VCS Quick Start (V1.0, March 19th, 2001)

Notations

s represents a slab/transient variable
x is the vcs canvas object (i.e. x=vcs.init())

Starting up

```
import vcs   # import the vcs module
x=vcs.init() # initiate the vcs window
x.plot(s) # Plot the slab s and return a "canvas" object that can be altered
```
To change the orientation:
```
x.open()  # pops up the vcs window
x.page() # toggles between portrait and landscape mode
x.portrait() # Set the window to portrait mode
x.landscape() # set the window to landscape mode
x.geometry(widht,height,xoff,yoff) # to specify your own window size
```

Defining VCS Objects and using them

Available graphic methods are:
'boxfill', 'isofill', 'isoline', 'vector', 'xvsy', 'outfill', 'outline', 'scatter', 'xyvsy', 'yxvsx', 'continents'
e.g. to list the 'isofill' methods available:
```
x.show('isofill')
```
To retrieve a graphic method, e.g. boxfill method 'quick':
```
box=x.getboxfill('quick')
```
To create a new graphic method, e.g isoline method named 'new' from 'quick':
```
isol=x.createisoline('new',['quick']), 'quick' is optional if not  passed then the 'default' is used
```
To plot a slab using a graphic method object: e.g "iso" object and s slab : x.plot(s,iso) or x.plot(iso,s), order doesn't matter
Another type of VCS objects is 'template', i.e. where things are plotted in the vcs canvas (see special section for more info)
```
tmpl=x.gettemplate('quick') or tmpl=x.createtemplate('new',['quick'])
```
You can plot in a specific template: e.g slab "s" with graphic method "iso" in the template "tmpl": x.plot(s,iso,tmpl) in any order
Other vcs objects exists, they represent specifications used by graphic-methods or template, these objects are:
'line', 'marker', 'fillarea', 'textorientation' and 'textable'
Once again you can read them in with the get or create new ones with create
e.g.: fa=x.createfillarea('new')
Alternative colormap can be set: x.setcolormap()
These colormap can be altered: e.g. to change the red,green,blue values ofcolor 20:
```
x.setcolorcell(20,red,green,blue)
```

Altering VCS Objects

vcsobject.list() list the modifiable attributes of "vcsobject" and their values
see reference manual for details on the attributes and their functions

Outputing

Image

to the screen: x.plot(s,[method],[template])
to the printer: x.printer('printername')
to plot in the background: add the keyword bg=1 to your plot command
to a file:

- postscript: x.postscript('filename[.ps]',[orientation]), orientation is 'l'/'p'
- eps: x.eps('filename[.ps]',[orientation]), orientation is 'l'/'p'
- gif: x.gif('filename[.gif]',[append],[orientation]), append: 'a'/'r'
- cgm: x.cgm('filename[.cgm]',[append])
- raster: x.raster('filename[.ras]',[append])

VCS object (obj)

A vcs object can be saved as either a python script or a vcs script by using its script() function: iso.script('filename'), vcs script will be saved if 'filename' ends with '.scr'

Examples

Usefull method available to create levels/colors/palette

The VCS module comes with some handy functionalities:

- min,max=vcs.minmax(slab or list of slabs) returns the min and max
- levs=vcs.mkevenlevels(n1,n2, [n]), returns a list of levels going from n1 to n2 and spanning n intervals (i.e. levs has n+1 values), the default value for n is 10 intervals
- levs=vcs.mkscale(n1,n2,[nmax],[zero]), return a list of levels containing n1 and n2, with a maximum of nmax values. Zero determines if zero can be a level (zero=1, default), has to be a level (zero=2) or cannot be a level (zero=-1)
- cols=vcs.getcolors(levs,[cols=range(16,240)],[white=240],[split=1), returns a set of colors spanning the list "cols". If "split" is set to 1 (default) the "cols" list is split in 2 halves one for >0 values, 1 for <0 values, moreover if a contour goes from <0 to >0 it will be set to the color "white" (default=241, i.e. White)
- lbls=vcs.mklabels(levs,output='dic') return a dictionnary containing "nicely" rewritten valus for the levels ex: 0.00000001 -> '1E-8', if output is not starting with 'dic' then a simple list is returned

Playing with the levels/colors/attributes

Let's assume we want to redefine the levels for displaying s, say from 250 up to 300, every 5, with extension arrows. Here is how to do it for different graphic methods:
Thereafter the levels wanted are in the "levs" list and the labels dictionnary in "lbls" and the colors are in "cols"

Boxfil

```
box=x.createboxfill('new')
box.level_1=levs[0]
box.level_2=levs[1]
box.ext_1='y'
box.ext_2='y'
box.legend=lbls
x.plot(s,box)
```

Isofill

```
iso=x.createisofill('new')
levs.append(1.E20)        # to have an extension arrow on the left
levs.insert(0,1.E20)      # to have an extension arrow on the right
iso.levels=levs           # sets the levels
iso.fillareacolors=cols   # set the colors
x.plot(s,iso)
```

Isoline

```
iso=x.createisoline('new')
levs.append(1.E20)
levs.insert(0,1.E20)
iso.level=levs
iso.line='dash', or list of line types
x.plot(s,iso)
```

yvsx (yxvsx or xyvsy), 2D plot (y(x))

```
mx,mn=x.mxmn(y)
levs=x.mkscale(mx,mn)
xy=x.createxvsy('new')
xy.datawc_y1=levs[0]
xy.datawc_y2=levs[-1]
xy.datawx_x1=x[0]
xy.datawc_x2=x[-1]
x.plot(x,y,xy)
```

Altering the Template

To create a vcs template object: tmpl=x.createtemplate('new')

The data area is in: tmpl.data and is represented by the 4 edges x1<x2 and y1<y2, the values are in % of the canvas. For example to plot your data for25 to 75 % of the page horizontally and from 30% to 70% vertically:
```
tmpl.data.x1=.25 ; tmpl.data.x2=.75 ; tmpl.data.y1=.3 ; tmpl.data.y2=.7
```
The box around the data is determined by box1, therefore we should do:
```
tmpl.box1.x1=.25 ; tmpl.box1.x2=.75 ; tmpl.box1.y1=.3 ; tmpl.box1.y2=.7
```
Finally the horizontal tickmarks are set by xtickmark and xtic1 (xtcik2 for the second ones), and the labels by xlabel1 (2)
```
tmpl.xtick1.y1=.29;tmpl.xtic1.y2=.3;tmpl.xlabel1.y=.27
```
Setting the vertical labels works the same way:
```
tmpl.ytick1.x1=.24;tmpl.ytic1.x2=.25;tmpl.ylabel1.x=.22
```
Note the actual values of the labels are set in the vcs graphics method, for example on isofill (but it works the same on ANY graphic method)
iso.xticlabels1=dictionnary of location/names

Finally any attributes of the template can be set on/off using the priority attribute, for example to turn the legend off: tmpl.legend.priority=0

Remember a list of the element of the templates that you can set is accessible using tmpl.list() function. Also if you know which element you wish to set a less exhaustive can be obtained by using the list function of this element for example: tmpl.data.list() will the attribute of "data" zone that you can set.