EXISTING METHODS

Introduction

The JGOFS data system is built on the idea that associated with each data set is a program, called a "method," which can interpret the data base. One easy way to prepare your data for the system is to set it up in a form that one of the existing methods can read. In this document, we describe the three methods which are supplied with the data system. If you choose to follow one or the other of these, you can work with your data easily and get it online with very little effort.

Default Method — def

Much oceanographic data can be represented as (1) a table of data for each station/ instrument along with (2) header information such as location,event number, etc. The default method allows you to build this kind of structure easily. We'll use an example of hydrographic stations.

Simple Tables — Data from One Station

For each station, there will be a file containing a table of data from that station. This is the simplest data structure. It consists of a header line containing the column titles and a series of data lines:

```
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
```

Here `nd` is used as a missing data indicator. Separators can be spaces, commas, tabs. Extra spaces are ignored.You cannot break rows of names or of data; however, the lines can be 256 characters long. Names can be enclosed in " marks — these will be stripped off (a convenience for LOTUS users).

If you put this in a file `stn3.dat`, you can look at it with

```
list "def(stn3.dat)"
```

or even more simply by

```
list "stn3.dat"
```

Comments

Comments can be included at the beginning of each file; preface them with a hash mark `#`:

```
# 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
```

Variable Attributes

Different variables may have important attributes which are associated with them. Units are the clearest example of this, but one might also want to specify an attribute such as `technique=Winkler` with one of the variables. Although this information can be included in comments, having it as an attribute of the `o2` variable would allow a program to search for this kind of information. Attributes are included in square brackets after the variable name (no space) and are separated by semicolons when there is more than one:

```
# This data prepared by someone
# Measurement at station 21 decibars contaminated
# 2/18/93
depth[m] temp[degC] sal[ppt] oxy[ml/l;technique=Winkler]
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
```

Structured Data — Header Files

To link the individual station files together, one creates a header file. As an example, the object `#test` consists of a header file, `t0` containing the most slowly varying variables, such as `leg`, a set of header files `t1, t2` containing information on the stations in each of the two legs, and then data files `s3--s10` for each station. The variables can be considered as being arranged in a hierarchy

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

A header file record looks like

```
# wunsch stations 3-5
# p<1000
station lat lon >
    press,temp,sal,o2,sigth
3     38.28 -73.53  s3
4     38.19 -73.52  s4
5     38.16 -73.26  s5
```

There are comments and then the list of variables contained in this file. The last one is the symbol >, indicating that the column contains the name of a subfile. This **must** be the last column in the header file. The next line tells the variable names which the subfiles contain. (Initial spaces are ignored.) The subfiles are here assumed to be in the same directory as the header file; however you can put explicit paths or ones relative to the header file's location. This file is `t1` in the data directory of the distribution and can be listed with

```
list ~jgofs/data/t1
```

and you can look at the subfiles with

```
list ~jgofs/data/s3
```

for example.

Header files can be nested up to five levels. The `#test` data set is equivalent to `~jgofs/data/t0` and the top level file looks like

```
# wunsch stations 3-10
# p<1000
leg year month >
    station lat lon >
        press,temp,sal,o2,sigth
1 81   6     t1
2 81   6     t2
```

Again the names section shows the lists of names in the files at inner levels of the hierarchy.

Data attributes can be put with the variable names in the header file (in which case, they are not needed in the subfiles), e.g.

```
# wunsch stations 3-10
# p<1000
leg year month >
    station lat[deg] lon[deg] >
        press[dbar],temp[degC],sal[psu;bottle],o2[ml/l],sigth[ppt]
1 81   6     t1a
2 81   6     t2
```

The simplest way, then, to get data into the system is to put your data for each station in a separate file in tabular form with a header line of names (and comments) and then build of a file of station data (which could be done as part of an event log and shared between users) and then add the column of filename pointers.

Multiple Stations, Single File (spreadsheet) — nm

This method provides a convenient way to put multiple stations in a file, including both header data and cast data. The format includes comments and attributes as well. The header data (and this includes integrated totals as well) are at the beginning of the cast and are written in the form `name=value`. Then there is a line of names for the variables in the table. Succeeding lines have the data values. Blank lines are ignored. Attributes are handled the same way as by the default method; they need not be repeated in repetitions of the names. Here is an example:

```
#  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
99.000, 12.133, 35.508,  5.180, 26.961

station =        7,    lat =   37.93,    lon =  -72.69

press,   temp,    sal,     o2,  sigth

5.000, 17.697, 33.787,  5.760, 24.419
25.000, 15.268, 35.001,  6.320, 25.915
49.000, 14.548, 35.298,  5.620, 26.302
99.000, 12.162, 35.426,  4.710, 26.892
```

Note that spaces around the = sign are ignored, and header variables retain their assignment from table to table, but all the header variables must be given a value in the first table.

Multiple Stations, Single File — lm

The output from `list` can be saved in a file

```
list "#test" listout
```

This gives a single file with multiple stations. You can scan the file by using `more`. But you can also use the data system to access such files if you want to subselect from them. There is a method for reading such files called `lm`. You can list the whole file with

```
list "lm(listout)"
```

or look at restricted parts

```
list "lm(listout,station=7)"
list "lm(listout,press<500)"
list "lm(listout,station,lat,lon)"
```

As in the case of `def`, the first argument to `lm` is a filename; the rest are selections and projections.

A file for `lm` has comments, header variables and values, station variables and values.

```
#  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
=======================
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
***********************
```

Comment lines begin with `#`. The line of one or more equals signs `========` indicates that the next line(s) will be lists of variables (comma or space separated). The line of dots .... separates header variable names from their values. The line of dashes `-----` separates variable names at the innermost level of the hierarchy from their values. Finally the line of asterisks `*****` indicates the end of the file. You can include attributes in the same way as in the default method.