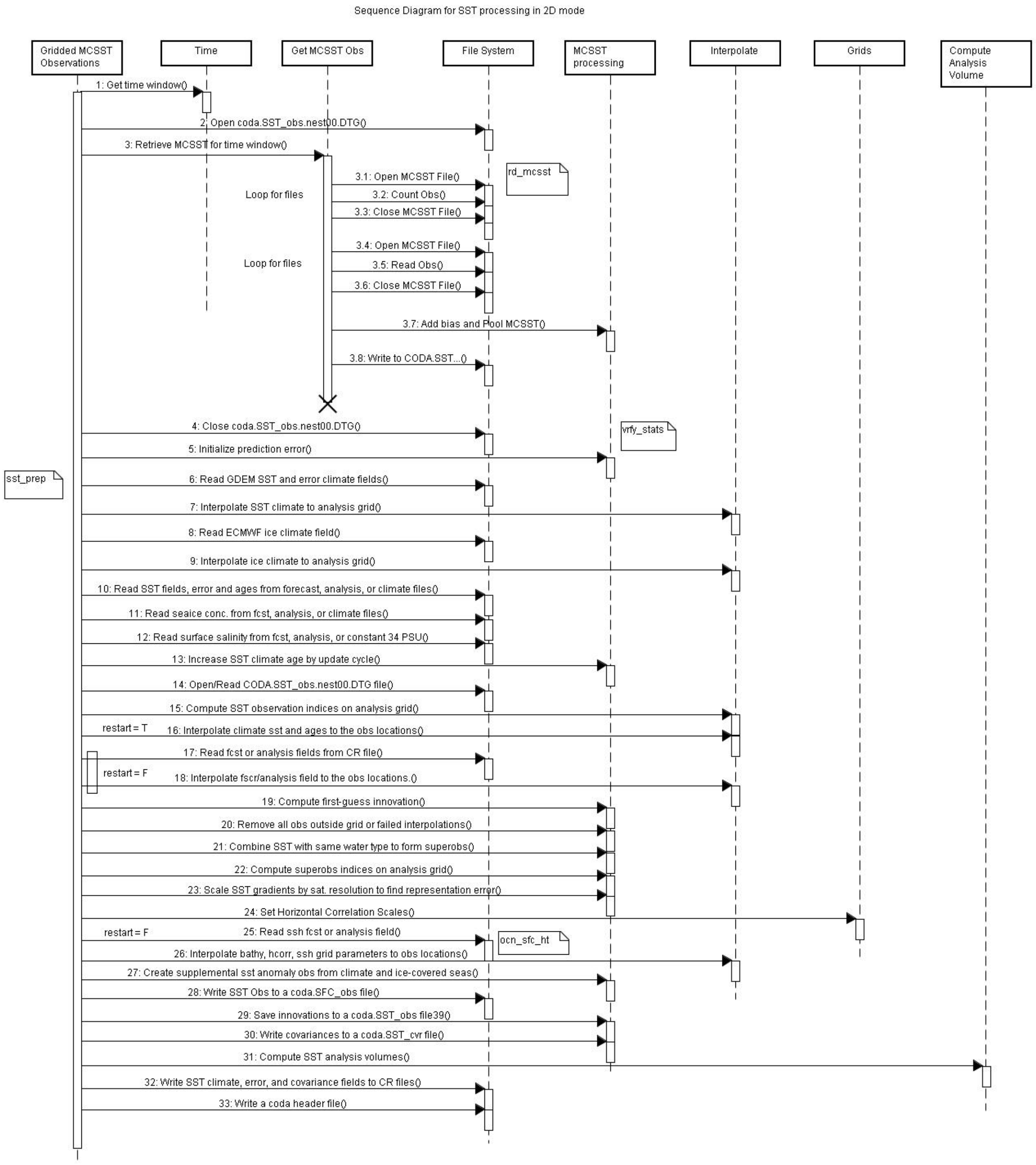
Sequence Diagram for SST processing in 2D mode
Gridded MCSST Observations
Time
Get MCSST Obs
File System
MCSST processing
Interpolate
Grids
Compute Analysis Volume
1: Get time window()
2: Open coda.SST_obs.nest00.DTG()
3: Retrieve MCSST for time window()
rd_mcsst
3.1: Open MCSST File()
Loop for files
3.2: Count Obs()
3.3: Close MCSST File()
3.4: Open MCSST File()
Loop for files
3.5: Read Obs()
3.6: Close MCSST File()
3.7: Add bias and Pool MCSST()
3.8: Write to CODA.SST...()
4: Close coda.SST_obs.nest00.DTG()
vrfy_stats
5: Initialize prediction error()
sst_prep
6: Read GDEM SST and error climate fields()
7: Interpolate SST climate to analysis grid()
8: Read ECMWF ice climate field()
9: Interpolate ice climate to analysis grid()
10: Read SST fields, error and ages from forecast, analysis, or climate files()
11: Read seaice conc. from fcst, analysis, or climate files()
12: Read surface salinity from fcst, analysis, or constant 34 PSU()
13: Increase SST climate age by update cycle()
14: Open/Read CODA.SST_obs.nest00.DTG file()
15: Compute SST observation indices on analysis grid()
restart = T
16: Interpolate climate sst and ages to the obs locations()
17: Read fcst or analysis fields from CR file()
restart = F
18: Interpolate fscr/analysis field to the obs locations.()
19: Compute first-guess innovation()
20: Remove all obs outside grid or failed interpolations()
21: Combine SST with same water type to form superobs()
22: Compute superobs indices on analysis grid()
23: Scale SST gradients by sat. resolution to find representation error()
24: Set Horizontal Correlation Scales()
restart = F
25: Read ssh fcst or analysis field()
ocn_sfc_ht
26: Interpolate bathy, hcorr, ssh grid parameters to obs locations()
27: Create supplemental sst anomaly obs from climate and ice-covered seas()
28: Write SST Obs to a coda.SFC_obs file()
29: Save innovations to a coda.SST_obs file39()
30: Write covariances to a coda.SST_cvr file()
31: Compute SST analysis volumes()
32: Write SST climate, error, and covariance fields to CR files()
33: Write a coda header file()

Figure 6.2-2. UML Sequence Diagram of NCODA_PREP for SST analysis in 2D command line mode. The objects are loosely based on the class diagram (Figure 2.2-1). The arrows indicate messages being passed to other objects from the main task (leftmost object). Time progresses from the top of the diagram. The note boxes show locations where specific subroutines operate.

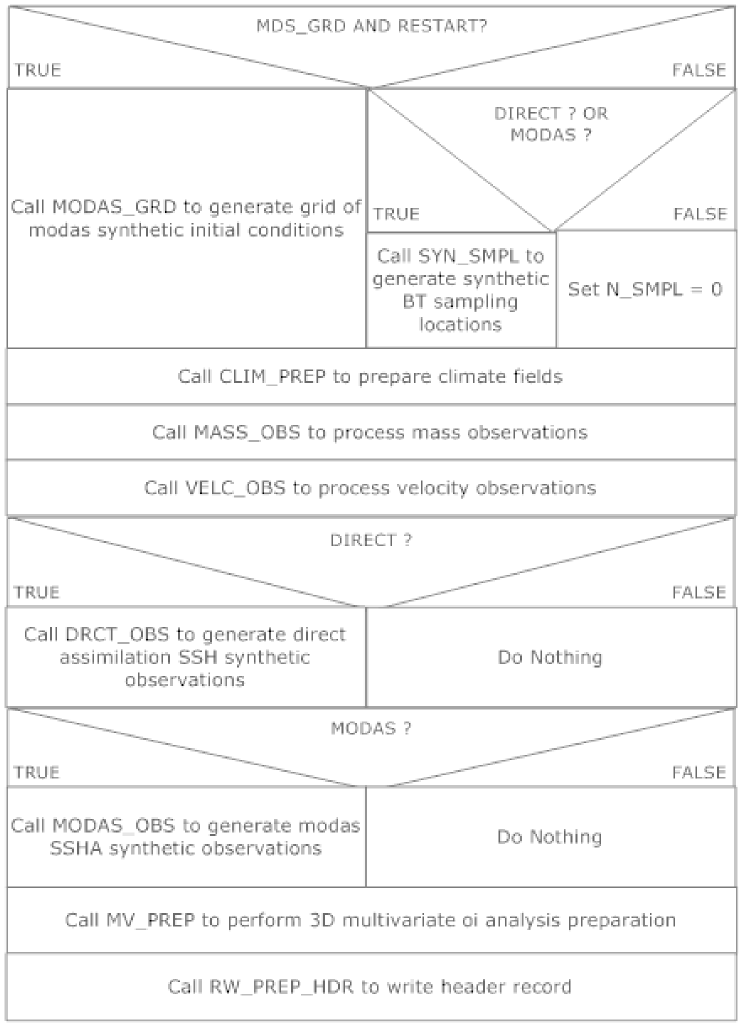
MDS_GRD AND RESTART?
TRUE
FALSE
Call MODAS_GRD to generate grid of modas synthetic initial conditions
DIRECT ? OR MODAS ?
TRUE
FALSE
Call SYN_SMPL to generate synthetic BT sampling locations
Set N_SMPL = 0
Call CLIM_PREP to prepare climate fields
Call MASS_OBS to process mass observations
Call VELC_OBS to process velocity observations
DIRECT ?
TRUE
FALSE
Call DRCT_OBS to generate direct assimilation SSH synthetic observations
Do Nothing
MODAS ?
TRUE
FALSE
Call MODAS_OBS to generate modas SSHA synthetic observations
Do Nothing
Call MV_PREP to perform 3D multivariate oi analysis preparation
Call RW_PREP_HDR to write header record

Figure 6.3-1. Nassi-Shneiderman diagram of the 3D analysis block from Figure 6.1-5.

7 Notes.

7.1 Update Cycle

The update cycle is critical in the detrending of the observations; therefore, it is useful to examine its assignment in some detail in this report. The variable *upd_cyc* is entered by the user in namelist *oanl*, which is included in the header file "oanl.h." A typical value is 24 hours. This header file is read by COAMOA. *Upd_cyc* is then passed to sub RW_DATAO as *upd*. If it is not 0, it is used as the increment to search for existing CR files for *field* = 'datahd' up to 14 days in the past. If it is zero, an increment of 6 hours is used. The value is not changed by RW_DATAO. It is passed to sub INIT_DATAO and written to array **datao**(20). INIT_DATAO also passes *upd_cyc* to IRREG_GRID as *upd* (*nproj* < 0), where it is used to search for a time-stamped latitude file. The value is not changed. It is then passed directly to IRREG_GRID from COAMOA in a redundant search for a latitude file. Variable *upd_cyc* is then passed to sub CODA_PREP as *upd*. A second update variable, *upd_tau*, is initialized in CODA_PREP as a local variable passed to sub RW_OCN_CNTRL along with *upd*, where its name reverts to *upd_cyc*. With *opt* = 'get', the value of *upd_cyc* is held in *upd_tau* and *file_dtg* is found by subtracting *upd_cyc* from *dtg* for a restarted run. RW_OCN_CNTRL then returns to CODA_PREP.

For a cold start, *upd_cyc* is used as a time increment to search for a CODA restart file for *field* = 'codaoi'. If this file exists, *upd_tau* is assigned the hour for which it exists but *upd_cyc* is unchanged. If the file is not found, *upd_tau* is assigned *upd_cyc* and *file_dtg* is found by subtracting *upd_tau* from *dtg*. This file dtg is then inserted into the **_dtg* variables for all data sources (e.g., amsr, ship, xbt) and written to a CR file for *field* = 'codaoi', after which control returns to CODA_PREP. Although *upd_tau* now has a useful value that could differ from *upd* (*upd_cyc*), *upd* is passed to sub OCN_DEPTH to search for a CODA restart file for field = 'depths'; its value is unchanged.

It is interesting that *upd_cyc* has the value of an available CR file for *field* = 'codaoi' but this is not used to look for a CR file for *field* = 'depth'. This is probably because *upd_tau* is the actual update time whereas *upd_cyc* is the increment for updating.

The same procedure is used for the ocean mask (sub OCN_MASK; *field* = 'maskls'), correlation length scales (sub GET_HSCL; *field* = 'grdscl'), and forecast fields (sub CHK_FCST; *field* = 'seatmp', 'salint', 'uucurr', 'vvcurr', 'seatmp', 'seahgt', 'seaice', and 'sigwht'). *Upd* (*upd_cyc*) is then passed to OCN_OBS as *upd_cyc*, where a local variable named *tmp_upd* is introduced. The behavior of *upd_cyc* in OCN_OBS is primarily determined by the value of *restart*. If *restart* = T, *upd_cyc* is assigned to *tmp_upd* and used to construct old *file_dtg* from *dtg*, which is then assigned to variables, *old_** for the types of observations (e.g., ship). *Tmp_upd* either uses one-half of *upd_cyc* (*fcst* = T) or no less than 24 hours otherwise. If *restart* = F, a *new_dtg* is found by subtracting one-half of *upd_cyc* (temporarily held in *tmp_upd*) from *dtg*. The value of *upd_cyc* is unchanged. Sub OCN_OBS passes *upd_cyc* to CR_TMP_FILE, where it is used to read observations (e.g., atsr) from files. The *dtg* variables derived in OCN_OBS (above) are changed only if *sst_time* = 'synt'. *Upd_cyc* is then passed to a series of subroutines (e.g., RD_MCSST) where it is used to set *obs_typ* for data sources. This last step is not required for ship, profile, or glider data. The same procedure is applied in subs CR_ICE_FIL, CR_SSH_FILE, and CR_SWH_FILE**.** The value of *upd_cyc* is unchanged.

Subroutine CODA_PREP passes *upd* (*upd_cyc*) to sub VRFY_STATS as *upd* where it is used to search for a CR file with *field* = 'cvstat'. *Upd* (*upd_cyc*) is then passed to SWH_PREP as *upd.* SWH_PREP then passes it to RD_SWH_ANL as *upd_tau,* where it is used to search for a CR file with *field* = 'seaice' and *type* = 'analfld'. It is passed to sub RD_OCN_ANL as *upd_tau,* which does the actual searching. Sub SWH_PREP then updates the age of *swh_age* by adding *upd.* The next time *upd* is passed from sub SWH_PREP is to sub OBS_DETREND, which is where the description continues below.

7.2 The first-guess appropriate time (FGAT)

Fgat < 0 indicates that there are no analyses to read, as on the first analysis of a region. If *fgat* > 0, there is a procedure that includes reading analysis files and interpolating them to the observation real-valued indices. The minimum age of the observations is found; it must be greater than one-half *upd_cyc*. *Upd_str* = -*n_hrs* and *upd_end* = *upd_cyc*/2. These are used as the loop limits to find the first-guess time field, by adding (subtracting) these hours from *dtg*. The increment variable, *fgat*, is the first-guess appropriate time update interval. *Fgat* is set in namelist "oanl". It has values for ice, sst, ssh, multivariate, and swh fields. Typical values are ~1 hr.

The first step is to find an appropriate first guess time interval that has analyses available. Otherwise there would be no point. This is done by finding the minimum age (hours) among all of the observations. Ages can be either positive or negative because the analysis does not have to be in real time; thus obs can be newer than the desired analysis. This condition must be covered. *Age_mn* is assigned a value of -*upd_cyc*/2, which initializes the minimum age of the obs. If an observed age is old, its age is negative. This bias is removed by taking its abs value. If this point age is less than *age_mn*, it becomes the new field minimum. Limited testing with this complicated algorithm indicated that it appears to find at least one time.

Loop over the entire first guess field time intervals (between *upd_str* and *upd_end*). Compile a list of observations for each time window [time(*dtg*) +/- *fgat*/2]. The index of available obs is saved to **ndx**(*nobs*) as well as its real-valued indexes on the analysis grid. The *dtg* is determined and the forecast time, *tau*, is found.

7.3 Error calculation condition

The prediction error estimation in SET_ERR is computed in a block that executes only when *restart* = F. Note that SET_ERR is called within the *nest* loop in COAMOA (see Figure 6.1-4). The default settings are used for *prd_err* for *restart* = T. The comments before the else statement indicate that the defaults are to be used for a 'cold start', which makes sense. However, the code from SET_ERR,

```
if  (.not. restart) then
        set prediction error from observed average values...
```

```
                    or...
                    interpolate prediction error from grid-averaged values...
               else
                    use default settings for cold start...
               endif
```

will apply the default settings for *restart* = T. *Cold_start* is a scalar logical variable that is input from namelist "oanl." It is passed by COAMOA to CHK_DATAO, where it causes all entries in the array **restart**(*mx_grds*) to be F if *cold_start* = F, whether a "datao" file exists or not. Thus, a cold start run is equivalent to a restart, except that the second applies to individual grids as necessary; i.e., individual grids may be added to an analysis and they will be set to *restart* = T.

The prediction error is scaled by offset and a tuning factor (*fct_bkg*, *ebkg* in "oanl"), which is often set to 1. Another restart/cold start block follows, again with a default value applied to the obs error on the current grid (see section 6.2) for *restart* = T. If *restart* = F, however, the obs error is found from the relative depth of the obs, its relative age (normalized to *mx_age*, *clm_scl* in "oanl"; a typical value for swh = 72 hrs), and *obs_rep* (a dummy variable = 0 in sub SWH_PREP), number of obs for a superob, and *err_obs* (*eobs* in "oanl"; swh = default typically). Finally, the *obs_err* is normalized by the *prd_err* found above.

The innovation error is checked if *inv_chk* (*linck* in "oanl"; default = T). This is done by normalizing to the *prd_err*, and again by the *obs_err*. If this error exceeds *tol* (*clm_scl* in "oanl"; typically 72 hrs for swh), this obs is rejected. Note that this check is only completed for *restart* = F, for which no detrending is performed on the obs. Control then returns to SWH_PREP, which checks to see if any obs were rejected by the innovation error check.

7.4 Logical Flags for Initialization

The synthetic initial condition discussed in section 3.4.1 depends on *mds_grd* = T and *restart* = T. Code checks found the following occurrences of *mds_grd* in the source files:

```
$ grep -in mds_grd *.f | more
coamoa.f:352:     *  mds_edit, mds_grd, mds_mld, mds_xtnd,
                     clm_scl, sal_adj,
coda.f:178:          if (mds_grd .and. restart) then
coda_prep.f:13:   *  ssh_hrs, ssh_time, mds_edit, mds_grd,
                     mds_mld,
coda_prep.f:109:  c  mds_grd  logical  input  (true) MODAS
                     initial conditions
coda_prep.f:346:     logical   mds_grd
coda_prep.f:912:     if (mds_grd .and. restart) then
```

Thus, the MODAS and GDEM initial condition will only occur on a restart (either *cold_start* or new grid) when the user specifies it. The alternative case (no synthetic initial condition) covers all cold starts and restarts unless over-ruled by user input in namelist "oanl."

A second set of flags controls using synthetic BT's. Sub SYN_SMPL will only be called if *direct* = T or *modas* = T. Supposedly, *direct* indicates a desire for direct MODAS assimilation and *modas* indicates a wish to perform a MODAS assimilation of altimeter ssh. Here is where "direct " shows up in logical statements:

```
$ grep -n direct *.f| grep "if ("
(1) chk_fcst.f:104:  if (direct .and. ssh_opt .eq. 'modl') then
(2) chk_fcst.f:111: if (direct) then
(3) coda_prep.f:568:      if (direct .and. bv_chk .gt. 0.) then
(4) coda_prep.f:938:            if (direct .or. modas) then
(5) coda_prep.f:996:      if (direct .and. n_smpl .gt. 0) then
(6) rd_ssh_anl.f:104:      if (direct) then
(7) ssh_prep.f:356:      if (direct) then
(8) ssh_prep.f:429:         if (direct) then
(9) ssh_prep.f:749:   if (direct .and. ssh_opt .eq. 'modl'
                         .and.(topo_src .eq. 'modas' .or.
                         topo_src .eq. 'model')) then
```

Occurrences (1) and (2) are associated with checking that forecast variables to be retrieved from CR files are availale. (1) is used if one analysis level is chosen (*n_lvl* = 1); it makes sure that *field* = 'seahgt' is available if *ssh_opt* (*ssh_mean* in "oanl") is set to 'modl' rather than 'clim'. (2) makes certain that 'seahgt' fields are available for *n_lvl* > 1. Occurrence (3) in CODA_PREP prints the Brunt-Vaisala frequency threshold (*bv_chk* in "oanl") because it affects the direct assimilation of MODAS synthetics and (4) calls SYN_SMPL to generate the synthetic BT sampling locations if either *direct* or *modas* = T, but only if either *mds_grd* or *restart* = F. This makes sense because, if *mds_grd* = T a full 3D grid of synthetics are available and if *restart* = T, it is an initial analysis for the current grid.

Occurrence (5) calls DRCT_OBS to compute T and S profiles at the selected *n_smpl* locations using model and altimeter ssh fields. These will be used as supplemental obs in the MVOI analysis.

Occurrence (6) in RD_SSH_ANL causes a ssh analysis field to be read if *ssh_opt* (*opt*) = 'modl'; otherwise, either forecast model or analysis background T and S fields are read and the surface geopotential field is saved. (7) in SSH_PREP retrieves or creates the mean ssh field and (8) scales the altimeter by the in-situ ssh variability using sub RD_SSH_STD. (9) writes the ssh background field to a CR file for all cases involving ssh.

8 References

Cummings, J. A. Operational multivariate ocean data assimilation. *Q. J. R. Meteorol. Soc.*, **131**, 3583-3604, 2005.

Goerss, J. S. and Phoebus, P. A. The Navy's operational atmospheric analysis. *Weather and Forecasting,* **7**, 232-249, 1992.

Horn, R. A. and Johnson, C. R. *Matrix Analysis,* Section 7.2. Cambridge University Press, 1985.

Lorenc, A. C. A global three-dimensional multivariate statistical interpolation scheme. *Mon. Weather Rev.*, **109**, 701-721, 1981.

Nassi, I. and Schneiderman, B. Flowchart techniques for structured programming. Tech. Contributions, SIGPLAN Notices No. 12, August 1973.

Page-Jones, M. *Fundamentals of Object-Oriented Design in UML.* Addison-Wesley, New York, 458 pp., 2000.

APPENDIX 1. Abbreviated List of Subroutine Calls for NCODA Prep.

The numbers refer to the line number in calling routine file.

```
program ncoda_prep
     105: getarg (3)
     131: dtgops
          268: getenv
          421: gmtime (2)
     143: getarg
     218: coamoa
          166: nstlvls
          197: move_datao
          275: rw_datao
          277: init_datao
               158: ij2ll
               176: irreg_grid
          283: chk_datao
          306: coamps_grid
          315: ngps_grid
          321: irreg_grid
          340: coda_prep
               715: rw_ocn_cntrl
               742: ocn_depth
               748: set_hgrd
               750: set_vgrd
               764: ocn_mask
               770: get_hscl
                    153: rd_rossby
               776: chk_fcst
               784: ocn_obs
                    724: cr_tmp_file
                         277: rd_mcsst
                         300: rd_metop
                         323: rd_goes
                         346: rd_lac
                         370: rd_metop_lac
                         394: rd_msg
                         417: rd_amsr
                         439: rd_atsr
                         462: rd_ship
                         504: prof_collect
                              286: rd_prof
                                   573: prof_err
                              302: prof_argo
                              313: prof_thin
                              323: prof_pool
                              331: prof_slct
                              340: prof_dupchk
                         553: gldr_collect
                              258: rd_gldr
                              276: prof_thin
                              286: gldr_slct
                              296: gldr_dupchk
                    760: cr_ice_file
                         109: rd_ssmi
                    767: cr_ssh_file
                         144: rd_ssh
                    779: cr_swh_file
                         121: rd_swh
               804: rw_prep_hdr
               818: vrfy_stats
               830: swh_prep
                    300: rd_swh_clim
                    303: rd_swh_anl
                         183: rd_ice_clim
                    317: rd_data_file
                    326: swh_bias
                    330: obs_index
                         89: ll2ij
                         104: srch_ij
               |         110: irreg_ll2ij
                         117: srch_kz
                    338: obs_detrend
                    343: obs_detrend
                    351: obs_remove
                    360: super_ob
                    369: obs_index
                         89: ll2ij
                         104: srch_ij
                         110: irreg_ll2ij
                         117: srch_kz
                    388: set_err
                    423: set_hcorr
                    460: cr_ovrlp_obs
                    473: rw_prep
                    479: rw_covr
                    484: set_volume
                         139: volume_init
                         142: cr_volumes
                         146: cr_ovrlp_vol
                              317: sort_vctr
               847: ice_prep
                    326: rd_ice_clim
                    328: rd_sst_clim
                    333: rd_ice_anl
                    346: coda_bndy
                    355: ngps_bndy
                    386: rd_data_file
                    395: obs_index
                         89: ll2ij
                         104: srch_ij
                         110: irreg_ll2ij
                         117: srch_kz
                    415: obs_detrend
                    420: obs_detrend
                    428: obs_remove
                    437: super_ob
                    446: obs_index
                         89: ll2ij
                         104: srch_ij
               |         110: irreg_ll2ij
                         117: srch_kz
                    464: grdnt_err
                    471: set_err
                    505: set_hcorr
                    510: fld2_trp (4)
                    554: cr_ovrlp_obs
                    567: rw_prep
                    576: rw_covr
                    582: set_volume
                         139: volume_init
                         142: cr_volumes
                         146: cr_ovrlp_vol
                              317: sort_vctr
               866: sst_prep
                    367: rd_sst_clim (2)
                    371: rd_ice_anl
                    376: rd_sst_anl
                    386: coda_bndy
                    395: ngps_bndy
                    410: wr_ocn_anl
                    426: rd_data_file
                    434: obs_index
                         89: ll2ij
                         104: srch_ij
                         110: irreg_ll2ij
                         117: srch_kz
                    454: obs_detrend
                    459: obs_detrend
                    467: obs_remove
                    476: super_ob
                    485: obs_index
                         89: ll2ij
                         104: srch_ij
                         110: irreg_ll2ij
                         117: srch_kz
                    503: grdnt_err
                    510: set_err
                    544: set_hcorr
                    573: cr_suppl_sst
                    639: cr_ovrlp_obs
                    651: rw_prep
                    661: rw_covr
                    667: set_volume
                         139: volume_init
                         142: cr_volumes
                         146: cr_ovrlp_vol
                              317: sort_vctr
                    682: wr_ocn_anl (3)
               886: ssh_prep
```

```
          360: get_ssh_mean
               164: cr_mean_ssh
                    157: gdem_grd (gdem_mod.f)
                    197: gdem_grd (gdem_mod.f)
                    238: coda_xtnd (2)
                    243: geo_ptl
               173: coda_xtnd
          368: rd_ssh_err
          373: rd_ssh_anl
          388: rd_data_file
          411: obs_index
               89: ll2ij
               104: srch_ij
               110: irreg_ll2ij
               117: srch_kz
          422: rd_ssh_std
          458: modas_topog
          485: modas_topog
          524: obs_detrend
          529: obs_detrend
          538: obs_remove
          547: super_ob
          556: obs_index
               89: ll2ij
               104: srch_ij
               110: irreg_ll2ij
               117: srch_kz
          574: grdnt_err
          581: set_err
          616: ocn_sfc_ht
          628: set_hcorr
          645: cr_suppl_ssh
          677: cr_ovrlp_obs
          690: rw_prep
          700: rw_covr
          703: set_volume
               139: volume_init
               142: cr_volumes
               146: cr_ovrlp_vol
                    317: sort_vctr
     906: rw_prep_hdr
```

```
916: modas_grid
     300: modas_topog
     422: modas_opn
     442: modas_syn
          266: modas_data
          322: modas_poly
          332: modas_coef
          341: modas_temptrp (3)
               202: modas_edit
               211: modas_mld
          362: modas_salt
          378: modas_clim
          387: modas_clim
     560: grd_clim
     562: grd_clim
     725: grd_clim (2)
     747: geo_ptl (2)
     853: geo_vel (2)
     1131: vrfy_stats
     1140: rw_prep_hdr
928: syn_smpl
943: clim_prep
     147: ocn_clim
          171: grd_clim
     150: ocn_clim
          171: grd_clim
     188: geo_ptl
     209: geo_vel
     244: ocn_clim
          171: grd_clim
     247: ocn_clim
          171: grd_clim
968: mass_obs
     241: mass_fld
     250: coda_mld
     258: sst_anl
     268: sst_obs
          173: rd_data_file
          188: ll2ij
          192: irreg_ll2ij
     284: sss_obs
```

```
          295: prof_obs
               61:  rd_data_file
               290: ll2ij
               294: irreg_ll2ij
               368: prof_xtnd
                    225: bathy_merge
                    228: bathy_merge
          329: wr_mass_obs
     991: drct_obs
          327: drct_fld
               271: coda_mld
          336: modas_topog
          410: ts_static
               127: salt_corr
          458: ssh_drct
               131: spline (2)
               156: ts_adj
          497: drct_adj
               217: ts_static
                    127: salt_corr
          643: wr_mass_obs
     1004: modas_obs
          478: modas_topog
          526: modas_opn
          546: modas_syn
               266: modas_data
                    167: rd_modas_data (2)
               322: modas_poly
               327: modas_trifit
               332: modas_coef
               341: modas_temp
                    174: modas_type (3)
                    202: modas_edit
                    211: modas_mld
               362: modas_salt
               378: modas_clim
               387: modas_clim
          604: wr_mass_obs
     1024: mv_prep
          333: rd_mvoi_obs (2)
          356: rd_mvoi_anl
```

```
364: obs_detrend
376: rd_mvoi_anl
379: rd_mvoi_anl
394: obs_detrend
400: set_err
415: rd_mvoi_anl
418: rd_mvoi_anl
433: obs_detrend
439: set_err
457: geo_ptl
459: rd_mvoi_anl
479: obs_geoptl
     190: bilnr
486: obs_detrend
492: set_err
550: rd_mvoi_anl
571: obs_lyrp
     330: ts_static
          127: salt_corr
     375: lyrp_err
578: set_err
613: obs_remove
629: rd_mvoi_obs
646: geo_vel
658: rd_mvoi_anl
661: rd_mvoi_anl
666: rd_mvoi_anl
681: rd_mvoi_anl
708: obs_detrend
714: obs_detrend
720: set_err (2)
744: obs_remove
836: ocn_sfc_ht
848: set_hcorr
850: set_vcorr
891: cr_ovrlp_obs
902: rw_prep
915: grd_conflct
     221: coamps_grid
     228: ocn_depth
     234: set_vgrd
```

```
                    239: get_hscl
                         153: rd_rossby
                    247: ocn_sfc_ht
                    251: ssh_conflct
                         3:   ll2ij
                    270: ocn_mask
                    276: set_hcorr
                    286: rw_covr
                    291: ll2ij
                    322: cr_ovrlp_obs
                    333: rw_prep
                    343: set_volume
                         139: volume_init
                         142: cr_volumes
                         146: cr_ovrlp_vol
               942: rw_covr
               944: set_volume
                    139: volume_init
                    142: cr_volumes
                    146: cr_ovrlp_vol
                         317: sort_vctr
          1046: rw_prep_hdr
          1072: rw_ocn_cntrl
     371: rw_datao
```

APPENDIX 2: Argument List Mapping for COAMOA calling CODA_PREP

```
VAR.  SUBROUTINE CODA_PREP   CALL CODA_PREP
--------------------------------------------------------------------------

1.    opt,                   opt,
2.    dtg,                   dtg,
3.    upd, -----------------> upd_cyc,
4.    out_dir, -------------> dsorff,
5.    datu_dir, ------------> dsoudat,
6.    datr_dir, ------------> dsordat,
7.    datc_dir, ------------> dsocdat,
8.    dats_dir, ------------> dsosdat,
9.    clim_dir, ------------> dsoclim,
10.   gdem_dir, ------------> dsogdem,
11.   modas_dir, -----------> dsomdas,
12.   ngps_dir, ------------> dsngff,
13.   wrk_dir, -------------> dsowrk,
14.   nest, ----------------> nn,
15.   n_lon, ---------------> mo(nn),
16.   n_lat, ---------------> no(nn),
17.   n_pgrd, --------------> np_grid(nn),
18.   n_lvl, ---------------> nl,
19.   z_lvl, ---------------> lvl,
20.   lvl_nmo,                lvl_nmo,
21.   f3d,                   f3d,
22.   ocn3d, ---------------> locn3d(nn),
23.   igrid, ---------------> nproj,
24.   reflat,                reflat,
25.   reflon,                reflon,
26.   iref, ----------------> iref(nn),
27.   jref, ----------------> jref(nn),
28.   stdlt1,                stdlt1,
29.   stdlt2,                stdlt2,
30.   stdlon,                stdlon,
31.   delx, ----------------> delx(nn),
32.   dely, ----------------> dely(nn),
33.   conflct,               conflct,
34.   anl_grd,               anl_grd,
35.   deny,                  deny,
36.   blist,                 blist,
37.   wlist,                 wlist,
38.   fgat,                  fgat,
39.   pool,                  pool,
40.   sal_std,               sal_std,
41.   ssh_std,               ssh_std,
42.   tmp_std,               tmp_std,
43.   datao,                 datao,
44.   grd_lat,               grd_lat,
45.   grd_lon,               grd_lon,
46.   f,                     f,
47.   grd_hx,                grd_hx,
48.   grd_hy,                grd_hy,
49.   xpos,                  xpos,
50.   os,                    ypos,
51.   v_opt,                 dbv_opt,
52.   dbv_res,               dbv_res,
53.   den_ds,                den_ds,
54.   diurnal,               diurnal,
55.   err_bkg, -------------> ebkg,
56.   err_mdl, -------------> emdl,
57.   err_obs, -------------> eobs,
58.   tol_fctr,              tol_fctr,
59.   qc_prb, --------------> qc_err,
60.   do_invc, -------------> linck,
61.   mask_opt,              mask_opt,
62.   lndz,                  lndz,
63.   ice_time,              ice_time,
64.   gldr_slct,             gldr_slct,
65.   prf_hrs,               prf_hrs,
66.   prf_opt,               prf_opt,
67.   prf_slct,              prf_slct,
68.   prf_time,              prf_time,
69.   prf_xtnd,              prf_xtnd,
70.   pt_anl,                pt_anl,
71.   rscl,                  rscl,
72.   del_ssh,               del_ssh,
73.   del_sst,               del_sst,
74.   direct,                direct,
75.   modas,                 modas,
76.   ssh_hrs,               ssh_hrs,
77.   ssh_time,              ssh_time,
78.   mds_edit,              mds_edit,
79.   mds_grd,               mds_grd,
80.   mds_mld,               mds_mld,
81.   mds_xtnd,              mds_xtnd,
82.   clm_scl,               clm_scl,
83.   sal_adj,               sal_adj,
84.   smpl,                  smpl,
85.   ssh_opt, -------------> ssh_mean,
86.   sst_time,              sst_time,
87.   st_asm,                st_asm,
88.   st_chain, ------------> st_chn(nn),
89.   st_grid, -------------> st_grd(nn),
90.   st_ntrvl,              st_ntrvl,
91.   st_smpl,               st_smpl,
92.   swh,                   swh,
93.   swh_da,                swh_da,
94.   swh_time,              swh_time,
95.   topo_mn,               topo_mn,
96.   topo_mx,               topo_mx,
97.   topo_src,              topo_src,
98.   hc_mdl,                hc_mdl,
99.   vc_mdl,                vc_mdl,
100.  vc_bkg,                vc_bkg,
101.  dv_dz,                  dv_dz,
102.  vscl, ----------------> vol_scl,
103.  warm,-----------------> warm_ice,
104.  restart,--------------> restart(nn),
105.  amsr_bias,              amsr_bias,
106.  amsr_dw,                amsr_dw,
107.  argo_bias,              argo_bias,
108.  atsr_bias,              atsr_bias,
109.  atsr_dw,                atsr_dw,
110.  goes_bias,              goes_bias,
111.  goes_dw,                goes_dw,
112.  lac_bias,               lac_bias,
113.  lac_dw,                 lac_dw,
114.  mcsst_bias,             mcsst_bias,
115.  mcsst_dw,               mcsst_dw,
116.  metop_bias,             metop_bias,
117.  metop_dw,               metop_dw,
118.  msg_bias,               msg_bias,
119.  msg_dw,                 msg_dw,
120.  model,                  model,
121.  offset,                 offset,
122.  bv_chk,                 bv_chk,
123.  dh_scl,                 dh_scl,
124.  mx_lyr_prs,             mx_lyr_prs,
125.  run_class,              run_class,
126.  global,                 global,
127.  debug,                  debug,
129.  spval,                  spval
```

APPENDIX 3. Sample FORTRAN program to generate a 1-record bathymetry file for the topography database.

```
      program DBVDBV
C
C..................START PROLOGUE.....................................
C
C  SCCS IDENTIFICATION:  %W% %G%
C                        %U% %P%
C
C  DESCRIPTION:  Extracts DBDBV fields for specified latitudes,
C    longitudes from the data base files.
C
C  PARAMETERS:
C     Name        Type       Usage             Description
C   --------    -------     -------   ----------------------------------
C   XLAT        Real        Input     Latitude Array
C                                     Degrees (+ North, - South)
C   XLON        Real        Input     Longitude Array
C                                     Degrees (+ East,  - West )
C                                     or 0-360 positive East
C   NP          Integer     Input     Dimension of XLAT, XLON, DEPTH
C   DEPTH       Real        Output    Depth Array (meters)
C   RES         Real        Input     Minimum resolution (minutes)
C   PATH        Character   Input     Directory path of DBDBV files
C   SOURCE      Character   Input     Data desired (Navy/DOD only or any)
C                                     SOURCE = 'DoD', use Navy files only
C                                     SOURCE = 'Navy', use Navy files only
C                                     SOURCE = ' ', use any dictionary file
C   ERRO        Integer     Input     Level of print diagnostics,
C                                     Bits 0-4 set output switches,
C                                     i.e., ERRO = B"01001" for 4 and 1.
C                           Output    Error flag on return,
c                                     0 = OK.
C                                     1 = OK, points missing though.
C                                     2 = Fail, bad or missing files.
C  FILES:
C     Name      Unit    File Type   Attribute  Usage     Description
C   ---------   ----   ----------- ----------- ------  -----------------
C DBV.dictionary 90      ASCII      Read         In    Identifies appro-
C                                                      priate data base
C                                                      file by lat/lon.
C   stdout        6      ASCII      Write        Out   Diagnostic output
C
C  DATA BASES:
C     Name           Table      Usage          Description
C   ---------   --------------  -----  --------------------------------
C   DBVHRBALTIC Bottom Depths    In    Bottom depths for Baltic Sea (1-min)
C   DBVMREATL   Bottom Depths    In    Bottom depths for E Atlantic (1-min)
C   DBVMRMED    Bottom Depths    In    Bottom depths for Med (5-min)
```

```
C   DBVHRMED    Bottom Depths    In    Bottom depths for Med (1-min)
C   DBVHRSCHINA Bottom Depths    In    Bottom depths for S China Sea (1-min)
C   DBVHRSEUSA  Bottom Depths    In    Bottom depths for SE US Coast (.5-min)
C   DBVHRWEST   Bottom Depths    In    Bottom depths for W US Coast (1-min)
C   DBVMRNATL   Bottom Depths    In    Bottom depths for N Atlantic Ocean
C   DBVMRSATL   Bottom Depths    In    Bottom depths for S Atlantic Ocean
C   DBVMRNPAC   Bottom Depths    In    Bottom depths for N Pacific Ocean
C   DBVMRSPAC   Bottom Depths    In    Bottom depths for S Pacific Ocean
C   DBVMRIND    Bottom Depths    In    Bottom depths for Indian Ocean
C   DBVMRARCTIC Bottom Depths    In    Bottom depths for Arctic
C   DBVMRANTARC Bottom Depths    In    Bottom depths for Antarctic
C   SASHRIND    Bottom Depths    In    Smith-Sandwell for Indian (2-min)
C   SASHRMED    Bottom Depths    In    Smith-Sandwell for Med (2-min)
C   SASHRNATL   Bottom Depths    In    Smith-Sandwell for N Atlantic (2-min)
C   SASHRNPAC   Bottom Depths    In    Smith-Sandwell for N Pacific (2-min)
C   SASHRSATL   Bottom Depths    In    Smith-Sandwell for S Atlantic (2-min)
C   SASHRSPAC   Bottom Depths    In    Smith-Sandwell for S Pacific (2-min)
C
C  ERROR CONDITIONS:
C        Condition                          Action
C    ---------------------      ----------------------------------------
C    Database Access Error      Parameter INFO (ERRO) returns = 2.
C                               Processing will continue for all
C                               requested points.  Diagnostic messages
C                               are written to stdout for all errors.
C    Database Access Error      Parameter INFO (ERRO) returns = 1.
C
C  ADDITIONAL COMMENTS:  This routine manages the extraction of the
C    bathymetry data base files.
C
C..................MAINTENANCE SECTION................................
C
C  MODULES CALLED:
C         Name           Description
C        --------     ------------------------------------------------
C        DBVINZ       Attaches and loads Dictionary file. (Calls DBVLOD)
C        DBVLOD       Loads the Dictionary data into memory.
C        DBVSCH       Search Dictionary for coverage at a point.
C        DBVOPN       Opens data base file for the point. (Calls DBVPRT)
C        DBVPRT       Prints contents of a header array.
C        DBVPTR       Determines record & cell linear index for a point.
C        DBVUPK       Unpacks bottom depth from the specified cell.
C
C  LOCAL VARIABLES AND
C           STUCTURES:
C    Name      Type              Description
C   -------  ---------  ------------------------------------------------
C   IBUF     Integer    record buffer.
C   KREC     Integer    record number to read.
C   KCELL    Integer    word num in record to extract depth from.
C   PATH     Character  Path for the dictionary and data base files.
C
C  METHOD:  Initialize the data access flags, the file names and unit
C    numbers.  Load the dictionary information into arrays.  For each
C    lat/lon point search the dictionary information arrays to see if
```

```
C    there is coverage at the point.  Looping on the number of points,
C    access the data bases for requested parameter and retrieve the
C    values.  If errors occur during the opening, reading or retrieval
C    of data base or dictionary information, the INFO output parameter
C    error flag is set to 2.  Errors less than or equal to 1 are
c    non-fatal, such as points over land.  Additionally, if any of the
c    output diagnostic flags, QFLAG, are true, informative messages are
c    written to output.
C
C  RECORD OF CHANGES:
C
C..................END PROLOGUE.......................................
C
c (XLAT, XLON, NP, DEPTH, RES, PATH, SOURCE, ERRO)
      implicit none
C
      integer ERRO, MAXBASN, MAXWDS, NP
      parameter (np=600)
      parameter (MAXBASN=20, MAXWDS=1009800)
C
      logical*1 btest, DONE(NP), QDBA, QFLAG(5), QBASN(MAXBASN)
C
      character*8 DPRTAB, PARAM
      character*20 DBPFNS(MAXBASN), PFNGRD
      character*256 DBVPATH, DICPFN, PFN
      character*256 PATH, SOURCE
C
      integer I, IBASN, IFO, IND(NP), INFO, IREC, IRM(3,180)
      integer IUDICT, IUGRD, IUNITS, IPTR, K, KCELL, KGDREC
      integer KLAT, KLON, KREC, Len_Trim, LRPFNS(MAXBASN)
      integer NL, NPTS, NREC, NRECT, NRM
      integer*4 IBUF(MAXWDS)
      integer lrec,nltcel,nltblk,nlncel,nlnblk
      integer IW, LBITS, NM, NSFT
C
      integer imin,ideg,nlt,nln
      character label1*40, label2*36,label3*4
      character*5 ns, ew

      real*4 HDR(30), xlt, ylt, xres
      real DEPTH(NP), RES, XLAT(NP), XLON(NP)
      real ALAT, ALON, DBD, DICT(9,124), DTYPE, clat
      real frec,fltcel,fltblk,flncel,flnblk

      integer im,jm
      parameter (im=20, jm=30)
      real h1(im,jm), buf(np/2)
      integer*2 ih1(np),ih2(np)
C
      equivalence (IBUF, BUF)
      data LBITS /16/
C
C * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C
C        Check for type of data to extract, Navy sources only or all
```

```
C
      nl = 1
        do i=1,20
      do k=30,1,-1
          h1(i,k) = i*10
          if (k .eq. 2) h1(i,k) = 33.
          if (i.ge.15) h1(i,k) = 1000.
          ih1(nl) = int(h1(i,k))
          nl = nl+1
        enddo
      enddo
      res = 30.
      path = './'
      label1 = 'TEST  01NPAC     1.0    10/20/0812:30:00'
      label2 = '                                    '
      clat = 2100.
      xlt =  20.
      ylt =   30.
      alon = 10740.
      xres = 30.
      frec = 300.
      fltcel = ylt
      fltblk = 1
      flncel = xlt
      flnblk = 1
c
c open direct access file and write header
c
      open (101, file='TESTBASIN', access='direct',recl=1200)
      write (101, rec=1) label1,label2,clat,ylt,res,alon,xlt,xres,
     s     frec,fltcel,fltblk,flncel,flnblk
c
c print to screen
c
      read (101,rec=1) hdr
      write (*, 900) (HDR(I), I = 1, 5)
  900 format (' Data base file header', 5X, 'PAR  ', 2A4, '  basin  '
     a        , A4, A2, '   version  ', A4)
      write (*, '(" Date/time ", 2A4, 1X, 2A4)') (HDR(I), I = 7, 10)
      CLAT = HDR(20)
      ALAT = 90.0 * 60.0 -CLAT
      IMIN = int(abs(ALAT) +0.5)
      IDEG = IMIN / 60
      IMIN = IMIN -60 * IDEG
      if (ALAT .ge. 0.0) then
         NS = 'NORTH'
      else
         NS = 'SOUTH'
      endif
      RES = HDR(22)
      NLT = int(HDR(21))
      write (*, 910) IDEG, IMIN, NS, RES, NLT
  910 format (5X, ' Latitude grid   ', I3, ' deg  ', I2, ' min ', A5,
     a        '    res ', F6.1, ' min', '   cells ', I6)
C
```

```
      ALON = HDR(23)
      IMIN = int(abs(ALON) +0.5)
      IDEG = IMIN / 60
      IMIN = IMIN -60 * IDEG
      if (ALON .ge. 0.0) then
         EW = 'EAST'
      else
         EW = 'WEST'
      endif
      RES = HDR(25)
      NLN = int(HDR(24))
      write (*, 920) IDEG, IMIN, EW, RES, NLN
  920 format (5X, ' Longitude grid  ', I3, ' deg  ', I2, ' min ', A5,
     a        '    res ', F6.1, ' min', '   cells ', I6)
      LREC = int(HDR(26))
      NLTCEL = int(HDR(27))
      NLTBLK = int(HDR(28))
      NLNCEL = int(HDR(29))
      NLNBLK = int(HDR(30))
      write (*, 930) LREC, NLTCEL, NLTBLK, NLNCEL, NLNBLK
  930 format (5X, ' Block structure', 4X, 'words ', I7, 5X, 'LAT ', I4,
     a        I4, 5X, 'LON ', I4, I4)
c
c write record 2 (record map and first record number)
c
      buf(1) = 1.
      buf(3) = 3.
      buf(4) = 0.
      buf(5) = 19.
      buf(5) = 1.
      write (101,rec=2) buf
      read (101,rec=2) buf
      print'( "Record map", 3X, I5, " entries")', int(buf(1))
c
c write record 3 (depth record)
c
      write (101,rec=3) ih1
      read (101,rec=3) ih2
      print'(20f5.0)',(real(ih2(k)), k=1,np)

c     read (101, rec=2) (IBUF(K), K = 1, LRPFNS(IPTR))
c     IW = (KCELL -1) / 2 +1
c     NSFT = LBITS * (mod(KCELL -1, 2))
C
C        Unload the cell
C
c     NM = ibits (IBUF(IW), NSFT, LBITS)
c     DEPTH = float(NM)

      close (101)
      stop
      end
```

The matching data for the DBV.dictionary file.

```
DICTIONARY          20 October 2008
  1  BOTTOM
 -1
  1     300 1 1 TESTBASIN
 -1
  1  3300.0  2400.0 10740.0 11340.0  30.       1.
 -1
```