OASIS 2.0
Ocean **A**tmosphere **S**ea **I**ce **S**oil

User's Guide
and Reference Manual

November 1995

Laurent Terray, Eric Sevault, Eric Guilyardi, Olivier Thual.

C.E.R.F.A.C.S.
42 avenue Coriolis, 31057 TOULOUSE cedex

Abstract

This user's guide contains a brief overview of the information needed for the straightforward use of OASIS 2.0. As far as we know, it is the best way to use it!

The aim of OASIS is to provide an efficient and easy-to-use tool for coupling independent general circulation models of the atmosphere and the ocean (AO-GCM) as well as other climate modules (sea-ice, vegetation, wave models...). Coupled GCMs (CGCMs) are the necessary tools to tackle current climatic paradigms such as the El Niño Southern Oscillation (ENSO) or the greenhouse gases global warming effect. The models can possibly run on various plateforms and in different environments (batch mode, interactive mode...). Modularity and flexibility have been particularly emphasized in the OASIS design. The use of OASIS does not change the way the models were previously running, except that it requires the addition of few communication routines. In case one wants to run distributed simulations, PVM version 3.3 or later needs to be installed on the different platforms.

Acknowledgements

We would like to thank the following people for their help and suggestions in the design of the OASIS software:

Dominique Astruc (IMFT)

Sophie Belamari (Météo-France)

Dominique Bielli (Météo-France)

Gilles Bourhis (IDRIS)

Pascale Braconnot (CEA)

Yves Chartier (RPN)

Philippe Courtier (CNES)

Philippe Dandin (Météo-France)

Michel Déqué (Météo-France)

Jean-Louis Dufresne (LMD)

Laurent Fairhead (LMD)

Gilles Garric (CERFACS)

Pierre Herchuelz (ACCRI)

Maurice Imbard (LODYC)

Claire Lévy (LODYC)

Olivier Marti (CEA)

Claude Mercier (IDRIS)

Marc Pontaud (CERFACS)

Tim Stockdale (ECMWF)

Rowan Sutton (UGAMP)

Véronique Taverne (CERFACS)

1. General Overview of OASIS

1.1. History and upgrades

The initial work on OASIS began in 1991 when the Climate Modelling team at CERFACS had to build up a french coupled model from existing General Circulation Models (GCMs) developped in an independent way by several laboratories (LODYC, CNRM, LMD). Quite clearly, the only way to handle this heterogeneous environment was to create a very modular and flexible tool (Terray et al. [10]).
The first version of the software was version 0 which was used in the forcing of an oceanic GCM by a set of atmospheric fluxes generated through an AMIP-type run. Version 1.0 was written in 1993 and has been used quite extensively at CERFACS, in particular for the first coupled 10-year simulation of the tropical Pacific (Terray et al. [11]).

Version 1.1, released in 1994, incorporated some bug corrections and several new features. Among them, one can note the possibility to couple an AGCM with a sea-ice model, several interpolation schemes including the case of identical grids (no interpolation) between the two models. This version has been used widely in the french community as well as ECMWF and UGAMP. In particular, it was used at CERFACS to perform two global coupled simulations (one of 50 years at T21 and one of 25 years at T42 with a high resolution global ocean) (Guilyardi et al. [5]; Pontaud et al. [7]).

Version 2.0 (this version) is a major rewriting of the software as the goal was much more ambitious. The modularity of OASIS 2.0 is much greater than before as it can handle now an arbitrary number of models and exchanged fields (while version 1.x could only deal with 2 models and some specific given number of fields). The time strategy can be serial or parallel and the models can now run asynchroneously. Furthermore, modularity is also enabled to the set of analysis one wants to perform on a given field. Lastly, OASIS 2.0 allows to run distributed simulations (the GCMs are on remote computers and OASIS on a workstation, for example) (Sevault et al [9]).
To the authors knowledge, few other coupling packages have been developped with modularity and generality in mind. These include MICASA developped at GFDL (Pacanowski et al. [6]) but with a different philosophy. It is the authors opinion that OASIS 2.0 is a really nice tool and we sure hope to convince you with the help of this manual. OASIS was written by Laurent Terray with contributions from Olivier Thual, Eric Guilyardi, Pascale Noyret, Eric Sevault, Rowan Sutton and Yves Chartier.

1.2. What is OASIS ?

OASIS is a complete self consistent set of fortran 77 routines divided into a main library , interpolation libraries and message passing libraries. The DOCTOR norm from ECMWF (Gibson [4]) with a few extensions has been used to write these routines in order to ease reading and debugging of the program. The extra conventions are described in Appendix 1.
Include files, (of the form *xxxxxx.h*), are needed to compile the code. In particular the file *parameter.h* is used to set up the key dimensions of the software arrays.
All the options of the simulation are defined by the user in an input file, *namcouple*,

which uses free formatting. The code is thorougly documented, i.e each variable or array is defined in the apropriate subroutine or include files headers. Furthermore, numerous comments are included within the code itself to facilitate the understanding of the code structure.

1.3. OASIS philosophy

Coupling numerical models is certainly not a new preoccupation in the climate research community and in other research fields such as electromagnetism, Computational Fluid Dynamics (CFD), etc.... In order to understand the physics of coupling, one needs to have a software interface which allows at a minimal cost the realization of coupled simulations. Ideally, it should permit testing of different coupling algorithms (time strategy or interpolation methods for instance) and objective intercomparison of coupled GCMs by changing one or both models (which implies being able to handle any O-AGCM). It should also allow the coupling of additional models (sea-ice, wave and biosphere models for instance).

One can see right away that these specifications strongly suggest modularity as the key concept in designing the interface structure. The different modular aspects of OASIS 2.0 are the following:

1.3.1. Independent models

The models being coupled are separate entities (different processes in the unix sense). There are unchanged in respect to their own main options (like I/O or multitasking) compared to the uncoupled mode. Few routines need to be added to deal with the time synchronization.

1.3.2. Dynamical definition of models and coupling fields

The models to be coupled and the fields to be exchanged during a simulation are defined ONLY at run-time through the input file. If the number of models or fields change, one needs NOT to recompile or modify the program. One just has to edit and modify the input file *namcouple*. This is done through a pseudo-dynamic allocation (this is standard fortran !!) within OASIS.

1.3.3. Time strategy

The time strategy of the coupling is defined ONLY at run-time through the input file. Models can be run sequentially, in parallel or a combination of both. They can be run synchroneously or asynchroneously.

1.3.4. Analysis

The set of analysis (i.e interpolation, extrapolation, ordering etc...) to be performed on a given field are defined ONLY at run-time through the input file. They can be different for different fields.

1.3.5. Computing environment

The CLIM library (Sevault et al. [8]) which is based on PVM (Geist et al. [3]) can be used to run the models in a distributed environment (i.e the AGCM on a Cray C90, the OGCM on a Cray T3D and OASIS on a front-end workstation). Within the framework of the CATHODE project (Cahouet and Beaucourt [1]), OASIS was used to couple two GCMs running on two remote computers (one in Paris, one in Toulouse) (Cassou et al. [2]).
Further developments will use the Message Passing Interface (MPI) as soon as some points will be added or lighted in the MPI specifications (processes spawning and location, informations on the current physical configuration...). Furthermore, the models can run interactively or in batch mode.
CLIM is an interface library built upon PVM 3.3 or later whose specifications are very close to the OASIS ones (See the CLIM manual). If one model runs on a parallel computer, OASIS (via CLIM) can deal efficiently with the parallel data decomposition of the coupling fields.
Due to the extensive use of standard fortran 77, the OASIS code should run on any usual target for scientific computing (IBM RS6000 and SPs, SPARCs, SGIs CRAY-2, CRAY C90 and CRAY J90 series). The code should also compile with a standard fortran f90 compiler.

2. Obtaining and Installing OASIS

2.1. The OASIS package

OASIS is available as a compressed archive file by anonymous ftp to `ftp.cerfacs.fr` in the `/pub/globc/exchanges/oasis` directory (ftp to ftp.cerfacs.fr, login anonymous, password my_email_adress and cd `/pub/globc/exchanges/oasis`) In this directory you will find the following files and sub-directories:

- `README_v2.0` to be read before doing anything else (!),
- `CHANGES` which records changes between successive releases,
- `oasis_version2.0.tar.Z`, the OASIS release 2.0 compressed file,
- `oasis_version2.0` the on-line OASIS tree for version 2.0,
- `oasis_version2.0beta` the on-line OASIS tree for version 2.0beta,

The `oasis_version2.0.tar.Z` file can be expanded by doing:

```
prompt% uncompress oasis_version2.0.tar.Z
prompt% tar xvf oasis_version2.0.tar
```

The following arborescence is built within the tar file and also on line if you want to explore the tree during the ftp session.

```
- oasis_version2.0/

          - doc/     doc.oasis.ps
                     ------------
                     the OASIS documentation (latex file available
                     on request)

          - include/ *.h
                     ---
                     all the include *.h files

          - lib/     anaisg/*.f
             anaism/*.f
                     fscint/*.f
                     ----------
                     the interpolation libraries

                     pipe/*.f
                     --------
                     the cray pipe message passing library

                     clim/clim.tar
                     ----------------
                     the CLIM library for distributed simulations
```

```
- src/     *.f
           namcouple
           ---------
           the source of the main code + input file

- tools/   fork.c
           datcpl.f
           script_nqs
           ----------
           the tools to run OASIS on a CRAY supercomputer
```

3. The OASIS tree

3.1. OASIS source subroutines summary

The main task of the OASIS software is to read in some coupling fields at specific times during the coupled simulation, to perform a certain number of analysis or operations on these fields and to write them out. As the coupling fields are usually real arrays, a generic definition of an analysis could be the following. Any transformation which either:

- modify some or all element array values
- change array dimensions
- rearrange array storage order

is an analysis which operates on a given coupling field. In what follows, we use indistinctly the words analysis or operation for the type of transformation defined above. Let us now detail the inner structure of the OASIS software.

The OASIS code has a main program, `couple`, which opens up the coupler output and input files, *cplout* and *namcouple* respectively. Then `couple` calls the driver routine `driver` which basically controls the whole simulation. It contains the initialization process, the coupler time iteration loop as well as the basic sequential structure of the OASIS timestepping scheme. The first time iteration differs from the others as the coupler needs no signal to read the input fields (no receive operation is performed). The initial fields must be present at the beginning of the simulation on their original grid. The coupler simply reads in these files and does the interpolation while the models pause initially, waiting for OASIS to provide them their initial boundary conditions.

In fact, the whole modularity concept underlying OASIS can be presented with a description of the routine `driver`. The main entity in the coupling sequence is the coupling field itself and not the model which is generating it. This has strong consequences on the type of input needed by OASIS. To each coupling field is associated the relevant grid-related data (i.e grid longitudes and latitudes, masked points and grid-square surfaces). For each OASIS iteration, `driver` determines which coupling fields have to be exchanged (i.e read in, treated and written out). The field identificators (basically the rank in which they appear in the input file) are stored in an index array. This index together with the total number of fields to be exchanged are passed as arguments to the "core" routines.

These routines can be divided in two categories. The first one deals with I/O issues and contain routines `getfld` and `givfld` which read in and write out the coupling fields, respectively. The second one deals with the analysis branching and contain routines `preproc, interp, cookart` and `postpro`. Routines `preproc` and `postpro` do the fields preprocessing and postprocessing, respectively. Routine `interp` performs the fields interpolation while routine `cookart` does some additional analysis. The inner structure of these routines is similar. The external loop is over the exchanged fields for the current iteration while the internal one runs over the whole set of analysis for the

appropriate field. Within the internal loop, there are a serie of branching (fortran IF statements) towards the analysis routines of the same family.
Finally `driver` also ensures correct termination of the simulation. The other subroutines of the main OASIS code can be divided in five sets as detailed in the following tables.

3.1.1. Initialization

The initialization process consists mainly in establishing communication between OASIS and the other models and in reading the input file and some necessary data such as all the grid-related parameters. Other tasks are calendar initialization and definition of I/O files and units.
The pseudo-dynamic memory allocation is set up in routine `inidya`. Use of this technique was absolutely necessary to handle the modularity regarding the number of coupling fields. The concept is as follows: we define a list of macro arrays whose size is defined thru parameter statements. The list includes arrays for grid parameters such as longitudes, latitudes, masks and grid-square surfaces (for both initial and final grids) as well as arrays for initial and final field values. To each coupling field, we associate one pointer (adress) and one size (proportional to the associated grid dimensions) for initial and final data (here data means field and the grid-related parameters defined above). The macro arrays are then used to read all the field-and-grid related data. All the subsequent analysis deal exclusively with the macro arrays, pointers and sizes defined in routine `inidya`. A check is performed to ensure that the cumulated size of the coupling field arrays does not exceed the dimension of the macro arrays. If it is the case, one needs to increase the value of parameters jpmxold and/or jpmxnew and/or jpmax.
Table 1 describes the different routines involved in the initialization.

Routine name	Description
	Initialization Process
`blkdata`	Block data routine, define default values and initialize constant basic variables.
`inicmc`	Initialize communication between OASIS and other models.
`inidya`	Initialize pseudo-dynamic memory allocation.
`inigrd`	Read in grid-related data files.
`iniiof`	Open up Input-Output files.
`inilun`	Assign logical unit numbers to all used files.
`inipar`	Read in input file *namcouple*.
`initim`	Initialize time iterative scheme parameters.

Table 1: OASIS initialization subroutines

3.1.2. Process Management

We specifically focus here on the `PIPE` version of OASIS. We refer to the `CLIM` manual for a detailed description of the `CLIM` routines. In the `PIPE` version, the GCMs and OASIS are in a child-parent relationship in a unix sense. The following routines are used to detect errors occuring in any of the models (the coupler must be able to stop the simulation if an error has occured in one of the models) as well as to ensure the correct ending of the simulation (the parent must wait until its children have finished). Table 2 describes the routines used for process management.

Routine name	Description
	Process Management
`ferror`	To be executed upon reception of signal SIGFLD. Mean one of the models has exited abnormally.
`getfpe`	To be executed upon reception of signal SIGFPE. Mean a floating point error occurred in the coupler.
`modsgc`	Switch the handler upon reception of signal SIGFLD. Set up the handler to ignore the signal.
`waitpc`	Pause the coupler until termination of all child processes, i.e the models.

Table 2: OASIS process management subroutines

3.1.3. Analysis Control

As previously described, the analysis control routines have a similar internal structure. Basically the external loop is over the active fields for the current iteration. The internal loop is over the number of analysis relative to each field. Within the latter, there are IF statements for each specific analysis relative to the control routine. As noted before, this structure allows complete modularity in the type of analysis to be performed for a given field. The four general types of analysis are preprocessing, interpolation, "cooking" and postprocessing.

The preprocessing of the fields deals with rearranging the arrays according to the OASIS convention, treats the sea-land mask mismatch between the different grids and corrects the fields with external data if required.

The interpolation stage includes algebric combination of the fields **before** interpolation, the interpolation per se and the blending in case of a non global coupling. The "cooking stage" includes the field conservation, algebric combination of the fields **after** interpolation and a conservative (to first order) interpolation with subgrid variability.

The post-processing puts the fields into the right format so they can be read and used directly by the apropriate models.

Table 3 describes the routines used to control the branching to different analysis for the coupling fields.

Routine name	Description
Analysis Control	
`cookart`	Controls the post-interpolation analysis of the coupling fields.
`interp`	Control the coupling fields interpolation.
`postpro`	Control the coupling fields postprocessing.
`preproc`	Control the coupling fields preprocessing.

Table 3: OASIS coupling fields I/O subroutines

3.1.4. Coupling Fields I/O

The modularity in the coupling field I/O part of OASIS is based on the following concept: to each coupling field is associated what we call a channel (whether it is a named pipe associated to a regular binary file or a unix socket). To each coupling field we associate (in the PIPE version) two pipe file names (one to import the field and one to export) and two regular file names where the data are physically located for I/O operations. After reading the input file, OASIS knows when (i.e at which iteration) it has to perform send/receive operations to/from a given channel. OASIS does not know which other process is going to read or write on a given channel.
Here we describe only the routines used in the PIPE version of OASIS. We refer the reader to the CLIM manual for a complete description of the CLIM routines.
Table 4 describes the routines used to deal with the transfer of coupling fields.

Routine name	Description
Coupling Fields I/O	
`getfld`	Control the synchronization and reading of the coupling fields data.
`givfld`	Control the synchronization and writing of the coupling fields data.
`locread`	Locate a character string within a file and reads in subsequent real array.
`locrint`	Locate a character string within a file and reads in subsequent integer array.
`locwrint`	Write a character string to a file followed by an integer array.
`locwrite`	Write a character string to a file followed by a real array.

Table 4: OASIS coupling fields I/O subroutines

3.1.5. Analysis Set

Given two fields F_a and F_b on two different grids G_a and G_b, representing the same (or closely related) physical entity, the analysis set can be viewed as the sequence of operations needed to transform (F_a, G_a) in (F_b, G_b). The current (OASIS 2.0) set of analysis contains fourteen different operations: BLASNEW, BLASOLD, CONSERV, CORRECT, EXTRAP, FILLING, GLORED, INTERP, INVERT, MASK, MOZAIC, REDGLO, REVERSE and SUBGRID.
The repartition in function of the four analysis control routines is as follows (in calling order within OASIS):

- preproc : CORRECT, EXTRAP, INVERT, MASK and REDGLO
- interp : BLASOLD, INTERP and MOZAIC
- cookart : CONSERV, FILLING and SUBGRID
- postpro : BLASNEW, GLORED and REVERSE

To each of these operations is associated one or more fortran routines. The FILLING, GLORED and SUBGRID operations call several subroutines. The INTERP analysis calls one routine, `fiasco`, but which is the interface module to the interpolation libraries.
Table 5 describes the routines which performs the different analysis mentionned above.

Any analysis requires a certain number of input parameters. In what follows, we list for each analysis the needed input parameters and the different possible choices if any. The DOCTOR convention is followed to name the generic variables.

- BLASNEW: this analysis requires at least one line of input with two parameters plus N lines (N being the value of the second parameter) with also two parameters. The generic entry is as follows:

```
# BLASNEW operation
                    XMULT     NBFIELDS
# nbfields lines
                    CNAME    AMULT
...
# End of BLASNEW input
```

  where `XMULT` is the multiplicative coefficient of the current field, `NBFIELDS` the number of additional fields to be combined with the current fields, `CNAME` and `AMULT` their symbolic name and multiplicative coefficient.

- BLASOLD: this analysis requires the same parameters as the previous one.
- CONSERV: this analysis requires one input line with one argument. The generic entry is as follows:

```
# CONSERV operation
                    CMETHOD
# End of CONSERV input
```

Routine name	Description
	Analysis Set
`blasnew`	Do linear combination of fields after interpolation.
`blasold`	Do linear combination of fields before interpolation.
`coasts`	Locate coastal points for extrapolation.
`conserv`	Ensure field conservation through the interpolation process.
`correct`	Apply algebric correction to coupled field.
`extrap`	Do field extrapolation over land.
`fiasco`	Monitor the interpolation process.
`filling`	Complete regional coupling field with global climatological data.
`glored`	Interpolate from a normal to a reduced gaussain grid.
`invert`	Order coupling fields according to coupler convention.
`masq`	Mask field with a constant value.
`mozaic`	Perform field mapping from a coarse to fine grid.
`redglo`	Interpolate from a reduced to a normal gaussian grid.
`reverse`	Reorder coupling fields in agreement with their own model convention.
`subgrid`	Conservative interpolation with subgrid variability.

Table 5: Analysis subroutines

where `CMETHOD` is the conservation method required. It is either GLOBAL or LOCAL. The LOCAL case has no effect as the algorithm is not implemented yet.

- CORRECT: this analysis requires at least one line of input with two parameters plus N lines (N being the value of the second parameters) with four parameters. The generic entry is as follows:

```
# CORRECT operation
                     XMULT     NBFIELDS
# nbfields lines
                     CLOC   AMULT   CFILE    NUMLU
...
# End of CORRECT input
```

  where `XMULT` is the multiplicative coefficient of the current field, `NBFIELDS` the number of additional fields to be combined with the current field, `CLOC` and `AMULT` their symbolic name and multiplicative coefficient, `CFILE` and `NUMLU` the files name and associated logical units on which they are to be read.

- EXTRAP: this analysis requires one input line with two arguments. The generic entry is as follows:

```
# EXTRAP operation
                    CMETHOD     NVOISIN
# End of EXTRAP input
```

  where `CMETHOD` is the extrapolation method (only one is currently available, NINENN) and `NVOISIN` the number of neighbors required to perform the extrapolation (see routine `extrap`).

- INTERP: this analysis requires one input line with at least five parameters. Depending on the interpolation method, 3 or 2 or 0 additional parameters are needed (GAUSSIAN, SURFMESH and all others, respectively). The generic entry is as follows:

```
# INTERP operation in case CMETHOD = anything but GAUSSIAN or SURFMESH
#
  CMETHOD  CGRDSRC  LPERSRC  LPERTGT  CFLDTYP
#
# INTERP operation in case CMETHOD = SURFMESH
#
CMETHOD  CGRDSRC  LPERSRC  LPERTGT  CFLDTYP  NID  NIO
#
# INTERP operation in case CMETHOD = GAUSSIAN
#
CMETHOD  CGRDSRC  LPERSRC  LPERTGT  CFLDTYP  NID  VAR  NIO
# End of INTERP input
```

where `CMETHOD` is the interpolation method (5 are available NNEIBOR, BILINEAR, BICUBIC, SURFMESH and GAUSSIAN), `CGRDSRC` the source grid type (5 types: L uniform; R reduced gaussian; G gaussian; Z irregular; U unstructured or streched) `LPERSRC` and `LPERTGT` the source and target grid periodicity (T or F, true or false), `CFLDTYP` the field type (SCALAR or VECTOR), `NID` the identificator for the ANAIS(M-G) relevant data (weights and adresses), `NIO` the flag which says if the ANAIS(M-G) data can be read or has to be calculated (0 read, 1 calculate) and `VAR` the variance of the gaussian to be used in the GAUSSIAN interpolation method.

- FILLING: this analysis requires one input line with three and possibly six arguments. The generic entry is as follows:

```
# FILLING operation
                     CFILE  NUMLU  CMETHOD
# If CMETHOD(4:6) = 'SST', there are one or three more parameters.
                     CFILE  NUMLU  CMETHOD  NFCOAST
# If CMETHOD(1:3) = 'SMO':
                     CFILE  NUMLU  CMETHOD  NFCOAST  CNAME  NUNIT
# End of FILLING input
```

where `CFILE` is the file name for the global data set, `NUMLU` the associated logical unit, `CMETHOD` the FILLING technique (character*8 variable, the first 3 characters are either SMO, smooth, or RAW, no smoothing ; the next three characters deal with the field name: in this version, it must be either SST or SIE, for sea surface temperature and sea-ice extent respectively ; the last two define the time characteristics of the global data file, MO, SE and AN for interannual monthly, climatological monthly and yearly), `NFCOAST` the flag for the calculation of the coastal correction (0 no, 1 yes), `CNAME` the symbolic name for the flux correction term and `NUNIT` the logical unit on which it is going to be written.

- GLORED: this analysis requires one input line with one argument. The generic entry is as follows:

```
# GLORED operation
                   NTRONCA
# End of GLORED input
```

where `NTRONCA` is the truncature of the gaussian grid (in the current version, it can be either 21, 42, 63 or 106).

- INVERT: this analysis requires one input line with two arguments. The generic entry is as follows:

```
# INVERT operation
                   CORLON  CORLAT
# End of INVERT input
```

where `CORLON` and `CORLAT` are the orientations of the fields in longitude and latitude, respectively (2 possible values for each, NORSUD and SUDNOR, ESTWST and WSTEST).

- MASK: this analysis requires one input line with one argument. The generic entry is as follows:

```
# MASK operation
                  VALMASK
# End of MASK input
```

  where `VALMASK` is the value to be given to the field over land points.

- MOZAIC: this analysis requires one input line with three arguments. The generic entry is as follows:

```
# MOZAIC operation
                   CFILE  NUMLU  NID
# End of MOZAIC input
```

  where `CFILE` and `NUMLU` are the file name and associated logical unit on which the mapping data is going to be read, `NID` the identificator for the relevant mapping dataset (weights and adresses).

- REDGLO: same input as GLORED.

- REVERSE: same input as INVERT

- SUBGRID: this analysis requires one input line with three arguments. The generic entry is as follows:

```
# SUBGRID operation
                    CCOARSE  CFINE  CDQDT
# End of SUBGRID input
```

  where `CCOARSE`, `CFINE` and `CDQDT` are the three symbolic names for the three fields appearing in the first-order Taylor development.

It is worth mentionning some remarks about the use of some of the analysis. The constraints are the following:

- when present, REDGLO and GLORED must be the first and last operations performed, respectively.

- when the EXTRAP operation is required, the MASK analysis has to be performed just before.

- the SUBGRID operation must be used with the MOZAIC one handling the interpolation.

3.1.6. The interpolation libraries

Different interpolation techniques can be used according to grid and field types. A quick description of each will be given here and we refer to the references mentionned below for a complete and detailed presentation.

- `Anaism` : this package does what is called mesh averaging. The minimum grid size of the lower resolution model(LRM) has to be more than the maximum grid size of the higher resolution model(HRM). So any LRM grid square overlaps at least two HRM grid squares. This method is used to interpolate from the HRM towards the LRM. The interpolated field on grid point I_{LRM} is the weighted average of grid points J_{HRM}, $J = 1, N$ with N being the number of overlapped HRM grid squares by the I_{LRM} grid square. The weight is the surface of the HRM grid square overlapped by the LRM grid square.

- `Anaisg` : this method is a nearest-neighbor(s) algorithm with gaussian weight involving the distance between the point on the target grid and its neighbors. The source grid can be unstructured and the number of neighbors as well as the gaussian variance are input parameters.

- `Fscint` : this software does nearest-neighbor, bilinear and bicubic interpolation. Different source grid types are handled such as gaussian, regular and irregular lat-lon grids. The bicubic option is attractive as it ensures continuity of the field and its first derivative. This is essential for the wind stress and its curl which plays a major role for Ekman pumping.

3.1.7. Auxilary Utilities

OASIS 2.0 contains many auxilary routines which perform secondary tasks. Most of them are used before the time loop to read the input file and for checking purposes. Table 6 describes the utility routines which performs auxilary tasks such as checking, calendar, character string manipulation, etc.

3.2. Necessary data files

The OASIS code needs some data files relevant to the models being coupled as well as a number of other files, for instance interpolation parameters or climatological files.
Table 7 gives the name and definition of these files.
All these files have the same structure. Character variables are written to the file to locate the data set corresponding to a given model or interpolation identificator. The data set is then written sequentially after its locator. Let us call "brick" the locator and its associated data set. The order in which the bricks are written doesn't matter. The files *areas, grids, masks and maskr* have character locators which represents a given grid type while, for *gweights* and *mweights*, they represent a given set of interpolation parameters. For instance, the Anaism interpolation can be used for two fields which are not defined on the same grid. Clearly then, the necessary parameters (i.e the interpolation coefficient) are going to differ and can be read properly with the locator trick from file *mweights*.

Routine name	Description
	Utilities Set
`calend`	Evaluate date according to Julian calendar.
`chkmem`	Check size of dynamic allocation with respect to the available memory.
`chkpar`	Ensure global consistency between the coupler and models time parameters.
`empty`	Fill up character string with blanks.
`grstat`	Calculate some grid-related geometrical parameters.
`icoor`	Given an index, find a 2D array row number.
`idivmax`	Find greatest common divider of a set of integers.
`ilenstr`	Find length of a character string.
`imaxim`	Find the greatest element of a set of integers.
`iminim`	Find the smallest element of a set of integers.
`izero`	Zero an integer array.
`jcoor`	Given an index, find a 2D array column number.
`nextbl`	Give the position of the next blank within a character string.
`nextch`	Give the position of the next non-blank character within a character string.
`parse`	Extract a non-blank character string from an eighty-character line.
`prcout`	Print out a character string followed by a character variable.
`prtout`	Print out a character string followed by an integer variable.
`sqdis`	Calculate distance between 2 points on a spheric grid.
`szero`	Zero a real array.
`updtim`	Update calendar date.

Table 6: Auxilary subroutines

Routine name	Description
Files Set	
`areas`	Contain all model mesh surfaces.
`grids`	Contain all model grid coordinates.
`gweights`	Contain weights and corresponding adresses for anaisg interpolation.
`masks`	Contain all model sea-land masks.
`maskr`	Contain reduced gaussian grid masks.
`mweights`	Contain weights and corresponding adresses for anaism interpolation.

Table 7: Necessary files

3.3. OASIS Parameters

Table 8 gives the value by default of some parameters defined in the `parameter.h` file of the OASIS program. When modifying any of these values (for example, increasing the maximum number of models), the complete main source should be re-compiled. The interpolation libraries need not to be recompiled. Most of these parameters are used to dimension macro and work arrays needed to handle the pseudo-dynamic allocation.

3.4. Input File

The OASIS input file, *namcouple* , contains pretty much all the necessary information to run a coupled simulation. Routine `inipar` is entirely devoted to the process of reading the input data file. The first part of the file is devoted to general input parameters such as the number of models involved in the simulation, the number of coupling fields, the type of time coupling strategy etc... The second part deals specifically with input regarding the coupling fields. In particular, it includes the whole list of analysis to be performed on each field as well as the appropriate parameters necessary to perform each analysis. A complete file appropriate for a Pacific ocean model coupled to a global atmospheric model is given in Appendix 3. In the following, we detail the first part of this file.

An important feature of the file *namcouple* is that it does not use any specific format (such as formatted I/O or NAMELIST). OASIS has its own library of character string handling routines which performs the scanning of the input file, the search of keywords and the extraction and traduction of the strings of interest. One can define a character string as follows: it is a finite and continuous serie of non-blank characters. This free-format aspect has important consequences: the keywords used to separate the different kind of data can appear in any order within the input file. The number of blanks between two character strings is non-significant. Furthermore, all lines beginning with # are ignored and considered as comments.

The first part of the file uses some predefined keywords (prefixed by the dollar sign)

Name	Default Value	Description
jpfield	8	Maximum number of coupling fields
jpmodel	2	Maximum number of models
jpanal	12	Maximum number of analysis
jpcomb	2	Maximum additional fields number to be combined in the *blas-xxx* routines
jpmxold	150000	Memory size of the macro arrays handling fields and grid data before interpolation
jpmxnew	150000	Memory size of the macro arrays handling fields and grid data after interpolation
jpmax	150000	Maximum of the last two
jpgrd	51200	Maximum grid size (T106)
jpred	5484	Maximum reduced gaussian grid size (T42)
jpwrk	330	Must be greater than maximum number of longitudes
jpparal	3	Number of parameters for parallel data decomposition (see CLIM manual)
jpwoa	50	Maximum number of underlying neighbors for Anaism interpolation
jpnoa	1	Number of neighbors for Anaisg interpolation
jpnfm	1	Maximum number of different Anaism interpolation data set
jpnfg	1	Maximum number of different Anaisg interpolation data set

Table 8: Main parameters

to locate the related input data. The dollar sign must be in the second column. The $END strings present in the example given in Appendix 3 are there just to nicely separate the different inputs by imitating the namelist-file looking.
The reading of data is done as follows : OASIS scans the input file until it finds the keyword of interest. Then, it skips to the next line and reads it as a character80 variable. This character variable or line is made of character strings (as defined above) separated by one or several blanks. Finally it parses the line to select and extract the character string of interest. If the input data is a character variable, it can be read directly; otherwise one needs to read the character string through an internal file in order to get the right type.

Name	Default	Meaning
`$SEQMODE`	1	Number of sequential models (1 means all parallel)
`$MACHINE`	CRAY	Type of machine (CRAY or IEEE)
`$CHANNEL`	PIPE	Type of message passing (PIPE or CLIM)
`$NFIELDS`	7	Total number of exchanged fields
`$JOBNAME`	CO0	Acronym for the current simulation
`$NBMODEL`	2	Number of models in the experiment and their names
`$RUNTIME`	432000	Total simulated time in seconds
`$INIDATE`	00000000	Initial date of the simulation
`$STRINGS`	None	Keyword to locate beginning of field input data

Table 9: Keywords and the default and meaning of the related variables

These keywords and the meaning of the associated variables are described in table 9. The first seven keywords have only one associated input variable but for $NBMODEL which has $1+N$ associated variables (N being the value of the first variable, the number of models). The N other variables (of the character6 type) are the model acronyms. The last keyword $STRINGS marks the beginning of the coupling fields related data.

So after the last keyword, one has to enter the field related data. The field input data has a similar format whatever field is considered. The first three lines deal with the field parameters while the others deal with the analysis parameters for the current field. A typical first line would be (for instance for a sea surface temperature field):

```
SSTOCEAN SSTATMOS  1   86400   5  sstocefl   sstatmfl  35   96
```

where the different words have the following meaning:

- SSTOCEAN : symbolic name for the field on its initial grid, here the ocean grid (must not exceed 8 characters).
- SSTATMOS : symbolic name for the field on its final grid, here the atmospheric grid (must not exceed 8 characters).
- 1 : refer to the first element of an array defined in routine `blkdata` containing the definitions of fields which can possibly be exchanged during an ocean-atmosphere simulation.

- 86400 : this is the coupling frequency for this field measured in seconds (i.e the frequency at which OASIS reads in the SST field from the ocean model, treats it and writes it out to the atmosphere model).
- 5 : this is the number of analysis to be performed for this field.
- sstocefl : this is the name of the file on which the SST data are going to be read by OASIS
- sstatmfl : this is the name of the file on which the SST data are going to be written by OASIS
- 35 : this is the number of the unit connected to the file on which the SST data are going to be read by OASIS
- 96 : this is the number of the unit connected to the file on which the SST data are going to be written by OASIS

The filenames must correspond to the filenames used in the ocean and atmosphere models to write and read the SST field, respectively. All the words (i.e 9) must be present on the first line. The number of blanks between the words does not matter (there must be at least one, indeed).

The second line for the same case would be:

```
228  94      128  64       topa  at42     1  0
```

where the different words have the following meaning:

- 228 : number of longitudes for the initial grid (i.e here the ocean grid)
- 94 : number of latitudes for the initial grid (i.e here the ocean grid)
- 128 : number of longitudes for the final grid (i.e here the atmosphere grid)
- 64 : number of latitudes for the final grid (i.e here the atmosphere grid)
- topa : root name used to read the grid parameters of the initial grid (here the ocean grid)
- at42 : root name used to read the grid parameters of the final grid (here the atmosphere grid)
- 1 : sequential index of the model generating the SST field, (i.e the ocean model). If the models run sequentially (i.e the SEQMODE input is greater than one), then one needs to say here if the model generating the initial field is in sequential position N.
- 0 : initialization flag. This is used to delay the field exchange in the case of parallel models. Useful if one wants to initialize the coupled system only with sst data. 0 means no delay while N means a delay of N times the coupling frequency.

The third line is the list of analysis to be performed for this field. Let us assume that our ocean model is restricted to the Pacific basin and our atmospheric model uses a spectral reduced gaussian grid. A typical list of analysis for the SST field would be then:

```
INTERP FILLING BLASNEW REVERSE GLORED
```

The first analysis is INTERP which means that the SST data read from the ocean model can be directly interpolated (so the ocean model has the same convention as OASIS). FILLING indicates that the model SST data have to be completed by climatological data to provide a global set which is required by the atmosphere model. BLASNEW (which does simple algebric operations) could be used here to go from Celsius to Kelvin units for the sea surface temperature. REVERSE implies that the AGCM convention differs from that of OASIS, so the field needs to be reversed. Finally, GLORED deals with interpolation from the global gaussian grid to the reduced one.

For the total heat flux field going from the atmospheric model to the ocean one, a typical list of analysis would be the following:

```
REDGLO  INVERT  CORRECT  MASK  EXTRAP  INTERP
```

OASIS first puts the field on the global gaussian grid (REDGLO). It then inverts the field ordering as the AGCM and OASIS conventions differ (INVERT). The CORRECT analysis modifies the total heat flux field with specified (in case one wants to make flux correction) or simulated data. The latter case occurs when the ocean model is a regional one. The model SSTs have then to be mixed with climatological SSTs and if the blending is smooth (i.e if one wants to modify the model SST to ensure a smooth transition), the heat flux needs to be modified to ensure thermodynamic consistency. The MASK analysis assigns a given value for all continental points. This is done in order to perform the EXTRAP analysis which iteratively extrapolates continental point values from ocean point values. Last, INTERP performs the spatial interpolation to the ocean grid. Note that if the fields given by the AGCM are the solar and non solar heat fluxes (while the OGCM needs the solar and total heat fluxes), the BLASOLD operation (which in this case will sum up the solar and non solar component) has to be performed before the interpolation.

The following lines deal with parameters relevant to each selected analysis. In the SST example, one would have the following lines:

```
# Interp
  SURFMESH    Z   F    T     SCALAR      1      0
# Filling
  climssta  21  SMOSSTMO   0    DELTASST  20
# Blasnew
  1.   1
  CONSTANT     273.15
# Reverse
  NORSUD   WSTEST
# Glored
  42
```

The first line describes the interpolation parameters. The first input, "SURFMESH" gives the type of required interpolation. The second, "Z", gives the source grid type while the third and fourth give the periodicity for the source and target grids, respectively (F means false, not periodic and T true, periodic). The fifth input gives the field type, "SCALAR" or "VECTOR". The last two are specific to the ANAIS interpolation: the sixth gives the rank (here it means the order in which the relevant parameters are going to be read from the ANAIS data files) and the last one says if the weights have to be calculated within OASIS or read from the apropriate file (0 means read and 1 calculated).

The second line gives parameters necessary for the FILLING operation. The first input gives the file name for the climatological data used to complete the regional field given by the model (here the OGCM). The second input is the logical unit number associated to the previous file. The third input contains three parameters. The first three letters can be "RAW" or "SMO" (smooth) and indicate the required type of blending. "RAW" means no blending while "SMO" means that the transition between the climatology and model has to be smooth. The next three letters give the field type. In this version, it is used mainly to differentiate between SST, SIE (sea ice extent) and all the other possible fields. The last two letters specify the type of global dataset used ("MO" for monthly interannual and "SE" and "AN" for climatology, monthly or yearly respectively). The last three inputs are read only if the current field is SST. The third input switches off/on the coast mismatch correction (0 means off, 1 means on). The last two inputs have to do with the necessary flux correction to perform if the blending type is smooth. The one-before-last gives the character string needed to locate the correction term data and the last one gives the logical unit on which it is going to be written.

The third line provides the options for the BLASNEW operation. The first input gives the multiplicative coefficient of the main field (1. here) and the second the number of additional fields to be combined with the original one (1 here). The next lines (one per additional field) give specific parameters for each additional field. The first input gives the field name (here "CONSTANT" which means all field values are equal to one) and the second its multiplicative coefficient (273.15 here as it is used to go from Celsius to Kelvin).

The fifth line is related to the REVERSE analysis. The inputs are the storage order for latitude (NORSUD means from north to south) and longitude (WSTEST means from west to east), respectively. The sixth line deals with the GLORED operation. The only needed parameter is the truncature of the considered gaussian grid (42 here). Note here that REVERSE has to be done before GLORED as OASIS can't reverse a field on a reduced gaussian grid.

3.5. Changes in the models

Few changes are necessary in the models to make them suitable for OASIS 2.0. The first thing is to establish the communication between OASIS and the models. We focus here on the PIPE method to illustrate the changes to be made within a given model, say an oceanic model. One needs to open 2 pipes related to the model itself in order to exchange timestep data between OASIS and the model. One pipe is for reading from OASIS, the other one is for writing to OASIS. Note that these pipes are also defined within OASIS by routine `PIPE_Init` and that they must have the same name. The names are specified in `PIPE_Init`: the pipes used to read from models begin with "Preadm" and the pipes used to write to models begin with "Pwritm". Then one adds the model integer identificator defined within OASIS after reading the input file (refer to the $NBMODEL entry). If the input file looks like:

```
 $NBMODEL
           2    model1    model2
 $END
```

model1 will be represented by identificator 01 et model2 by 02. If the oceanic model is model2, the 2 pipes used between OASIS and the ocean model are going to be "Preadm02" (the ocean model writes to this pipe and the coupler reads it) and "Pwritm02" (the coupler writes to this one and the ocean model reads it). The fortran lines to create the pipes within the models can then be directly copied from the equivalent lines within OASIS. An integer array of dimension four must then be exchanged between OASIS and the models (see `PIPE_Init`). In the ocean model, the fortran lines could be the following:

```
C
      INTEGER GETPID
      DIMENSION ISEND(4), IRECV(4)
C  KITRO  : total number of timesteps in oceanic model
C  KEXCO  : frequency of exchange for the fluxes (in time steps)
C  KSTPO  : timestep value (in seconds)
C  NPIOC  : process identificator
      NPIOC = GETPID()
      IMESS(1) = KITRO
      IMESS(2) = KEXCO
      IMESS(3) = KSTPO
      IMESS(4) = NPIOC
      WRITE(Preadm02) IMESS
      READ(Pwritm02)
...
```

Pipes must also be created for each exchanged field within the ocean model. The pipes are not used to transfer the data per se but rather to send a message saying that the data has been written on file. For instance, the ocean model provides the sea surface temperature field and needs the heatflux field. So a reading pipe needs to be defined for the latter and a writing one for the former. Their names are defined by the *namcouple* file. For instance, if the first line of input for the sea surface temperature and heatflux fields are:

```
SSTOCEAN  SSTATMOS  1   86400   5   sstocefl   sstatmfl   35   96
...
HFXATMOS  HFXOCEAN  6   86400   7   flxatmos  flxocean    95   31
```

SSTOCEAN is the name of the pipe used within OASIS to read the SST field from the ocean model. So SSTOCEAN is also the name which has to be used within the ocean model to write the SST field. The fortran lines to create them can be copied directly from routine `PIPE_Define` of the coupler.
The other modifications deal with the writing and reading of the exchanged fields. For the ocean model and the SST field, the fortran code could be something like (with the input line just described):

```
C
C Write the data on field
C
      REWIND(35)
      WRITE(35) 'SSTOCEAN'
      WRITE(35) SST
C Trick to simulate a rewind
      CLOSE(JPTSO)
      OPEN(UNIT=35,STATUS='UNKNOWN',FORM='UNFORMATTED',
     $                FILE='sstocefl')
C
C Write to the pipe (ISTEP is the current timestep)
C
      WRITE(SSTOCEAN) ISTEP
C
```

For the reading of the heatflux field, one could have the following lines:

```
C
C The ocean model waits until the coupler sends a message
C ICSTEP is the coupler current iteration
C
       READ(HFXOCEAN) ICSTEP
C
C The ocean model reads the data
C Q the heat flux, ISIZE the array size, 31 the logical unit associated
C to file flxocean
C
       CALL locread ('HFXOCEAN', Q, ISIZE, 31, IERR)
```

So it means that routine `locread` has to be included in the models in order to read correctly the data.

A. Additional conventions in writing OASIS 2.0 routines

This Appendix contains a brief description of the coding rules followed within OASIS 2.0. All the routines (but those of the library fscint) have been written with the DOCTOR norm (Gibson [4]) with the following extensions:

- the fortran keywords are written in capital letters while all the other character strings are in regular letters.
- some specific indentation rules have been followed in particular for IF and DO statements: two blanks for DO loops and four for IF structures.

Other rules widely used within OASIS 2.0 are the following:

- every routine is subdivided in numbered paragraphs. The DO-loop and others labels within one paragraph begin with the paragraph number.
- all routines (but some auxilary ones) first print a banner including the name of the routine and a short description of the task being performed. A FLUSH is performed on the coupler output file *cplout* at the end of every routine.
- two (1 integer, 1 real) big arrays (nwork, work) with fixed dimensions (defined in parameter.h) are used as temporary arrays in many OASIS routines. This technique is safe as long as one zeroes the arrays before using them.

A. Structure of auxilary files

This Appendix contains a brief description of the format which has to be followed within OASIS 2.0 for a number of auxilary data files related to some analysis, namely FILLING, CORRECT and MOZAIC.
The `filling` routine needs a file with either global climatological or interannual data. The climatology can be annual or monthly while the interannual data is necessarily monthly. In case of monthly data, the file must be written in the following way:

```
      WRITE(NLU_fil) field_january_year_01
      WRITE(NLU_fil) field_february_year_01
      WRITE(NLU_fil) field_march_year_01
      etc...
      WRITE(NLU_fil) field_december_year_01
C
C if climatology, one stops here
C
      WRITE(NLU_fil) field_january_year_02
      etc...
```

Note that the first month need not to be a january. NLU_fil is the logical unit associated to the global data file and is defined in the coupler input file. The connection between the file name (also defined in the coupler input file) and the unit is done in routine `iniiof`. This is the only file within OASIS 2.0 which is not accessed through the `locxxxx` family routines.
In some circumstances (see main text), the FILLING analysis automatically generates a flux correction file defined through a logical unit number given by the coupler input file. This logical unit must correspond to the logical unit defined within the CORRECT analysis to account for this specific flux modification. The `filling` routine writes a locator (defined again through the coupler input file) followed by the flux correction data. A file name is also defined through the input file within the CORRECT analysis and they are connected in routine `iniiof`. Furthermore, a locator is also read through the input file for the CORRECT analysis and it must correspond to the one defined by the FILLING analysis.
The MOZAIC analysis needs data to perform the mapping operation. A file name, a logical unit number and an integer identificator are read from the input file. Locators are built with the identificator in routine `mozaic` which does the reading of the following data: the adresses of the overlapped neighbors (see description of the `mozaic` routine) and the corresponding weights. the locators are "WEIGHTS*Id*" and "ADRESSE*Id*", respectively.

A. A complete input file for OASIS 2.0

This Appendix gives the OASIS 2.0 input file *namcouple* relevant to a simulation with a Pacific ocean model coupled to a global atmospheric model. Refer to the main text for details about the parameters appropriate to each analysis and for the overall file format.
This file tells OASIS 2.0 to run a 5-day simulations during which:

- the two models run in parallel.
- the PIPE library is used for message passing and the computer is a CRAY one.
- the atmospheric model needs two fields, namely the sea surface temperature and the sea-ice extent. The ocean model needs five fields, the heat flux, the solar heat flux, the zonal and meridional wind stress as well as the freshwater flux. These fields are exchanged every day.

The *namcouple* file is as follows:

```
%prompt cat namcouple

########################################################################
# This file can be used AS IT IS for the coupled simulation
#
 $SEQMODE
                1
 $END
#
 $MACHINE
                CRAY
 $END
#
 $CHANNEL
                PIPE
 $END
#
 $NFIELDS
                7
 $END
#
 $JOBNAME
                JR1
 $END
#
 $NBMODEL
                2  arpege   opa
 $END
#
 $RUNTIME
```

```
               432000
 $END
#
 $INIDATE
               19860101
 $END
#
#
 $STRINGS
#
# Field 1 : sea surface temperature
#
SOSSTSST SISUTESU 1 86400  5  sstocean   sstatmos  35   96
228  94    128  64    topa  at42     1  0
 INTERP FILLING BLASNEW REVERSE GLORED
#
# 5 analysis
#
  SURFMESH    Z   F    T     SCALAR      1      0
climssta  21  SMOSSTSE   0    DELTASST  20
  1.   1
  CONSTANT     273.15
  NORSUD   WSTEST
  42
#
# Field 2 : sea ice extent
#
SOICECOV  SIICECOV  2   86400  4  sstocean   sstatmos  35   96
228  94    128  64   topa   at42     1  0
INTERP FILLING REVERSE GLORED
#
# 4 analysis
#
SURFMESH  Z  F    T    SCALAR    1     0
climssta  RAWSIEMO
NORSUD WSTEST
 42
#
# Field 3 : non solar heat flux
#
CONSFTOT SOHEFLDO 6 86400  7   flxatmos  flxocean 95 31
128 64  228 94  at42     topa  1  0
REDGLO INVERT CORRECT MASK EXTRAP BLASOLD  INTERP
#
# 7 analysis
#
42
```

```
NORSUD    WSTEST
1.0     1
DELTASST  -3.456000e+06   fluxcorr   20
9999.999999e+06
NINENN  2
1.157407e-05  1
 COSHFTOT  1.157407e-05
BICUBIC  R  T    F   SCALAR
#
# Field 4 : solar heat flux
#
COSHFTOT  SOSHFLDO  7 86400  6   flxatmos  flxocean 95 31
128 64  228 94  at42     topa  1  0
REDGLO INVERT MASK EXTRAP BLASOLD  INTERP
#
# 6 analysis
#
42
NORSUD    WSTEST
9999.999999e+06
NINENN  2
1.157407e-05  0
BICUBIC  R   T    F   SCALAR
#
# Field 5 : zonal wind stress
#
COZOTAUX  SOZOTAUX   23  86400  6   flxatmos  flxocean 95 31
128 64  228 94  at42     uopa  1  0
REDGLO INVERT MASK EXTRAP BLASOLD  INTERP
#
# 6 analysis
#
42
NORSUD    WSTEST
9999.999999e+06
NINENN  2
1.157407e-05  0
BICUBIC  R    T   F    VECTOR
#
# Field 6 : meridional wind stress
#
COMETAUY  SOMETAUY   24  86400  6   flxatmos  flxocean 95 31
128 64  228 94  at42     vopa  1  0
REDGLO INVERT MASK EXTRAP BLASOLD  INTERP
#
# 6 analysis
#
```

```
42
NORSUD    WSTEST
9999.999999e+06
NINENN  2
1.157407e-05  0
BICUBIC  R  T    F    VECTOR
#
# Field 7 : water flux
#
COWATFLU  SOWAFLDO  30  86400  6  flxatmos  flxocean 95 31
128 64  228 94  at42     topa  1  0
REDGLO INVERT MASK EXTRAP BLASOLD  INTERP
#
# 6 analysis
#
42
NORSUD    WSTEST
9999.999999e+06
NINENN  2
1.157407e-05  0
BICUBIC  R     T    F     SCALAR
 $END
#####################################################################
```

A. Reference pages of OASIS 2.0 routines

This Appendix contains a brief overview of all the routines within OASIS 2.0. The input and output arguments are presented together with a short discussion of what the routine does. Error messages, if any, are also stated. The OASIS main routines are listed below in alphabetical order:

- `blasnew`
- `blasold`
- `blkdata`
- `calend`
- `chkmem`
- `chkpar`
- `coasts`
- `conserv`
- `cookart`
- `correct`
- `couple`
- `driver`
- `empty`
- `extrap`
- `ferror`

- `fiasco`
- `filling`
- `filred`
- `getfld`
- `getfpe`
- `givfld`
- `glored`
- `grstat`
- `icoor`
- `idivmax`
- `ilenstr`
- `imaxim`
- `iminim`
- `inicmc`
- `inidya`
- `inigrd`
- `iniiof`
- `inilun`

- inipar
- initim
- interp
- invert
- isumi
- izero
- jcoor
- locread
- locrint
- locwrint
- locwrite
- mask
- modsgc
- mozaic
- nextbl
- nextch
- parse
- postpro

- `prcout`
- `preproc`
- `prtout`
- `redglo`
- `reset`
- `reverse`
- `revmsk`
- `sqdis`
- `ssumr`
- `subgrid`
- `szero`
- `updtim`
- `waitpc`

Synopsis

```
call blasnew ( pfild ,kmsize, kfield, pmcoeff,
               kaux, kaddr, kasize, pacoeff,
               ksiztot, paux)
```

Arguments

In

`pfild`	-	field array to be modified (real 1D)
`kmsize`	-	size of the field array (integer)
`kfield`	-	field identificator number (integer)
`pmcoeff`	-	field multiplicative coefficient (real)
`kaux`	-	number of combined auxilary fields (integer)
`kaddr`	-	pointer for auxilary fields (integer 1D)
`kasize`	-	size of auxilary fields (integer 1D)
`pacoeff`	-	auxilary field multiplicative coefficient (real 1D)
`ksiztot`	-	total size of work memory used (integer)
`paux`	-	global work array (real 1D)

Out

`pfild`	-	new field array (real 1D)

Discussion

`blasnew` performs a linear combination of the current coupling field with auxilary fields. These can be other coupling fields after interpolation or constant fields. This routine can be used to switch between different units used in the models or to add coupling fluxes together to give a new one (for instance the solar and non solar heat flux to make the total heat flux). Note that a similar routine, `blasold`, exists for combining fields before interpolation. The need for `blasnew` comes from the case in which one field has to be completed by climatology (the `filling` routine) after interpolation. It is then more logical to perform the linear combination once the final global field has been assembled.

Errors

`blasnew` checks the size of the auxilary fields compared to that of the main field. If they differ, it stops the simulation.

Synopsis

```
call blasold ( pfild ,kmsize, kfield, pmcoeff,
               kaux, kaddr, kasize, pacoeff,
               ksiztot, paux)
```

Arguments

IN

`pfild`	-	field array to be modified (real 1D)
`kmsize`	-	size of the field array (integer)
`kfield`	-	field identificator number (integer)
`pmcoeff`	-	field multiplicative coefficient (real)
`kaux`	-	number of combined auxilary fields (integer)
`kaddr`	-	pointer for auxilary fields (integer 1D)
`kasize`	-	size of auxilary fields (integer 1D)
`pacoeff`	-	auxilary field multiplicative coefficient (real 1D)
`ksiztot`	-	total size of work memory used (integer)
`paux`	-	global work array (real 1D)

OUT

`pfild`	-	new field array (real 1D)

Discussion

`blasold` performs a linear combination of the current coupling field with auxilary fields. These can be other coupling fields before interpolation or constant fields. This routine can be used to switch between different units used in the models or to add coupling fluxes together to give a new one (for instance the solar and non solar heat flux to make the total heat flux).Note that a similar routine, `blasnew`, exists for combining fields after interpolation.

Errors

`blasold` checks the size of the auxilary fields compared to that of the main field. If they differ, it stops the simulation.

Synopsis

`None`

Arguments

IN

`None`

OUT

`None`

Discussion

`blkdata` is used to initialize at compile time variables named in common blocks. The initialized variables are all the general parameters of the simulation to be performed (number of models, fields, time of the simulation, ...). Field labels are initialized too in the `blkdata` routine which can be used to fill up the field part of the *namcouple* file. The default for the names of the models grid-related files is also defined in `blkdata`. The date of the current run as well as the beginning date of the whole simulation are also initiated here. Last, data related to reduced gaussian grids is provided, namely the number of latitude points by longitude circle for troncature T21, T42, T63 and T106. Check if they correspond to the values you are using within your spectral model.

Errors

No error code for this routine.

Synopsis

```
call calend ( kdini, kmini, kyini, kinc,
              kdfin, kmfin, kyfin, klmo)
```

Arguments

IN

`kdini`	-	initial day (integer)
`kmini`	-	initial month (integer)
`kyini`	-	initial year (integer)
`kinc`	-	date increment in days (integer)

OUT

`kdfin`	-	final day (integer)
`kmfin`	-	final month (integer)
`kyfin`	-	final year (integer)
`klmo`	-	length of months (integer)

Discussion

`calend` updates the date at each coupling timestep. It takes into account leap years. Note that `calend` is called from within the main time loop but outside of the sequential model loop. So it must agree with the models calendar.

Errors

No error code for this routine.

Synopsis

```
call chkmem ( kneed)
```

Arguments

IN

`kneed` - requested amount of memory (integer 1D)

OUT

`None`

Discussion

`chkmem` checks that the requested memory for the big macro arrays does not exceed the available storage defined through the parameters jpmxold (initial fields) and jpmxnew (final fields). The amount of memory needed depends on the number of initial and final fields as well as on the related grid sizes.

Errors

If the program stops in `chkmem` because of insufficient memory, one has to increase jpmxold and/or jpmxnew in the *parameter.h* file, recompile the code and launch the simulation another time.

Synopsis

```
call chkpar
```

Arguments

IN

`None`

OUT

`None`

Discussion

`chkpar` checks time parameters compatibility between the coupler and the models. In particular, it checks that the total simulation time and the smallest coupling frequency of the models are compatible with the coupler input data.

Errors

The program stops in `chkpar` if one of the model coupling frequency is not compatible with the coupler input. Check oasis input file *namcouple* as well as the data sent by the models. In case the durations are different, the coupler prints out a warning to point out it is doing an asynchroneous simulation.

Synopsis

```
call coasts ( plon, plat, kmsk, kmskval,
              klon, klat, kmesh)
```

Arguments

IN

`plon`	-	target grid longitude array (real 2D)
`plon`	-	target grid latitude array (real 2D)
`kmsk`	-	target grid sea-land mask (integer 2D)
`kmskval`	-	sea-land mask value (integer)
`klon`	-	number of longitude for target grid (integer)
`klat`	-	number of latitude for target grid (integer)
`kmesh`	-	overlap array (integer 2D)

OUT

`None`

Discussion

`coasts` is designed to detect coastal points where there is mismatch between the atmosphere and ocean land sea masks and where this mismatch could result in the atmosphere (undesirably) 'seeing' climatological SST's directly adjacent to ocean model SST's. Where this situation arises, suitable neighbours from where ocean model SST's may be extrapolated are located.

Note

This is only for use with ANAISM interpolation of SST's.

Synopsis

```
call conserv ( pfldb, ksizb, kmskb, psurfb,
               pflda, ksiza, kmska, psurfa,
               psgrb, psgra, cdmet)
```

Arguments

IN

- `pflda` - initial field on target grid (real 1D)
- `pfldb` - field on source grid (real 1D)
- `kmska` - mask for target grid (integer 1D)
- `kmskb` - mask for source grid (integer 1D)
- `psurfa` - surfaces for target grid meshes (real 1D)
- `psurfb` - surfaces for source grid meshes (real 1D)
- `ksiza` - target arrays size (integer)
- `ksizb` - source arrays size (integer)
- `psgra` - work array (real 1D)
- `psgrb` - work array (real 1D)
- `cdmet` - conservation method (character string)

OUT

- `pflda` - field on target grid after conservation

Discussion

`conserv` performs global flux conservation. The method is a trivial one: the flux is integrated on both source and target grids and the residual (target - source) is calculated. Then all flux values on the target grid are uniformly modified, according to their surface. The previous calculations do not consider masked points.

Errors

No error code for this routine.

Synopsis

```
call cookart ( kindex, kfield)
```

Arguments

IN

`kindex` - field identificators array (integer 1D)
`kfield` - current number of active fields (integer)

OUT

`None`

Discussion

`cookart` performs the branching for additional analysis after the interpolation stage. Namely, subgrid variability, flux conservation and basic linear combination can be applied to the coupling fields. The branching is performed as follows: the number of active fields for the current iteration as well as their identificators are searched in `driver` and passed as arguments to `cookart`. Then the main loop within `cookart` is over the active fields and, for each of them, one checks if one or several of the above analysis has to be performed.

Errors

No error code for this routine.

Synopsis

```
call correct ( pfild, ksize, pmcoef, kaux,
               pacoef, pwork, kunit,
               cffic, cdfld)
```

Arguments

IN

`pfild`	-	field on source grid (real 1D)
`ksize`	-	size of field array (integer)
`pmcoef`	-	main field coefficient (real)
`kaux`	-	number of auxilary fields (integer)
`pacoef`	-	auxilary field coefficients (real 1D)
`pwork`	-	work array to read auxilary fields (real 1D)
`kunit`	-	logical unit numbers for data files (integer)
`cdfic`	-	filenames for external data (character 1D)
`cdfld`	-	auxilary field names (character 1D)

OUT

`pfild`	-	corrected field on source grid (real 1D)

Discussion

`correct` uses external data to modify coupling fields. The data are read on files and a symbolic name is used to define the data to be read. This routine can be used to perform flux correction technique on some or all coupling fields. It can also be used to read data generated by OASIS itself in order to use it for its internal use. For instance, in case of a regional coupling, the SST along and within the domain boundaries must be blended with the climatological SST. In order to be consistant, the heat fluxes must be changed accordingly in these regions. The `filling` routine generates the flux data which is used by `correct` to modify the heat fluxes. See description of `filling` for details.

Errors

No error code for this routine.

Synopsis

```
call couple
```

Arguments

IN

`None`

OUT

`None`

Discussion

`couple` is the main program of the OASIS source code. It performs the opening of the OASIS input and output files. It calls `driver` which controls the simulation.

Errors

No error code for this routine.

Synopsis

```
call driver
```

Arguments

IN

`None`

OUT

`None`

Discussion

`driver` is the main routine of the OASIS program. It basically controls the whole simulation. First, there is an initialization stage performed by routines of the `inixxx` type. It is followed by the temporal loop of the OASIS code which contains the import, preprocessing, interpolation and export of the coupling fields. A serie of checks is made to get which fields have to be treated at a given iteration. The OASIS timestep value is defined as the greatest common divisor of all field coupling frequencies. At each iteration and for each field, one checks the coupling frequency, the sequential index (indicating the order in a sequential simulation) and the initialization flag (to allow for a delay in the treatment of a given field). The total number of fields to be treated is then passed to the I/O and analysis routines together with the field indexes (i.e their order of appearance in the input file *namcouple*. Note that a coupled simulation with OASIS 2.0 starts with the interpolation of the boundary conditions from their original grid to the target grid in contrast with the previous versions. Consequently, the GCMs pause initially until the coupling variables have been interpolated. Note also that the time loop goes from 0 to niter-1 in contrast with previous versions. The iteration 0 of oasis does not increment the simulation time.

Errors

No error code for this routine.

Synopsis

```
call empty ( cdchar, klen)
```

Arguments

IN

`cdchar`	-	string to be filled up with blanks (character)
`klen`	-	string length (integer)

OUT

`cdchar`	-	string filled up with blanks

Discussion

`empty` fills up a character string with blanks.

Errors

No error code for this routine.

Synopsis

```
call extrap ( pfild, pmask, pwork, kxlon, kylat,
              knbor, kwork, cdextmet, ldper)
```

Arguments

IN

`pfild`	-	field to be extrapolated (real 2D)
`pmask`	-	mask value (real)
`pwork`	-	work array (real 2D)
`kxlon`	-	number of longitudes (integer)
`kylat`	-	number of latitudes (integer)
`knbor`	-	number of neighbors for extrapolation (integer)
`kwork`	-	work array dimension (integer)
`cdextmet`	-	extrapolation method (character)
`ldper`	-	grid periodicity (logical)

OUT

`pfild`	-	field extrapolated (real 2D)

Discussion

`extrap` performs the extrapolation of a field over land points using sea points values by a progressive nearest-neighbor(s) filling method. The routine accounts for periodicity in longitude as well as correct values in case of a global grid in latitude. It must be used in relation with the routine `masq` which gives a specified value to land points so they can be identified within `extrap`. This is needed as the extrapolation method currently used is an iterative procedure where the set of masked points evolves at each iteration. The number of sea neighbors required to extrapolate is a variable which is defined for each field in the input file *namcouple*.

Errors

No error code for this routine.

Synopsis

```
call ferror
```

Arguments

IN

`None`

OUT

`None`

Discussion

`ferror` is executed each time signal sigcld is caught. This routine is used only when the PIPE version has been chosen. In the PIPE version, OASIS and the other models are in a parent-child relationship. So whenever one of the child terminates, it sends away signal sigcld to the parent. Upon reception of signal sigcld, `ferror` aborts the simulation to avoid an eternal deadlock.

Errors

No error code for this routine.

Synopsis

```
call fiasco ( poutp, ptlon, ptlat, ptsurf,
              ktmsk, ktlon, ktlat, ldtper,
              pinpt, pslon, pslat, pssurf,
              ksmsk, kslon, kslat, ldsper,
              plonz, platz, psgrb, psgra, psfrb, psfra,
              kfield, cdint, cdtyp, cddim)
```

Arguments

IN

- `ktlon` - number of longitudes for target grid (integer)
- `ktlat` - number of latitudes for target grid (integer)
- `ptlon` - longitudes of target grid (real 2D)
- `ptlat` - latitudes of target grid (real 2D)
- `ktmsk` - mask of target grid (integer 2D)
- `ptsurf` - target grid square surfaces (real 2D)
- `ldtper` - target grid periodicity (logical)
- `pinpt` - input field on source grid (real 2D)
- `kslon` - number of longitudes for source grid
- `kslat` - number of latitudes for source grid
- `pslon` - longitudes of source grid (real 2D)
- `pslat` - latitudes of source grid (real 2D)
- `ksmsk` - mask of source grid (integer 2D)
- `pssurf` - source grid square surfaces (real 2D)
- `ldsper` - source grid periodicity (logical)
- `kfield` - current field number (integer)
- `cdint` - type of interpolation (character)
- `cdtyp` - type of source grid (character)
- `cddim` - type of field (scalar or vector) (character)

OUT

- `poutp` - output field on target grid (real 2D)

Discussion

`fiasco` is the interpolation driver. All the field related data required for the interpolation is passed to `fiasco` as arguments (grid data, field type, grid type etc...). The only other argument is the type of interpolation to be performed which can indeed be different for different fields. OASIS currently uses three main libraries of interpolation routines: ANAISG, ANAISM and FSCINT. ANAISG is a nearest neighbor(s) interpolation technique with gaussian weight function. It can be used with any source or target grids. ANAISM is a surface-averaging interpolation which can be used to go from a fine to a coarse grid (the source grid must be finer over the whole domain). For a target grid square, one finds all the underlying source grid squares and the target grid field value is the weighed sum of the source grid field values by the overlapped surfaces. Note that for both ANAISG and ANAISM, **NOTHING** is done for the masked points. FSCINT contains three interpolators: a nearest-neighbor, a bilinear and a bicubic algorithms. All the grid points are treated including the masked ones. So before interpolation, one has to deal with sea-land mismatch between the models (see routine `extrap`). It can be used with different types of grids: regular, gaussian, A-B-C Arakawa grids etc... See the interpolation part of the documentation for details about these libraries. Another type of interpolation is available, namely grid mapping with or without subgrid variability (see description of routines `mozaic` and `subgrid`).

Errors

No error code for this routine.

Synopsis

```
call filling ( pfild, pworka, pworkb, pworkc,
               plon, plat, klon, klat, kmask, kmesh,
               cdfic, cdmet)
```

Arguments

IN

`pfild` - field to be completed (real 2D)
`pworka` - work array (real 2D)
`pworkb` - work array (real 2D)
`pworkc` - work array (real 2D)
`plon` - grid longitude array (real 2D)
`plat` - grid latitude array (real 2D)
`klon` - number of longitude (integer)
`klat` - number of latitude (integer)
`kmask` - mask array (integer 2D)
`kmesh` - overlapped neighbors array (integer 2D)
`cdfic` - climatological data filename (character)
`cdmet` - filling method (character)

OUT

`pfild` - field completed (real 2D)

Discussion

`filling` performs the blending of a regional data set with a global climatological or interannual one. This occurs for the SST field in case of the coupling of a non-global ocean model with a global atmospheric model. The filling method argument encapsulates necessary information for the blending stage. Namely, the global dataset can be either monthly or yearly and the blending can be smooth or abrupt (only model values within the model domain , and climatological values outside). If the blending is smooth, one performs a linear interpolation between the two fields **WITHIN** the model domain over narrow bands along the boundaries. The user must provide the climatological data file with a specific format described in Appendix 2. When one uses `filling` for SST with smooth blending, thermodynamics consistency requires to modify the heat fluxes over the blending regions. The correction term is proportional to the difference $\delta(SST) = SST_{bl} - SST_i$ where SST_{bl} is the blended SST on the atmospheric grid and SST_i the simulated SST interpolated on the atmospheric grid. The $\delta(SST)$ term is calculated within `filling` and written out on a file which is to be read later. This is done with a call to routine `correct` (see relevant description). In case the non global ocean model is a multi-basin one, one can ask also for coastal points correction due to sea-land mask mismatch (see description of routine `coasts`). This is done within `filling`.

Errors

No error code for this routine.

Synopsis

```
call getfld ( kindex, kfield, kiter)
```

Arguments

IN

`kindex` - field identificators array (integer 1D)
`kfield` - current number of active fields (integer)
`kiter` - current iteration number (integer)

OUT

`None`

Discussion

`getfld` imports the currently active coupling fields using either the PIPE or the CLIM library. In the PIPE case, OASIS reads in the named pipe associated to the current active coupling field (see routine `PIPE_Recv`). Once a message has been written on it, OASIS reads in the field data on a regular binary file through the routine `locread`. In the CLIM case, both actions are done by the call to routine `CLIM_Import` (see the CLIM documentation). Note the special treatment of the first iteration (number 0): OASIS does not have to wait as the coupled simulation must begin by the interpolation of the boundary conditions from their original model grid to the grid where they are going to be used. A simple call to `locread` is done to read the data before to proceed to the analysis part.

Errors

If OASIS can't read a coupling field, it stops the simulation and gives a meaningful error code in case the CLIM library is being used.

Synopsis

```
call getfpe
```

Arguments

IN

`None`

OUT

`None`

Discussion

`getfpe` is executed each time signal SIGFPE is caught by the coupler. It stops the simulation right away in a clean fashion.

Errors

No error code for this routine.

Synopsis

```
call givfld ( kindex, kfield, kiter)
```

Arguments

IN

`kindex` - field identificators array (integer 1D)
`kfield` - current number of active fields (integer)
`kiter` - current iteration number (integer)

OUT

`None`

Discussion

`givfld` exports the currently active coupling fields using either the PIPE or the CLIM library. In the PIPE case, OASIS writes out the field data on a regular binary file through the routine `locwrite`. Then a signal is being sent through the named pipe associated to the current active coupling field (see routine `PIPE_Send`). In the CLIM case, both actions are done by the call to routine `CLIM_Export` (see the CLIM documentation).

Errors

If OASIS can't write a coupling field, it stops the simulation and gives a meaningful error code in case the CLIM library is being used.

Synopsis

```
call glored ( pzgg, pworkgr, klon, klat, ktronca)
```

Arguments

IN

`pzgg`	-	field identificators array (real 2D)
`pworkgr`	-	current number of active fields (real 2D)
`klon`	-	number of longitudes of global grid (integer)
`klat`	-	number of latitudes of both grids (integer)
`ktronca`	-	truncature of gaussian grid (integer)

OUT

`pzgg`	-	field on reduced grid (real 2D)

Discussion

`glored` performs a linear interpolation of a field from a full gaussian grid to a reduced one. The interpolation is done by latitude circle. The reduced grid data are read in routine `blkdata` and the user must check if they correspond to its spectral model implementation.

Errors

Note that no mask is used in the interpolation. It is assumed that a preprocessing of the field on the full grid was done to allocate ocean values to land points.

Synopsis

```
call grstat ( pcorx, pcory, psgr, kmsk, kngx, kngy,
              pdr, psr, phhi, kvma)
```

Arguments

IN

`pcorx`	-	the grid points longitude (real 2D)
`pcory`	-	the grid points latitude (real 2D)
`psgr`	-	surface elements (real 2D)
`kmsk`	-	the grid mask (integer 2D)
`kngx`	-	the grid size in direction 1
`kngy`	-	the grid size in direction 2
`kvma`	-	integer value of the mask

OUT

`pdr`	-	the average grid square size
`psr`	-	the total surface
`phhi`	-	inverse of sum of surface elements squared

Discussion

`grstat` performs some statistical analysis of a given grid. Namely, it determines the average grid square size, the total surface and the inverse of sum of surface elements squared. These can be useful to specify values for parameters of the ANAISG interpolation as well as straightforward checks of grid data.

Errors

No error code for this routine.

Synopsis

```
ii = icoor ( kind, kimax)
```

Arguments

IN

`kind` - index (integer)
`kimax` - total number of rows (integer)

OUT

`None`

Discussion

Given an index value, kind, for a 2D array which has kimax rows, this function returns the row number.

Errors

No error code for this routine.

Synopsis

```
ii = idivmax ( ka, kna, kguess)
```

Arguments

IN

`ka`	-	array to be searched (integer 1D)
`kna`	-	current number of active fields (integer)
`kguess`	-	initial guess for the divisor (integer)

OUT

`None`

Discussion

`idivmax` searches the greatest common divisor of all elements of an integer array. This is used to determine the OASIS timestep from the set of field coupling frequencies. Thus, the timestep is equal or less than the minimum coupling frequency. At each OASIS iteration, the whole set or a subset of fields is then treated. There can be iterations where no field is treated.

Errors

No error code for this routine.

Synopsis

```
ii = ilenstr ( cdstr, knstr)
```

Arguments

IN

`cdstr`	-	string to be searched (character)
`knstr`	-	total length of the string (integer)

OUT

`None`

Discussion

`ilenstr` finds the length of a character string (no blank characters). This function is used in the reading of the input file *namcouple*.

Errors

No error code for this routine.

Synopsis

```
i = imaxim ( ka, kna)
```

Arguments

IN

`ka` - array to be searched (integer 1D)
`kna` - array dimension (integer)

OUT

`None`

Discussion

`imaxim` searches the maximum of the elements of an integer array.

Errors

No error code for this routine.

Synopsis

```
i == iminim ( ka, kna)
```

Arguments

IN

`ka` - array to be searched (integer 1D)
`kna` - array dimension (integer)

OUT

`None`

Discussion

`iminim` searches the minimum of the elements of an integer array.

Errors

No error code for this routine.

Synopsis

```
call inicmc
```

Arguments

IN

`None`

OUT

`None`

Discussion

`inicmc` starts up the communication with the outside world. First, it defines and creates the communication channels with all the other models through calls to routine `XXXX_Init` where XXXX is either CLIM or PIPE. See these routines for details. Then, it defines and creates the communication channels for all the fields to be exchanged between the models and OASIS. This is done through calls to routine `XXXX_Define` where XXXX is either CLIM or PIPE. Last, it exchanges timestep data through the model channels in order to perform some basic consistency checkings in routine `chkpar`.

Errors

No error code for this routine.

Synopsis

```
call inidya
```

Arguments

IN

`None`

OUT

`None`

Discussion

`inidya` initializes to zero all the macro arrays. It then determines the small array adresses within the macro arrays from their size. It then checks if the global parameters for the macro arrays are big enough given the number and size of fields to be treated. See routine `chkmem` for details.

Errors

No error code for this routine.

Synopsis

```
call inigrd
```

Arguments

IN

`None`

OUT

`None`

Discussion

`inigrd` reads in all the grid parameters and stores them in the grid macro arrays. Specifically, it reads longitudes, latitudes, sea-land mask and grid-square surfaces for both initial and final grids. Four files contain all the data: *grids*, *masks*, *maskr* and *areas*. The longitudes and latitudes are in *grids*, the masks are in *masks* but for the reduced gaussian grid mask which is in *maskr*, and the grid-square surfaces in *areas*. All the grid data follows the OASIS convention (South to North and West to East from Greenwhich) but for the reduced gaussian grid mask which must be as in the model using it. All the reading is performed through calls to routine `locread`. The locators (8-character strings) used by `inigrd` are made of two pieces. The first one (prefix) refers to the type of grid while the second one (suffix) describes the type of data to be read. The prefix is read from the input file while the suffix is defined in routine `blkdata`. For instance, it is ".lon" for longitudes and ".msk" for the masks.

Errors

Each call to routine `locread` generates an error flag which is cumulated in the `inigrd` error flag. If an error occurs, an informative message is written to the OASIS output file *cplout* and OASIS stops the simulation at the end of the routine.

Synopsis

```
call iniiof
```

Arguments

IN

`None`

OUT

`None`

Discussion

`iniiof` opens up files needed by OASIS. Specifically, OASIS needs grid related files, data transfer files and interpolation or other analysis-related data files.

Errors

All the OPEN operations are checked and OASIS stops the simulation if one of the OPEN fails. A description of the error is then printed in the OASIS output file *cplout*.

Synopsis

```
call inilun
```

Arguments

IN

`None`

OUT

`None`

Discussion

`inilun` sets up logical unit numbers related to the grid data files as well as ANAIS related data files (output and weights data files).

Errors

No error code for this routine.

Synopsis

```
call inipar
```

Arguments

IN

`None`

OUT

`None`

Discussion

`inipar` reads in the OASIS input file *namcouple* which contains all the data and parameters needed to perform a coupled simulation. The input data is read through recurrent calls to routine `parse` which selects the n^{th} character string within a line. If the variables to be read are of type integer or real, an internal READ is performed to make the type conversion. All the input variables are then printed on the OASIS output file *cplout*.

Errors

The first part of the input file contains the simulation general options. If these variables are not present within *namcouple*, the default values defined in routine `blkdata` are used and a warning is issued in the output file *cplout*. Furthermore, OASIS stops the simulation if it finds an unknown type of analysis or parallel decomposition (see the CLIM documentation).

Synopsis

```
call initim
```

Arguments

IN

`None`

OUT

`None`

Discussion

`initim` determines the iterative scheme parameters. First, it calculates the OASIS timestep and the total number of iterations. At this point, OASIS knows all the field exchange frequencies and finds the greatest common divisor which becomes the OASIS timestep. Then the total number of iterations is obtained as the total length of the simulation has been read from the input file. The initial date is also printed on the OASIS output file *cplout*.

Errors

No error code for this routine.

Synopsis

```
call interp ( kindex, kfield)
```

Arguments

IN

`kindex`	-	current field identificators array (integer 1D)
`kfield`	-	current number of active fields (integer)

OUT

`None`

Discussion

`interp` monitors the interpolation step which is done with a call to routine `fiasco`, the interpolation driver. All the interpolation parameters and needed data are passed as arguments to `fiasco`. So if one wants to add a new interpolation method, one has just to change the code within `fiasco` and below. The other analysis called from `interp` are BLASOLD, FILLING and MOZAIC. BLASOLD must be done first as it deals with simple algebric operations on the fields before interpolation. MOZAIC is a very simple mapping scheme: it basically performs the mapping of a field from a coarse (CG) to a fine grid (FG). In case there is an integer number of FG grid-squares underneath any CG grid-square, the field value on the FG grid-points is simply that of the overlapping CG grid-point. In general, the field value on the FG grid-points is a weighted sum over several CG grid-points with weights given by the overlapped fraction of the FG grid-square surface. FILLING completes a regional dataset with a global climatological or interannual one.

Errors

No error code for this routine.

Synopsis

```
call invert ( pfild ,kxlon ,kylat, cdxord, cdyord)
```

Arguments

IN

`pfild` - field identificators array (real 2D)
`kxlon` - number of longitudes (integer)
`kylat` - number of latitudes (integer)
`cdxord` - longitude ordering (character)
`cdyord` - latitude ordering (character)

OUT

`pfild` - field reordered (real 2D)

Discussion

`invert` reorders field *pfild* according to arguments *cdxord* and *cdyord*. The OASIS convention is to go from South to North and from Greenwhich to the East. So if some fields (i.e models) have a different convention, these need to be inverted before being treated by OASIS.

Errors

No error code for this routine.

Synopsis

```
i = isumi ( ka, kna)
```

Arguments

IN

`ka` - array to be summed (integer 1D)
`kna` - array dimension (integer)

OUT

`None`

Discussion

`isumi` sums up the elements of an integer 1D array.

Errors

No error code for this routine.

Synopsis

```
call izero ( ka, kna)
```

Arguments

In

`ka`	-	array to be zeroed (integer 1D)
`kna`	-	array dimension (integer)

Out

`ka`	-	array filled up with zeros (integer 1D)

Discussion

`izero` zeroes integer array ka from element 1 to element kna.

Errors

No error code for this routine.

Synopsis

```
i = jcoor ( kind, kimax)
```

Arguments

IN

`kind`	-	index (integer)
`kimax`	-	number of rows (integer)

OUT

`None`

Discussion

Given an index value, kind, for a 2D array which has kimax rows, this function returns the column number.

Errors

No error code for this routine.

Synopsis

```
call locread ( cdfldn, pfield, kdimax, knulre, kflgre)
```

Arguments

IN

`cdfldn`	-	character string locator (character)
`kdimax`	-	dimension of field to be read (integer)
`knulre`	-	logical unit to be read (integer)

OUT

`pfield`	-	field array (real 1D)
`kflgre`	-	error status flag (integer)

Discussion

`locread` reads in a character string followed by a real array from logical unit *knulre* with binary format. The character string is used as a locator during the search.

Errors

`kflgre = 1`	-	OASIS could not read array *pfield* from unit number *knulre*.

Synopsis

```
call locrint ( cdfldn, kfield, kdimax, knulre, kflgre)
```

Arguments

IN

`cdfldn`	-	character string locator (character)
`kdimax`	-	dimension of field to be read (integer)
`knulre`	-	logical unit to be read (integer)

OUT

`kfield`	-	field array (integer 1D)
`kflgre`	-	error status flag (integer)

Discussion

`locrint` reads in a character string followed by an integer array from logical unit *knulre* with binary format. The character string is used as a locator during the search.

Errors

`kflgre = 1`	-	OASIS could not read array *kfield* from unit number *knulre*.

Synopsis

```
call locwrint ( cdfldn, kfield, kdimax, knulre, kflgre)
```

Arguments

IN

`cdfldn`	-	character string locator (character)
`kdimax`	-	dimension of field to be writen (integer)
`knulre`	-	logical unit to be written (integer)
`kfield`	-	field array (integer 1D)

OUT

`kflgre`	-	error status flag (integer)

Discussion

`locwrint` writes a character string followed by an integer array on logical unit *knulre* with binary format. The character string is used as a locator during the search by routine similar to the `locrint` routine.

Errors

`kflgre = 1`	-	OASIS could not write array *kfield* on unit number *knulre*.

Synopsis

```
call locwrite ( cdfldn, pfield, kdimax, knulre, kflgre)
```

Arguments

IN

`cdfldn`	-	character string locator (character)
`kdimax`	-	dimension of field to be writen (integer)
`knulre`	-	logical unit to be written (integer)
`pfield`	-	field array (real 1D)

OUT

`kflgre`	-	error status flag (integer)

Discussion

`locwrite` writes a character string followed by a real array on logical unit *knulre* with binary format. The character string is used as a locator during the search by routine similar to the `locread` routine.

Errors

`kflgre = 1`	-	OASIS could not write array *pfield* on unit number *knulre*.

masq — Mask field

Synopsis

```
call masq ( pfild, ksize, pmask, kmask)
```

Arguments

In

`pfild`	-	field to be masked (real 1D)
`ksize`	-	array size (integer)
`pmask`	-	mask value given from input (real)
`kmask`	-	mask array: 0 -¿ ocean , 1 -¿ land (integer 1D)

Out

`pfild`	-	field masked (real 1D)

Discussion

`masq` is used before the analysis EXTRAP. A given value is assigned to all land points so they can be detected by routine `extrap`.

Errors

No error code for this routine.

Synopsis

```
call modsgc
```

Arguments

IN

`None`

OUT

`None`

Discussion

`modsgc` modifies the SIGCLD signal handler in the PIPE case. This is performed at the end of the time loop when the last iteration has been done. The new OASIS behavior upon reception of SIGCLD is to ignore it as it means the models have finished their simulations.

Errors

No error code for this routine.

Synopsis

```
call mozaic ( pfldn, ksizn, pfldo, ksizo,
              cdfic, kunit, cdgrd, knumb,
              pwork, kwork, knbor)
```

Arguments

IN

- `pfldo` - field on source grid (real 1D)
- `ksizn` - size of field array on target grid (integer)
- `ksizo` - size of field array on source grid (integer)
- `kunit` - logical unit numbers for mapping file (integer)
- `cdfic` - filename for mapping data (character)
- `cdgrd` - locator to read mapping data in cdfic (character)
- `knumb` - mapping dataset identification number (integer)
- `pwork` - work array to read mapping weights (real 1D)
- `kwork` - work array to read mapping array (integer 1D)
- `knbor` - maximum number of overlapped neighbors on source grid for a given target grid-square (integer)

OUT

- `pfldn` - field on target grid (real 1D)

Discussion

`mozaic` performs field mapping from a coarse grid (CG) to a fine grid (FG). In order to perform this mapping, the routine reads some data on a file whose name is defined through the input file (see description of MOZAIC analysis). The data file contains, for each FG grid-square, the adresses of the CG grid-squares having a non-zero intersection. It contains also the weights associated to each CG grid-square, i.e the fraction of the FG grid-square which is overlapped. If F_j is a coupling field defined on a CG and ϕ_i the corresponding mapped field on a FG, we have: $\phi_i = \sum_{j=1}^{n} \beta_{ij} F_j$ where β_{ij} represents the fraction of the FG grid-square i overlapped by the CG grid-square j and n the number of overlapped CG grid-squares. This routine is usually used with the `subgrid` routine to perform a conservative interpolation to first order. Note that it can also be used as the identity transformation if one has identical source and target grids. In this case OASIS can be used just to synchronize the exchanges.

Errors

No error code for this routine.

Synopsis

```
i = nextbl ( cdstr, kistr, knstr)
```

Arguments

IN

`cdstr` - field identificators array (character 1D)
`kistr` - initial search position within the string (integer)
`knstr` - final search position within the string (integer)

OUT

`None`

Discussion

`nextbl` performs the branching for additional analysis after

Errors

No error code for this routine.

Synopsis

```
ii = nextch ( cdstr, kistr, knstr)
```

Arguments

IN

`cdstr`	-	string to be searched (character 1D)
`kistr`	-	initial search position within the string (integer)
`knstr`	-	final search position within the string (integer)

OUT

`None`

Discussion

`nextch` finds the first non-blank character in a character string.

Errors

No error code for this routine.

Synopsis

```
call parse ( cdone, cdtwo, knumb, klen, kleng)
```

Arguments

IN

`cdone`	-	line to be parsed (character 1D)
`knumb`	-	rank within the line of the extracted string (integer)
`klen`	-	length of the input line (integer)

OUT

`cdtwo`	-	extracted character string (character 1D)
`kleng`	-	length of the extracted string (integer)

Discussion

`parse` finds the knumb'th word in string cdone and puts it in cdtwo. A word is defined as a continuous set of non-blanks characters.

Errors

`kleng = -1` - there is less than knumb strings within character line cdone.

Synopsis

```
call postpro ( kindex, kfield)
```

Arguments

IN

`kindex`	-	field identificators array (integer 1D)
`kfield`	-	current number of active fields (integer)

OUT

`None`

Discussion

`postpro` does the field postprocessing, after the interpolation and subsequent analysis (see routine `cookart`). The postprocessing includes routines `reverse` and `glored`. The internal structure of `postpro` is similar to that of the other control routines: the external loop is over the current active fields while the internal one is over the total number of analysis for each field. So the fields are entirely preprocessed one by one. Note that the GLORED analysis must be performed after the REVERSE analysis as routine `reverse` cannot handle fields on a reduced gaussian grid. Within the GLORED operation, OASIS does the following steps: to account for the different masks between the reduced and complete gaussian grid, one has to extrapolate the field over land points from sea point values for the complete gaussian grid. In the mask stage of the extrapolation, a check is performed on the field ordering to see if it is the same as the OASIS convention. If not, as the REVERSE operation has already been performed, one has to switch the mask ordering for the extrapolation stage. Once the extrapolation has been performed, the routine `glored` is called. Note that the parameter values used in the call to routine `extrap` are specified in `postpro`.

Errors

No error code for this routine.

Synopsis

```
call prcout ( cdtext, cdstring, kstyle)
```

Arguments

IN

`cdtext`	-	character string to be printed (character 1D)
`cdstring`	-	character variable to be printed (character 1D)
`kstyle`	-	printing style (integer)

OUT

`None`

Discussion

`prcout` prints out character string and one character variable.

Errors

No error code for this routine.

Synopsis

```
call preproc ( kindex, kfield)
```

Arguments

IN

`kindex`	-	field identificators array (integer 1D)
`kfield`	-	current number of active fields (integer)

OUT

`None`

Discussion

`preproc` does the branching to the different preprocessing routines. The preprocessing includes routines `masq`, `invert`, `correct`, `extrap` and `redglo`. The internal structure of `preproc` is similar to that of the other control routines: the external loop is over the current active fields while the internal one is over the total number of analysis for each field. So the fields are entirely preprocessed one by one. Note that extrapolation (`extrap`) must always be preceded by masking (`masq`). Also, if routine `redglo` is called (i.e if one asks for the REDGLO analysis), it must preceed the call to routine `invert` (i.e the INVERT analysis). Last, if both INVERT and EXTRAP(with MASK) are required, the call to `invert` must preceed those of `masq` and `extrap`.

Errors

No error code for this routine.

Synopsis

```
call prtout ( cdtext, kvalue, kstyle)
```

Arguments

IN

`cdtext`	-	character string to be printed (character 1D)
`kvalue`	-	integer variable to be printed (integer)
`kstyle`	-	printing style (integer)

OUT

`None`

Discussion

`prtout` prints out a character string and one integer value.

Errors

No error code for this routine.

Synopsis

```
call redglo ( pzgg, pzgr, kredu, kinip, klon, klat,
              kmask, kland, ktronca, pmask)
```

Arguments

IN

`pzgg`	-	field on reduced grid (real 2D)
`kredu`	-	number of points on reduced grid (integer)
`kinip`	-	number of longitudes per latitude value (integer 1D)
`klon`	-	number of longitudes of global grid (integer)
`klat`	-	number of latitudes of both grids (integer)
`pmask`	-	reduced grid mask value (real)
`ktronca`	-	truncature of gaussian grid (integer)
`pzgr`	-	work array to store field on reduced grid (real 1D)
`kland`	-	number of masked points per latitude on reduced grid (integer 1D)

OUT

`pzgg`	-	field on global grid (real 2D)
`kmask`	-	mask on reduced grid (integer 1D)

Discussion

`redglo` does the interpolation from a reduced gaussian grid to a complete one. The interpolation is linear by latitude circle. As the interpolation per se does not account for the sea-land mask, one has first to extrapolate the land points values from the sea values on the reduced grid. The data regarding the reduced grid (i.e number of longitudes per latitude circle) is given in routine `blkdata`. The reduced grid mask is read within `redglo`. Note that it does not follow the OASIS convention for latitude ordering (it has the same convention as the spectral model which is using it).

Errors

No error code for this routine.

Synopsis

```
call reset
```

Arguments

IN

`None`

OUT

`None`

Discussion

`reset` reinitializes to zero the field macro arrays after each iteration. This is a basic security action to avoid any wrong use of field values from the previous iteration.

Errors

No error code for this routine.

Synopsis

```
call reverse ( pfild ,kxlon ,kylat, cdxord, cdyord)
```

Arguments

IN

`pfild`	-	field to be reordered (real 2D)
`kxlon`	-	number of longitudes (integer)
`kxlon`	-	number of longitudes (integer)
`cdxord`	-	longitude ordering (character)
`cdyord`	-	latitude ordering (character)

OUT

`pfild`	-	field reordered (real 2D)

Discussion

`reverse` reorders field pfild according to arguments cdxord and cdyord. The OASIS convention is to go from South to North and from Greenwhich to the East. So if some fields (i.e models) have a different convention, these need to be reversed before being written out by OASIS.

Errors

No error code for this routine.

Synopsis

```
call revmsk ( kfild ,kxlon ,kylat, cdxord, cdyord)
```

Arguments

IN

- `kfild` - mask to be reordered (integer 2D)
- `kxlon` - number of longitudes (integer)
- `kxlon` - number of longitudes (integer)
- `cdxord` - longitude ordering (character)
- `cdyord` - latitude ordering (character)

OUT

- `kfild` - field reordered (integer 2D)

Discussion

`revmsk` reorders mask kfild according to arguments cdxord and cdyord. This routine is used within the GLORED analysis if the field(model) to be treated has a convention which differs from that of OASIS(the OASIS convention is to go from South to North and from Greenwhich to the East). See routine `postpro` for details about the use of `revmsk`.

Errors

No error code for this routine.

Synopsis

```
call sqdis ( px1, py1, px2, py2)
```

Arguments

IN

- `px1` - longitude of first point (real)
- `py1` - latitude of first point (real)
- `px2` - longitude of second point (real)
- `py2` - latitude of second point (real)

OUT

`None`

Discussion

`sqdis` calculates the distance squared between 2 points on a spheric grid.

Errors

The points coordinates must be in degrees.

Synopsis

```
zs = ssumr ( pa, kna)
```

Arguments

IN

`pa`	-	array to be summed (real 1D)
`kna`	-	array dimension (integer)

OUT

`None`

Discussion

`ssumr` sums up the elements of a real array.

Errors

No error code for this routine.

Synopsis

```
call subgrid ( pfild, ksize, pcoar, pfine, pdqdt)
```

Arguments

IN

`pfild`	-	initial field on target grid (real 1D)
`ksize`	-	size of field array (integer)
`pcoar`	-	coarse grid additional field (real 1D)
`pfine`	-	fine grid additional field (real 1D)
`pdqdt`	-	coarse grid coupling ratio (real 1D)

OUT

`pfild`	-	corrected field on target grid (real 1D)

Discussion

`subgrid` can be used if one grid is always finer than the other one. This interpolation is rigorously conservative to first order if one can treat exactly the sea-land mask mismatch between the two grids and if information is exchanged at each timestep of the models. One performs a first-order Taylor expansion of the field on the coarse grid relatively to a state variable (for instance SST if the field is the total heat flux): $\Phi_i = F + \partial F / \partial T (T - T_i)$ where Φ_i (F) is the heat flux on the fine(coarse) grid (i.e here the ocean(atmos) grid), T_i (T) the SST on the fine(coarse) grid and $\partial F / \partial T$ the derivative of the flux versus SST on the coarse grid. The `subgrid` routine just performs the algebric operation which implies that all the involved fields are already on the same grid (the fine one). This means that all the fields coming from the coarse grid habe been interpolated before BUT T which comes from the interpolation of SST from the fine to the coarse grid. So within `subgrid`, one has to map the coarse grid SST to the fine grid as was done previously for the other coarse grid fields (see routine `mozaic`.

Errors

No error code for this routine.

Synopsis

```
call szero ( pa, kna)
```

Arguments

IN

`pa` - array to be zeroed (real 1D)
`kna` - array dimension (integer)

OUT

`pa` - array filled up with zeros (real 1D)

Discussion

`szero` zeroes real array pa from element 1 to element kna.

Errors

No error code for this routine.

Synopsis

```
call updtim ( kiter)
```

Arguments

IN

`kiter` - iteration number (integer)

OUT

`None`

Discussion

`updtim` increments the date since run start. It takes into account leap years. It prints out the exact date (including hours) for the coupler and gets calendar data necessary for reading climatological or interannual files. It is called at the beginning of the time loop. Note it is not incremented for the first iteration as its number is zero. This is consistent with the fact that OASIS has first to interpolate boundary conditions for the other models before the simulation really starts.

Errors

No error code for this routine.

Synopsis

```
call waitpc
```

Arguments

IN

`None`

OUT

`None`

Discussion

`waitpc` halts OASIS as long as the other models haven't finished their own simulation. In the PIPE case, this is necessary given the child-parent relationship between the models being coupled and the coupler. This is done thru either the wait CRAY function for the PIPE option or the routine `CLIM_Quit` in the CLIM case.

Errors

Error messages are issued if problems occur.

Synopsis

```
call PIPE_Define (cdfread, cdfwrite, cdpread, cdpwrite, kinfo )
```

Arguments

IN

`cdfread` - alias filename for read pipe (char string)
`cdfwrite` - alias filename for write pipe (char string)
`cdpread` - symbolic reading pipe name (char string)
`cdpwrite` - symbolic writing pipe name (char string)

OUT

`kinfo` - error status (integer)

Discussion

The `PIPE_Define` routine associates a named pipe (FIFO) to a symbolic name and alias it to a filename. Both names cannot exceed eight characters. Each exchanged field has two pipes associated to it: one to read messages from model A, one to write to model B. The pipes are not precisely used to transfer the field data but to synchronize the models with the coupler. An *assign* command is used within the fortran to alias a filename to the pipe which can be then used through WRITE and READ statements. Then, a *mknod* command is used to create the pipe. These two functions work on any CRAY machines (non MPP). As the pipes have also to be created in the models, the error status for the *mknod* command can be non zero if the pipes have already been opened in the models. This has no consequence. So the `kinfo` argument gathers only the error codes relative to the *assign* calls.

Errors

`kinfo = 0` - The *assign* calls are executed correctly.
`kinfo = N` - M *assign* calls failed. Examination of the *cplout* file shows where the problem has occured.

Synopsis

```
call PIPE_Init (cdjobnam, cdmodnam, kmodel, kinfo)
```

Arguments

IN

`cdjobnam`	-	experiment name (char string)
`cdmodnam`	-	remote model name (char string)
`kmodel`	-	number of models (integer)

OUT

`kinfo`	-	error status (integer)

Discussion

`PIPE_Init` initializes communication between the remote models and the coupler. A pipe is created between the coupler and each model, as in `PIPE_Define`, in order to exchange timestep data. See `PIPE_Define` for a description of the pipe creation. Furthermore two calls to function *fsigctl* are performed. The first one catches all occurences of signal SIGCLD which says to the coupler that one of its child processes has died off unexpectedly. The coupler then executes routine `ferror` which terminates the simulation. The second one catches all occurences of signal SIGFPE which means that when the coupler encounters a floating point error, it terminates in a clean way by executing routine `getfpe`.

Errors

`kinfo = 0`	-	The *assign* calls are executed correctly.
`kinfo = N`	-	M *assign* calls failed. Examination of the *cplout* file shows where the problem has occured.

Synopsis

```
call PIPE_Send (cdpname, kiter)
```

Arguments

IN

`cdpname`	-	pipe alias filename (char string)
`kiter`	-	iteration number (integer)

OUT

`None`

Discussion

The `PIPE_Send` routine writes into the pipe `cdpname` the data `kiter` which is the current coupler iteration. The aim of this routine is to give to the "outside world" the following information: the field associated to pipe `cdpname` has been treated by the coupler and written on the transfer file. It is then ready to be used by the "outside world". Jointly with `PIPE_Recv` , `PIPE_Send` does the synchronization with the remote models.

Errors

No error code for this routine.

PIPE_Recv

Receive message on pipe

Synopsis

```
call PIPE_Recv (cdpname, kiter)
```

Arguments

IN

`cdpname`	-	pipe alias filename (char string)
`kiter`	-	iteration number (integer)

OUT

`None`

Discussion

The `PIPE_Recv` routine waits for data coming on the pipe `cdpname` from the "outside world" at the coupler iteration `kiter`. The aim of this routine is to pause the coupler while the field associated to pipe `cdpname` is being calculated in the "outside world". As soon as the coupler does receive the message, it goes on to the following instructions. Jointly with `PIPE_Send` , `PIPE_Recv` does the synchronization with the remote models.

Errors

No error code for this routine.

Synopsis

```
call PIPE_Stepi (cdmodnam, kmodel, kstep, kfcpl, kdt, kinfo)
```

Arguments

IN

`cdmodnam` - name of the remote model (char string)

OUT

`kstep`	-	number of time steps (integer)
`kfcpl`	-	frequency of coupling in time steps (integer)
`kdt`	-	length of a time step in seconds (integer)
`kinfo`	-	error status (integer)

Discussion

`PIPE_Stepi` returns timestepping information from the remote model `cdmodnam`. First, it sends out timestepping data to the model `cdmodnam` by writing on pipe `cpwnam` . Then it reads in the data from model `cdmodnam` on pipe `cprnam` . A check is performed on the process-id of the remote model to be sure it is within the system physical limits.

Errors

`kinfo = -1` - The model `cdmodnam` has a wrong process-id number.

Synopsis

```
call CLIM_Define ( cport,kinout,ktype,kparal,kinfo )
```

Arguments

IN

`cport` - symbolic port name (char string)
`kinout` - port status (integer)
`ktype` - port data type (integer)
`kparal` - parallel decomposition (integer array)

OUT

`kinfo` - error status (integer)

Discussion

The `CLIM_Define` function associates a port to a symbolic name (no more than `CLIM_Clength` characters) and some specifications:

The port status `kinout` can be:
`CLIM_In`
`CLIM_Out`
`CLIM_InOut`

Data types recognized by CLIM are :
`CLIM_Integer` : implicit 32-bits integers.
`CLIM_Real` : 32-bits floats.
`CLIM_Double` : 64-bits floats.

The array `kparal` of size `CLIM_ParSize` describes the parallel decomposition associated to this port. It should always be filled using pre-defined constants associated to several strategies as following:

- `kparal ( CLIM_Strategy ) = CLIM_Serial`
  It's the simplest strategy when the calling model views the entire data field. There is no data distribution, and when importing through this port, the model expects to retrieve the complete field. Note, that this doesn't mean that the field is coming from serial models, but rather that the calling model at least is serial. Then, there is only one argument to give for describing the

field - its length:
`kparal ( CLIM_Length )` = *length-of-the-field*

- `kparal ( CLIM_Strategy ) = CLIM_Apple`
  The calling model handles a contiguous set of data, as a part of a global one splitted over several processes. The strip is described by its length and its offset in the global field (the starting point minus one, see the Appendice IIa for an illustrated example):
  `kparal ( CLIM_Length )` = *length-of-the-strip*
  `kparal ( CLIM_Offset )` = *offset-in-the-global-field*

- `kparal ( CLIM_Strategy ) = CLIM_Box`
  The calling model handles a rectangular subset of the global field, described with 2D parameters (see the Appendice IIb for an illustrated example):
  `kparal ( CLIM_Offset )` = *offset-in-the-global-field*
  `kparal ( CLIM_SizeX )` = *size-in-the-first-direction*
  `kparal ( CLIM_SizeY )` = *size-in-the-second-direction*
  `kparal ( CLIM_LdX )` = *leading-dimension-of-the-global-field*

- `kparal ( CLIM_Strategy ) = CLIM_Orange`
  This is the most general parallel distribution of data since the user describes, segment per segment, the subset of data handled by the calling process. Thus, the following informations should be provided in the `kparal` array (see the Appendice IIc for an illustrated example):
  `kparal ( CLIM_Segments )` = *number-of-segments*
  `kparal ( CLIM_Segments +1 )` = *offset-of-segment#1*
  `kparal ( CLIM_Segments +2 )` = *length-of-segment#1*
  `kparal ( CLIM_Segments +3 )` = *offset-of-segment#3*
  and so on...

Example

```
c
      integer myid, npar ( CLIM_ParSize ), info
      ...
c
c*    global field and process numbers:
c            ------------
c            |1111122222|
c            |2222211111|
c            ------------
c
      npar ( CLIM_Strategy ) = CLIM_Orange
      if ( myid.eq.1 )
         npar ( CLIM_Segments ) = 2
         npar ( CLIM_Segments + 1 ) = 0
         npar ( CLIM_Segments + 2 ) = 5
```

```
            npar ( CLIM_Segments + 3 ) = 15
            npar ( CLIM_Segments + 2 ) = 5
         else
            npar ( CLIM_Segments ) = 1
            npar ( CLIM_Segments + 1 ) = 5
            npar ( CLIM_Segments + 2 ) = 10
         endif
c
         call CLIM_Define ( 'port-1-2', CLIM_Out, CLIM_Real, npar, info )
         ...
```

Errors

`CLIM_DoubleDef`	-	The specified port name is already used with the same port status.
`CLIM_BadType`	-	the `ktype` argument is not recognized.

Synopsis

`call CLIM_Delta ( cnam,pdt,pde,kinfo )`

Arguments

IN

`cnam`	-	remote model name (char string)

OUT

`pdt`	-	difference with the remote model's wall clock in seconds (double real)
`pde`	-	elapsed time needed to measure pdt in seconds (double real)
`kinfo`	-	error status (integer)

Discussion

`CLIM_Delta` returns a difference between two wall clocks. Therefore if model a asks for his "delta time" $\Delta(b)$ with model b which had done the measurement T_b, the expression $T_b + \Delta(b)$ is comparable to a time measurement done by model a.

The measurement of $\Delta(b)$ is obtained through several *ping-pong* exchanges. The $\Delta(b)$ value selected is associated to the smallest elapsed time measured on the local model for performing a *ping-pong*. This value is returned in the `pde` argument for information purpose on the accuracy of the `pdt` value.

Example

```
c
      integer info
      real*8  dt, de, t1, t2
      ...
      call CLIM_Delta ( 'MODEL2', dt, de, info )
      ...
      call CLIM_Time ( t1, info )
      call CLIM_Import ( 'Time MODEL2', CLIM_Void, t2, info )
      write(6,*) 'Homogeneous timing: ', t2+dt-t1
```

Errors

`CLIM_BadName`	-	The `cnam` model was not found.

Synopsis

```
call CLIM_Export ( cport,kstep,pfield,kinfo )
```

Arguments

IN

`cport`	-	symbolic port name (char string)
`kstep`	-	time step (integer)
`pfield`	-	data buffer (handle)

OUT

`kinfo`	-	error status (integer)

Discussion

The `CLIM_Export` functions delivers to the "outside world" the data starting at the address `pfield` on the port `cport`. If this action is not related to an iterative scheme, `kstep` can be set to `CLIM_Void`.

Example

```
...
call CLIM_Export ( 'A Great Example', CLIM_Void, null, info )
```

Errors

`CLIM_BadPort`	-	The specified port was not found.
`CLIM_NotStep`	-	`kstep` is not a coupling step of the model as it was defined by the `CLIM_Init` call.
`CLIM_IncStep`	-	`kstep` is an incompatible coupling step for the matching models.
`CLIM_NoTask`	-	One of the remote processes associated to this port doesn't exist any more.
`CLIM_BadTaskId`	-	One of the PVM task identifier describing a model associated to this port doesn't exist any more.
`CLIM_Down`	-	The Parallel Virtual Machine is partially or completly down. You'd better stop.
`CLIM_InitBuff`	-	The process is unable to get a PVM buffer for sending the message (see the `pvmfinitsend` error code).
`CLIM_Pack`	-	an error occured when packing the message (see the `pvmfpack` error code.
`CLIM_Pvm`	-	A trouble occured when sending the message (see the `pvmfsend` error code.

Synopsis

```
call CLIM_Import ( cport,kstep,pfield,kinfo )
```

Arguments

IN

`cport`	-	symbolic port name (char string)
`kstep`	-	time step (integer)

OUT

`pfield`	-	data buffer (handle)
`kinfo`	-	error status (integer)

Discussion

The `CLIM_Import` function waits for data coming on the port `cport` from the "outside world". The data received is stored in the buffer `pfield`. If this action is not related to an iterative scheme, `kstep` can be set to `CLIM_Void`

Example

```
...
call CLIM_Import ( 'One More', CLIM_Void, anything, info )
```

Errors

`CLIM_BadPort`	-	The specified port was not found.
`CLIM_NotStep`	-	`kstep` is not a coupling step of the model as it was defined by the `CLIM_Init` call.
`CLIM_IncStep`	-	`kstep` is an incompatible coupling step for the matching models.
`CLIM_NoTask`	-	One of the remote processes associated to this port doesn't exist any more.
`CLIM_BadTaskId`	-	One of the PVM task identifier describing a model associated to this port doesn't exist any more.
`CLIM_Down`	-	The Parallel Virtual Machine is partially or completly down. You'd better stop.
`CLIM_UnPack`	-	an error occured when unpacking the message (see the `pvmfunpack` error code.
`CLIM_Pvm`	-	A trouble occured when sending the message (see the `pvmfsend` error code.
`CLIM_TimeOut`	-	A time out occured when receiving the message.

Synopsis

```
call CLIM_Init ( cexp,cnam,kno,ktrout,
                 kstep,kfcpl,kdt,
                 ktiret,ktiogp,ktiout,kinfo )
```

Arguments

IN

`cexp`	-	name of the experiment (char string)
`cnam`	-	name of the model (char string)
`kno`	-	number of models in the experiment (integer)
`ktrout`	-	unit number of the trace file (integer)
`kstep`	-	number of time steps (integer)
`kfcpl`	-	frequency of coupling in time steps (integer)
`kdt`	-	length of a time step in seconds (integer)
`ktiret`	-	time in seconds between two checks when gathering models (integer)
`ktiogp`	-	time out in seconds when gathering models (integer)
`ktiout`	-	time out in seconds when receiving a message (integer)

OUT

`kinfo`	-	error status (integer)

Discussion

`CLIM_Init` should be the first call to the CLIM library. It performs miscellaneous initiliazations:

- The `CLIM_Init` call registers the model `cnam` in the experiment `cexp` which is supposed to include `kno` different processes.

- `ktrout` specifies an unit number for the CLIM trace file associated to the calling process.

- If the model is not related to an iterative time step scheme, the arguments `kstep`, `kfcpl` and `kdt` can be set to `CLIM_Void`. However, it is safer to create a artificial time stepping (by counting the messages for exemple).

- In order to save CPU time when waiting for models to join the experiment (this action is performed by CLIM_Start), a process is sleeping `ktiret` seconds between two checks, and exit after `ktiogp` seconds if some model is missing. For all the remaining communications, the calling process will never wait more than `ktiout` seconds when receiving a message.

Example

```
c*    A climatic experiment
c       ... the atmospheric point of view:
c
      call CLIM_Init ( 'Climat', 'Atmos', 2, 99,
     +                 480, 96, 900, 60, 3600, 3600, info )
c
c     means: 2 models a "Climat" experiment,
c            Atmos will perform 480 time steps of 900 seconds each
c            and expect coupled fields every day (96 time steps).
c            A check for the other model will occur every minute when
c            starting the application,
c            and both time out values are set to one hour.
```

Errors

`CLIM_FirstCall`	-	An error occured when calling `pvmfmytid`. The Parallel Virtual Machine may be badly configured.
`CLIM_PbRoute`	-	An error occured when setting the route option (see the `pvmfsetopt` error code).

Synopsis

```
call CLIM_Quit ( kstop,kinfo )
```

Arguments

IN

`kstop` - exit code for PVM (integer)

OUT

`kinfo` - error status (integer)

Discussion

`CLIM_Quit` properly exits from the CLIM experiment according to the `kstop` value:

- `CLIM_ContPvm`: the calling process needs to stay in the Parallel Virtual Machine.
- `CLIM_StopPvm`: the `CLIM_Quit` call also exits from the Parallel Virtual Machine.

Before leaving the eperiment, the models are synchronized.

Example

```
c*    time to leave work...
c
      call CLIM_Quit ( CLIM_StopPvm )
```

Errors

`CLIM_Group`	-	An error occured when calling a PVM group function (see the trace file).
`CLIM_PvmExit`	-	An error occured when exiting from PVM.

Synopsis

```
call CLIM_Reset ( kinfo )
```

Arguments

IN

none

OUT

`kinfo` - error status (integer)

Discussion

Calling `CLIM_Reset` allows the user to dynamically restart a new coupled application in the same running executable. All the port and model definitions are removed and a new sequence of `CLIM_Init`-`CLIM_Define`- `CLIM_Start` can be started for a new experiment. A global barrier is performed in the current experiment before leaving. The user should give a new unit number for the trace file of the next experiment.

Example

```
c
      if ( info.ne.CLIM_Ok ) then
         call CLIM_Reset ( info )
c        re-start the experiment !
         ...
      endif
```

Errors

`CLIM_Group` - An error occured when calling a PVM group function (see the trace file).

Synopsis

```
call CLIM_Start ( kmxtag,kinfo )
```

Arguments

IN

`none`

OUT

`kmxtag`	-	Maximum tag number to use
`kinfo`	-	error status (integer)

Discussion

`CLIM_Start` is the last part of the initialization scheme after `CLIM_Init` and the ports definitions (`CLIM_Define`). It performs the gathering of the models using the same experiment name, and exchanges ports definitions in order to create all the possible *links*.

In order to distinguish the use of PVM by CLIM from the rest of the application, a maximum tag number is provided. Using tag numbers lower than `kmxtag` guaranties that the calling process will not interfere with internal CLIM messages.

Example

```
c*    Init and ports definitions are done, so let's start:
c
      call CLIM_Start ( imaxtag, info )
```

Errors

`CLIM_Group`	-	An error occured when calling a PVM group function (see trace file).
`CLIM_TimeOut`	-	A time out occured either when gathering models or when receiving models' internal descriptions.
`CLIM_IncSize`	-	A link cannot be created because of incompatible port sizes.

Synopsis

```
call CLIM_Stepi ( cnam,kstep,kfcpl,kdt,kinfo )
```

Arguments

IN

`cnam` - name of the remote model (char string)

OUT

`kstep` - number of time steps (integer)
`kfcpl` - frequency of coupling in time steps (integer)
`kdt` - length of a time step in seconds (integer)
`kinfo` - error status (integer)

Discussion

`CLIM_Stepi` returns time steps informations from the `cnam` model.

Example

```
...
call CLIM_Stepi ( 'Remote Model', nb_steps, fr_steps, dt_steps, info )
```

Errors

`CLIM_BadName` - The `cnam` model was not found.

Synopsis

```
call CLIM_Time ( pt,kinfo )
```

Arguments

IN

`pt` - absolute time in seconds (double real)

OUT

`kinfo` - error status (integer)

Discussion

`CLIM_Time` returns an absolute elapsed time in seconds. An elapsed time in the application should be the difference between two calls to `CLIM_Time` (possibly using `CLIM_Delta` if one measurement is coming from a remote host).

Example

```
c
      real*8 t0
c
      call CLIM_Time ( t0, info )
      write(6,*) 'Absolute time in seconds: ', t0
```

Errors

Synopsis

```
call CLIM_Trace ( ksend,krecv,kinfo )
```

Arguments

IN

OUT

`ksend`	-	number of bytes already sent (integer)
`krecv`	-	number of bytes already received (integer)
`kinfo`	-	error status (integer)

Discussion

`CLIM_Trace` returns some crude statistics about the number of bytes sent and received by the calling process in the CLIM experiment.

Example

```
c
      integer nb_send, nb_recv, info
c
      call CLIM_Trace ( nb_send, nb_recv, info )
      write(6,*) 'I sent ', nb_send, ' bytes'
      write(6,*) 'and received ', nb_recv
```

Errors

Synopsis

`call CLIM_Wait ( cport,kstep,kinfo )`

Arguments

IN

none

OUT

`cport`	-	symbolic port name (char string)
`kstep`	-	time step (integer)
`kinfo`	-	error status (integer)

Discussion

`CLIM_Wait` is probing for any message to the calling process. The user should be aware that this mechanism is not CPU free. If a message is received before the time out, the associated port name and time step are returned. It could happenned than the message doesn't belong to the CLIM experiment. Therefore the calling process should read (receive) this message and then call back the `CLIM_Wait` function in order to check for a CLIM message.

Example

```
      character*(CLIM_Clength) cport
      integer istep, info
c
      call CLIM_Wait ( cport, istep, info )
      if ( info.eq.CLIM_NotClim ) then
         call RecvNext
      else
         call CLIM_Import ( cport, istep, buff, info )
      endif
```

Errors

`CLIM_Down`	-	The Parallel Virtual Machine is partially or completly down. You'd better stop.
`CLIM_TimeOut`	-	No message present before the time limit.
`CLIM_NotClim`	-	The first messqge queued is not a "Clim" message, but a private message of the parallel application.

References

[1] J. Cahouet and D. Beaucourt. the cathode project. Technical report, EdF/DER, 1994.

[2] C. Cassou, P. Noyret, E. Sevault, O. Thual, L. Terray, D. Beaucourt, and M. Imbard. Distributed ocean-atmosphere modelling and sensitivity to the coupling flux precision: the cathode project. *Monthly Weather Review*, page To be submitted, 1996.

[3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM 3 User's Guide and Reference Manual TM-12187*, May 1994.

[4] J. Gibson. the doctor norm. Technical report, ECMWF, 1986.

[5] E. Guilyardi, G. Madec, L. Terray, M. Déqué, M. Pontaud, M. Imbard, D. Stephenson, M.-A. Filiberti, D. Cariolle, P. Delecluse, and O. Thual. Simulation couplée océan-atmosphère de la variabilité du climat. *C.R. Acad. Sci. Paris*, t. 320, série IIa:683–690, 1995.

[6] R.C. Pacanowski, K. Dixon, and A. Rosati. Gfdl modular interface for coupled air sea applications (micasa). Technical report, GFDL, 1991.

[7] M. Pontaud, L. Terray, E. Guilyardi, E. Sevault, D. B. Stephenson, and O. Thual. Coupled ocean-atmosphere modelling - computing and scientific aspects. In *2nd UNAM-CRAY supercomputing conference, Numerical simulations in the environmental and earth sciences, 21-24 june 1995*, page accepted, Mexico-city, Mexico, 1995.

[8] E. Sevault, P. Noyret, and L. Terray. Clim 1.2 user guide and reference manual. Technical Report TR/CGMC/95-47, CERFACS, 1995.

[9] E. Sevault, P. Noyret, L. Terray, and O. Thual. Distributed and coupled ocean-atmosphere modelling. In *In Proceedings of the Sixth ECMWF Workshop on the Use of Parallel Processors in Meteorology, ECMWF, November 21-25*, page In press, Shinfield Park, Reading UK, 1994. ECMWF.

[10] L. Terray and O. Thual. Oasis: le couplage océan-atmosphère. *La Météorologie*, 10:50–61, 1995.

[11] L. Terray, O. Thual, S. Belamari, M. Déqué, P. Dandin, C. Lévy, and P. Delecluse. Climatology and interannual variability simulated by the arpege-opa model. *Climate Dynamics*, 11:487–505, 1995.