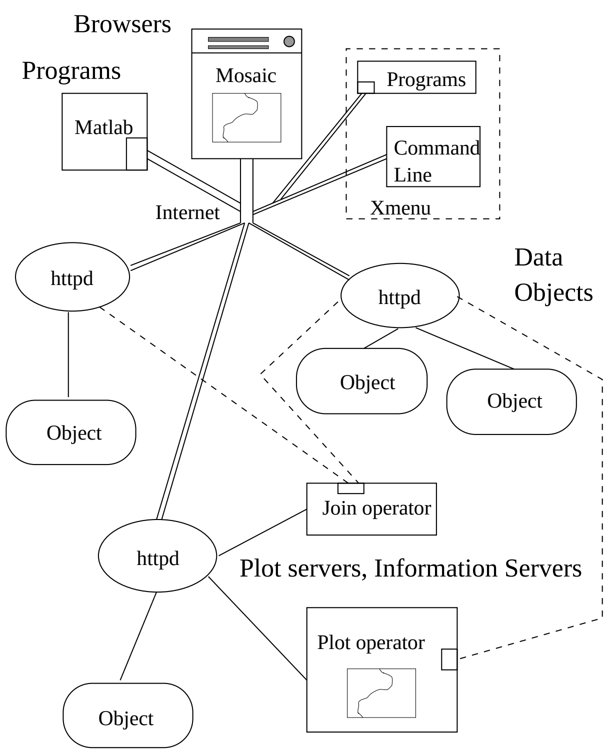
JGOFS Object-Based Data System

Glenn R. Flierl, MIT
James K. Bishop, Lamont-Doherty
David M. Glover, Woods Hole
Satish Paranjpe, Lamont-Doherty

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

System Elements:

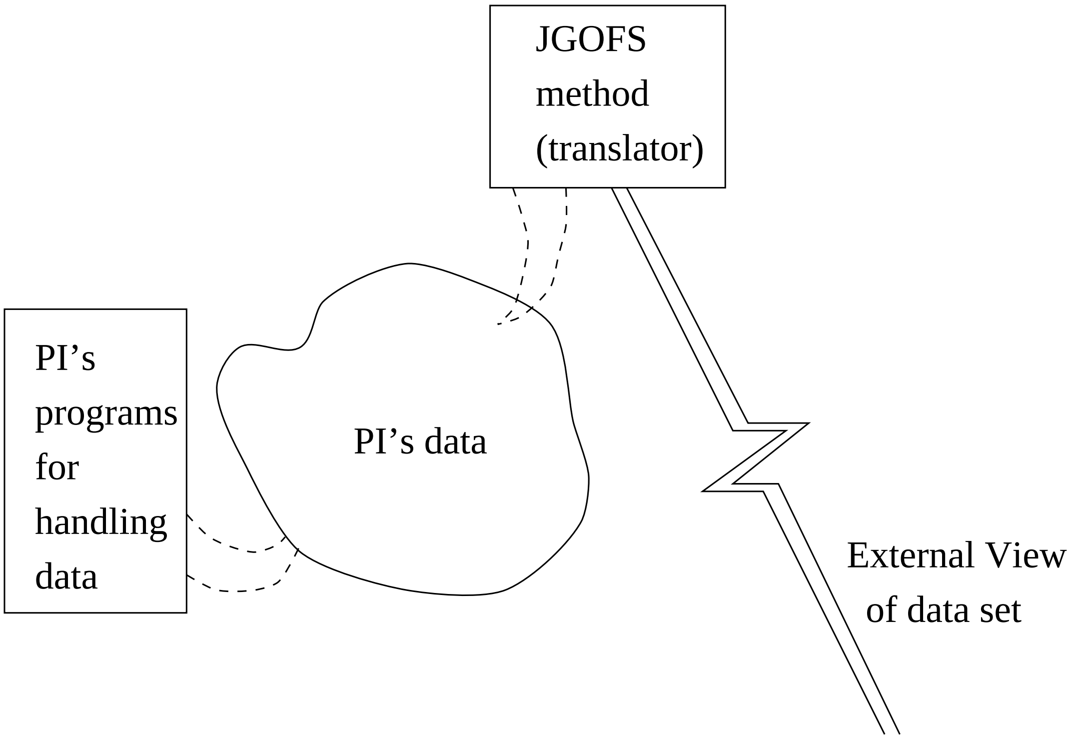


Basic Principle

The system is designed to unify and provide simple access to distributed, heterogeneous data sets.

· When possible, PIs keep the data sets on their own machines in their own formats and can manipulate the data with their own programs.

· The JGOFS data system provides a viewport by which other investigators can access the same data set. This viewport is provided by an **executable program** (called a "method" in object-based terminology — you can also think of it as a "translator"), which converts the requested data on–the–fly into a uniform structure and format.



These programs are responsible for

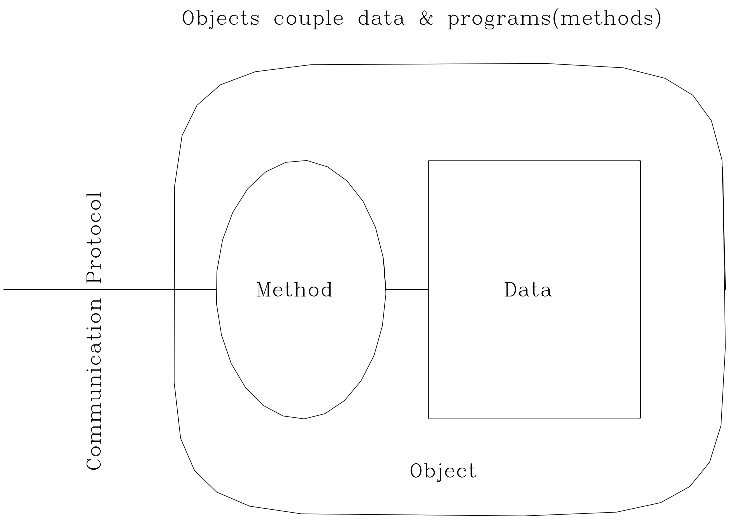
1) receiving requests for subselections of the data
2) gathering the requested information from the data set
3) translating the information into the common internal form used for transferring data
4) sending the information through the communication line to the process which made the request.

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).

Data Objects:

Data objects combine a program (method/ translator) and data. The only access to the data from outside is through the method. The coupling is made explicitly in a `.objects` file.

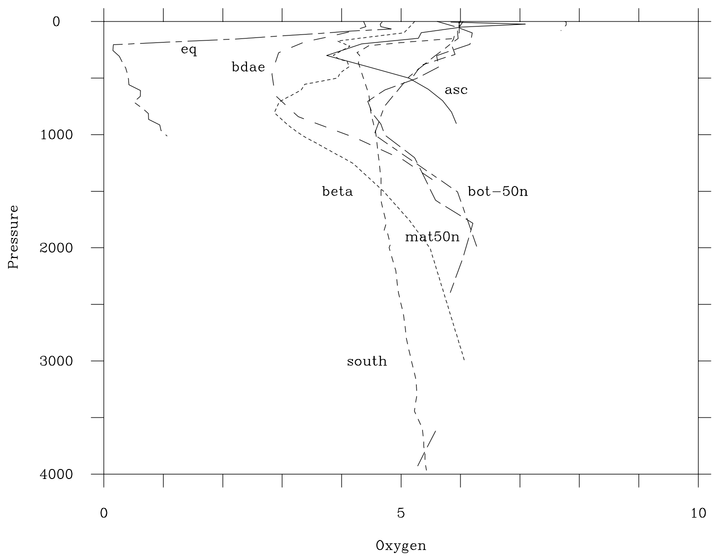
```
   test=def(/usr/users/jgofs/data/t0)
objname  method   req.  params.
```



Example:

The following graph shows the output from a single program `dash` requesting "o2" and "press" data from each object and displaying it with a specified dash pattern. The objects are

object	location	machine/storage method
asc	France	HP/ flat ASCII file
beta	MIT	Ultrix/ scaled bin. int.
bdae	Dartmouth	IBM/ scaled bin. int.
mat50n	MIT	Sun/ MATLAB bin. float
eq	U. Victoria	Sun/ multiple files/dirs
bot-50n	WHOI	Sun/ multiple files
south	U. Chicago	IRIX/ One file, mult. sta



1) Data is transmitted with a common protocol

→ Programs can work with any data in the system

The communications protocol includes:

- Comments (text)
- Variable descriptions
  - Name
  - [Dimensions for vectors/ matrices/ tensors]
  - Attributes (e.g., units)
  - Hierarchical structure
- Data
  - Strings or numbers

The protocol allows hierarchical structuring

- leg
- year
- month
  - station
  - lat
  - lon
    - press
    - temp
    - sal
    - o2
    - sigth

---

2) Data objects behave like functions

The arguments instruct the data object to do certain operations before transmitting the information

Projection (subsetting by variable name)

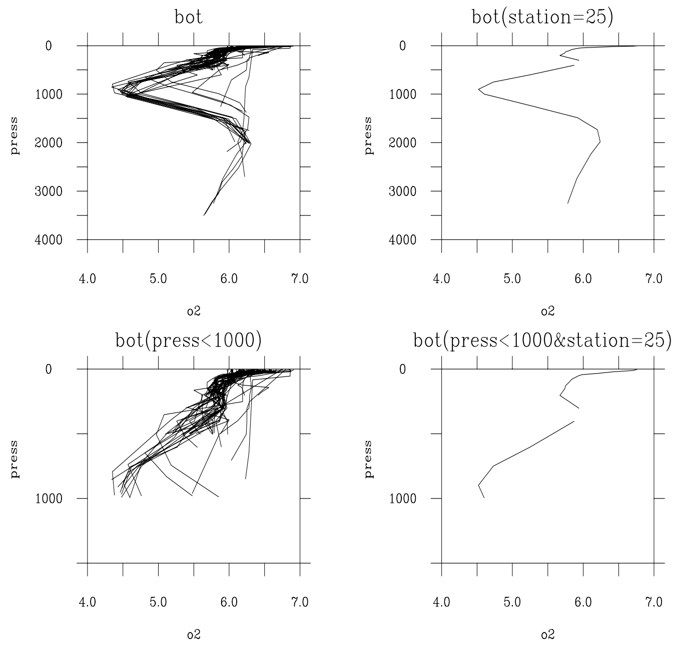
Selection (subsetting by variable values)

→ Can minimize transmission of data

→ Individual objects may have other functions

These plots show the results of "selections" on a data object

---



3) Data name specifies location and operations

The name one gives specifies both the directory name (analogous to the file name) and additional arguments (operations) you want done:

/jgofs/bloom/bot
/jgofs/bloom/bot(eventno,lat,lon)
/jgofs/bloom/bot(eventno=272022)
/jgofs/bloom/bot(eventno=272022,
press,o2,no3,po4)

In order to access a data object, you need to know its *full name*. For local objects, this includes the method name and the parameters; for remote objects, it contains the machine name, the object name in the remote directory, and the parameters. Often, parts can be abbreviated.

In the JGOFS directory, the /jgofs/

eqpac/tt007/ctd object is listed in the
`.remoteobjects` file as

```
ctd=//dataone.whoi.edu/jgofs/eqpac/
   tt007/ctd
- J.Murray
- Final version CTD w/beam attenuation
```

On dataone.whoi.edu, the local directory has in its `.objects` file

```
ctd=def(/data/eqpac/ctd/tt007/header7.def)
```

where `def` is the method name and
`/data/eqpac/ctd/tt007/header7.def` is the first parameter to be passed to `def`

A request such as ctd(sta=5,press,fluor) translates effectively into

```
//dataone.whoi.edu/def(/data/eqpac/ctd/tt007/header7.def,sta=5,press,fluor)
```

Putting Data on System

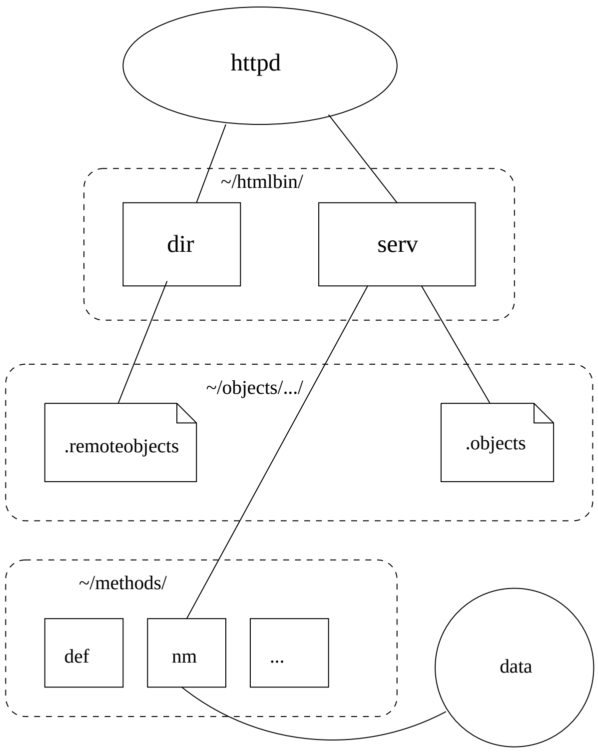
In JGOFS and Globec GB, there are a number of ways to submit your data:

1) **On a workstation:**
   - Install JGOFS data server
   - Identify method (translator) for data
     Use existing method (`def, lm, nm`)
     Write new method to fit data structure
   - Construct local directory information (.objects) which links method to data and gives it an object name
   - Send global directory entry to Data Management Office (.remoteobjects)
     machinename & objectname
       (including path)
     descriptive information

---

2) **Send to Data Management Office:**
   - Data must be in acceptable format, with necessary information
   - E-mail, disks

Data Servers:



URL's

A Universal Resource Locator has several parts:

protocol:
: Protocol the server understands. In our case it's `httpd:`.

//machinename
: The network name of the machine serving the information.

:port
: The network port (default is 80 for `httpd`, and that's what we u se).

/directorystructure/file
: The path from the DocumentRoot to the file.

.mimetype
: The type of file - indicates how it is to be viewed on the receiving end. In our case, this is `NULL`, `.flat#`, or `.html#`

#subsection

Section of document [not used]

?querystring

Information to be passed as a query.

JGOFS software uses Universal Resource Locators, but fills in parts of them by default. Thus the object

```
//puddle.mit.edu/jgofs/bloom/ctd(station=20,press<=200)
```

becomes the URL

```
http://puddle.mit.edu/jg/serv/jgofs/
   bloom/ctd?station=20,press<=200
```

when data is passed to a program. For a browser, the URL might look like

```
http://puddle.mit.edu/jg/serv/jgofs/
   bloom/ctd.html1?station=20,
   press<=200
```

CGI

The CGI (common gateway interface) program, `serv`, receives the environmental variables

- `PATH_INFO` = `/jgofs/bloom/ctd.html1`
- `QUERY_STRING` = `station=20,press<=200`

and looks up `ctd` in
`objects/jgofs/bloom/.objects`
to find

```
ctd=jgbl(/d5/jgofs/bloom/ctd)
```

and overlays itself with the command

```
jgbl "/d5/jgofs/bloom/ctd" "station=20" "press<=200"
```

Directories:

You can think of data objects as residing in a directory tree, much as files do. For example, part of the JGOFS directory structure looks like:

/
  jgofs/
    test
    eqpac/
      sedtrapannual, sedtrapnorth,
      10mEPOCS5...
      tt007/
        tco2, pp, phyc, ...
      tt008/
        snow, pp, ox18...
      tt011/
        tco2, dist-pp, insitupp...

---

      tt012/
        pp, ox18, phyc...
      tt013/
        ctd, carbcore
    nabe
      atlantisII
        grazing, zoomass, optics...
      endeavor
        pp
    southern
      solar

.remoteobjects & .objects

These two files may be present in each data subdirectory and summarize the data objects “contained” therein.

The `.remoteobjects` file lists the externally visible objects and gives the reference to the machine which actually contains them. The format is

```
objectname=//machinename/dir/objectname
- PI name
- [URL] Brief description
```

(There could be qualifiers on the remote name as well.)

The `.remoteobjects` files are intended to be passed freely from machine to machine; they do not contain information about specific files or methods.

The `.objects` file lists all the objects actually residing on this machine. The format is

```
objectname=methodname(parameterlist[,optionalparameters])
     or
objectname=reference--to--other--obj(params)]
     or
objectname=methodname(parameters) n1=v1 n2=v2 ...
```

In the latter case, the `n1=v1` etc. strings are inserted into the environment before the method is executed.

The parameters in the `.objects` file are merged with those in the request.

1) required parameters from `.objects`
2) parameters from request
3) if request did not include projections,
   optional parameters for `.objects`
   enclosed in `[ ]`

Existing methods: data formats:

def

```
# stations 3-5
# p<1000
station lat lon >
    depth[m],temp,sal,oxy[ml/l;
       type=Winkler]
3     38.28 -73.53  s3
4     38.19 -73.52  s4
5     38.16 -73.26  s5
-----------------------------------

# 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
-----------------------------------

# This data prepared by someone
# Measurement at station 21 decibars contaminated
# 2/18/93
sta lat lon  depth[m] temp[degC] sal[ppt] oxy[ml/l]
3 38.28 -73.53  1.000    21.800   25.380   5.700
3 38.28 -73.53  3.300      nd       nd       nd
3 38.28 -73.53  5.000    21.800   25.580   5.600
```

---

nm

```
#  wunsch stations 3-10
#  p<1000
```

```
leg = 1  year = 81 month = 6
station = 3  lat[deg] = 38.28,
lon[deg] = -73.53

press,   temp,sal[psu],o2[ml/l],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

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

leg = 2,   year = 81,  month = 6
station =  6,    lat =   38.03,
lon =  -72.99

press,   temp,    sal,     o2,  sigth
 5.000, 16.942, 33.097,  6.020, 24.069
25.000, 14.871, 34.715,  6.380, 25.782
49.000, 10.605, 34.557,  5.820, 26.503
```

---

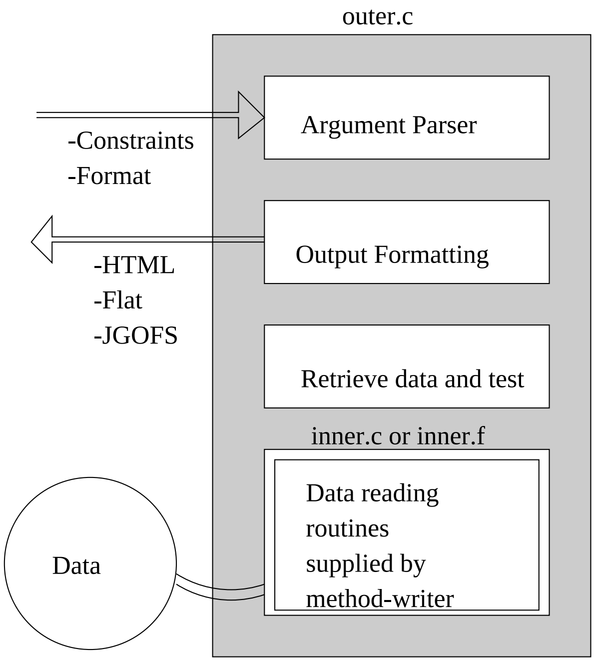
lm

```
#  wunsch stations 3-5
#  p<200
=======================
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
=======================
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
***********************
```

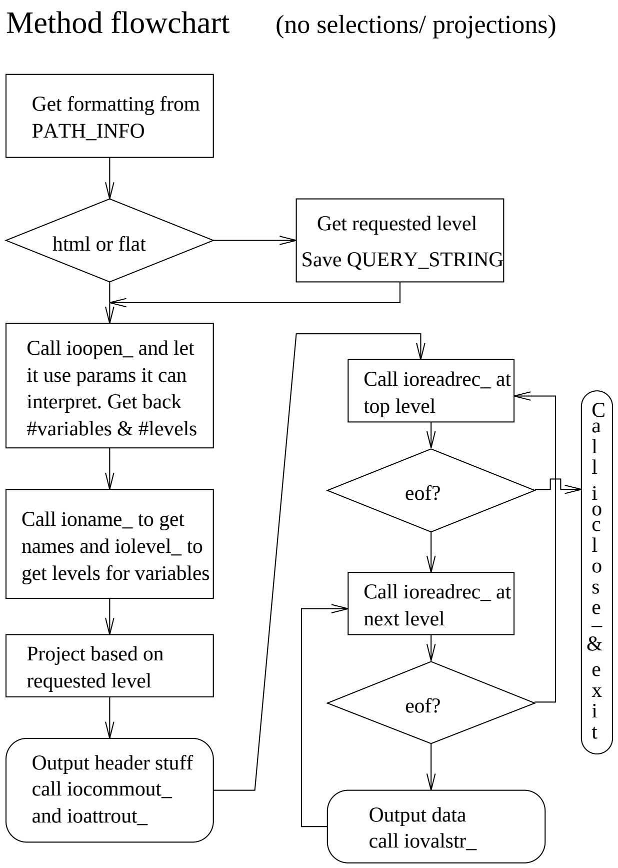
Methods



Subroutines in "inner"

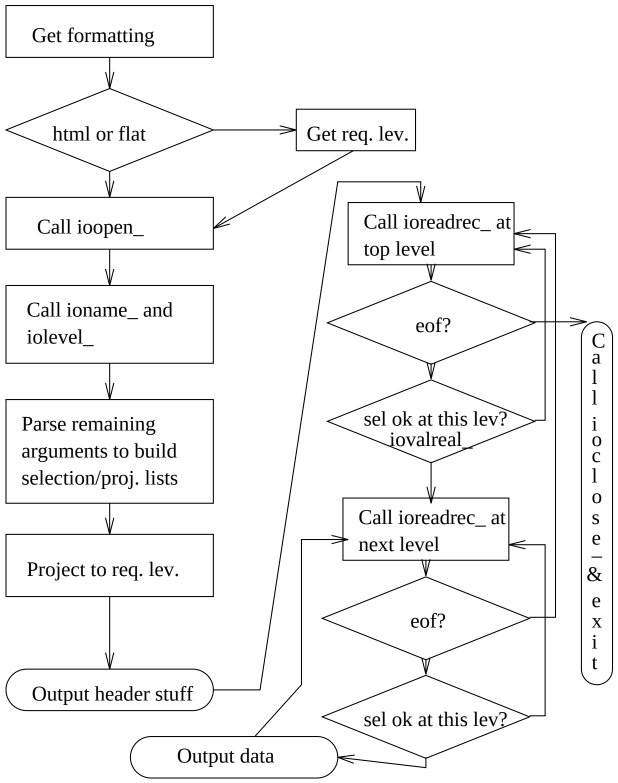
```
int ioopen_(s,nparams,ntotal)
   char *s[ ]; int *nparams,*ntotal;
ioname_(vn,s)
   int *vn; char *s;
int iovarlevel_(vn)
   int *vn;
int iocommout_(s)
   char *s;
ioattrout_(vn, str)
   int *vn; char *str;
int ioreadrec(lev)
   int *lev;
iovalreal_(vn,f)
   int *vn; float *f;
iovalstr_(vn,tmp)
   int *vn; char *tmp;
```

ioclose_()



Method flowchart
(no selections/ projections)
Get formatting from PATH_INFO
html or flat
Get requested level
Save QUERY_STRING
Call ioopen_ and let it use params it can interpret. Get back #variables & #levels
Call ioname_ to get names and iolevel_ to get levels for variables
Project based on requested level
Output header stuff call iocommout_ and ioattrout_
Call ioreadrec_ at top level
eof?
Call ioreadrec_ at next level
eof?
Output data call iovalstr_
Call ioclose_ & exit

Method flowchart (selections/ projections)



Information available to method:

argv & argc:

The command line arguments pass the parameters to the method: i.e., the output from the object

```
def(/usr/users/jgofs/data/t0,station=5,
   station,lat,lon)
```

is the same as would be obtained by typing

```
methods/def /usr/users/jgofs/data/t0 station=5
   station lat lon
```

The arguments generally contain
- Parameters such as filenames
- Selections
- Projections
- Other method–specific input

PATH_INFO:

- NULL or no extension: Return information in JGOFS protocol
- .flat#: Return information in flat–file format from level 0 to and including level #
- .html#: Return information in HTML browser format from level 0 to level #. Data becomes an anchor for getting the next level of data with a selection.
  When this is used, other environmental variables will also be required:
  - QUERY_STRING: Gives the original full query.
  - OPTIONSERVER: Gives the machine/CGI to use for the Other Options anchor.

Getting Data off System

There are also many ways to get data out of the system:

1) **Connecting:**
   - Workstation running JGOFS software
     - Typed commands
     - Menu interface
     - User programs
     - Shell scripts
   - Workstation/ PC/ Mac running Mosaic or Netscape
     - Other program information
     - Direct access to data system
   - Telnet
   - Dial-up

2) **Looking at Data:**
   - Listings
   - Plots
   - Data selection and manipulation
3) **Downloading Data:**
   - Several file formats
4) **Importing Data:**
   - Simple subroutine calls
   - Interfaces for other packages

Subroutine interface

```
int maxlevel = jdbopen_(handle,obj,names,
      namesize,num)
   int *handle; char *obj; char *names;
   int *namesize; int *num;
int ok = jdbcomments_(handle,outcom)
   int *unit; char *outcom;
int level = jdblevel_(handle,varnum)
   int *handle; int *varnum;
int ok = jdbattributes_(handle,varnum,attr)
   int *handle; int *varnum; char *attr;
int level = jdbread_(handle,values)
   int *handle; float values[ ];
int level=jdbreada_(handle,avalues,valuesize)
   int *handle; char *avalues; int *valuesize;
jdbclose_(handle)
   int *handle;
```

JGOFS programs

Listing:

listvar obj
list [-n] [-s] [-t] [-f] [-b] [-c] [-z] object [outfile]
readdct directory

Plotting:

axis [-l] x/y ticmar label numtic format
ch file var1=val1 ...
dash name [-l] xvar [-l] yvar dashpattern
ers
ez name v1 ... vn model
p name [-r] [-l] xvar [-r] [-l] yvar
  [+/-sym] marker# [-size] markersize
  [-brk] breakvar [-nobrk]
p2 name [-r] [-l] xvar1 [-r] [-l] xvar2
  [-r] [-l] yvar
pl name [-l] xvar [-l] yvar
  [+/-sym] marker# [-size] markersize
  [-brk] breakvar [-nobrk]
pns name [-l] xvar1 [-l] yvar
  [+/-sym] marker# [-size] markersize
  [-brk] breakvar [-nobrk]
text "label" x/-x/y/-y size xpos ypos [u]
view sxmin symin sxmax symax
window xmin ymin xmax ymax

Misc:

transfer url
updatedct
xmenu menu
xterm
tekps [-f n] [-w n] [infile] [outfile]
tekprint

Shell Script:

```
window 0 4000 10 0
axis x 1 Oxygen 5 dd
axis y 500 Pressure 1000 dddd
dash asc o2 press ----------------
text asc x 2 6 600 u
dash bdae o2 press ----....----....
text bdae x 2 2.5 400 u
dash mat50n o2 press ------..------..
text mat50n x 2 5.6 1900 u
dash south o2 depth --..--..--..--..
text south x 2 4.5 3000 u
```

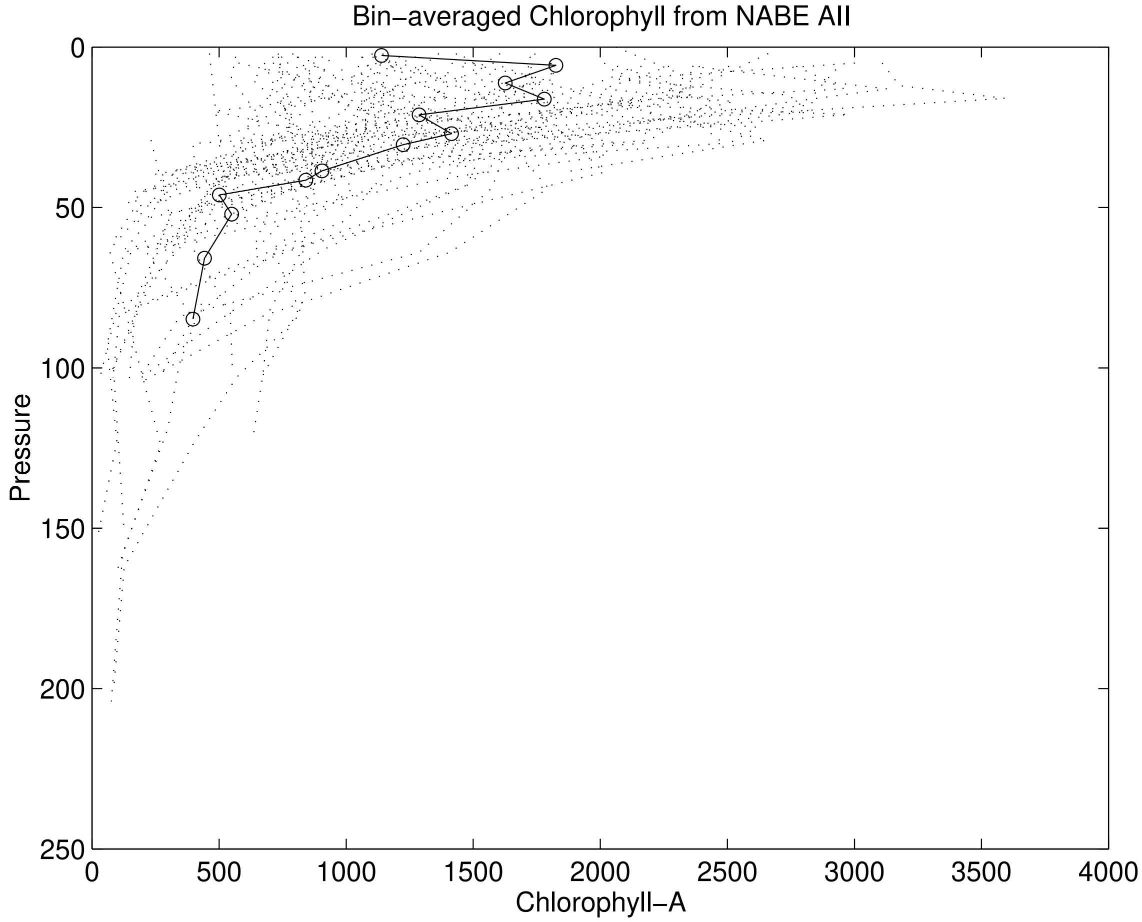
```
dash bot50n o2 press ----..--..----..
text bot50n x 2 6.7 1500 u
dash eq o2 press ----------..--..
text eq x 2 1.5 240 u
dash beta o2 press -.-.-.-.-.-.-.-.
text beta x 2 4 1500 u
```

.objects file

```
asc=//ccrv.obs-vlfr.fr/test(station=4)
mat50n=/d3/glenn/matlab/matjg(/d3/glenn/matlab/50n,station=3860)
bot50n=//dataone.whoi.edu/jgofs/nabe/atlantisII/bottle(sta=10)
bdae=//fundy.dartmouth.edu/bdae(station=5225)
beta=//wind.mit.edu:3377/beta2(station=128)
eq=//fireglo.seaoar.uvic.ca/jgofs/eqpac/tt007/bot(cast=1)
south=math(//popeye.uchicago.edu/jgofs/geosecs(sta=90),o2=o2/44.6)
```

MATLAB interface:

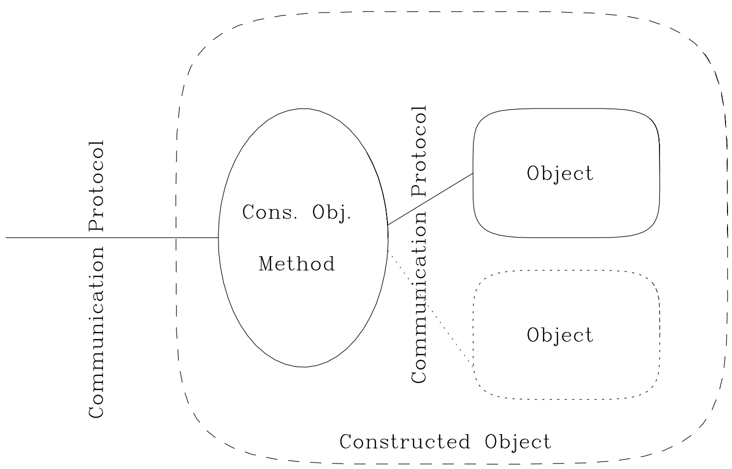
```
% Compute bin-averaged chlorophyll-a profile
%
% read in data from JGOFS data system
loadjg('//dataone.whoi.edu/jgofs/nabe/atlantisII/pigments(event,press,
   chla)','NaN');
% bin-average the press and chlorophyll-A
pav=binav([0:5:50,60:20:100],press,press);
cav=binav([0:5:50,60:20:100],press,chla);
% and plot
plot(cav,pav,cav,pav,'o',chla,press,':');
axis('ij');
xlabel('Chlorophyll-A');ylabel('Pressure');
title('Bin-averaged Chlorophyll from NABE
   AII');
```



Database Manipulations: "Constructed Objects"

These are objects which behave much like filters, taking the output from one or more objects, transforming it and passing it to the parent.
Since the functions are defined by independent programs, they can be added to at any time without affecting existing functions. This "extensibility" allows the data system to do specifically oceanographic functions (e.g., AOU, dynamic calculations, ...) as well as procedures which might, for example, join two data sets by interpolating one against the other.

Constructed Objects build new objects from old ones



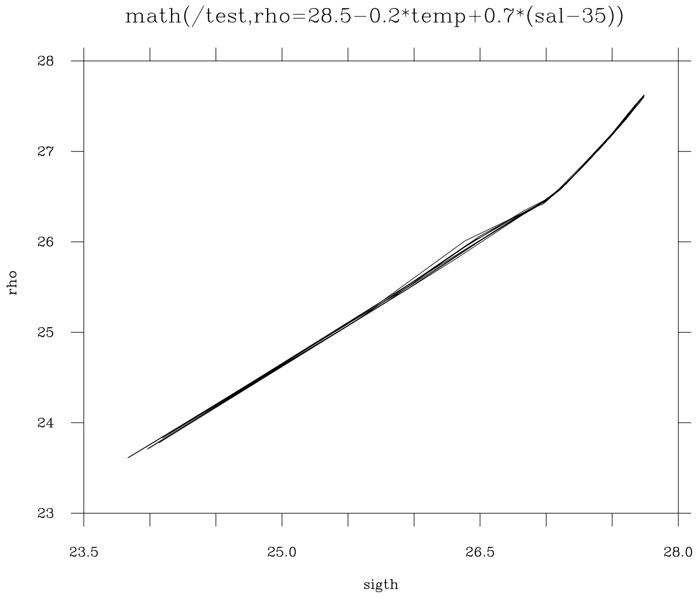
For example, we can add a column to the /test (hydrographic data) which gives a linearized estimate of density:

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

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

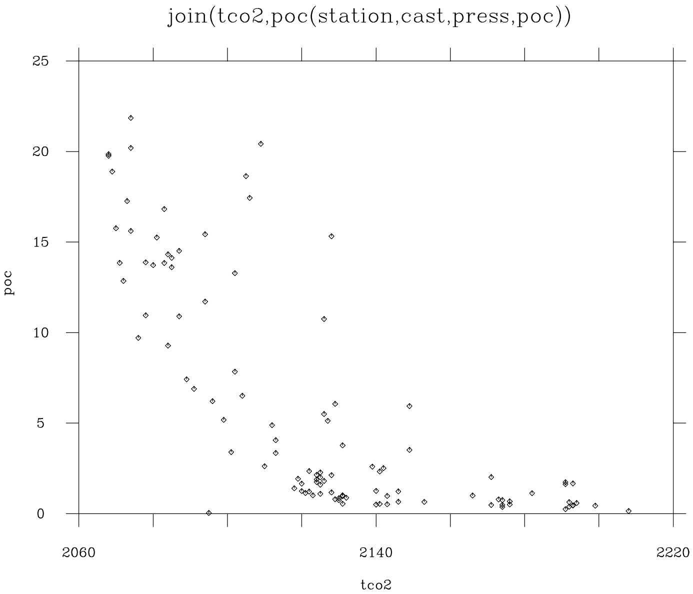
```
math(/test,rho=28.5-0.2*temp+
   0.7*(sal-35))
```

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



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))


join(tco2,poc(station,cast,press,poc))
25
20
15
10
5
0
poc
2060
2140
2220
tco2