Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

JGOFS Data System Overview

Glenn R. Flierl, MIT
James K.B. Bishop, LDEO
David M. Glover, WHOI
Satish Paranjpe, LDEO

Index

Obtaining the Software

- Installation instructions and tar file generation

Full Document

A PDF version of this document is also available.

Further Documentation

A collection of Postscript documents describing technical details of the system are available here.

back to top

U.S. JGOFS
U.S. Joint Global Ocean Flux Study

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Introduction

Large oceanographic programs such as JGOFS (The Joint Global Ocean Flux Study) require data management systems which enable the exchange and synthesis of extremely diverse and widely spread data sets. We have developed a distributed, object-based data management system for multidisciplinary, multi-institutional programs. It provides the capability for all JGOFS scientists to work with the data without regard for the storage format or for the actual location where the data resides. The approach used yields a powerful and extensible system (in the sense that data manipulation operations are not predefined) for managing and working with data from large scale, on-going field experiments.

In the “object-based” system, user programs obtain data by communicating with a program (the “method”) which can interpret the particular data base. Since the communication protocol is standard and can be passed over a network, user programs can obtain data from any data object anywhere in the system. Data base operations and data transformations are handled by methods which read from one or more data objects, process that information, and write to the user program.

Purpose:

- Permit scientists to use data without concern for storage technique, location, or format
- Networked interchange of data sets
- Access to most recent versions of data sets during experiments
- Handle multidimensional data
- Transmit metadata
- Extensible data manipulation routines
- Usable interactively or from programs

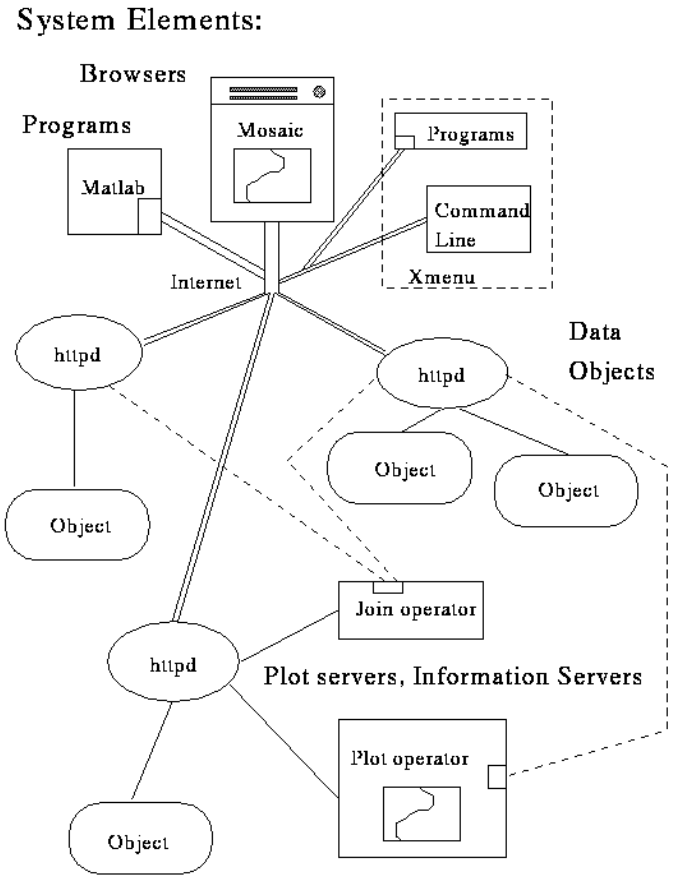
▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Basic Elements:

- **data objects** which receive requests and respond with data
- application programs/interfaces to other software:
  - data can be **imported directly** into packages such as MATLAB
  - **simple listing and plotting programs** (supplied)
  - open/read/close FORTRAN/C **interface**
- a **server** which connects applications to objects



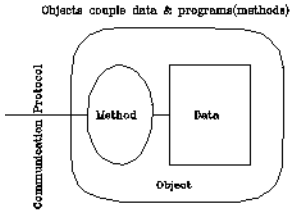
▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Data Objects

Data Objects package together a program (the **translator or method**) and data. User programs never look at the data directly; rather, they communicate with the data object.



Data Objects communicate with a common protocol
--> All data objects present the same appearance to outside (described **here**)
--> Programs can work with any data in the system (**Example**)
Data Objects handle
Projection (subsetting by variable name)
Selection (subsetting by variable values)
--> Can minimize transmission of data
--> Individual objects may have other functions

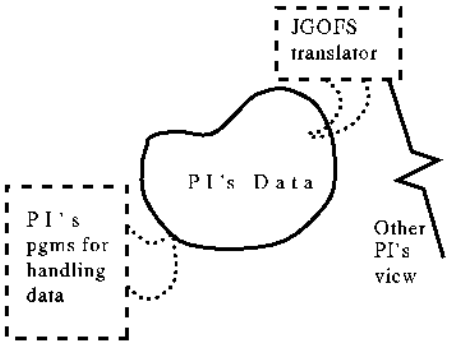
▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Translators / methods

The “translators” (or “methods” in object-based terminology) are **programs** which give other PI's a viewport into a data set. The program both makes the data set visible to the outside world and shields outside users from needing to know the details of where and how the data is stored.



These programs are responsible for

- receiving requests for subselections of the data
- gathering the requested information from the data set
- translating the information into the internal form used for transferring data
- sending the information through the communication line to the process which made the request.

One translator may serve several different data sets -- the translators depend on the format chosen by the PI, but generally not on the information itself, though there can be exceptions.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Data Model - appearance to applications

The JGOFS data model is the critical part of the communications protocol. It includes:

- Comments (text)
- Variable descriptions
  - Name
  - Dimensions for vectors/ matrices/ tensors [not implemented!]
  - Attributes (e.g., units)
  - Hierarchical structure
- Data
  - Strings or numbers
- End of data set indicator

The hierarchical structuring is an important way of organizing many kinds of data. It groups the least rapidly changing variables (e.g., header data), then the next-most rapidly changing information, etc. For example, a hydrographic section might look like

```
leg
year          [lowest (0) level]
month
   station
   lat        [level 1]
   lon
   date
      press
      temp
      sal     [level 2]
      o2
      sigth
```

A current meter mooring might have

```
mooring_id
lat
lon                [level 0]
nominal_depth
start_time
end_time
   time
   u               [level 1]
   v
   temp
```

Often one scans the lower level information first to pick out the desired station or mooring and then retrieves the information only for that subset of the data base.

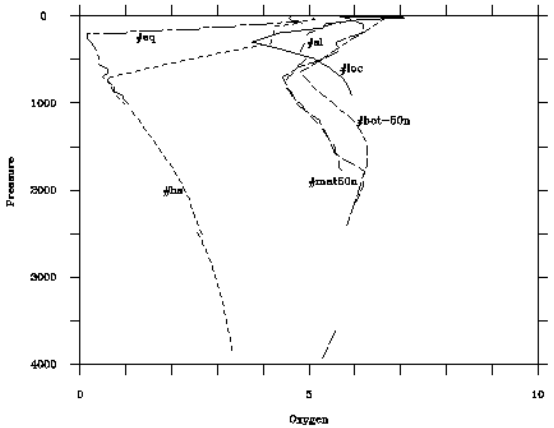
▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Example of Accessing Different Objects

As an example, we show a figure with plots from 6 different data objects



The commands which created the plot are

```
window 0 4000 10 0
   [sets lower left and upper right corner]
axis x 2 Oxygen 5 dd
   [draws x axis with tic mark at 2 units and label at 5 units
    format is two digits]
axis y 500 Pressure 1000 dddd
dash "#loc" o2 press ----------------
   [draws first line using object #loc, x variable "o2",
    y variable "press", and dash pattern indicated, in this
    case solid]
dash "#sl" o2 press ----....----....
dash "#mat50n" o2 press -------..-------..
dash "#hs" o2 press --..--..--..--..
dash "#bot-50n" o2 press ----..--..------..
dash "#eq" o2 press ----------.--..
```

This shows a single program **dash** requesting ``o2'' and ``press'' data from each object and displaying it with the specified dash pattern. The objects are

```
  object  location  machine type  storage method
-------------------------------------------------------------
  #loc      MIT         Sun    flat ASCII file
  #sl       RSMAS       Alpha  scaled binary integers
  #mat50n   MIT         Sun    MATLAB binary floating point
  #hs       WHOI        Sun    multiple files/directories
  #bot-50n  MIT         Sun    multiple files
  #eq       U. Chicago  IRIX   Single file/ multiple stations
```

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Examples of Projection

Projection -- choosing to display only particular variables -- is accomplished by listing the desired variables as the argument to the data object.

objectname(variablename1,variablename2,...)

Thus if we have a data object **hyd** (displayed by the program**list)**

```
list "hyd"
#  wunsch stations 3-5
#  p<1000
======================
    leg,   year,  month
......................        [lowest (0) level data]
      1,     81,      6
======================
station,    lat,    lon
......................        [first level data]
      3,  38.28, -73.53
======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 18.334, 33.570,  5.970, 24.096
 25.000, 12.848, 34.159,  6.990, 25.773
 49.000, 11.070, 34.523,  6.060, 26.394       [second level]
 99.000, 11.093, 35.090,  5.340, 26.831
149.000, 11.906, 35.487,  5.020, 26.990
199.000, 10.819, 35.435,  4.210, 27.152
300.000,  8.293, 35.126,  3.730, 27.334
400.000,  6.363, 35.046,  4.640, 27.546
500.000,  5.724, 35.019,  4.980, 27.608
600.000,  5.031, 34.990,  5.460, 27.670
701.000,  4.633, 34.981,  5.680, 27.710
801.000,  4.515, 34.980,  5.850, 27.722
901.000,  4.376, 34.979,  5.880, 27.737
======================
station,    lat,    lon
......................
      4,  38.19, -73.52
======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 17.516, 33.160,  5.840, 23.981
 25.000, 12.315, 33.958,  7.090, 25.721
 49.000,  9.612, 34.192,  6.020, 26.387
 99.000, 12.095, 35.402,  5.340, 26.887
149.000, 12.407, 35.625,  5.290, 27.000
199.000, 11.287, 35.487,  4.340, 27.108
300.000,  8.059, 35.120,  3.750, 27.365
400.000,  6.606, 35.053,  4.440, 27.520
500.000,  5.467, 34.997,  5.130, 27.622
600.000,  5.003, 34.983,  5.460, 27.667
701.000,  4.680, 34.979,  5.700, 27.702
801.000,  4.489, 34.977,  5.850, 27.722
901.000,  4.374, 34.978,  5.930, 27.737
======================
station,    lat,    lon
......................
      5,  38.16, -73.26
======================
  press,   temp,    sal,     o2,  sigth
-----------------------
```

```
  5.000, 18.382, 33.647,  5.770, 24.143
 25.000, 12.040, 34.196,  6.660, 25.959
 49.000, 11.951, 34.925,  5.510, 26.543
 99.000, 11.914, 35.390,  5.100, 26.912
149.000, 12.045, 35.547,  5.070, 27.010
199.000, 11.976, 35.589,  4.940, 27.057
300.000,  9.425, 35.250,  3.620, 27.251
400.000,  7.003, 35.075,  4.210, 27.483
500.000,  5.827, 35.009,  4.910, 27.589
600.000,  5.252, 34.988,  5.300, 27.643
701.000,  4.845, 34.980,  5.610, 27.684
801.000,  4.635, 34.980,  5.710, 27.709
901.000,  4.444, 34.978,  5.940, 27.729
****** End of object ***
```

then we can select to list some of the variables

```
list "hyd(month,station,press,sigth)"
#  wunsch stations 3-5
#  p<1000
=======================
  month
.......................
      6
=======================
station
.......................
      3
=======================
  press,  sigth
-----------------------
  5.000, 24.096
 25.000, 25.773
 49.000, 26.394
 99.000, 26.831
149.000, 26.990
199.000, 27.152
300.000, 27.334
400.000, 27.546
500.000, 27.608
600.000, 27.670
701.000, 27.710
801.000, 27.722
901.000, 27.737
=======================
station
.......................
      4
=======================
  press,  sigth
-----------------------
  5.000, 23.981
 25.000, 25.721
 49.000, 26.387
 99.000, 26.887
149.000, 27.000
199.000, 27.108
300.000, 27.365
400.000, 27.520
500.000, 27.622
600.000, 27.667
701.000, 27.702
801.000, 27.722
901.000, 27.737
=======================
station
.......................
      5
=======================
  press,  sigth
-----------------------
  5.000, 24.143
 25.000, 25.959
 49.000, 26.543
 99.000, 26.912
149.000, 27.010
199.000, 27.057
300.000, 27.251
400.000, 27.483
500.000, 27.589
```

```
600.000, 27.643
701.000, 27.684
801.000, 27.709
901.000, 27.729
****** End of object ***
```

or

```
list "hyd(station,lat,lon)"
#  wunsch stations 3-5
#  p<1000
======================
......................
======================
station,    lat,    lon
----------------------
      3,  38.28, -73.53
      4,  38.19, -73.52
      5,  38.16, -73.26
****** End of object ***
```

Note that data at levels higher than the requested information will not be returned.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Examples of Selection

Selection -- choosing to display information only when particular variables satisfy specified criteria -- is accomplished by including a Boolean combination of comparisons as an argument to the data object.

objectname(variablename1=value1&variablename2=value2...)

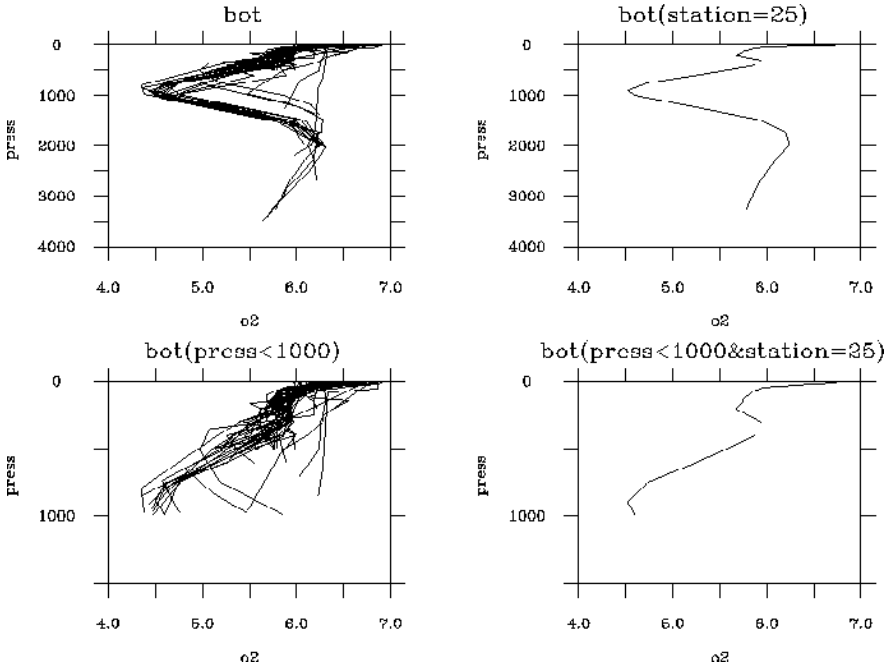
Permissable comparison operators are

<,=,>,<=,*<>,>=

while the Boolean operations are

&(and),|(or),!(not)

and grouping is accomplished with left and right parentheses. As an example we show some subselected data sets:



Thus if we have a data object **hyd** (displayed by the program**list)**

```
list "hyd"
#  wunsch stations 3-5
#  p<1000
=======================
    leg,   year,  month
.......................        [lowest (0) level data]
      1,     81,      6
=======================
station,    lat,    lon
.......................        [first level data]
```

```
      3,  38.28, -73.53
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 18.334, 33.570,  5.970, 24.096
 25.000, 12.848, 34.159,  6.990, 25.773
 49.000, 11.070, 34.523,  6.060, 26.394       [second level]
 99.000, 11.093, 35.090,  5.340, 26.831
149.000, 11.906, 35.487,  5.020, 26.990
199.000, 10.819, 35.435,  4.210, 27.152
300.000,  8.293, 35.126,  3.730, 27.334
400.000,  6.363, 35.046,  4.640, 27.546
500.000,  5.724, 35.019,  4.980, 27.608
600.000,  5.031, 34.990,  5.460, 27.670
701.000,  4.633, 34.981,  5.680, 27.710
801.000,  4.515, 34.980,  5.850, 27.722
901.000,  4.376, 34.979,  5.880, 27.737
=======================
station,    lat,    lon
.........................
      4,  38.19, -73.52
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 17.516, 33.160,  5.840, 23.981
 25.000, 12.315, 33.958,  7.090, 25.721
 49.000,  9.612, 34.192,  6.020, 26.387
 99.000, 12.095, 35.402,  5.340, 26.887
149.000, 12.407, 35.625,  5.290, 27.000
199.000, 11.287, 35.487,  4.340, 27.108
300.000,  8.059, 35.120,  3.750, 27.365
400.000,  6.606, 35.053,  4.440, 27.520
500.000,  5.467, 34.997,  5.130, 27.622
600.000,  5.003, 34.983,  5.460, 27.667
701.000,  4.680, 34.979,  5.700, 27.702
801.000,  4.489, 34.977,  5.850, 27.722
901.000,  4.374, 34.978,  5.930, 27.737
=======================
station,    lat,    lon
.........................
      5,  38.16, -73.26
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 18.382, 33.647,  5.770, 24.143
 25.000, 12.040, 34.196,  6.660, 25.959
 49.000, 11.951, 34.925,  5.510, 26.543
 99.000, 11.914, 35.390,  5.100, 26.912
149.000, 12.045, 35.547,  5.070, 27.010
199.000, 11.976, 35.589,  4.940, 27.057
300.000,  9.425, 35.250,  3.620, 27.251
400.000,  7.003, 35.075,  4.210, 27.483
500.000,  5.827, 35.009,  4.910, 27.589
600.000,  5.252, 34.988,  5.300, 27.643
701.000,  4.845, 34.980,  5.610, 27.684
801.000,  4.635, 34.980,  5.710, 27.709
901.000,  4.444, 34.978,  5.940, 27.729
****** End of object ***
```

we can select a depth range by

```
list "hyd(press<=500)"
#  wunsch stations 3-5
#  p<1000
=======================
    leg,   year,  month
.........................
      1,     81,      6
=======================
station,    lat,    lon
.........................
      3,  38.28, -73.53
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 18.334, 33.570,  5.970, 24.096
 25.000, 12.848, 34.159,  6.990, 25.773
 49.000, 11.070, 34.523,  6.060, 26.394
 99.000, 11.093, 35.090,  5.340, 26.831
```

```
149.000, 11.906, 35.487,  5.020, 26.990
199.000, 10.819, 35.435,  4.210, 27.152
300.000,  8.293, 35.126,  3.730, 27.334
400.000,  6.363, 35.046,  4.640, 27.546
500.000,  5.724, 35.019,  4.980, 27.608
=======================
station,    lat,    lon
.......................
      4,  38.19, -73.52
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 17.516, 33.160,  5.840, 23.981
 25.000, 12.315, 33.958,  7.090, 25.721
 49.000,  9.612, 34.192,  6.020, 26.387
 99.000, 12.095, 35.402,  5.340, 26.887
149.000, 12.407, 35.625,  5.290, 27.000
199.000, 11.287, 35.487,  4.340, 27.108
300.000,  8.059, 35.120,  3.750, 27.365
400.000,  6.606, 35.053,  4.440, 27.520
500.000,  5.467, 34.997,  5.130, 27.622
=======================
station,    lat,    lon
.......................
      5,  38.16, -73.26
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 18.382, 33.647,  5.770, 24.143
 25.000, 12.040, 34.196,  6.660, 25.959
 49.000, 11.951, 34.925,  5.510, 26.543
 99.000, 11.914, 35.390,  5.100, 26.912
149.000, 12.045, 35.547,  5.070, 27.010
199.000, 11.976, 35.589,  4.940, 27.057
300.000,  9.425, 35.250,  3.620, 27.251
400.000,  7.003, 35.075,  4.210, 27.483
500.000,  5.827, 35.009,  4.910, 27.589
****** End of object ***
```

or a particular station by

```
list "hyd(station=4)"
#  wunsch stations 3-5
#  p<1000
=======================
    leg,   year,  month
.......................
      1,     81,      6
=======================
station,    lat,    lon
.......................
      4,  38.19, -73.52
=======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 17.516, 33.160,  5.840, 23.981
 25.000, 12.315, 33.958,  7.090, 25.721
 49.000,  9.612, 34.192,  6.020, 26.387
 99.000, 12.095, 35.402,  5.340, 26.887
149.000, 12.407, 35.625,  5.290, 27.000
199.000, 11.287, 35.487,  4.340, 27.108
300.000,  8.059, 35.120,  3.750, 27.365
400.000,  6.606, 35.053,  4.440, 27.520
500.000,  5.467, 34.997,  5.130, 27.622
600.000,  5.003, 34.983,  5.460, 27.667
701.000,  4.680, 34.979,  5.700, 27.702
801.000,  4.489, 34.977,  5.850, 27.722
901.000,  4.374, 34.978,  5.930, 27.737
****** End of object ***
```

or combine these operations

```
list "hyd(station=4&press<=500)"
#  wunsch stations 3-5
#  p<1000
=======================
    leg,   year,  month
.......................
      1,     81,      6
```

```
======================
station,    lat,    lon
.....................
      4,  38.19, -73.52
======================
  press,   temp,    sal,     o2,  sigth
-----------------------
  5.000, 17.516, 33.160,  5.840, 23.981
 25.000, 12.315, 33.958,  7.090, 25.721
 49.000,  9.612, 34.192,  6.020, 26.387
 99.000, 12.095, 35.402,  5.340, 26.887
149.000, 12.407, 35.625,  5.290, 27.000
199.000, 11.287, 35.487,  4.340, 27.108
300.000,  8.059, 35.120,  3.750, 27.365
400.000,  6.606, 35.053,  4.440, 27.520
500.000,  5.467, 34.997,  5.130, 27.622
****** End of object ***
```

A more complex selection might look like

```
list "hyd((station=4|station=5)&press>=200&press<=500)"
#  wunsch stations 3-10
#  p<1000
======================
    leg,   year,  month
.....................
      1,     81,      6
======================
station,    lat,    lon
.....................
      4,  38.19, -73.52
======================
  press,   temp,    sal,     o2,  sigth
-----------------------
300.000,  8.059, 35.120,  3.750, 27.365
400.000,  6.606, 35.053,  4.440, 27.520
500.000,  5.467, 34.997,  5.130, 27.622
======================
station,    lat,    lon
.....................
      5,  38.16, -73.26
======================
  press,   temp,    sal,     o2,  sigth
-----------------------
300.000,  9.425, 35.250,  3.620, 27.251
400.000,  7.003, 35.075,  4.210, 27.483
500.000,  5.827, 35.009,  4.910, 27.589
****** End of object ***
```

Projection can be combined with selection by adding the list of variables to be returned also.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Putting Data on the system

To add a new data object to the system, one needs a translator/method which can properly interpret the data. The options are:

- Write a new translator to conform to the data. If there is a large, established database with existing programs for updating and access, this may be the best procedure. Often this translator may also glue together a number of different files to form a full database.
- Transform the data into a form compatible with an existing translator/method. This may be the easiest thing to do when a measurement program is just beginning.

Two existing methods, shipped with the system, are the default method, def, and the method for reading output from the list program, nm.

def

This is intended for data with each station (or mooring, etc.) in a single file, with header files linking them. Thus a hydrographic data set might look like:

Header file

```
# Gulf Stream Cruise Stations 3-5
# p<1000
station lat lon >    [variable names for this file's data]
 press  temp sal o2 sigth [variable names for the next level files]
  3     38.28 -73.53  s3
  4     38.19 -73.52  s4
  5     38.16 -73.26  s5
```

file s3

```
# Station 3
# lat=38.28, lon=-73.53
# This data prepared by someone
# Measurement at station 21 decibars contaminated
# 2/18/93
depth   temp    sal     oxy
 1.000  21.800  25.380   5.700
 3.300  nd      nd        nd
 5.000  21.800  25.580   5.600
10.000  21.400  25.670   5.400
13.000  21.000  25.850   5.000
15.000  20.500  26.020   5.000
21.000  19.900  26.400   5.000
```

The # sign indicates comments; the > in the header variable name list indicates that item points to a subfile containing more detailed information.

nm

This method is for a single file with multiple stations.

```
# Gulf Stream Cruise Stations 3-5
# p<1000

 station = 3    lat = 38.28,    lon = -73.53

  press,   temp,    sal,     o2,  sigth
  5.000, 18.334, 33.570,  5.970, 24.096
 25.000, 12.848, 34.159,  6.990, 25.773
 49.000, 11.070, 34.523,  6.060, 26.394
 99.000, 11.093, 35.090,  5.340, 26.831
149.000, 11.906, 35.487,  5.020, 26.990
199.000, 10.819, 35.435,  4.210, 27.152

station =       4,    lat =   38.19,    lon =  -73.52
```

```
  press,   temp,    sal,     o2,  sigth

  5.000, 17.516, 33.160,  5.840, 23.981
 25.000, 12.315, 33.958,  7.090, 25.721
 49.000,  9.612, 34.192,  6.020, 26.387
 99.000, 12.095, 35.402,  5.340, 26.887
149.000, 12.407, 35.625,  5.290, 27.000
199.000, 11.287, 35.487,  4.340, 27.108

station = 5,    lat=38.16,    lon=-73.26

  press,   temp,    sal,     o2,  sigth

  5.000, 18.382, 33.647,  5.770, 24.143
 25.000, 12.040, 34.196,  6.660, 25.959
 49.000, 11.951, 34.925,  5.510, 26.543
 99.000, 11.914, 35.390,  5.100, 26.912
149.000, 12.045, 35.547,  5.070, 27.010
149.000, 12.045, 35.547,  5.070, 27.010
199.000, 11.976, 35.589,  4.940, 27.057
```

Comment lines begin with #. The lines with an equals sign = contain assignments for variables at level 0 (comma or space separated). The assigments need only be done when the variable changes. The first line without an equals sign contains the names of the level 1 variables (comma or space separated).

▲ back to top

Data Management

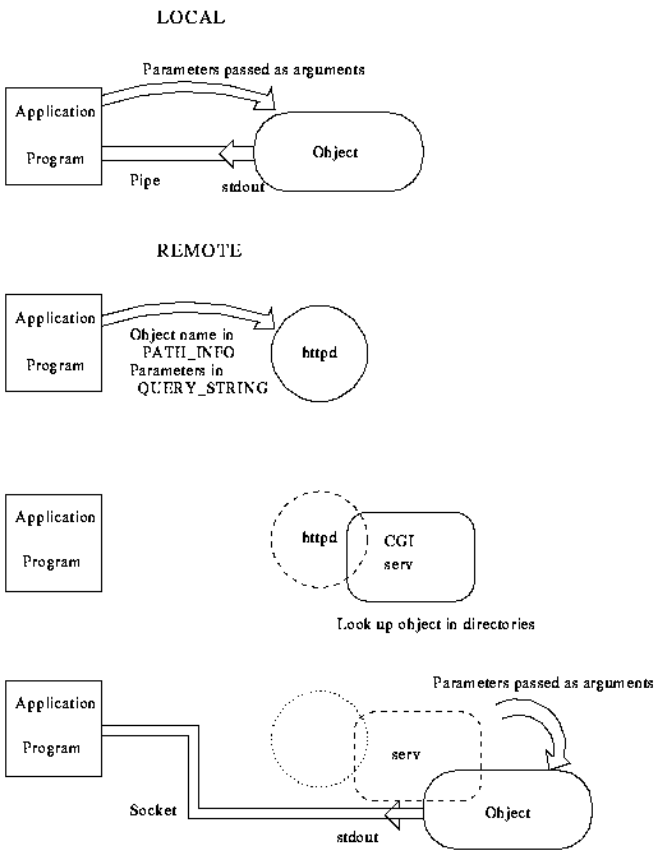
Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Communications

There are two parts to the problem of communicating information from the object on one machine to the application on another:

- The "physical" connection which involves setting up a data transfer pathway between the two processes on the different machines. To do this, the software uses NCSA's **HTTPD** and a **JGOFS data server program**.
- The **protocol** for the communication which ensures that the processes understand the requests and replies.

All exchanges between the user's application program (process 1) and the method/ translator (process 2 -- perhaps on another machine) are made via interprocess communications using "pipes" or "sockets" as defined in Berkeley UNIX. In the case of a locally defined object, a pipe is opened between the application and the method processes. For a remotely defined object, the application opens a socket to the **HTTP daemon** on the other machine and starts the **server**. The server effectively connects the standard output stream on the method to the socket in the application. The processes then begin exchanging information according to the JGOFS **protocol**.



▲ back to top

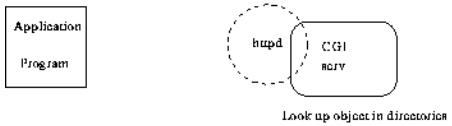
Data Management

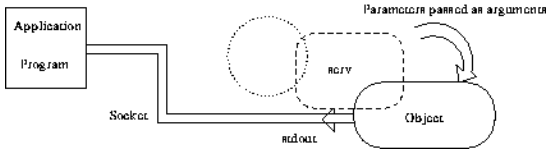
Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Servers and Dictionaries

Servers

Each JGOFS system which is providing data must have the **HTTPD** process running as a background task. When a request for data comes in, HTTPD starts our **server** process. This process consults the dictionary, starts up the method process and passes it the requested subselections and other parameters.





The method analyzes the request, gets the information from the data files or database, and writes out the results (in the JGOFS **protocol**). These pass through the communication pathway to the application program. In this sense, the method acts like an input subroutine which the main program calls to get data from files. However, the the data can be gathered from across the network.

Dictionaries: .objects files

The server works with two dictionaries, the user's (in the current working directory) and a tree of system dictionaries (set up when the software is built). These translate between a shorthand notation for the object and the detailed description either of where the object is [what machine it's on], or, if it's locally held, what method is used, and what default arguments are to be passed to the method. Thus the user can generally deal with brief names.

So users can specify objects in the following forms:

1. **method(parameters)**
   In this case, the software will use the method named as the translator, passing it the parameters. Methods are stored in the **methods** subdirectory of the JGOFS software directory. The parameters are passed as command line arguments to the process.
2. **datafilename** or **datafilename(parameters)**
   In this case, the software assumes the **default method**, **def**, is being used.
3. **nameindictionary** or **nameindictionary(parameters)**
   The name is looked up in a file, .objects, in the present directory and replaced with the information found therein. The parameters are merged. For example, if the local **.objects** file contains

   ```
   stuff=nm(myfile)
   farstuff=//jgofs.whoi.edu/test
   ```

   Then a request for **stuff(press<100)** will translate to **nm(myfile,press<100)** and then be reinterpreted by the first rule. A request for **farstuff(press<100)** will be translated to **//jgofs.whoi.edu/test (press<100)** and reinterpreted by the fifth rule below.

4. **/path/nameindictionary** or **/path/nameindictionary(parameters)**
   The name is looked up in a file, .objects, in the JGOFS system directory, following the path given. The ``root'' of the objects tree is the subdirectory **objects** of the JGOFS software directory. Replacement occurs as above.

5. **//machinename/path/nameindictionary** or **//machinename/path/nameindictionary(parameters)**
   The path, name, and parameters are transferred to the remote machine which then follows the procedure outlined just above.

Dictionaries have two types of entries:

Local entries
   These map the name to a method on this machine and (usually) some required parameters: e.g.,
   bot=jgbl2(/d5/glenn/bloom/bot)
Remote entries
   Usually, these just map a name on this machine to a name on the other machine. Thus if a data object on the remote machine is moved or replaced, only the dictionary on that machine needs to be updated. This also shields remote users from needing any details about the remote filesystem, methods, or data locations. An entry of this type looks like
   bot=//puddle.mit.edu/jgofs/bloom/bot

Dictionaries: .remoteobjects files

In addition, the system supports a set of dictionaries which tell the outside world what objects are available on this machine. In addition, other information about the object is provided, usually with links to an HTML page giving textual description of the information in the object, the variables, etc. Such a file looks like

```
tco2=//puddle.mit.edu/jgofs/bloom/tco2
- P.Brewer
- Total carbon dioxide
optics=//dataone.whoi.edu/jgofs/bloom/optics
- C.Davis
- Bio Optical Profiler Data
poc=//puddle.mit.edu/jgofs/bloom/poc
- H.Ducklow
- Particulate C, N
stuff=//puddle.mit.edu/test
-
- http://puddle.mit.edu/notready.html This will contain good stuff
```

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

HTTP daemon

We use the Hypertext Transport Protocol Daemon (HTTPD) from the National Center for Supercomputer Applications, NCSA. This software is intended to

- Serve HTML pages, including graphics, and other static information to users on the World-Wide Web.
- Start programs on the server machine, using what's called the Common Gateway Interface (CGI), which can take information from the request and construct pages or graphic information to be sent back to the user. This ability is used by our **data server**.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Protocol for Communication

Methods provide three different kinds of data stream. You can view all of these from browsers:

- HTML for browser display. See **this example** (and view at the "source").
- Flat listings browser display. See **this example**.
- Protocol for application programs. See **this example**.

The rest of this document concentrates on the last case.

Example

We illustrate the communication protocol with a simple example: for a data object which looks like

```
list "test(station<=5&press<100,station,lat,lon,press,o2)"

#  wunsch stations 3-10
#  p<1000
======================
station,    lat,    lon
......................
      3,  38.28, -73.53
======================
  press,     o2
------------------------
  5.000,  5.970
 25.000,  6.990
 49.000,  6.060
 99.000,  5.340
======================
station,    lat,    lon
......................
      4,  38.19, -73.52
======================
  press,     o2
------------------------
  5.000,  5.840
 25.000,  7.090
 49.000,  6.020
 99.000,  5.340
======================
station,    lat,    lon
......................
      5,  38.16, -73.26
======================
  press,     o2
------------------------
  5.000,  5.770
 25.000,  6.660
 49.000,  5.510
 99.000,  5.100
======================
```

The dictionary entry is assumed to be

test=def(/usr/users/jgofs/data/t0)

The communications look like:

list -> method (def)

```
argv = [/usr/users/jgofs/data/t0,station<=5&press<100,station,lat,lon,press,o2
def -> list
&c**********************
 wunsch stations 3-10
 p<1000
&v0======================

&v1======================
station lat     lon
&v2======================
press   o2
&r======================
&c**********************
 wunsch stations 3-5
 p<1000
&d0----------------------

&d1----------------------
3       38.28   -73.53
&d2----------------------
5.000   5.970
25.000  6.990
49.000  6.060
99.000  5.340
&d1----------------------
4       38.19   -73.52
&d2----------------------
5.000   5.840
25.000  7.090
49.000  6.020
99.000  5.340
&d1----------------------
5       38.16   -73.26
&d2----------------------
5.000   5.770
25.000  6.660
49.000  5.510
99.000  5.100
&e**** End of object ****
```

Thus the application begins by sending the parameters to the method and then reading the blocks of data. The blocks are indicated by commands with an **&** in the first position. There are four types of protocol blocks: comments, variable names, data, and end.

Protocol blocks

Comments

The **&c** introduces the plain text comments section. Comments consist of lines of no more than 80 characters.

Variables

This section gives the names, dimensions, and attributes of variables at each hierarchical level. The outermost level, 0, is defined first and then we work our way inward. The signal is **&vn** with n=0...9 the level indicator. Each variable definition has:

1. The name (avoid embedded blanks --- use _)
2. Attribute list appended to the variable name surrounded by []. The attribute list is a comma-separated set of strings, usually (except for units) of the form **attribute=value**. The variables section is closed by a record marker, **&r**.

Variable fields are tab-separated.

Data

The data is likewise presented in a hierarchical fashion. The **&dn** intoduce the data from the n'th level. Note that the innermost level can drop the **&dn**. The data from the outermost levelis sent, followed by the next level, up to the innermost level. The innermost level repeats until the next level up changes or the data ends. Data fields are tab-separated.

End

This indicates the end of the data object. The indicator is **&e**.

Errors

Errors are indicated by the method returning **&x [descriptive string]** and exiting.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

User Applications

There are many different ways of using the JGOFS data system:

1. Browsers: World-Wide Web browser clients can be used to list and (for those which support imaging) plot data.
2. To list data, one can use a **user interface** to construct calls to the data system, acquire the data, and do certain operations on it. The user interface is *not* a core part of the system, but is built using the programming interface; and supports both command line and web browser client interfaces.
3. Data can be imported into commercial packages, in some cases quite directly. Thus we have a MATLAB function, **loadjg**, which can read data directly into matrices from the data system (including all the data manipulation operations). See **here**.
4. We provide a simple set of **subroutine calls** by which C and Fortran programs can read data.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

User Interfaces

Command Line Interface

After setting one's path appropriately to include the JGOFS binaries, one can type commands which directly invoke programs that communicate with the data system. Such commands can also be incorporated in a shell

script (perhaps with arguments) for repetitive operations. The basic commands are

Listing an Object

```
listvar "object"
        Lists the variables in the object, with indenting indicating the hierarchical structure.
list "object"
        Lists the data
        In addition, there are options for other kinds of listings:
        list [-n] [-s] [-t] [-f] [-b] [-c] [-z] object [outfile]

Typing the application name without any arguments returns usage instructions.
For example:

% list
Usage: list [-n] [-s|-t] [-z] [-m [new_missing]] [-c] [-l] [-f|-b|-r|-rs|-rt]
         [-errout errfile[+]] [-noerrecho] [-nopipeerr] [-finishinp]
         [-forceheader]
         object [outfile[+]]
Options:
   -n   nonstop
   -s   space-separated   |  -t   tab-separated
   -z   delete extra spaces
   -m   missing value converted to new_missing (default -9999.0)
   -c   no comments
   -l   no limit on output line length
   -f   flat file output   |  -b   brief flat file (no header info)
   -r   reps-on-one-line output
   -rs   -r w/reps space-separated   |  -rt   -r w/reps tab-separated
   -h   prints this message
   ?    prints this message
   -v   prints version information

-errout       send err messages to errfile instead of /dev/stdout after command line parsing
-noerrecho    do not echo err messages to outfile if errfile is different
-nopipeerr    do not produce err message if output ends w/ "broken pipe"
-finishinp    read input until end-of-data even if output errors
-forceheader  produce variable list even if there is no data from object
           (unless listing is in -b format)

list version 1.6 28 Dec 2003

# to return a list of the 'test' object data
#
% list -c -f -n -z /test
leg,year,month,station,lat,lon,press,temp,sal,o2,sigth
1,81,6,3,38.28,-73.53,5.000,18.334,33.570,5.970,24.096
1,81,6,3,38.28,-73.53,25.000,12.848,34.159,6.990,25.773
1,81,6,3,38.28,-73.53,49.000,11.070,34.523,6.060,26.394
1,81,6,3,38.28,-73.53,99.000,11.093,35.090,5.340,26.831
1,81,6,3,38.28,-73.53,149.000,11.906,35.487,5.020,26.990
. . .
```

```
% listvar
Usage: /usr/local/dmo/JGddms/bin/listvar [-a][-l] object
Options:
 -a   follow variable names with attributes
 -l   precede variable names with level at which it occurs as X, variable

listvar version 1.3  28 Jul 1998

# to return a list of the variables and their respective levels from 'test' object data
#
% listvar -l /test
0, leg
0, year
0, month
1, station
1, lat
1, lon
2, press
2, temp
2, sal
2, o2
2, sigth
```

Browser Interface

The browser interface can be used from any platform running a web browser client. You can try it **here**. In addition, It is possible to install software on UNIX machines which allows you to use the local server and obtain full functionality through browers.

▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

MATLAB [c. Mathworks] access

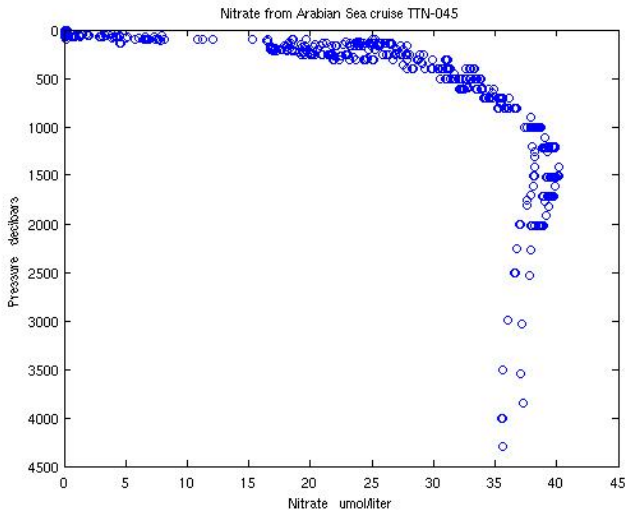
The MATLAB interface uses the **loadjg** function. The code to build **loadjg** is distributed with some versions of the DDMS software distribution package.

loadjg

This is a cmex program (like an M file, but compiled and written in C) which functions like MATLAB's **load** command but can accept both .mat files and JGOFS object names. The syntax is
>> loadjg('objectname'[,'NaN'])
For example, to load the Arabian Sea Niskin bottle data from cruise TTN-045
>> loadjg('//usjgofs.whoi.edu/jgofs/arabian/ttn-045/bottle')

Here's an example:

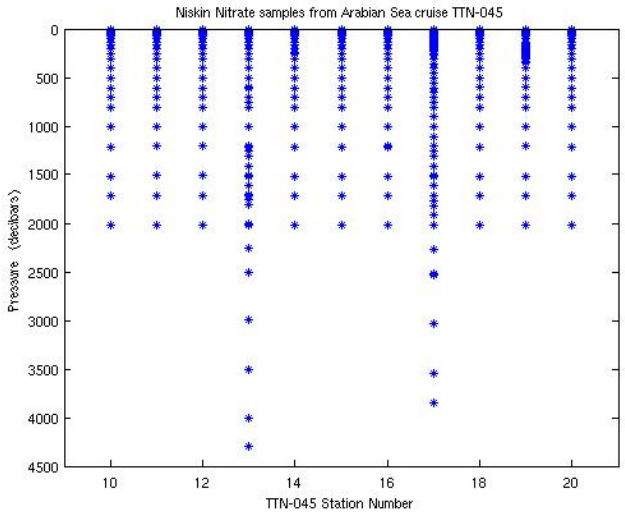
```
% Plot nitrate profile for a data selection
%
% read in data from JGOFS data system
% Arabian Sea TTN-045 bottle nitrate data for stations 10 through 20
loadjg('//usjgofs.whoi.edu/jgofs/arabian/ttn-045/bottle(sta,press,NO3,sta>=10,sta<=20)','NaN');
% and plot
figure(1);
plot(NO3,press,'bo');
axis ij;
xlabel('Nitrate  umol/L');ylabel('Pressure  decibars');
title('Nitrate from Arabian Sea TTN-045');
```



and another example:

```
% Plot nitrate sample distribution for a data selection
%
% read in data from JGOFS data system
% Arabian Sea TTN-045 bottle nitrate data for stations 10 through 20
loadjg('//usjgofs.whoi.edu/jgofs/arabian/ttn-045/bottle(sta,press,NO3,sta>=10,sta<=20)','NaN');
% and plot
figure(2);
```

```
plot(sta,press,'b*');
axis ij;
axis([9,21,0,4500]);
xlabel('TTN-045 Station Number');ylabel('Pressure   (decibars)');
title('Niskin Nitrate samples from Arabian Sea cruise TTN-045');
```



▲ back to top

Data Management

Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Application Program Interface

From a C programs, you can call the folowing functions to obtain information from the data system:

maxlev = jdbopen_(&unit,obj,names,&namesize,&num)
Opens a data object. The variable **char obj[1024]** is a long string containing the object name, including parameters if desired for selections, etc. The array of **char names[][namesize]** contains **num** names for the variables to be returned (if **num** > 0) or space for **|num|** names if **num**< 0. In the latter case, the number of variables found is also returned in **num**. The result of the call is the maximum heirachical level of the dataset. Negative values are error returns.
lev = jdblevel_(&unit,&varnum)
Return the level in the hierarchy (0=outermost, 1=next,...) of the variable number **varnum**.
lev = jdbread_(&unit,values)
Read the next realization of the data from the object. The subroutine fills in **num** values in the array of **floats**.
lev = jdbreada_(&unit,values,&valuesize)
Same as above, but the values are read into strings
ok = jdbcomments_(&unit,outcom)
Return the next comment in the string **outcom**. The returned value is 0 if there are no more comments.
ok = jdbattributes_(&unit,&id,outcom)
Return the next attribute of variable number **id** in the string **outcom**. The returned value is 0 if there are no more attributes.
jdbclose_(&unit)
Close the object.

In Fortran, the calls are

```
maxlev = jdbopen(unit,obj,names,namesize,num)
lev = jdblevel(unit,varnum)
lev = jdbread(unit,values)
lev = jdbreada(unit,values,valuesize)
ok = jdbcomments(unit,outcom)
call jdbclose(unit)
```

▲ back to top

Data Management

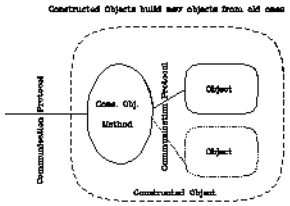
Detailed Information About the JGOFS Distributed Database Management System (DDMS)

Manipulating Data

The system has two built-in operations -- functions which each data object is expected to carry out:

- **Projection** (subsetting by variable name)
- **Selection** (subsetting by variable values)

But, although these are the most common operations, they do not in and of themselves satisfy all the requirements for a data system. One significant advantage of an object-based data system is that new operations can be added at any time. One simply builds a "method" which takes as its input information that supplied by one (or more) objects rather than data files, transforms the information in some way, and passes it on to the user application.



We call the combination of the new method and the sub-objects a "constructed object." You can also think of these as similar to UNIX filters.

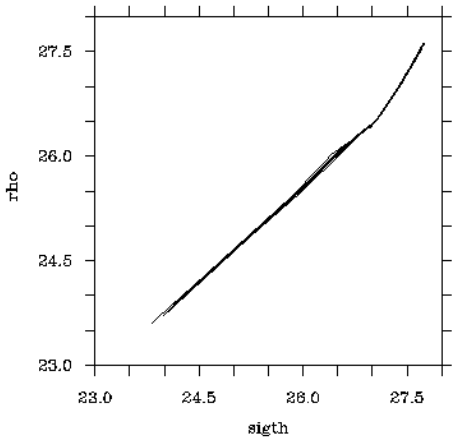
For example, we can add a column to the **/test** (hydrographic data) which gives a linearized estimate of density:

rho=28.5-0.2 T +0.7 (S-35)

by using the ``math'' constructed object which takes as parameters an input object name and formulae for changing/ adding columns. The new object name is

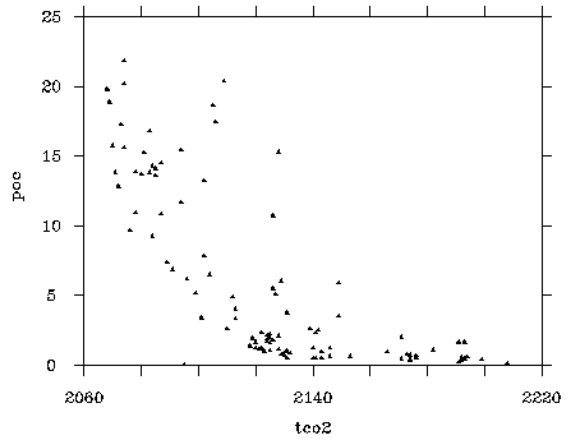
math(/test,rho=28.5-0.2*temp+0.7*(sal-35))

and this can be used by the lister/plotter/... in exactly the same way as any other object --- see figure.



As another example, there is a plot from two data objects joined together by common station, cast, and pressure.

join(/tco2,/poc(station,cast,press,poc))



▲ back to top