# Unlocking CDAT's Best Kept Secrets

Charles Doutriaux[1]

Dean Williams[1]

Bob Drach[1]

Presented at

Department of Geophysical Sciences

University of Chicago


[1]Lawrence Livermore National Laboratory Livermore, CA

UCRL-PRES-204106
This work was performed under the auspices of the U.S. Department of Energy by the University of California at Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48

# Preliminary Remarks

- Who can benefit from this presentation?
  - Everybody
    - Has a way to discover new CDAT capabilities
  - Experienced Users
    - Has a “cheat-sheet” on how to do things
- Pace of the talk: Relatively Fast
  - You’ll get the talk
  - You can come ask me later
  - Please stop me anyway if needed

# Which kind of secrets ?

- Type of secrets
  - Command line/Scripts
  - Little known/undocumented features
  - Novelties since 3.3
  - Tricks
- Talk organized per Module
  - Python Tips
  - Data I/O
    - cdms
    - Not self described files (ASCII/Binary)
  - Cdutil (climate data specific utilities)
    - times
    - vertical
  - Genutil (general utilities)
    - udunits
    - statusbar
    - misc
  - VCS (graphics)
    - Templates
    - Text
    - Other primitives
    - Projections
  - Contributed Packages
    - Thermodynamic diagrams
    - Misc

# Python Tricks
# prompt

- "_" represents last object returned
- Automatic loading of module, execution of some commands
  - Set environement variable "PYTHONSTARTUP" to a file
  - Type the commands you want to execute at starting time (usually import some modules)
  - Warning: it does not work with python –i script.py
- Automatic completion
  - Add these lines to your startup file
    - import rlcompleter
    - import readline
    - readline.parse_and_bind("tab: complete")

# Python Tricks

- Queries/loops multiple elements
  - if/for v in [1,2,3,4,7]
- Python objects are always instance of something
  - print object.__class__
- Querying object type:
  - if isinstance(a,(int,float,str,list,tuple))
- Most object/function are self documented
  - print obj.__doc__

# DATA I/O: CDMS (1)

- Best way to ingest/write data!
- Opening a file for reading
  - F=cdms.open(*file_name*)
  - It will open an existing file protected against writing
- Opening a new file for writing
  - F=cdms.open(*file_name,'w'*)
  - It will create a new file even if it already exists
- Opening an existing file for writing
  - F=cdms.open(*file_name,'r+'*) # or 'a'
  - It will open an existing file ready for writing or reading

# DATA I/O: CDMS (2)

- Multiple way to retrieve data
  - All of it, omitted dimensions are retrieved entirely
    - s=f('var')
  - Specifying dimension type and values
    - S=f('var', time=(time1,time2))
    - Known types: time, level, latitude, longitude (t,z,y,x)
  - Dimension names and values
    - S=f('var',dimname1=(val1,val2))
  - Sometimes indices are more useful than actual values
    - S=f('var',time=slice(indice1,indice2,step))

# DATA I/O: CDMS (3)

- Special Case: Time dimension
  - Raw values are not necessarily meaningful
  - 2 Solutions
    - Use strings as “value”
      - S=f(var,time=(‘2004’,’2004-4-29 10:30:0.0’))
    - Use cdtime object (cdms doc, Chapter 3, pg 111)
      - T1=cdtime.comptime(2004)
      - T2=cdtime.comptime(2004,4,29,10,30)
      - S=f(var,time=(T1,T2))

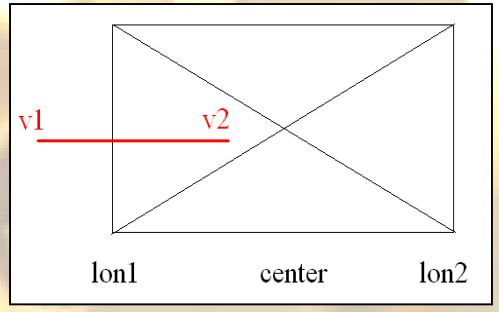
# DATA I/O: CDMS (4)
# The mysterious “third argument”

- OK, we understood s=f(‘var’,time=(t1,t2))
- But what’s the heck is this mysterious 3rd argument defaulted to ‘ccn’ ?
  - The first 2 letters represents the bounds of the retrieved segment they can be “c” or “o” as in “**C**losed” or “**O**pened”:
    - » ‘cc’ : [v1,v2]
    - » ‘co’ : [v1,v2[
    - » ‘oo’ : ]v1, v2[
  - The third letter represents the search method, it can be ‘b’, ’n’, ‘e’ or ‘s’ as in ‘**B**ounds’,’**N**ode’, ‘**E**xtranode’ or ‘**S**elect’
    - i.e the cell will be considered valid if the bounds or node are within the interval defined
    - In the example at left:

      (v1,v2,’ccb’) selects

      (v1,v2,’ccn’) does not select

    - ‘e’: same as n but add an extra node
    - ‘s’: select axis elements for which the cell boundary is a subset of the interval



# DATA I/O: CDMS (5)

- Other known keywords for data ingestion (cdms page 103)
  - squeeze=0/1 # deletes dimensions of length 1
  - order='…zyxt(mydim)…' # Reorders the data
  - cdms selectors
    - Cdutil predefined
    - Cdms doc page 104
      - *from cdms.selectors import Selector*
      - *sel = Selector(time=('1979-1-1','1979-2-1'), level=1000.)*
      - *x1 = v1(sel)*
      - *x2 = v2(sel)*
  - required
  - raw
  - grid

# Data I/O: Not self-describing files (ASCII)

- Use Python “string” Module
- Use VCDAT
- In general use browser.gui_ascii.read
  - `browser.gui_ascii.read( text_file ,header=0, ids=None, shape=None, next='-----',separators=[';',',',':'])`
- Data in columns use browser.gui_ascii_cols.read
  - `bowser.gui_ascii_cols.read( text_file ,header=0, cskip=0, cskip_type='columns', axis=0, ids=None, idrow=0, separators=[';',',', ':'])`

# Data I/O: Not self-describing files (Binary)

- Use Python “struct” Module
- Use VCDAT
- Use browser.gui_read_Struct.read
  - `browser.gui_read_Struct.read( file ,format="", endian='@', datatype='f', ids=[], shape=[], separator="")`:

# cdutil.times

- Climatology, Departures, Anomalies Tools works on BOUNDS, NOT on time values
- 4.0 does not generate time bounds automatically anymore (if not in file)
- In order to fix that use:
  - cdutil.setTimeBoundsMonthly(Obj)
  - cdutil.setTimeBoundsYearly(Obj)
  - cdutil.setTimeBoundsDaily(Obj, frequency=1)
  - Obj can be slab or time axis
- Create your own seasons:
  - DJFM=cdutil.times.Seasons('DJFM')

# cdutil.vertical

- Allows for vertical interpolation
- cdutil.vertical.reconstructPressureFromHybrid
  - Given PS, A, B, P0: P=B*Ps+A*Po
- cdutil.vertical.linearInterpolation(S,I,levels
  - Given S, I( i.e. Pressure/Depth) : Makes linear interpolation to levels
- cdutil.vertical.logLinearInterpolation(S,I,levels)
  - Given S, I( i.e. Pressure/Depth) : Makes log-linear interpolation to levels

# genutil.udunits (or unidata.udunits)

- UNIDATA/UDUNITS Python Object
  - initialization: a=unidata.udunits(value,units)
  - a=unidata.udunits(5,'m')
  - b=unidata.udunits(6,'in')
  - c=a+b # udunits(5.1524,"m")
- CONVERSION
  - a.units='feet' ; print a # 16.4041994751 feet
  - c=a.to('km') # udunits(0.005,"km")
  - c=unidata.udunits(7,'K') ; factor, offset = c.how('degF') # (1.8, -459.67)
- WHICH UNITS ?
  - lst = c.available_units() # returns list of all known units
  - dict = c.known_units() # dictionary: units (keys) / type (values)
  - dict['k'] # returns : 'THERMODYNAMIC TEMPERATURE'
  - dict = c.known_units(bytype=1) # returns a dictionary of units type (keys) associated with a list of units for each type
  - dict['THERMODYNAMIC TEMPERATURE'] # ['degree_Kelvin', 'degree_Celsius', …]

# genutil.statusbar

- For long script with loops or incremental steps it might be usefull to know if how far along you are.

```
for i in range(1000):
    a=genutil.statusbar(i+1.,1000.)
```

- Sometimes you might want a graphical bar

```
prev=-1
for i in range(1000):
    prev=genutil.statusbar(i+1.,1000.,prev=prev, tk=1)
```

# genutil: Miscellaneous

- Xmgrace: Background mode: use “xvfb”
- C,D = genutil.grower(a,b)
  - Adds to a and b any axis not already present that exist in the other
  - E.g: a(x,t) b(y) -> C(x,t,y), D(x,t,y)
  - Order is a dims then b dims
- Min, Max = genutil.minmax(s,[2,34],(s2))
  - Returns the min and max of everything passed to it, independently of type
  - Note: vcs.minmax, identical but masked everything > 1.E20 in absolute value (since they’re not drawn in VCS)
- p=genutil.picker(dim1=list1,dim2=list2,match=1) # Selector
  - Retrieves for each dimensions the actual values passed, not a range
  - match=0, returns missing values if value does not exist.
  - match=1, raise an exception if a value does not exists.
- genutil.colors.rgb2str and genutil.colors.str2rgb
  - Convert between r,g,b values and “names”
- genutil.filters
  - genutil.filters.runningaverage, genutil.filters.smooth12, genutil.filters.custom1D

# Vcs –templates manipulation- X=vcs.init(), T=x.createtemplate('new')

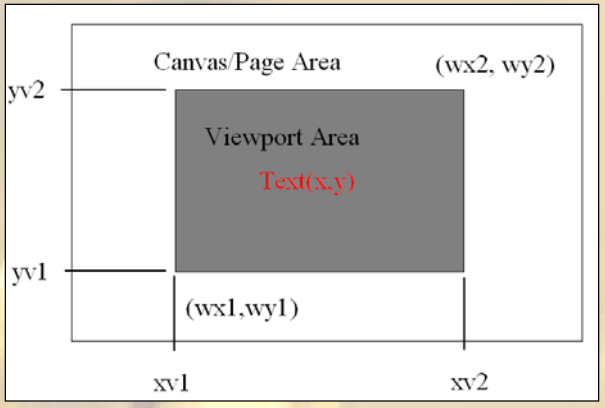
- Template ratio
  - X.ratio=2 : y is twice as big as x
  - X.ratio='auto'
  - X.ratio='2t' : also moves tick marks
- Template scaling (lower left data area unchanged)
  - T.scale(.5) # half size
  - T.scale(.5, axis='x') #half size in X, font unchanged
  - T.scale(.5, axis='x', font=1) # also alter fonts
- Template moving
  - T.move(.2, .4) # move by 20% in x, 40% in y
    - Positive values means up/right
  - T.moveto(x,y) # move lower left corner of data to x,y

# Vcs –primitives (1)- Text

- text=x.createtext('new')

```
----------Text Table (Tt) member (attribute) listings ----------
Tt_name = new
font = 1
spacing = 2
expansion = 100
color = 1
priority = 1
string = None
viewport = [0, 1, 0, 1]
worldcoordinate = [0, 1, 0, 1]
x = None
y = None
projection = default
----------Text Orientation (To) member (attribute) listings ----------
To_name = new
height = 14
angle = 0
path = right
halign = left
valign = half
```

# Vcs –primitives (1)- Others




- fa=x.createfillarea('new')
- l=x.createline('new')
- m=x.createmarker('new')
- Each primitive has the 2 following attributes:
  - Prim.viewport=[xv1,xv2,yv1,yv2] # default: [0,1,0,1]
    - In % of page, area of the primitive extends
  - Prim.worldcoordinates = [x1,x2,y1,y2] # defalut [0,1,0,1]
    - Coordinates corresponding to xv1,xv2,yv1,yv2
    - Primitive units are in the worldcoordinate system
  - Example
    - text.viewport=[.25,.75,.25,.75] # define smaller zone on page
    - text.worldcoordinate=[-180, -90, 180, 90] # Define the coordinate system
    - text.x=[-122.4428 ]
    - text.y=[37.7709 ]
    - text.string=['San Francisco, CA, 94117']
- For overlay with an existing graphic method
  - Prim.viewport # set to your template.data
  - Prim.worldcoordinates # set to your graphic method.data.wc

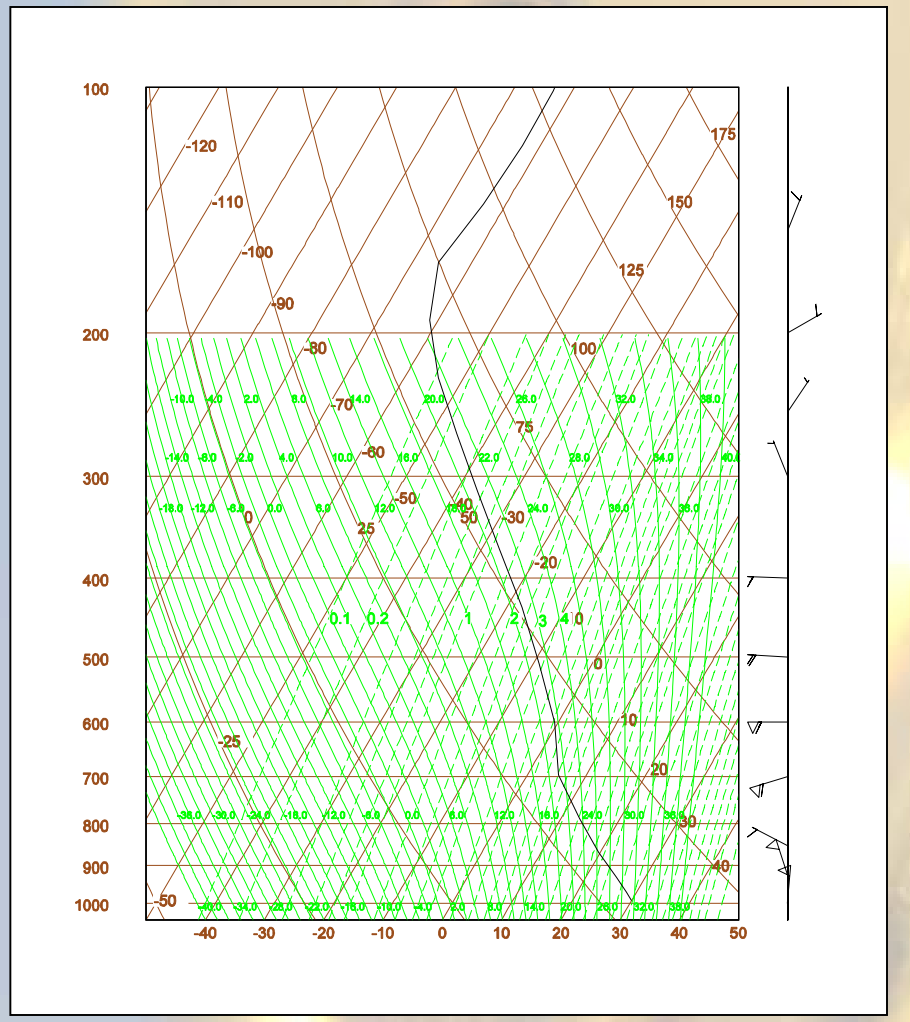
# Vcs –projections-

- P=x.createprojection('new')
- Graphicmethod.projection=P
- P.type=n
  - N can be one of 28 possible
    - print P.__doc__
  - Each type has specific parameters
    - P.list()

# Contributed Packages: Thermodynamic Diagrams

(skewT, emagram, stuve, tephigram, custom)

- import thermo
- th=thermo.Gth(x=x,name='test')
- Entirely customizable
- Lines/fills are vcs graphic method
- Can define your own T,P -> X,Y relation
- Plotting
  - th.plot(t,template=tmpl)
  - T is 1D and axis represents pressure
  - th.plot_windbarb(u,v,P=p)
- Detailed example in source



# Contributed Packages
# Others

- PYCLIMATE: FFT, Filters, etc..
- IaGraph: VCS wrapper
- Pyncl: NCL wrapper
- GMT: GMT wrapper (soon)
- ComparisonStatistics: GNU Fortran compatible

- With special build (ask Dean)
  - R
  - VTK

# Final Note

- Best Sources on “HowTo”?
  1. CDAT Discussion List
  2. Co-Workers
  3. You ! (with some time and effort)
     - If you thought about it, it can be done
     - Do it and share it!
  4. Or ask the developers…
     - Please, no abuse!