VCS Documentation

Release 2.4.1

AIMS Team

Sep 26, 2016

CHAPTER

ONE

VCS

1.1 VCS: Visualization and Control System

1.1.1 Authors

Creator: Dean Williams (LLNL, AIMS Team)

Lead Developer: Charles Doutriaux (LLNL, AIMS Team)

Contributors: https://github.com/UV-CDAT/uvcdat/graphs/contributors

Support Email: uvcdat-support@llnl.gov

Project Site: http://uvcdat.llnl.gov/

Project Repo: https://github.com/UV-CDAT/uvcdat/graphs/contributors

1.1.2 Description

VCS is a visualization library for scientific data. It has a simple model for defining a plot, that is decomposed into three parts:

1. **Data**: If it's iterable, we'll plot it... or at least try! Currently we support numpy arrays, lists (nested and not), and CDMS2 variables (there's some special support for metadata from CDMS2 that gives some niceties in your plot, but it's not mandatory).
2. **Graphics Method**: We have a variety of plot types that we support out-of-the box; you can easily customize every aspect of them to create the effect that you're looking for. If you can't, we also support defining your own graphics methods, which you can share with other users using standard python infrastructure (conda, pip).
3. **Template**: Templates control the appearance of everything that *isn't* your data. They position labels, control fonts, adjust borders, place legends, and more. They're very flexible, and give the fine-grained control of your plot that is needed for the truly perfect plot. Once you've customized them, you can also save them out for later use, and distribute them to other users.

vcs.**init**(*mode=1*, *pause_time=0*, *call_from_gui=0*, *size=None*, *backend='vtk'*, *geometry=None*, *bg=None*)

Initialize and construct a VCS Canvas object.

Example

```
import vcs

# Portrait orientation of 1 width per 2 height
portrait = vcs.init(size=.5)
```

```
# also accepts "usletter"
letter = vcs.init(size="letter")
a4 = vcs.init(size="a4")

import vtk
# Useful for embedding VCS inside another application
my_win = vtk.vtkRenderWindow()
embedded = vcs.init(backend=my_win)

dict_init = vcs.init(geometry={"width": 1200, "height": 600})
tuple_init = vcs.init(geometry=(1200, 600))

bg_canvas = vcs.init(bg=True)
```

Parameters

- **size** (*float or case-insensitive str*) – Aspect ratio for canvas (width / height)
- **backend** (str, *vtk.vtkRenderWindow*) – Which VCS backend to use
- **geometry** (*dict or tuple*) – Size (in pixels) you want the canvas to be.
- **bg** (*bool*) – Initialize a canvas to render in "background" mode (without displaying a window)

Returns an initialized canvas

Return type *vcs.Canvas.Canvas*

1.2 Canvas

Canvas The object onto which all plots are drawn.

Usually created using *vcs.init()*, this object provides easy access to the functionality of the entire VCS module.

class vcs.Canvas.**Canvas**(*mode=1*, *pause_time=0*, *call_from_gui=0*, *size=None*, *backend='vtk'*, *geometry=None*, *bg=None*)

The object onto which all plots are drawn.

Usually created using *vcs.init()*, this object provides easy access to the functionality of the entire VCS module.

addfont(*path*, *name=''*)

Add a font to VCS.

Parameters

- **path** (*str*) – Path to the font file you wish to add (must be .ttf)
- **name** (*str*) – Name to use to represent the font.

boxfill(**args*, ***parms*)

Generate a boxfill plot given the data, boxfill graphics method, and template. If no boxfill class object is given, then the 'default' boxfill graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('boxfill') # Show all the existing boxfill graphics␣
↪methods
********************Boxfill Names List**********************
...
********************End Boxfill Names List**********************
>>> box=a.getboxfill('quick') # Create instance of 'quick'
>>> array=[range(10) for _ in range(10)]
>>> a.boxfill(array, box) # Plot array using specified box and␣
↪default template
<vcs.displayplot.Dp ...>
>>> template = a.gettemplate('quick') # get quick template
>>> a.clear() # Clear VCS canvas
>>> a.boxfill(array, box, template) # Plot array using␣
↪specified box and template
<vcs.displayplot.Dp ...>
>>> a.boxfill(box, array, template) # Plot array using␣
↪specified box and template
<vcs.displayplot.Dp ...>
>>> a.boxfill(template, array, box) # Plot array using␣
↪specified box and template
<vcs.displayplot.Dp ...>
>>> a.boxfill(template, box, array) # Plot array using␣
↪specified box and template
<vcs.displayplot.Dp ...>
>>> a.boxfill(array, 'hovmuller', 'quick') # Use 'hovmuller'␣
↪template and 'quick' boxfill
<vcs.displayplot.Dp ...>
>>> a.boxfill('hovmuller', array, 'quick') # Use 'hovmuller'␣
↪template and 'quick' boxfill
<vcs.displayplot.Dp ...>
>>> a.boxfill('hovmuller', 'quick', array) # Use 'hovmuller␣
↪template and 'quick' boxfill
<vcs.displayplot.Dp ...>
```

Note: As shown above, the array, 'template', and 'box' parameters can be provided in any order. The 'template' and 'box' parameters can either be VCS template and boxfill objects, or string names of template and boxfill objects.

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D

- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt

- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **slab** (*array*) – (Ex: [[0, 1]]) Data at least 2D, last 2 dimensions will be plotted

Returns Display Plot object representing the plot.

Return type *vcs.displayplot.Dp*

canvasid(**args*)
Get the ID of this canvas.

This ID number is found at the top of the VCS Canvas, as part of its title.

canvasinfo(**args*, ***kargs*)
Obtain the current attributes of the VCS Canvas window.

Returns Dictionary with keys: "mapstate" (whether the canvas is opened), "height", "width", "depth", "x", "y"

Return type dict

cgm(*file*, *mode='w'*)
Export an image in CGM format.

Parameters

- **file** – Filename to save
- **mode** – Ignored.

change_display_graphic_method(*display*, *type*, *name*)
Changes the type and graphic method of a plot.

Parameters

- **display** (*str or vcs.displayplot.Dp*) – Display to change.
- **type** (*str*) – New graphics method type.
- **name** (*str*) – Name of new graphics method.

check_name_source(*name*, *source*, *typ*)
make sure it is a unique name for this type or generates a name for user

clean_auto_generated_objects(*type=None*)
Cleans up all automatically generated VCS objects.

This function will delete all references to objects that VCS created automatically in response to user actions but are no longer in use. This shouldn't be necessary most of the time, but if you're running into performance/memory issues, calling it periodically may help.

Parameters **type** (*None, str, list/tuple (of str)*) – Type of objects to remove. By default, will remove everything.

clear(**args*, ***kargs*)
Clears all the VCS displays on a page (i.e., the VCS Canvas object).

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.clear() # clear VCS displays from the page
```

close (**args*, ***kargs*)
Close the VCS Canvas. It will not deallocate the VCS Canvas object. To deallocate the VCS Canvas, use the destroy method.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.close() #close the vcs canvas
```

copyfontto (*font1*, *font2*)
Copy 'font1' into 'font2'.

Parameters

- **font1** (*str or int*) – Name/number of font to copy
- **font2** (*str or int*) – Name/number of destination

create3d_dual_scalar (*name=None*, *source='default'*)
Create a new dv3d graphics method given the the name and the existing dv3d graphics method to copy the attributes from. If no existing dv3d graphics method is given, then the default dv3d graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('3d_dual_scalar') # show all available 3d_dual_scalar
********************3d_dual_scalar Names List**********************
...
********************End 3d_dual_scalar Names␣
↪List**********************
>>> ex=vcs.create3d_dual_scalar('3d_dual_scalar_ex1') # Create 3d_
↪dual_scalar '3d_dual_scalar_ex1' that inherits from 'default'
>>> vcs.listelements('3d_dual_scalar') # should now contain the '3d_
↪dual_scalar_ex1' 3d_dual_scalar
[...'3d_dual_scalar_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a 3d_dual_scalar or a string name of a 3d_dual_scalar*) – The object to inherit from

Returns A 3d_dual_scalar graphics method object

Return type vcs.dv3d.Gf3DDualScalar

create3d_scalar (*name=None*, *source='default'*)
Create a new dv3d graphics method given the the name and the existing dv3d graphics method to copy the attributes from. If no existing dv3d graphics method is given, then the default dv3d graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('3d_scalar') # show all available 3d_scalar
********************3d_scalar Names List**********************
...
********************End 3d_scalar Names List**********************
>>> ex=vcs.create3d_scalar('3d_scalar_ex1') # Create 3d_scalar '3d_
↪scalar_ex1' that inherits from 'default'
>>> vcs.listelements('3d_scalar') # should now contain the '3d_
↪scalar_ex1' 3d_scalar
[...'3d_scalar_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a 3d_scalar or a string name of a 3d_scalar*) – The object to inherit from

Returns A 3d_scalar graphics method object

Return type vcs.dv3d.Gf3Dscalar

create3d_vector(*name=None*, *source='default'*)

Create a new dv3d graphics method given the the name and the existing dv3d graphics method to copy the attributes from. If no existing dv3d graphics method is given, then the default dv3d graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('3d_vector') # show all available 3d_vector
********************3d_vector Names List**********************
...
********************End 3d_vector Names List**********************
>>> ex=vcs.create3d_vector('3d_vector_ex1') # Create 3d_vector '3d_
↪vector_ex1' that inherits from 'default'
>>> vcs.listelements('3d_vector') # should now contain the '3d_
↪vector_ex1' 3d_vector
[...'3d_vector_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a 3d_vector or a string name of a 3d_vector*) – The object to inherit from

Returns A 3d_vector graphics method object

Return type vcs.dv3d.Gf3Dvector

createboxfill(*name=None*, *source='default'*)

Create a new boxfill graphics method given the the name and the existing boxfill graphics method to copy the attributes from. If no existing boxfill graphics method is given, then the default boxfill graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('boxfill') # show all available boxfill
********************Boxfill Names List**********************
...
********************End Boxfill Names List**********************
>>> ex=vcs.createboxfill('boxfill_ex1') # Create boxfill 'boxfill_
↪ex1' that inherits from 'default'
>>> vcs.listelements('boxfill') # should now contain the 'boxfill_
↪ex1' boxfill
[...'boxfill_ex1'...]
>>> ex2=vcs.createboxfill('boxfill_ex2','polar') # create 'boxfill_
↪ex2' from 'polar' template
>>> vcs.listelements('boxfill') # should now contain the 'boxfill_
↪ex2' boxfill
[...'boxfill_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a boxfill or a string name of a boxfill*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number

- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A boxfill graphics method object

Return type *vcs.boxfill.Gfb*

createcolormap(*Cp_name=None*, *Cp_name_src='default'*)

Create a new colormap secondary method given the the name and the existing colormap secondary method to copy the attributes from. If no existing colormap secondary method is given, then the default colormap secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('colormap') # show all available colormap
******************Colormap Names List**********************
...
******************End Colormap Names List**********************
>>> ex=vcs.createcolormap('colormap_ex1') # Create colormap
 ↪'colormap_ex1' that inherits from 'default'
>>> vcs.listelements('colormap') # should now contain the 'colormap_
 ↪ex1' colormap
[...'colormap_ex1'...]
>>> ex2=vcs.createcolormap('colormap_ex2','rainbow') # create
 ↪'colormap_ex2' from 'rainbow' template
>>> vcs.listelements('colormap') # should now contain the 'colormap_
 ↪ex2' colormap
[...'colormap_ex2'...]
```

Parameters

- **Cp_name** (*str*) – The name of the created object
- **Cp_name_src** (*a colormap or a string name of a colormap*) – The object to inherit

Returns A VCS colormap object

Return type *vcs.colormap.Cp*

createfillarea(*name=None*, *source='default'*, *style=None*, *index=None*, *color=None*, *priority=1*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

Create a new fillarea secondary method given the the name and the existing fillarea secondary method to copy the attributes from. If no existing fillarea secondary method is given, then the default fillarea secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('fillarea') # show all available fillarea
******************Fillarea Names List**********************
...
******************End Fillarea Names List**********************
>>> ex=vcs.createfillarea('fillarea_ex1') # Create fillarea
 ↪'fillarea_ex1' that inherits from 'default'
```

```
>>> vcs.listelements('fillarea') # should now contain the 'fillarea_
↪ex1' fillarea
[...'fillarea_ex1'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – a fillarea, or string name of a fillarea
- **style** (*str*) – One of "hatch", "solid", or "pattern".
- **index** – Specifies which pattern to fill with.

Accepts ints from 1-20.

Parameters **color** – A color name from the X11 Color Names list,

or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))

Parameters

- **priority** (*int*) – The layer on which the fillarea will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A fillarea object

Return type *vcs.fillarea.Tf*

createisofill(*name=None*, *source='default'*)

Create a new isofill graphics method given the the name and the existing isofill graphics method to copy the attributes from. If no existing isofill graphics method is given, then the default isofill graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('isofill') # show all available isofill
********************Isofill Names List**********************
...
********************End Isofill Names List**********************
>>> ex=vcs.createisofill('isofill_ex1') # Create isofill 'isofill_
↪ex1' that inherits from 'default'
>>> vcs.listelements('isofill') # should now contain the 'isofill_
↪ex1' isofill
[...'isofill_ex1'...]
>>> ex2=vcs.createisofill('isofill_ex2','polar') # create 'isofill_
↪ex2' from 'polar' template
```

```
>>> vcs.listelements('isofill') # should now contain the 'isofill_
↪ex2' isofill
[...'isofill_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*an isofill object, or string name of an isofill object*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot

- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns An isofill graphics method

Return type *vcs.isofill.Gfi*

createisoline(*name=None*, *source='default'*)

Create a new isoline graphics method given the the name and the existing isoline graphics method to copy the attributes from. If no existing isoline graphics method is given, then the default isoline graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('isoline') # show all available isoline
******************Isoline Names List**********************
...
******************End Isoline Names List**********************
>>> ex=vcs.createisoline('isoline_ex1') # Create isoline 'isoline_
↪ex1' that inherits from 'default'
>>> vcs.listelements('isoline') # should now contain the 'isoline_
↪ex1' isoline
[...'isoline_ex1'...]
```

```
>>> ex2=vcs.createisoline('isoline_ex2','polar') # create 'isoline_
↪ex2' from 'polar' template
>>> vcs.listelements('isoline') # should now contain the 'isoline_
↪ex2' isoline
[...'isoline_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*an isoline object, or string name of an isoline object*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)

- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns An isoline graphics method object

Return type *vcs.isoline.Gi*

createline(*name=None*, *source='default'*, *ltype=None*, *width=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *projection=None*)

Create a new line secondary method given the the name and the existing line secondary method to copy the attributes from. If no existing line secondary method is given, then the default line secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('line') # show all available line
********************Line Names List**********************
...
********************End Line Names List**********************
>>> ex=vcs.createline('line_ex1') # Create line 'line_ex1' that␣
↪inherits from 'default'
```

```
>>> vcs.listelements('line') # should now contain the 'line_ex1'␣
↪line
[...'line_ex1'...]
>>> ex2=vcs.createline('line_ex2','red') # create 'line_ex2' from
↪'red' template
>>> vcs.listelements('line') # should now contain the 'line_ex2'␣
↪line
[...'line_ex2'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – a line, or string name of a line
- **ltype** (*str*) – One of "dash", "dash-dot", "solid", "dot", or "long-dash".
- **width** (*int*) – Thickness of the line to be created
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the line will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS line secondary method object

Return type *vcs.line.Tl*

createmarker(*name=None*, *source='default'*, *mtype=None*, *size=None*, *color=None*, *priority=1*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *projection=None*)

Create a new marker secondary method given the the name and the existing marker secondary method to copy the attributes from. If no existing marker secondary method is given, then the default marker secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('marker') # show all available marker
******************Marker Names List**********************
...
******************End Marker Names List**********************
>>> ex=vcs.createmarker('marker_ex1') # Create marker 'marker_ex1'␣
↪that inherits from 'default'
>>> vcs.listelements('marker') # should now contain the 'marker_ex1
↪' marker
```

```
[...'marker_ex1'...]
>>> ex2=vcs.createmarker('marker_ex2','red') # create 'marker_ex2'␣
↪from 'red' template
>>> vcs.listelements('marker') # should now contain the 'marker_ex2
↪' marker
[...'marker_ex2'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – A marker, or string name of a marker
- **mtype** (*str*) – Specifies the type of marker, i.e. “dot”, “circle”
- **size** (*int*) –
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A secondary marker method

Return type *vcs.marker.Tm*

createmeshfill (*name=None*, *source='default'*)

Create a new meshfill graphics method given the the name and the existing meshfill graphics method to copy the attributes from. If no existing meshfill graphics method is given, then the default meshfill graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('meshfill') # show all available meshfill
******************Meshfill Names List**********************
...
******************End Meshfill Names List**********************
>>> ex=vcs.createmeshfill('meshfill_ex1') # Create meshfill
↪'meshfill_ex1' that inherits from 'default'
>>> vcs.listelements('meshfill') # should now contain the 'meshfill_
↪ex1' meshfill
[...'meshfill_ex1'...]
>>> ex2=vcs.createmeshfill('meshfill_ex2','a_polar_meshfill') #␣
↪create 'meshfill_ex2' from 'a_polar_meshfill' template
```

```
>>> vcs.listelements('meshfill') # should now contain the 'meshfill_
↪ex2' meshfill
[...'meshfill_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a meshfill or a string name of a meshfill*) – The object to inherit from

Returns A meshfill graphics method object

Return type *vcs.meshfill.Gfm*

createprojection(*name=None*, *source='default'*)

Create a new projection graphics method given the the name and the existing projection graphics method to copy the attributes from. If no existing projection graphics method is given, then the default projection graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('projection') # show all available projection
******************Projection Names List*********************
...
******************End Projection Names List*********************
>>> ex=vcs.createprojection('projection_ex1') # Create projection
↪'projection_ex1' that inherits from 'default'
>>> vcs.listelements('projection') # should now contain the
↪'projection_ex1' projection
[...'projection_ex1'...]
>>> ex2=vcs.createprojection('projection_ex2','polar') # create
↪'projection_ex2' from 'polar' template
>>> vcs.listelements('projection') # should now contain the
↪'projection_ex2' projection
[...'projection_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a projection or a string name of a projection*) – The object to inherit from

Returns A projection graphics method object

Return type *vcs.projection.Proj*

createscatter(*name=None*, *source='default'*)

Create a new scatter graphics method given the the name and the existing scatter graphics method to copy the attributes from. If no existing scatter graphics method is given, then the default scatter graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('scatter') # show all available scatter
******************Scatter Names List**********************
...
******************End Scatter Names List**********************
>>> ex=vcs.createscatter('scatter_ex1') # Create scatter 'scatter_
↪ex1' that inherits from 'default'
>>> vcs.listelements('scatter') # should now contain the 'scatter_
↪ex1' scatter
[...'scatter_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a scatter or a string name of a scatter*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot

- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A scatter graphics method

Return type *vcs.unified1D.G1d*

createtaylordiagram(*name=None*, *source='default'*)

Create a new taylordiagram graphics method given the the name and the existing taylordiagram graphics method to copy the attributes from. If no existing taylordiagram graphics method is given, then the default taylordiagram graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('taylordiagram') # show all available taylordiagram
******************Taylordiagram Names List**********************
...
******************End Taylordiagram Names␣
↪List**********************
>>> ex=vcs.createtaylordiagram('taylordiagram_ex1') # Create␣
↪taylordiagram 'taylordiagram_ex1' that inherits from 'default'
>>> vcs.listelements('taylordiagram') # should now contain the
↪'taylordiagram_ex1' taylordiagram
[...'taylordiagram_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a taylordiagram or a string name of a*) – The object to inherit from

Returns A taylordiagram graphics method object

Return type *vcs.taylor.Gtd*

createtemplate(*name=None*, *source='default'*)

Create a new template graphics method given the the name and the existing template graphics method to copy the attributes from. If no existing template graphics method is given, then the default template graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('template') # show all available template
******************Template Names List**********************
...
******************End Template Names List**********************
>>> ex=vcs.createtemplate('template_ex1') # Create template
↪'template_ex1' that inherits from 'default'
>>> vcs.listelements('template') # should now contain the 'template_
↪ex1' template
[...'template_ex1'...]
>>> ex2=vcs.createtemplate('template_ex2','polar') # create
↪'template_ex2' from 'polar' template
>>> vcs.listelements('template') # should now contain the 'template_
↪ex2' template
[...'template_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object

- **source** (*a template or a string name of a template*) – The object to inherit from

Returns A template

Return type *vcs.template.P*

createtext(*Tt_name=None*, *Tt_source='default'*, *To_name=None*, *To_source='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*, *projection=None*)

Create a new textcombined secondary method given the the name and the existing textcombined secondary method to copy the attributes from. If no existing textcombined secondary method is given, then the default textcombined secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('textcombined') # show all available textcombined
******************Textcombined Names List*********************
...
******************End Textcombined Names List**********************
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> vcs.listelements('textcombined') # should now contain the 'qa_
↪tt:::left_tto' textcombined
[...'qa_tt:::left_tto'...]
```

Parameters

- **Tt_name** (*str*) – Name of created object
- **Tt_source** (*str or vcs.texttable.Tt*) – Texttable object to inherit from. Can be a texttable, or a string name of a texttable.
- **To_name** (*str*) – Name of the textcombined's text orientation (to be created)
- **To_source** (*str or vcs.textorientation.To*) – Name of the textorientation to inherit. Can be a textorientation, or a string name of a textorientation.
- **font** (*int or str*) – Which font to use (index or name).
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)

- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*int*) – Angle of the text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of ["left", "center", "right"].
- **valign** (*str*) – Vertical alignment of the text. One of ["top", "center", "botom"].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS text object

Return type *vcs.textcombined.Tc*

createtextcombined(*Tt_name=None*, *Tt_source='default'*, *To_name=None*, *To_source='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*, *projection=None*)

Create a new textcombined secondary method given the the name and the existing textcombined secondary method to copy the attributes from. If no existing textcombined secondary method is given, then the default textcombined secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('textcombined') # show all available textcombined
******************Textcombined Names List**********************
...
******************End Textcombined Names List**********************
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> vcs.listelements('textcombined') # should now contain the 'qa_
↪tt:::left_tto' textcombined
[...'qa_tt:::left_tto'...]
```

Parameters

- **Tt_name** (*str*) – Name of created object
- **Tt_source** (*str or vcs.texttable.Tt*) – Texttable object to inherit from. Can be a texttable, or a string name of a texttable.
- **To_name** (*str*) – Name of the textcombined's text orientation (to be created)
- **To_source** (*str or vcs.textorientation.To*) – Name of the textorientation to inherit. Can be a textorientation, or a string name of a textorientation.
- **font** (*int or str*) – Which font to use (index or name).

- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*int*) – Angle of the text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of ["left", "center", "right"].
- **valign** (*str*) – Vertical alignment of the text. One of ["top", "center", "botom"].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS text object

Return type *vcs.textcombined.Tc*

createtextorientation(*name=None*, *source='default'*)

Create a new textorientation secondary method given the the name and the existing textorientation secondary method to copy the attributes from. If no existing textorientation secondary method is given, then the default textorientation secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('textorientation') # show all available textorientation
******************Textorientation Names List**********************
...
******************End Textorientation Names␣
↪List***********************
>>> ex=vcs.createtextorientation('textorientation_ex1') # Create␣
↪textorientation 'textorientation_ex1' that inherits from 'default'
>>> vcs.listelements('textorientation') # should now contain the
↪'textorientation_ex1' textorientation
[...'textorientation_ex1'...]
>>> ex2=vcs.createtextorientation('textorientation_ex2','bigger') #␣
↪create 'textorientation_ex2' from 'bigger' template
```

```
>>> vcs.listelements('textorientation') # should now contain the
↪'textorientation_ex2' textorientation
[...'textorientation_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a textorientation or a string name of a textorientation*) – The object to inherit from

Returns A textorientation secondary method

Return type *vcs.textorientation.To*

createtexttable(*name=None*, *source='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

Create a new texttable secondary method given the the name and the existing texttable secondary method to copy the attributes from. If no existing texttable secondary method is given, then the default texttable secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('texttable') # show all available texttable
******************Texttable Names List**********************
...
******************End Texttable Names List**********************
>>> ex=vcs.createtexttable('texttable_ex1') # Create texttable
↪'texttable_ex1' that inherits from 'default'
>>> vcs.listelements('texttable') # should now contain the
↪'texttable_ex1' texttable
[...'texttable_ex1'...]
>>> ex2=vcs.createtexttable('texttable_ex2','bigger') # create
↪'texttable_ex2' from 'bigger' template
>>> vcs.listelements('texttable') # should now contain the
↪'texttable_ex2' texttable
[...'texttable_ex2'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – a texttable, or string name of a texttable
- **font** (*int or string*) – Which font to use (index or name).
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the texttable will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas

- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A texttable graphics method object

Return type *vcs.texttable.Tt*

createvector (*name=None*, *source='default'*)

Create a new vector graphics method given the the name and the existing vector graphics method to copy the attributes from. If no existing vector graphics method is given, then the default vector graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('vector') # show all available vector
********************Vector Names List**********************
...
********************End Vector Names List**********************
>>> ex=vcs.createvector('vector_ex1') # Create vector 'vector_ex1'␣
↪that inherits from 'default'
>>> vcs.listelements('vector') # should now contain the 'vector_ex1
↪' vector
[...'vector_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a vector or a string name of a vector*) – The object to inherit from

Returns A vector graphics method object

Return type *vcs.vector.Gv*

createxvsy (*name=None*, *source='default'*)

Create a new xvsy graphics method given the the name and the existing xvsy graphics method to copy the attributes from. If no existing xvsy graphics method is given, then the default xvsy graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('xvsy') # show all available xvsy
********************Xvsy Names List**********************
...
```

```
********************End Xvsy Names List**********************
>>> ex=vcs.createxvsy('xvsy_ex1') # Create xvsy 'xvsy_ex1' that␣
↪inherits from 'default'
>>> vcs.listelements('xvsy') # should now contain the 'xvsy_ex1'␣
↪xvsy
[...'xvsy_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a xvsy or a string name of a xvsy*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)

- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A XvsY graphics method object

Return type *vcs.unified1D.G1d*

createxyvsy(*name=None*, *source='default'*)

Create a new xyvsy graphics method given the the name and the existing xyvsy graphics method to copy the attributes from. If no existing xyvsy graphics method is given, then the default xyvsy graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('xyvsy') # show all available xyvsy
********************Xyvsy Names List**********************
...
********************End Xyvsy Names List**********************
>>> ex=vcs.createxyvsy('xyvsy_ex1') # Create xyvsy 'xyvsy_ex1' that␣
↪inherits from 'default'
```

```
>>> vcs.listelements('xyvsy') # should now contain the 'xyvsy_ex1'␣
↪xyvsy
[...'xyvsy_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a xyvsy or a string name of a xyvsy*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot

- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A XYvsY graphics method object

Return type *vcs.unified1D.G1d*

createyxvsx(*name=None*, *source='default'*)

Create a new yxvsx graphics method given the the name and the existing yxvsx graphics method to copy the attributes from. If no existing yxvsx graphics method is given, then the default yxvsx graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('yxvsx') # show all available yxvsx
********************Yxvsx Names List**********************
...
********************End Yxvsx Names List**********************
>>> ex=vcs.createyxvsx('yxvsx_ex1') # Create yxvsx 'yxvsx_ex1' that␣
↪inherits from 'default'
>>> vcs.listelements('yxvsx') # should now contain the 'yxvsx_ex1'␣
↪yxvsx
[...'yxvsx_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a yxvsy or a string name of a yxvsy*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)

- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A YXvsX graphics method object

Return type *vcs.unified1D.G1d*

destroy()
Destroy the VCS Canvas. It will deallocate the VCS Canvas object.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.destroy()
```

drawfillarea(*name=None, style=1, index=1, color=241, priority=1, viewport=[0.0, 1.0, 0.0, 1.0], worldcoordinate=[0.0, 1.0, 0.0, 1.0], x=None, y=None, bg=0*)
Generate and draw a fillarea object on the VCS Canvas.

Example

```
>>> a=vcs.init()
>>> a.show('fillarea') # Show all the existing fillarea objects
********************Fillarea Names List**********************
...
********************End Fillarea Names List**********************
>>> fa=a.drawfillarea(name='red', style=1, color=242,
...                   priority=1, viewport=[0, 1.0, 0, 1.0],
...                   worldcoordinate=[0,100, 0,50],
```

```
...                 x=[0,20,40,60,80,100],
...                 y=[0,10,20,30,40,50], bg=0 ) # Create instance of␣
↪fillarea object 'red'
>>> a.fillarea(fa) # Plot using specified fillarea object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object
- **style** (*str*) – One of "hatch", "solid", or "pattern".
- **index** – Specifies which pattern

to fill the fillarea with. Accepts ints from 1-20.

Parameters **color** – A color name from the X11 Color Names list,

or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))

Parameters

- **priority** (*int*) – The layer on which the fillarea will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **bg** (*bool*) – Boolean value. True => object drawn in background (not shown on canvas). False => object shown on canvas.

Returns A fillarea object

Return type *vcs.fillarea.Tf*

drawline(*name=None, ltype='solid', width=1, color=241, priority=1, viewport=[0.0, 1.0, 0.0, 1.0], worldcoordinate=[0.0, 1.0, 0.0, 1.0], x=None, y=None, projection='default', bg=0*)

Generate and draw a line object on the VCS Canvas.

Example

```
>>> a=vcs.init()
>>> a.show('line') # Show all the existing line objects
********************Line Names List**********************
...
********************End Line Names List**********************
>>> ln=a.drawline(name='red', ltype='dash', width=2,
...               color=242, priority=1, viewport=[0, 1.0, 0, 1.0],
...               worldcoordinate=[0,100, 0,50],
...               x=[0,20,40,60,80,100],
...               y=[0,10,20,30,40,50] )
>>> a.line(ln) # Plot using specified line object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object

- **ltype** (*str*) – One of “dash”, “dash-dot”, “solid”, “dot”, or “long-dash”.
- **width** (*int*) – Thickness of the line to be drawn
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the line will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS line object

Return type *vcs.line.Tl*

drawlogooff()
Hide UV-CDAT logo on the canvas

Example

```
>>> a=vcs.init()
>>> a.drawlogooff()
>>> a.getdrawlogo()
False
```

drawlogoon()
Show UV-CDAT logo on the canvas

Example

```
>>> a=vcs.init()
>>> a.drawlogoon()
>>> a.getdrawlogo()
True
```

drawmarker(*name=None, mtype='solid', size=1, color=241, priority=1, viewport=[0.0, 1.0, 0.0, 1.0], worldcoordinate=[0.0, 1.0, 0.0, 1.0], x=None, y=None, bg=0*)
Generate and draw a marker object on the VCS Canvas.

Example

```
>>> a=vcs.init()
>>> a.show('marker')  # Show all the existing marker objects
******************Marker Names List**********************
...
*******************End Marker Names List**********************
>>> mrk=a.drawmarker(name='red', mtype='dot', size=2,
...                  color=242, priority=1, viewport=[0, 1.0, 0, 1.0],
...                  worldcoordinate=[0,100, 0,50],
...                  x=[0,20,40,60,80,100],
...                  y=[0,10,20,30,40,50] ) # Create instance of marker␣
↪object 'red'
```

```
>>> a.marker(mrk) # Plot using specified marker object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object
- **mtype** (*str*) – Marker type, i.e. 'dot', 'plus', 'star, etc.
- **size** (*int*) – Size of the marker to draw
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A drawmarker object

Return type *vcs.marker.Tm*

drawtext(*Tt_name=None, To_name=None, string=None, font=1, spacing=2, expansion=100, color=241, height=14, angle=0, path='right', halign='left', valign='half', priority=1, viewport=[0.0, 1.0, 0.0, 1.0], worldcoordinate=[0.0, 1.0, 0.0, 1.0], x=None, y=None, bg=0*)

Generate and draw a textcombined object on the VCS Canvas.

Example

```
>>> a=vcs.init()
>>> a.show('texttable')  # Show all the existing texttable objects
********************Texttable Names List**********************
...
********************End Texttable Names List**********************
>>> tc=a.drawtextcombined(Tt_name = 'std_example', To_name='7left_
↪example', string='Hello example!', spacing=5,
...                    color=242, priority=1, viewport=[0, 1.0, 0, 1.
↪0],
...                    worldcoordinate=[0,100, 0,50],
...                    x=[0,20,40,60,80,100],
...                    y=[0,10,20,30,40,50]) # Create instance of␣
↪texttable object 'red'
>>> a.textcombined(tc) # Plot using specified texttable object
```

Parameters

- **name** (*str*) – Name of created object
- **style** (*str*) – One of "hatch", "solid", or "pattern".
- **index** (*int*) – Specifies which pattern to fill the fillarea with. Accepts ints from 1-20.
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))

- **priority** (*int*) – The layer on which the fillarea will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **bg** (*bool*) – Boolean value. True => object drawn in background (not shown on canvas). False => object shown on canvas.

Returns A texttable object

Return type *vcs.texttable.Tt*

drawtextcombined(*Tt_name=None, To_name=None, string=None, font=1, spacing=2, expansion=100, color=241, height=14, angle=0, path='right', halign='left', valign='half', priority=1, viewport=[0.0, 1.0, 0.0, 1.0], worldcoordinate=[0.0, 1.0, 0.0, 1.0], x=None, y=None, bg=0*)

Generate and draw a textcombined object on the VCS Canvas.

Example

```
>>> a=vcs.init()
>>> a.show('texttable')  # Show all the existing texttable objects
********************Texttable Names List**********************
...
********************End Texttable Names List**********************
>>> tc=a.drawtextcombined(Tt_name = 'std_example', To_name='7left_
↪example', string='Hello example!', spacing=5,
...                    color=242, priority=1, viewport=[0, 1.0, 0, 1.
↪0],
...                    worldcoordinate=[0,100, 0,50],
...                    x=[0,20,40,60,80,100],
...                    y=[0,10,20,30,40,50]) # Create instance of␣
↪texttable object 'red'
>>> a.textcombined(tc) # Plot using specified texttable object
```

Parameters

- **name** (*str*) – Name of created object
- **style** (*str*) – One of "hatch", "solid", or "pattern".
- **index** (*int*) – Specifies which pattern to fill the fillarea with. Accepts ints from 1-20.
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the fillarea will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].

- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **bg** (*bool*) – Boolean value. True => object drawn in background (not shown on canvas). False => object shown on canvas.

Returns A texttable object

Return type *vcs.texttable.Tt*

eps (*file*, *mode='r'*, *orientation=None*, *width=None*, *height=None*, *units='inches'*, *textAsPaths=True*)
In some cases, the user may want to save the plot out as an Encapsulated PostScript image. This routine allows the user to save the VCS canvas output as an Encapsulated PostScript file. This file can be converted to other image formats with the aid of xv and other such imaging tools found freely on the web.

Example

```
>>> a=vcs.init()
>>> array = [range(10) for _ in range(10)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.postscript('example') # Overwrite a postscript file
>>> a.postscript('example', 'a') # Append postscript to an existing␣
↪file
>>> a.postscript('example', 'r') # Overwrite an existing file
>>> a.postscript('example', mode='a') # Append postscript to an␣
↪existing file
>>> a.postscript('example', width=11.5, height= 8.5) # US Legal␣
↪(default)
>>> a.postscript('example', width=21, height=29.7, units='cm') # A4
```

Parameters

- **file** (*str*) – String name of the desired output file
- **mode** (*str*) – The mode in which to open the file. One of 'r' or 'a'.
- **orientation** (*None*) – Deprecated.
- **width** (*float*) – Width of the output image, in the unit of measurement specified
- **height** (*float*) – Height of the output image, in the unit of measurement specified
- **units** (*str*) – One of ['inches', 'in', 'cm', 'mm', 'pixel', 'pixels', 'dot', 'dots']. Defaults to 'inches'.

ffmpeg (*movie*, *files*, *bitrate=1024*, *rate=None*, *options=None*)
MPEG output from a list of valid files. Can output to more than just mpeg format.

Note: ffmpeg ALWAYS overwrites the output file

Audio configuration

via the options arg you can add audio file to your movie (see ffmpeg help)

Example

```
>>> a=vcs.init()
>>> import cdms2
>>> f = cdms2.open(vcs.sample_data+'/clt.nc')
>>> v = f('v') # use the data file to create a cdms2 slab
>>> u = f('u') # use the data file to create a cdms2 slab
>>> png_files = [] # for saving file names to make the mpeg
>>> plots = [] # for saving plots for later reference
>>> for i in range(10): # create a number of pngs to use for an mpeg
...     a.clear()
...     if (i%2):
...         plots.append(a.plot(u,v))
...     else:
...         plots.append(a.plot(v,u))
...     a.png('my_png__%i' % i)
...     png_files.append('my_png__%i.png' % i)
>>> a.ffmpeg('mymovie.mpeg',png_files) # generates from list of␣
↪files
True
>>> a.ffmpeg('mymovie.mpeg',files="my_png__[0-9]*\.png") # generate␣
↪from files with name matching regex
True
>>> a.ffmpeg('mymovie.mpeg',png_files,bitrate=512) # generates mpeg␣
↪at 512kbit
True
>>> a.ffmpeg('mymovie.mpeg',png_files,rate=50) # generates movie␣
↪with 50 frame per second
True
```

Parameters

- **movie** (*str*) – Output video file name
- **files** (*str, list, or tuple*) – String file name
- **rate** (*str*) – Desired output framerate
- **options** (*str*) – Additional FFMPEG arguments

Returns The output string generated by ffmpeg program

Return type str

fillarea(**args*, ***parms*)

Generate a fillarea plot

Plot a fillarea segment on the Vcs Canvas. If no fillarea class object is given, then an error will be returned.

Example

```
>>> a=vcs.init()
>>> a.show('fillarea') # Show all the existing fillarea objects
********************Fillarea Names List**********************
...
********************End Fillarea Names List**********************
>>> fa=a.createfillarea() # Create instance of default fillarea
>>> fa.style=1 # Set the fillarea style
>>> fa.index=4 # Set the fillarea index
>>> fa.color = 242 # Set the fillarea color
>>> fa.x=[0.25,0.75] # Set the x value points
>>> fa.y=[0.2,0.5] # Set the y value points
```

```
>>> a.fillarea(fa) # Plot using specified fillarea object
<vcs.displayplot.Dp ...>
```

Returns A fillarea object

Return type *vcs.displayplot.Dp*

flush(**args*)
The flush command executes all buffered X events in the queue.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.flush()
```

geometry(**args*)
The geometry command is used to set the size and position of the VCS canvas.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.geometry(450,337)
```

get3d_dual_scalar(*Gfdv3d_name_src='default'*)
VCS contains a list of graphics methods. This function will create a dv3d class object from an existing VCS dv3d graphics method. If no dv3d name is given, then dv3d 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.create3d_dual_scalar()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('3d_dual_scalar') # Show all the existing 3d_
↪dual_scalar graphics methods
[...]
>>> ex=vcs.get3d_dual_scalar()  # instance of 'default' 3d_dual_
↪scalar graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.plot(ex, slab1, slab2) # plot using specified 3d_dual_scalar␣
↪object
<vcs.displayplot.Dp ...>
```

Parameters **Gfdv3d_name_src** (*str*) – String name of an existing 3d_dual_scalar VCS object

Returns A pre-existing 3d_dual_scalar VCS object

Return type vcs.dv3d.Gf3DDualScalar

get3d_scalar(*Gfdv3d_name_src='default'*)

VCS contains a list of graphics methods. This function will create a dv3d class object from an existing VCS dv3d graphics method. If no dv3d name is given, then dv3d 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.create3d_scalar()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('3d_scalar') # Show all the existing 3d_scalar␣
↪graphics methods
[...]
>>> ex=vcs.get3d_scalar()  # instance of 'default' 3d_scalar␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.plot(ex, slab1) # plot using specified 3d_scalar object
<vcs.displayplot.Dp ...>
```

Parameters **Gfdv3d_name_src** (*str*) – String name of an existing 3d_scalar VCS object.

Returns A pre-existing 3d_scalar VCS object

Return type vcs.dv3d.Gf3Dscalar

get3d_vector(*Gfdv3d_name_src='default'*)

VCS contains a list of graphics methods. This function will create a dv3d class object from an existing VCS dv3d graphics method. If no dv3d name is given, then dv3d 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.create3d_vector()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('3d_vector') # Show all the existing 3d_vector␣
↪graphics methods
[...]
>>> ex=vcs.get3d_vector()  # instance of 'default' 3d_vector␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
```

```
>>> a.plot(ex, slab1, slab2) # plot using specified 3d_vector object
<vcs.displayplot.Dp ...>
```

Parameters **Gfdv3d_name_src** (*str*) – String name of an existing 3d_vector VCS object

Returns A pre-existing 3d_vector VCS object

Return type vcs.dv3d.Gf3Dvector

get_selected_display()
Deprecated since version 2.0: This function is no longer supported.

getboxfill(*Gfb_name_src='default'*)
VCS contains a list of graphics methods. This function will create a boxfill class object from an existing VCS boxfill graphics method. If no boxfill name is given, then boxfill 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createboxfill()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('boxfill') # Show all the existing boxfill␣
↪graphics methods
[...]
>>> ex=vcs.getboxfill()  # instance of 'default' boxfill graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.boxfill(ex, slab1) # plot using specified boxfill object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getboxfill('polar')  # instance of 'polar' boxfill␣
↪graphics method
>>> a.boxfill(ex2, slab1) # plot using specified boxfill object
<vcs.displayplot.Dp ...>
```

Parameters

- **Gfb_name_src** (*str*) – String name of an existing boxfill VCS object
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis

- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks

- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A pre-existing boxfill graphics method

Return type *vcs.boxfill.Gfb*

getcolorcell(**args*)

Gets the colorcell of the provided object's colormap at the specified cell index. If no object is provided, or if the provided object has no colormap, the default colormap is used.

Example

```
>>> a=vcs.init()
>>> b=vcs.createboxfill()
>>> b.colormap='rainbow'
>>> a.getcolorcell(2,b)
[85, 85, 85, 100.0]
```

Parameters

- **cell** (*int*) – An integer value indicating the index of the desired colorcell.
- **obj** (*Any VCS object capable of containing a colormap*) – Optional parameter containing the object to extract a colormap from.

Returns The RGBA values of the colormap at the specified cell index.

Return type *list*

getcolormap(*Cp_name_src='default'*)

VCS contains a list of secondary methods. This function will create a colormap class object from an existing VCS colormap secondary method. If no colormap name is given, then colormap 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createcolormap()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('colormap') # Show all the existing colormap␣
↪secondary methods
[...]
>>> ex=vcs.getcolormap()  # instance of 'default' colormap␣
↪secondary method
>>> ex2=vcs.getcolormap('rainbow')  # instance of 'rainbow'␣
↪colormap secondary method
```

Parameters **Cp_name_src** (*str*) – String name of an existing colormap VCS object

Returns A pre-existing VCS colormap object

Return type *vcs.colormap.Cp*

getcolormapname()
Returns the name of the colormap this canvas is set to use by default.

To set that colormap, use **:ref:'setcolormap'_**.

getcontinentstype(**args*)
Retrieve continents type from VCS; either an integer between 0 and 11 or the path to a custom continentstype.

Example

```
>>> a=vcs.init()
>>> cont_type = a.getcontinentstype() # Get the continents type
```

Returns An int between 1 and 0, or the path to a custom continentstype

Return type int or system filepath

getdrawlogo()
Returns value of draw logo. By default, draw logo is set to True.

Example

```
>>> a=vcs.init()
>>> a.getdrawlogo()
True
>>> a.drawlogooff()
>>> a.getdrawlogo()
False
```

Returns Boolean value of system variable which indicates whether logo will be drawn

Return type bool

getfillarea(*name='default'*, *style=None*, *index=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)
VCS contains a list of secondary methods. This function will create a fillarea class object from an existing VCS fillarea secondary method. If no fillarea name is given, then fillarea 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createfillarea()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('fillarea') # Show all the existing fillarea␣
↪secondary methods
[...]
>>> ex=vcs.getfillarea()  # instance of 'default' fillarea␣
↪secondary method
>>> a.fillarea(ex) # plot using specified fillarea object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – String name of an existing fillarea VCS object
- **style** (*str*) – One of "hatch", "solid", or "pattern".

- **index** (*int*) – Specifies which pattern to fill with. Accepts ints from 1-20.
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the texttable will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A fillarea secondary object

Return type *vcs.fillarea.Tf*

getfont (*font*)
Get the font name/number associated with a font number/name

Example

```
>>> a=vcs.init()
>>> font_names=[]
>>> for i in range(1,17):
...     font_names.append(str(a.getfont(i))) # font_names is now␣
↪filled with all font names
>>> font_names
['default', ...]
>>> font_numbers = []
>>> for name in font_names:
...     font_numbers.append(a.getfont(name)) # font_numbers is now␣
↪filled with all font numbers
>>> font_numbers
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

Parameters **font** (*int or str*) – The font name/number

Returns If font parameter was a string, will return the integer associated with that string. If font parameter was an integer, will return the string associated with that integer.

Return type int or str

getfontname (*number*)
Retrieve a font name for a given font index.

Parameters **number** (*int*) – Index of the font to get the name of.

getfontnumber (*name*)
Retrieve a font index for a given font name.

Parameters **name** (*str*) – Name of the font to get the index of.

getisofill (*Gfi_name_src='default'*)
VCS contains a list of graphics methods. This function will create a isofill class object from an existing VCS isofill graphics method. If no isofill name is given, then isofill 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createisofill()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('isofill') # Show all the existing isofill␣
↪graphics methods
[...]
>>> ex=vcs.getisofill()  # instance of 'default' isofill graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.isofill(ex, slab1) # plot using specified isofill object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getisofill('polar')  # instance of 'polar' isofill␣
↪graphics method
>>> a.isofill(ex2, slab1) # plot using specified isofill object
<vcs.displayplot.Dp ...>
```

Parameters

- **Gfi_name_src** (*str*) – String name of an existing isofill VCS object
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot

- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns The specified isofill VCS object

Return type *vcs.isofill.Gfi*

getisoline(*Gi_name_src='default'*)

VCS contains a list of graphics methods. This function will create a isoline class object from an existing VCS isoline graphics method. If no isoline name is given, then isoline 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createisoline()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('isoline') # Show all the existing isoline graphics methods
[...]
>>> ex=vcs.getisoline()  # instance of 'default' isoline graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.isoline(ex, slab1) # plot using specified isoline object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getisoline('polar')  # instance of 'polar' isoline graphics method
>>> a.isoline(ex2, slab1) # plot using specified isoline object
<vcs.displayplot.Dp ...>
```

Parameters

- **Gi_name_src** (*str*) – String name of an existing isoline VCS object
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot

- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns The requested isoline VCS object

Return type *vcs.isoline.Gi*

getline(*name='default'*, *ltype=None*, *width=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a line class object from an existing VCS line secondary method. If no line name is given, then line 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createline()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('line') # Show all the existing line secondary methods
[...]
>>> ex=vcs.getline()  # instance of 'default' line secondary method
>>> a.line(ex) # plot using specified line object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getline('red')  # instance of 'red' line secondary method
>>> a.line(ex2) # plot using specified line object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object
- **ltype** (*str*) – One of "dash", "dash-dot", "solid", "dot", or "long-dash".
- **width** (*int*) – Thickness of the line to be created
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A VCS line object

Return type *vcs.line.Tl*

getmarker(*name='default'*, *mtype=None*, *size=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a marker class object from an existing VCS marker secondary method. If no marker name is given, then marker 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createmarker()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('marker') # Show all the existing marker␣
↪secondary methods
[...]
>>> ex=vcs.getmarker()  # instance of 'default' marker secondary␣
↪method
>>> a.marker(ex) # plot using specified marker object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getmarker('red')  # instance of 'red' marker secondary␣
↪method
>>> a.marker(ex2) # plot using specified marker object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – A marker, or string name of a marker
- **mtype** (*str*) – Specifies the type of marker, i.e. "dot", "circle"
- **size** (*int*) – Size of the marker
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A marker graphics method object

Return type *vcs.marker.Tm*

getmeshfill(*Gfm_name_src='default'*)

VCS contains a list of graphics methods. This function will create a meshfill class object from an existing VCS meshfill graphics method. If no meshfill name is given, then meshfill 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createmeshfill()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('meshfill') # Show all the existing meshfill␣
↪graphics methods
[...]
>>> ex=vcs.getmeshfill()  # instance of 'default' meshfill graphics␣
↪method
```

```
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.meshfill(ex, slab1) # plot using specified meshfill object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getmeshfill('a_polar_meshfill')  # instance of 'a_polar_
↪meshfill' meshfill graphics method
>>> a.meshfill(ex2, slab1) # plot using specified meshfill object
<vcs.displayplot.Dp ...>
```

Parameters **Gfm_name_src** (*str*) – String name of an existing meshfill VCS object

Returns A meshfill VCS object

Return type *vcs.meshfill.Gfm*

getplot(*Dp_name_src='default'*, *template=None*)

Deprecated since version 2.0: The getplot function is deprecated. Do not use it.

This function will create a display plot object from an existing display plot object from an existing VCS plot. If no display plot name is given, then None is returned.

Parameters

- **Dp_name_src** (*str*) – String name of an existing display plot object
- **template** – The displayplot template to inherit from

Returns A VCS displayplot object

Return type *vcs.displayplot.Dp*

getprojection(*Proj_name_src='default'*)

VCS contains a list of graphics methods. This function will create a projection class object from an existing VCS projection graphics method. If no projection name is given, then projection 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createprojection()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('projection') # Show all the existing␣
↪projection graphics methods
[...]
>>> ex=vcs.getprojection()  # instance of 'default' projection␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.plot(ex, slab1) # plot using specified projection object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getprojection('polar')  # instance of 'polar'␣
↪projection graphics method
```

```
>>> a.plot(ex2, slab1) # plot using specified projection object
<vcs.displayplot.Dp ...>
```

Parameters **Proj_name_src** (*str*) – String name of an existing VCS projection object

Returns A VCS projection object

Return type *vcs.projection.Proj*

getscatter(*GSp_name_src='default'*)

VCS contains a list of graphics methods. This function will create a scatter class object from an existing VCS scatter graphics method. If no scatter name is given, then scatter ''**default_scatter_**'' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createscatter()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('scatter') # Show all the existing scatter␣
↪graphics methods
[...]
>>> ex=vcs.getscatter('default_scatter_')  # instance of ''default_
↪scatter_'' scatter graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.scatter(ex, slab1, slab2) # plot using specified scatter␣
↪object
<vcs.displayplot.Dp ...>
```

Parameters

- **GSp_name_src** (*str*) – String name of an existing scatter VCS object.
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D

- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A scatter graphics method object

Return type *vcs.unified1D.G1d*

gettaylordiagram(*Gtd_name_src='default'*)

VCS contains a list of graphics methods. This function will create a taylordiagram class object from an existing VCS taylordiagram graphics method. If no taylordiagram name is given, then taylordiagram 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtaylordiagram()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('taylordiagram') # Show all the existing␣
↪taylordiagram graphics methods
[...]
>>> ex=vcs.gettaylordiagram()  # instance of 'default'␣
↪taylordiagram graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.taylordiagram(ex, slab1) # plot using specified taylordiagram␣
↪object
<vcs.displayplot.Dp ...>
```

Parameters **Gtd_name_src** (*str*) – String name of an existing taylordiagram VCS object

Returns A taylordiagram VCS object

Return type *vcs.taylor.Gtd*

gettemplate(*Pt_name_src='default'*)

VCS contains a list of graphics methods. This function will create a template class object from an existing VCS template graphics method. If no template name is given, then template 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtemplate()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('template') # Show all the existing template␣
↪graphics methods
[...]
>>> ex=vcs.gettemplate()  # instance of 'default' template graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
```

```
>>> a.plot(ex, slab1) # plot using specified template object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.gettemplate('polar')  # instance of 'polar' template␣
↪graphics method
>>> a.plot(ex2, slab1) # plot using specified template object
<vcs.displayplot.Dp ...>
```

Parameters **Pt_name_src** – String name of an existing template VCS object

Returns A VCS template object

Return type *vcs.template.P*

gettext(*Tt_name_src='default'*, *To_name_src=None*, *string=None*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*)

VCS contains a list of secondary methods. This function will create a textcombined class object from an existing VCS textcombined secondary method. If no textcombined name is given, then textcombined 'EXAMPLE_tt:::EXAMPLE_tto' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtextcombined()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('textcombined') # Show all the existing␣
↪textcombined secondary methods
[...]
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> ex=vcs.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto')  # instance␣
↪of 'EXAMPLE_tt:::EXAMPLE_tto' textcombined secondary method
>>> a.textcombined(ex) # plot using specified textcombined object
<vcs.displayplot.Dp ...>
```

Parameters

- **Tt_name_src** (*str*) – Name of created object
- **To_name_src** (*str*) – Name of parent textorientation object
- **string** – Text to render
- **string** – list of str
- **font** (*int or str*) – Which font to use (index or name)
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.

- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*list of int*) – Angle of the rendered text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of ["left", "center", "right"]
- **valign** (*str*) – Vertical alignment of the text. One of ["top", "center", "bottom"]

Returns A textcombined object

Return type *vcs.textcombined.Tc*

gettextcombined(*Tt_name_src='default'*, *To_name_src=None*, *string=None*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*)

VCS contains a list of secondary methods. This function will create a textcombined class object from an existing VCS textcombined secondary method. If no textcombined name is given, then textcombined 'EXAMPLE_tt:::EXAMPLE_tto' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtextcombined()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('textcombined') # Show all the existing␣
↪textcombined secondary methods
[...]
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> ex=vcs.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto')  # instance␣
↪of 'EXAMPLE_tt:::EXAMPLE_tto' textcombined secondary method
>>> a.textcombined(ex) # plot using specified textcombined object
<vcs.displayplot.Dp ...>
```

Parameters

- **Tt_name_src** (*str*) – Name of created object
- **To_name_src** (*str*) – Name of parent textorientation object
- **string** – Text to render
- **string** – list of str

- **font** (*int or str*) – Which font to use (index or name)
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*list of int*) – Angle of the rendered text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of [”left”, “center”, “right”]
- **valign** (*str*) – Vertical alignment of the text. One of [”top”, “center”, “bottom”]

Returns A textcombined object

Return type *vcs.textcombined.Tc*

gettextextent (*textobject*)

Returns the coordinate of the box surrounding a text object once printed

Example

```
>>> a=vcs.init()
>>> t=a.createtext()
>>> t.x=[.5]
>>> t.y=[.5]
>>> t.string=['Hello World']
>>> a.gettextextent(t)
[[...]]
```

Parameters **textobject** (textcombined) – A VCS text object

Returns list of floats containing the coordinates of the text object's bounding box.

Return type *list*

gettextorientation (*To_name_src='default'*)

VCS contains a list of secondary methods. This function will create a textorientation class object from an existing VCS textorientation secondary method. If no textorientation name is given, then textorientation 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtextorientation()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('textorientation') # Show all the existing␣
↪textorientation secondary methods
[...]
>>> ex=vcs.gettextorientation()  # instance of 'default'␣
↪textorientation secondary method
>>> ex2=vcs.gettextorientation('bigger')  # instance of 'bigger'␣
↪textorientation secondary method
```

Parameters **To_name_src** (*str*) – String name of an existing textorientation VCS object

Returns A textorientation VCS object

Return type *vcs.textorientation.To*

gettexttable(*name='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a texttable class object from an existing VCS texttable secondary method. If no texttable name is given, then texttable 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtexttable()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('texttable') # Show all the existing texttable␣
↪secondary methods
[...]
>>> ex=vcs.gettexttable()  # instance of 'default' texttable␣
↪secondary method
>>> ex2=vcs.gettexttable('bigger')  # instance of 'bigger'␣
↪texttable secondary method
```

Parameters

- **name** (*str*) – String name of an existing VCS texttable object
- **font** – ???
- **expansion** – ???
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the texttable will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A texttable graphics method object

Return type *vcs.texttable.Tt*

getvector(*Gv_name_src='default'*)

VCS contains a list of graphics methods. This function will create a vector class object from an existing VCS vector graphics method. If no vector name is given, then vector 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createvector()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('vector') # Show all the existing vector␣
↪graphics methods
[...]
>>> ex=vcs.getvector()  # instance of 'default' vector graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.vector(ex, slab1, slab2) # plot using specified vector object
<vcs.displayplot.Dp ...>
```

Parameters **Gv_name_src** (*str*) – String name of an existing vector VCS object

Returns A vector graphics method object

Return type *vcs.vector.Gv*

getxvsy(*GXY_name_src='default'*)

VCS contains a list of graphics methods. This function will create a xvsy class object from an existing VCS xvsy graphics method. If no xvsy name is given, then xvsy '**default_xvsy_**' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createxvsy()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('xvsy') # Show all the existing xvsy graphics␣
↪methods
[...]
>>> ex=vcs.getxvsy()  # instance of 'default_xvsy_' xvsy graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
```

```
>>> a.xvsy(ex, slab1, slab2) # plot using specified xvsy object
<vcs.displayplot.Dp ...>
```

Parameters

- **GXY_name_src** (*str*) – String name of a 1d graphics method
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)

- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A XvsY graphics method object

Return type *vcs.unified1D.G1d*

getxyvsy(*GXy_name_src='default'*)

VCS contains a list of graphics methods. This function will create a xyvsy class object from an existing VCS xyvsy graphics method. If no xyvsy name is given, then xyvsy ''**default_xyvsy_**'' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createxyvsy()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('xyvsy') # Show all the existing xyvsy␣
↪graphics methods
[...]
>>> ex=vcs.getxyvsy('default_xyvsy_')  # instance of ''default_
↪xyvsy_'' xyvsy graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.xyvsy(ex, slab1) # plot using specified xyvsy object
<vcs.displayplot.Dp ...>
```

Parameters

- **GXy_name_src** (*str*) – String name of an existing Xyvsy graphics method

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean

- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns An XYvsY graphics method object

Return type *vcs.unified1D.G1d*

getyxvsx(*GYx_name_src='default'*)

VCS contains a list of graphics methods. This function will create a yxvsx class object from an existing VCS yxvsx graphics method. If no yxvsx name is given, then yxvsx '**default_yxvsx_**' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createyxvsx()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('yxvsx') # Show all the existing yxvsx␣
↪graphics methods
[...]
>>> ex=vcs.getyxvsx()  # instance of 'default_yxvsx_' yxvsx␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.yxvsx(ex, slab1) # plot using specified yxvsx object
<vcs.displayplot.Dp ...>
```

Parameters

- **GYx_name_src** (*str*) – String name of an existing Yxvsx graphics method
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D

- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot

- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A Yxvsx graphics method object

Return type *vcs.unified1D.G1d*

gif (*filename='noname.gif'*, *merge='r'*, *orientation=None*, *geometry='1600x1200'*)

In some cases, the user may want to save the plot out as a gif image. This routine allows the user to save the VCS canvas output as a SUN gif file. This file can be converted to other gif formats with the aid of xv and other such imaging tools found freely on the web.

By default, the page orientation is in Landscape mode (l). To translate the page orientation to portrait mode (p), set the orientation = 'p'.

The GIF command is used to create or append to a gif file. There are two modes for saving a gif file: 'Append' mode (a) appends gif output to an existing gif file; 'Replace' (r) mode overwrites an existing gif file with new gif output. The default mode is to overwrite an existing gif file (i.e. mode (r)).

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.gif(filename='example.gif', merge='a', orientation='l',␣
↪geometry='800x600')
>>> a.gif('example') # overwrite existing gif file (default is␣
↪merge='r')
>>> a.gif('example',merge='r') # overwrite existing gif file
>>> a.gif('example',merge='a') # merge gif image into existing gif␣
↪file
>>> a.gif('example',orientation='l') # merge gif image into␣
↪existing gif file with landscape orientation
>>> a.gif('example',orientation='p') # merge gif image into␣
↪existing gif file with portrait orientation
>>> a.gif('example',geometry='600x500') # merge gif image into␣
↪existing gif file and set gif geometry
```

grid (**args*)

Set the default plotting region for variables that have more dimension values than the graphics method. This will also be used for animating plots over the third and fourth dimensions.

Example

```
>>> a=vcs.init()
>>> a.grid(12,12,0,71,0,45)
```

Not Yet Implemented

:py:func‘vcs.Canvas.grid‘_ does not work.

---

isinfile(*GM*, *file=None*)

Checks if a graphic method is stored in a file if no file name is passed then looks into the initial.attributes file

Parameters

- **GM** (*str*) – The graphics method to search for
- **file** (*str*) – String name of the file to search

Returns

???

Return type

???

islandscape()

Indicates if VCS's orientation is landscape.

Returns a 1 if orientation is landscape. Otherwise, it will return a 0, indicating false (not in landscape mode).

Example

```
>>> a=vcs.init()
>>> array = [range(10) for _ in range(10)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> if a.islandscape():
...     a.portrait() # Set VCS's orientation to portrait mode
```

Returns Integer indicating VCS is in landscape mode (1), or not (0)

Return type int

isofill(**args*, ***parms*)

Generate a isofill plot given the data, isofill graphics method, and template. If no isofill class object is given, then the 'default' isofill graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('isofill') # Show all the existing isofill graphics␣
↪methods
********************Isofill Names List**********************
...
********************End Isofill Names List**********************
>>> iso=a.getisofill('quick') # Create instance of 'quick'
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to␣
↪open a data file
>>> slab = f('clt') # use the data file to create a cdms2 slab
>>> a.isofill(slab,iso) # Plot array using specified iso and␣
↪default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
```

```
>>> template = a.gettemplate('hovmuller')
>>> a.isofill(slab,iso,template) # Plot array using specified
↪iso and template
<vcs.displayplot.Dp ...>
```

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)

- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **slab** (*array*) – (Ex: [[0, 1]]) Data at least 2D, last 2 dimensions will be plotted

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS displayplot object.

rtype vcs.displayplot.Dp

isoline(**args*, ***parms*)

Generate a isoline plot given the data, isoline graphics method, and template. If no isoline class object is given, then the 'default' isoline graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('isoline') # Show all the existing isoline graphics␣
↪methods
********************Isoline Names List**********************
...
********************End Isoline Names List**********************
>>> iso=a.getisoline('quick') # Create instance of 'quick'
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.isoline(array,iso) # Plot array using specified iso and␣
↪default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> template = a.gettemplate('hovmuller')
>>> a.isoline(array,iso,template)  # Plot array using␣
↪specified iso and template
<vcs.displayplot.Dp ...>
```

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean

- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **slab** (*array*) – (Ex: [[0, 1]]) Data at least 2D, last 2 dimensions will be plotted

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS displayplot object.

rtype vcs.displayplot.Dp

isopened()

Returns a boolean value indicating whether the canvas is opened or not.

Returns A boolean value indicating whether the Canvas is opened (1), or closed (0)

Return type bool

isportrait()

Indicates if VCS's orientation is portrait.

Example

```
>>> a=vcs.init()
>>> array = [range(10) for _ in range(10)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> if a.isportrait():
...     a.landscape() # Set VCS's orientation to landscape mode
```

Returns Returns a 1 if orientation is portrait, or 0 if not in portrait mode

Return type bool

landscape(*width=-99*, *height=-99*, *x=-99*, *y=-99*, *clear=0*)

Change the VCS Canvas orientation to Landscape.

Note: The (width, height) and (x, y) arguments work in pairs. That is, you must set (width, height) or (x, y) together to see any change in the VCS Canvas.

If the portrait method is called with arguments before displaying a VCS Canvas, then the arguments (width, height, x, y, and clear) will have no effect on the canvas.

Warning: If the visible plot on the VCS Canvas is not adjusted properly, then resize the screen with the point. Some X servers are not handling the threads properly to keep up with the demands of the X client.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.landscape() # Change the VCS Canvas orientation and set␣
↪object flag to landscape
>>> a.landscape(clear=1) # Change the VCS Canvas to landscape and␣
↪clear the page
>>> a.landscape(width = 400, height = 337) # Change to landscape␣
↪and set the window size
>>> a.landscape(x=100, y = 200) # Change to landscape and set the x␣
↪and y screen position
>>> a.landscape(width = 400, height = 337, x=100, y = 200, clear=1)
↪# landscape with more settings
```

Parameters

- **width** (*int*) – Width of the canvas, in pixels
- **height** (*int*) – Height of the canvas, in pixels
- **x** (*int*) – Unused
- **y** (*int*) – Unused
- **clear** (*int*) – Indicates the canvas should be cleared (1), or should not be cleared (0), when orientation is changed.

line (**args*, ***parms*)

Plot a line segment on the Vcs Canvas. If no line class object is given, then an error will be returned.

Example

```
>>> a=vcs.init()
>>> a.show('line')  # Show all the existing line objects
********************Line Names List**********************
...
********************End Line Names List**********************
>>> ln=a.getline('red') # Create instance of 'red'
>>> ln.width=4 # Set the line width
>>> ln.color = 242 # Set the line color
>>> ln.type = 4 # Set the line type
>>> ln.x=[[0.0,2.0,2.0,0.0,0.0], [0.5,1.5]] # Set the x value points
>>> ln.y=[[0.0,0.0,2.0,2.0,0.0], [1.0,1.0]] # Set the y value points
>>> a.line(ln) # Plot using specified line object
<vcs.displayplot.Dp ...>
```

Returns A VCS displayplot object.

Return type *vcs.displayplot.Dp*

listelements(**args*)

Returns a Python list of all the VCS class objects.

The list that will be returned: [‘1d’, ‘3d_dual_scalar’, ‘3d_scalar’, ‘3d_vector’, ‘boxfill’, ‘colormap’, ‘display’, ‘fillarea’,

‘font’, ‘fontNumber’, ‘isofill’, ‘isoline’, ‘line’, ‘list’, ‘marker’, ‘meshfill’, ‘projection’, ‘scatter’, ‘taylordiagram’, ‘template’, ‘textcombined’, ‘textorientation’, ‘texttable’, ‘vector’, ‘xvsy’, ‘xyvsy’, ‘yxvsx’]

Example

```
>>> a=vcs.init()
>>> a.listelements()
['1d', '3d_dual_scalar', '3d_scalar', ...]
```

Returns A list of string names of all VCS class objects

Return type *list*

marker(**args*, ***parms*)

Plot a marker segment on the Vcs Canvas. If no marker class object is given, then an error will be returned.

Example

```
>>> a=vcs.init()
>>> a.show('marker') # Show all the existing marker objects
********************Marker Names List**********************
...
********************End Marker Names List**********************
>>> mrk=a.getmarker('red') # Create instance of 'red'
>>> mrk.size=4 # Set the marker size
>>> mrk.color = 242 # Set the marker color
>>> mrk.type = 4 # Set the marker type
>>> mrk.x=[[0.0,2.0,2.0,0.0,0.0], [0.5,1.5]] # Set the x value␣
↪points
>>> mrk.y=[[0.0,0.0,2.0,2.0,0.0], [1.0,1.0]] # Set the y value␣
↪points
>>> a.marker(mrk) # Plot using specified marker object
<vcs.displayplot.Dp ...>
```

Returns a VCS displayplot object

Return type *vcs.displayplot.Dp*

meshfill(**args*, ***parms*)

Generate a meshfill plot given the data, the mesh, a meshfill graphics method, and a template. If no meshfill class object is given, then the ‘default’ meshfill graphics method is used. Similarly, if no template class object is given, then the ‘default’ template is used.

Format: This function expects 1D data (any extra dimension will be used for animation) In addition the mesh array must be of the same shape than data with 2 additional dimension representing the vertices coordinates for the Y (0) and X (1) dimension Let’s say you want to plot a spatial assuming mesh containing 10,000 grid cell, then data must be shape (10000,) or (n1,n2,n3,...,10000) if additional dimensions exist (ex time,level), these dimension would be used only for animation and will be ignored in the rest of this example. The shape of the mesh, assuming 4 vertices per grid cell, must be (1000,2,4), where the array [:,0,:] represent the Y coordinates of the vertices (clockwise or counterclockwise) and the array [:,1:]

represents the X coordinates of the vertices (the same clockwise/counterclockwise than the Y coordinates) In brief you'd have: data.shape=(10000,) mesh.shape=(10000,2,4)

Example

```
>>> a=vcs.init()
>>> a.show('meshfill') # Show all the existing meshfill graphics␣
↪methods
******************Meshfill Names List*********************
...
******************End Meshfill Names List*********************
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab = f('clt') # use the data file to create a cdms2 slab
>>> mesh=a.getmeshfill() # Create instance of 'default'
>>> a.meshfill(slab,mesh) # Plot array using specified mesh and␣
↪default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> a.meshfill(slab,mesh,'quick','a_polar_meshfill')  # Plot slab␣
↪with polar mesh, quick template
<vcs.displayplot.Dp ...>
```

Returns A VCS displayplot object.

Return type *vcs.displayplot.Dp*

objecthelp(**arg*)

Print out information on the VCS object. See example below on its use.

Example

```
>>> a=vcs.init()
>>> ln=a.getline('red') # Get a VCS line object
>>> a.objecthelp(ln) # This will print out information on how to␣
↪use ln
```

open(*width=None*, *height=None*, ***kargs*)

Open VCS Canvas object. This routine really just manages the VCS canvas. It will popup the VCS Canvas for viewing. It can be used to display the VCS Canvas.

Example

```
>>> a=vcs.init()
>>> a.open()
>>> a.open(800,600)
```

Parameters

- **width** (*int*) – Integer representing the desire width of the opened window in pixels
- **height** (*int*) – Integer representing the desire height of the opened window in pixels

orientation(**args*, ***kargs*)

Return canvas orientation.

The current implementation does not use any args or kargs.

Example

```
>>> a=vcs.init()
>>> a.orientation() # Show current orientation of the canvas
'landscape'
```

Returns A string indicating the orientation of the canvas, i.e. 'landscape' or 'portrait'

Return type str

pdf (*file*, *width=None*, *height=None*, *units='inches'*, *textAsPaths=True*)
PDF output is another form of vector graphics.

Note: The textAsPaths parameter preserves custom fonts, but text can no longer be edited in the file

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.pdf('example') # Overwrite a postscript file
>>> a.pdf('example', width=11.5, height= 8.5) # US Legal
>>> a.pdf('example', width=21, height=29.7, units='cm') # A4
```

Parameters

- **file** (*str*) – Desired string name of the output file
- **width** (*int*) – Integer specifying the desired width of the output, measured in the chosen units
- **height** (*int*) – Integer specifying the desired height of the output, measured in the chosen units
- **units** (*str*) – Must be one of ['inches', 'in', 'cm', 'mm', 'pixel', 'pixels', 'dot', 'dots']. Default is 'inches'.
- **textAsPaths** (*bool*) – Specifies whether to render text objects as paths.

plot (**actual_args*, ***keyargs*)

Plot an array(s) of data given a template and graphics method. The VCS template is used to define where the data and variable attributes will be displayed on the VCS Canvas. The VCS graphics method is used to define how the array(s) will be shown on the VCS Canvas.

Plot Usage:

```
plot(array1=None, array2=None, template_name=None,
    graphics_method=None, graphics_name=None,
    [key=value [, key=value [, ...]]])
```

Note: array1 and array2 are NumPy arrays.

Plot attribute keywords:

Note: More specific attributes take precedence over general attributes. In particular, specific attributes override variable object attributes, dimension attributes and arrays override axis objects, which override grid objects, which override variable objects.

For example, if both 'file_comment' and 'variable' keywords are specified, the value of 'file_comment' is used instead of the file comment in the parent of variable. Similarly, if both 'xaxis' and 'grid' keywords are specified, the value of 'xaxis' takes precedence over the x-axis of grid.

- ratio [default is none]
  - None: let the self.ratio attribute decide
  - 0,'off': overwrite self.ratio and do nothing about the ratio
  - 'auto': computes an automatic ratio
  - '3',3: y dim will be 3 times bigger than x dim (restricted to original tempalte.data area
  - Adding a 't' at the end of the ratio, makes the tickmarks and boxes move along.
- Dimension attribute keys (dimension length=n):
  - x or y Dimension values

```
[x|y|z|t|w]array = NumPy array of length n
[x|y|z|t|w]array = NumPy array of length n
```

  - x or y Dimension boundaries

```
[x|y]bounds = NumPy array of shape (n,2)
```

- CDMS object:
  - x or y Axis

```
[x|y|z|t|w]axis = CDMS axis object
```

  - Grid object (e.g. grid=var.getGrid())

```
grid = CDMS grid object
```

  - Variable object

```
variable = CDMS variable object
```

- Other:
  - Reverse the direction of the x or y axis:

```
[x|y]rev = 0|1
```

Note: For example, xrev = 1 would reverse the direction of the x-axis

  - Continental outlines:

```
continents = 0,1,2,3,4,5,6,7,8,9,10,11
# VCS line object to define continent appearance
continents_line = vcs.getline("default")
```

Note: If continents >=1, plot continental outlines. By default: plot of xaxis is longitude, yaxis is latitude -OR- xname is 'longitude' and yname is 'latitude'

    - List of continents-type values (integers from 0-11)

·0 signifies “No Continents”
·1 signifies “Fine Continents”
·2 signifies “Coarse Continents”
·3 signifies “United States”
·4 signifies “Political Borders”
·5 signifies “Rivers”

Note: Values 6 through 11 signify the line type defined by the files data_continent_other7 through data_continent_other12.

–To set whether the displayplot object generated by this plot is stored

```
donotstoredisplay = True|False
```

–Whether to actually render the plot or not (useful for doing a bunch of plots in a row)

```
render = True|False
```

–VCS Display plot name (used to prevent duplicate display plots)

```
display_name = "__display_123"
```

–Ratio of height/width for the plot; autot and auto will choose a “good” ratio for you.

```
ratio = 1.5|"autot"|"auto"
```

–Plot the actual grid or the dual grid

```
plot_based_dual_grid = True | False
```

Note: This is based on what is needed by the plot: isofill, isoline, vector need point attributes, boxfill and meshfill need cell attributes the default is True (if the parameter is not specified).

–Graphics Output in Background Mode:

```
# if ==1, create images in the background
bg = 0|1
```

Example

```
>>> a=vcs.init()
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use␣
↪cdms2 to open a data file
>>> slab1 = f('u') # use the data file to create a cdms2␣
↪slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.plot(slab1) # this call will use default settings␣
↪for template and boxfill
<vcs.displayplot.Dp ...>
>>> a.plot(slab1, 'polar', 'isofill', 'polar') # this is␣
↪specifying the template and graphics method
<vcs.displayplot.Dp ...>
>>> t=a.gettemplate('polar') # get the polar template
>>> vec=a.getvector() # get default vector
>>> a.plot(slab1, slab2, t, vec) # plot the data as a␣
↪vector using the 'AMIP' template
```

```
<vcs.displayplot.Dp ...>
>>> a.clear() # clear the VCS Canvas of all plots
>>> box=a.getboxfill() # get default boxfill graphics␣
↪method
>>> a.plot(box,t,slab2) # plot array data using box 'new
↪' and template 't'
<vcs.displayplot.Dp ...>
```

Parameters

- **slab2** (*array*) – Data at least 1D, last dimension(s) will be plotted
- **template** (*str/vcs.template.P*) – (‘default’) vcs template to use
- **gm** (*VCS graphics method object*) – (Ex: ‘default’) graphic method to use
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)

- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **slab_or_primary_object** (*array*) – Data at least 1D, last dimension(s) will be plotted, or secondary vcs object

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS display plot object

rtype vcs.displayplot.Dp

png (*file*, *width=None*, *height=None*, *units=None*, *draw_white_background=True*, ***args*)

PNG output, dimensions set via setbgoutputdimensions

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.png('example') # Overwrite a png file
```

Parameters

- **file** (*str*) – Output image filename
- **width** (*float*) – Float representing the desired width of the output png, using the specified unit of measurement
- **height** (*float*) – Float representing the desired height of the output png, using the specified unit of measurement
- **units** (*str*) – One of ['inches', 'in', 'cm', 'mm', 'pixel', 'pixels', 'dot', 'dots']. Defaults to 'inches'.
- **draw_white_background** (*bool*) – Boolean value indicating if the background should be white. Defaults to True.

portrait (*width=-99*, *height=-99*, *x=-99*, *y=-99*, *clear=0*)

Change the VCS Canvas orientation to Portrait.

Note: If the current orientation of the canvas is already portrait, nothing happens.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.portrait()  # Change the VCS Canvas orientation and set␣
↪object flag to portrait
>>> a.portrait(clear=1) # Change the VCS Canvas to portrait␣
↪and clear the page
>>> a.portrait(width = 337, height = 400) # Change to portrait␣
↪and set the window size
>>> a.portrait(x=100, y = 200) # Change to portrait and set␣
↪the x and y screen position
>>> a.portrait(width = 337, height = 400, x=100, y = 200,␣
↪clear=1) # portrait, with more specifications
```

Parameters

- **width** (*int*) – Width to set the canvas to (in pixels)
- **height** (*int*) – Height to set the canvas to (in pixels)
- **x** (*None*) – Unused.
- **y** (*None*) – Unused.
- **clear** (*int*) – 0: Do not clear the canvas when orientation is changed. 1: clear the canvas when orientation is changed.

postscript (*file*, *mode='r'*, *orientation=None*, *width=None*, *height=None*, *units='inches'*, *textAsPaths=True*)

Postscript output is another form of vector graphics. It is larger than its CGM output counter part, because it is stored out in ASCII format.

There are two modes for saving a postscript file: 'Append' (a) mode appends postscript output to an existing postscript file; and 'Replace' (r) mode overwrites an existing postscript file with new postscript output. The default mode is to overwrite an existing postscript file (i.e. mode (r)).

Note: The textAsPaths parameter preserves custom fonts, but text can no longer be edited in the file

---

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.postscript('example') # Overwrite a postscript file
>>> a.postscript('example', 'a') # Append postscript to an␣
↪existing file
>>> a.postscript('example', 'r') # Overwrite an existing file
>>> a.postscript('example', mode='a') # Append postscript to␣
↪an existing file
>>> a.postscript('example', width=11.5, height= 8.5) # US␣
↪Legal (default)
>>> a.postscript('example', width=21, height=29.7, units='cm')
↪# A4
```

Parameters

- **file** (*str*) – String name of the desired output file
- **mode** (*str*) – The mode in which to open the file. One of 'r' or 'a'.
- **orientation** (*None*) – Deprecated.
- **width** (*int*) – Desired width of the postscript output, in the specified unit of measurement
- **height** (*int*) – Desired height of the postscript output, in the specified unit of measurement
- **textAsPaths** (*bool*) – Specifies whether to render text objects as paths.

pstogif (*filename*, **opt*)

In some cases, the user may want to save the plot out as a gif image. This routine allows the user to convert a postscript file to a gif file.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.pstogif('filename.ps') # convert the postscript file to␣
↪a gif file (l=landscape)
>>> a.pstogif('filename.ps','l') # convert the postscript file␣
↪to a gif file (l=landscape)
>>> a.pstogif('filename.ps','p') # convert the postscript file␣
↪to a gif file (p=portrait)
```

Parameters

- **filename** (*str*) – String name of the desired output file
- **opt** (*str*) – One of 'l' or 'p', indicating landscape or portrait mode, respectively.

Returns

???

Return type

???

raisecanvas(**args*)
Raise the VCS Canvas to the top of all open windows.

remove_display_name(**args*)
Removes a plotted item from the canvas.
Parameters **args** (*str list*) – Any number of display names to remove.

removeobject(*obj*)
The user has the ability to create primary and secondary class objects. The function allows the user to remove these objects from the appropriate class list.

Note, To remove the object completely from Python, remember to use the "del" function.

Also note, The user is not allowed to remove a "default" class object.
Example

```
>>> a=vcs.init()
>>> line=a.getline('red') # To Modify an existing line object
>>> iso=a.createisoline('dean') # Create an instance of an␣
↪isoline object
>>> a.removeobject(line) # Removes line object from VCS list
'Removed line object red'
>>> a.removeobject(iso) # Remove isoline object from VCS list
'Removed isoline object dean'
```

Parameters **obj** (*VCS object*) – Any VCS primary or secondary object

Returns String indicating the specified object was removed

Return type str

saveinitialfile()
At start-up, VCS reads a script file named initial.attributes that defines the initial appearance of the VCS Interface. Although not required to run VCS, this initial.attributes file contains many predefined settings to aid the beginning user of VCS.
Example

```
>>> a=vcs.init()
>>> a.saveinitialfile()
```

Warning: This removes first ALL objects generated automatically (i.e. whose name starts with '__') in order to preserve this, rename objects first e.g:

```
>>> b=vcs.createboxfill()
>>> b.rename('MyBoxfill') # graphic method is now preserved
```

scatter(**args*, ***parms*)
Generate a scatter plot given the data, scatter graphics method, and template. If no scatter class object is given, then the 'default' scatter graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('scatter') # Show all the existing scatter␣
↪graphics methods
********************Scatter Names␣
↪List**********************
...
```

```
******************End Scatter Names␣
↪List*********************
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use␣
↪cdms2 to open a data file
>>> slab1 = f('u') # use the data file to create a␣
↪cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.scatter(slab1, slab2) # Plot array using␣
↪specified sct and default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> template=a.gettemplate('hovmuller')
>>> a.scatter(slab1, slab2, template) # Plot array␣
↪using specified sct and template
<vcs.displayplot.Dp ...>
```

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values

- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **slab_or_primary_object** (*array*) – Data at least 1D, last dimension(s) will be plotted, or secondary vcs object

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS displayplot object.

rtype vcs.displayplot.Dp

scriptobject(*obj*, *script_filename=None*, *mode=None*)

Save individual attributes sets (i.e., individual primary class objects and/or secondary class objects). These attribute sets are saved in the user's current directory in one of two formats: Python script, or a Javascript Object.

Note: If the the filename has a ".py" at the end, it will produce a Python script. If the filename has a ".scr" at the end, it will produce a VCS script. If neither extensions are given, then by default a Javascript Object will be produced.

Attention: VCS does not allow the modification of 'default' attribute sets, it will not allow them to be saved as individual script files. However, a 'default' attribute set that has been copied under a different name can be saved as a script file.

VCS Scripts Deprecated

SCR scripts are no longer generated by this function

Example

```
>>> a=vcs.init()
>>> i=a.createisoline('dean') # Create an instance of default␣
↪isoline object
>>> a.scriptobject(i,'ex_isoline.py') # Save isoline object as␣
↪a Python file 'isoline.py'
>>> a.scriptobject(i,'ex_isoline2') # Save isoline object as a␣
↪JSON object 'isoline2.json'
```

Parameters

- **script_filename** (*str*) – Name of the output script file.
- **mode** (*str*) – Mode is either "w" for replace or "a" for append.
- **obj** (*VCS object*) – Any VCS primary class or secondary class object.

setantialiasing(*antialiasing*)

Set antialiasing rate.

Parameters **antialiasing** (*int*) – Integer from 0-64, representing the antialising rate (0 means no antialiasing).

setbgoutputdimensions(*width=None*, *height=None*, *units='inches'*)

Sets dimensions for output in bg mode.

Example

```
>>> a=vcs.init()
>>> a.setbgoutputdimensions(width=11.5, height= 8.5) # US Legal
>>> a.setbgoutputdimensions(width=21, height=29.7, units='cm')
↪# A4
```

Parameters

- **width** (*float*) – Float representing the desired width of the output, using the specified unit of measurement
- **height** (*float*) – Float representing the desired height of the output, using the specified unit of measurement.
- **units** (*str*) – One of ['inches', 'in', 'cm', 'mm', 'pixel', 'pixels', 'dot', 'dots']. Defaults to 'inches'.

setcolorcell(**args*)

Set a individual color cell in the active colormap. If default is the active colormap, then return an error string.

If the the visul display is 16-bit, 24-bit, or 32-bit TrueColor, then a redrawing of the VCS Canvas is made evertime the color cell is changed.

Note, the user can only change color cells 0 through 239 and R,G,B value must range from 0 to 100. Where 0 represents no color intensity and 100 is the greatest color intensity.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.setcolormap("AMIP")
>>> a.setcolorcell(11,0,0,0)
>>> a.setcolorcell(21,100,0,0)
>>> a.setcolorcell(31,0,100,0)
>>> a.setcolorcell(41,0,0,100)
>>> a.setcolorcell(51,100,100,100)
>>> a.setcolorcell(61,70,70,70)
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
```

setcolormap(*name*)

It is necessary to change the colormap. This routine will change the VCS color map.

If the the visual display is 16-bit, 24-bit, or 32-bit TrueColor, then a redrawing of the VCS Canvas is made every time the colormap is changed.

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
>>> a.setcolormap("AMIP")
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
```

Parameters **name** (*str*) – Name of the colormap to use

setcontinentsline(*line='default'*)

One has the option of configuring the appearance of the lines used to draw continents by providing a VCS Line object.

Example

```
>>> a = vcs.init()
>>> line = vcs.createline()
>>> line.width = 5
>>> a.setcontinentsline(line) # Use custom continents line
>>> a.setcontinentsline("default") # Use default line
```

Parameters **line** (str or *vcs.line.Tl*) – Line to use for drawing continents. Can be a string name of a line, or a VCS line object

setcontinentstype(*value*)

One has the option of using continental maps that are predefined or that are user-defined. Predefined

continental maps are either internal to VCS or are specified by external files. User-defined continental maps are specified by additional external files that must be read as input.

The continents-type values are integers ranging from 0 to 11, where: 0 signifies "No Continents" 1 signifies "Fine Continents" 2 signifies "Coarse Continents" 3 signifies "United States" (with "Fine Continents") 4 signifies "Political Borders" (with "Fine Continents") 5 signifies "Rivers" (with "Fine Continents")

6 uses a custom continent set

You can also pass a file by path.

Example

```
>>> a=vcs.init()
>>> a.setcontinentstype(3)
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array,'default','isofill','quick')
<vcs.displayplot.Dp ...>
```

Parameters **value** (*int*) – Integer representing continent type, as specified in function description

setdefaultfont (*font*)

Sets the passed/def show font as the default font for vcs

Parameters **font** (*str or int*) – Font name or index to use as default

show (**args*)

Creator: Dean Williams (LLNL, AIMS Team)

Lead Developer: Charles Doutriaux (LLNL, AIMS Team)

Contributors: https://github.com/UV-CDAT/uvcdat/graphs/contributors

Support Email: uvcdat-support@llnl.gov

Project Site: http://uvcdat.llnl.gov/

Project Repo: https://github.com/UV-CDAT/uvcdat/graphs/contributors

VCS is a visualization library for scientific data. It has a simple model for defining a plot, that is decomposed into three parts:

1. **Data**: If it's iterable, we'll plot it... or at least try! Currently we support numpy arrays, lists (nested and not), and CDMS2 variables (there's some special support for metadata from CDMS2 that gives some niceties in your plot, but it's not mandatory).
2. **Graphics Method**: We have a variety of plot types that we support out-of-the box; you can easily customize every aspect of them to create the effect that you're looking for. If you can't, we also support defining your own graphics methods, which you can share with other users using standard python infrastructure (conda, pip).
3. **Template**: Templates control the appearance of everything that *isn't* your data. They position labels, control fonts, adjust borders, place legends, and more. They're very flexible, and give the fine-grained control of your plot that is needed for the truly perfect plot. Once you've customized them, you can also save them out for later use, and distribute them to other users.

svg (*file*, *width=None*, *height=None*, *units='inches'*, *textAsPaths=True*)

SVG output is another form of vector graphics.

Note: The textAsPaths parameter preserves custom fonts, but text can no longer be edited in the file

---

Example

```
>>> a=vcs.init()
>>> array = [range(1, 11) for _ in range(1, 11)]
>>> a.plot(array)
<vcs.displayplot.Dp ...>
>>> a.svg('example') # Overwrite a postscript file
>>> a.svg('example', width=11.5, height= 8.5) # US Legal
>>> a.svg('example', width=21, height=29.7, units='cm') # A4
```

Parameters

- **file** –
- **width** (*float*) – Float to set width of output SVG, in specified unit of measurement
- **height** (*float*) – Float to set height of output SVG, in specified unit of measurement
- **units** (*str*) – One of ['inches', 'in', 'cm', 'mm', 'pixel', 'pixels', 'dot', 'dots']. Deafults to 'inches'.
- **textAsPaths** (*bool*) – Specifies whether to render text objects as paths.

switchfonts(*font1*, *font2*)

Switch the font numbers of two fonts.

Parameters

- **font1** (*int or str*) – The first font
- **font2** (*int or str*) – The second font

taylordiagram(**args*, ***parms*)

Generate a taylordiagram plot given the data, taylordiagram graphics method, and template. If no taylordiagram class object is given, then the 'default' taylordiagram graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('taylordiagram') # Show all the existing␣
↪taylordiagram graphics methods
********************Taylordiagram Names␣
↪List**********************
...
********************End Taylordiagram Names␣
↪List**********************
>>> td= a.gettaylordiagram() # Create instance of 'default'
>>> array=[range(1, 11) for _ in range(1, 11)]
>>> a.taylordiagram(array,td) # Plot array using specified iso␣
↪and default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> template=a.gettemplate('hovmuller')
>>> a.taylordiagram(array,td,template) # Plot array using␣
↪specified iso and template
<vcs.displayplot.Dp ...>
```

Returns A VCS displayplot object.

Return type *vcs.displayplot.Dp*

text(**args*, ***parms*)

Plot a textcombined segment on the Vcs Canvas. If no textcombined class object is given, then an error will be returned.

Note: The text() function is an alias for textcombined(). See example for usage.

Example

```
>>> a=vcs.init()
>>> a.clean_auto_generated_objects()
>>> a.show('texttable') # Show all the existing texttable objects
******************Texttable Names List**********************
...
******************End Texttable Names List**********************
>>> a.show('textorientation') # Show all the existing textorientation objects
******************Textorientation Names List**********************
...
******************End Textorientation Names List**********************
>>> vcs.createtext('qa_tta', 'qa', '7left_tto', '7left') # Create instance␣
↪of 'std_tt' and '7left_to'
<vcs.textcombined.Tc object at ...>
>>> tc=a.gettext('qa_tta', '7left_tto')
>>> tc.string='Text1' # Show the string "Text1" on the VCS Canvas
>>> tc.font=2 # Set the text size
>>> tc.color=242 # Set the text color
>>> tc.angle=45 # Set the text angle
>>> tc.x=[0.5]
>>> tc.y=[0.5]
>>> a.textcombined(tc) # Plot using specified text object
<vcs.displayplot.Dp ...>
```

Returns A fillarea object

Return type *vcs.displayplot.Dp*

textcombined(**args*, ***parms*)

Plot a textcombined segment on the Vcs Canvas. If no textcombined class object is given, then an error will be returned.

Note: The text() function is an alias for textcombined(). See example for usage.

Example

```
>>> a=vcs.init()
>>> a.clean_auto_generated_objects()
>>> a.show('texttable') # Show all the existing texttable objects
******************Texttable Names List**********************
...
******************End Texttable Names List**********************
>>> a.show('textorientation') # Show all the existing textorientation objects
******************Textorientation Names List**********************
...
******************End Textorientation Names List**********************
>>> vcs.createtext('qa_tta', 'qa', '7left_tto', '7left') # Create instance␣
↪of 'std_tt' and '7left_to'
<vcs.textcombined.Tc object at ...>
```

```
>>> tc=a.gettext('qa_tta', '7left_tto')
>>> tc.string='Text1' # Show the string "Text1" on the VCS Canvas
>>> tc.font=2 # Set the text size
>>> tc.color=242 # Set the text color
>>> tc.angle=45 # Set the text angle
>>> tc.x=[0.5]
>>> tc.y=[0.5]
>>> a.textcombined(tc) # Plot using specified text object
<vcs.displayplot.Dp ...>
```

Returns A fillarea object

Return type *vcs.displayplot.Dp*

update(**args*, ***kargs*)

If a series of commands are given to VCS and the Canvas Mode is set to manual, then use this function to update the plot(s) manually.

Example

```
>>> a=vcs.init()
>>> import cdms2 # We need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to␣
↪open a data file
>>> s = f('clt') # use the data file to create a slab
>>> a.plot(s,'default','boxfill','quick')
<vcs.displayplot.Dp ...>
>>> a.mode = 0 # Go to manual mode
>>> box=a.getboxfill('quick')
>>> box.color_1=100
>>> box.xticlabels('lon30','lon30')
>>> box.xticlabels('','')
>>> box.datawc(1e20,1e20,1e20,1e20)
>>> box.datawc(-45.0, 45.0, -90.0, 90.0)
>>> a.update() # Update the changes manually
```

updateorientation(**args*)

Deprecated since version 2.0: Use *landscape()* or *portrait()* instead.

vector(**args*, ***parms*)

Generate a vector plot given the data, vector graphics method, and template. If no vector class object is given, then the 'default' vector graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('vector') # Show all the existing vector graphics␣
↪methods
********************Vector Names List**********************
...
********************End Vector Names List**********************
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to␣
↪open a data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # vector needs 2 slabs, so get another
>>> a.vector(slab1, slab2) # plot vector using slab and␣
↪default vector
<vcs.displayplot.Dp ...>
```

```
>>> a.clear() # Clear VCS canvas
>>> template=a.gettemplate('hovmuller')
>>> a.vector(slab1, slab2, template) # Plot array using␣
↪default vector and specified template
<vcs.displayplot.Dp ...>
```

Returns A VCS displayplot object.

Return type *vcs.displayplot.Dp*

xvsy (**args*, ***parms*)

Generate a XvsY plot given the data, XvsY graphics method, and template. If no XvsY class object is given, then the 'default' XvsY graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('xvsy') # Show all the existing XvsY␣
↪graphics methods
********************Xvsy Names␣
↪List**********************
...
********************End Xvsy Names␣
↪List**********************
>>> xy=a.getxvsy('default_xvsy_') # Create instance of␣
↪default xvsy
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use␣
↪cdms2 to open a data file
>>> slab1 = f('u') # use the data file to create a␣
↪cdms2 slab
>>> slab2 = f('v') # use the data file to create a␣
↪cdms2 slab
>>> a.xvsy(slab1,slab2,xy) # Plot array using␣
↪specified xy and default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> template=a.gettemplate('hovmuller')
>>> a.xvsy(slab1,slab2,xy,template) # Plot array using␣
↪specified xy and template
<vcs.displayplot.Dp ...>
```

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis

- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks

- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **slab_or_primary_object** (*array*) – Data at least 1D, last dimension(s) will be plotted, or secondary vcs object

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS displayplot object.

rtype vcs.displayplot.Dp

xyvsy (**args*, ***parms*)

Generate a Xyvsy plot given the data, Xyvsy graphics method, and template. If no Xyvsy class object is given, then the 'default' Xyvsy graphics method is used. Similarly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('xyvsy') # Show all the existing Xyvsy␣
↪graphics methods
********************Xyvsy Names␣
↪List**********************
...
********************End Xyvsy Names␣
↪List**********************
>>> xyy=a.getxyvsy('default_xyvsy_') # Create instance␣
↪of default xyvsy
>>> array=[range(1, 11) for _ in range(1, 11)]
>>> a.xyvsy(array,xyy) # Plot array using specified␣
↪xyy and default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> template=a.gettemplate('hovmuller')
>>> a.xyvsy(array,xyy,template) # Plot array using␣
↪specified xyy and template
<vcs.displayplot.Dp ...>
```

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D

- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode

- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **slab** (*array*) – (Ex: [1, 2]) Data at least 1D, last dimension will be plotted

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS displayplot object.

rtype vcs.displayplot.Dp

yxvsx(**args*, ***parms*)

Generate a Yxvsx plot given the data, Yxvsx graphics method, and template. If no Yxvsx class object is given, then the 'default' Yxvsx graphics method is used. Simerly, if no template class object is given, then the 'default' template is used.

Example

```
>>> a=vcs.init()
>>> a.show('yxvsx') # Show all the existing Yxvsx␣
↪graphics methods
********************Yxvsx Names␣
↪List**********************
...
********************End Yxvsx Names␣
↪List**********************
>>> yxx=a.getyxvsx('default_yxvsx_') # Create instance␣
↪of default yxvsx
>>> array=[range(1, 11) for _ in range(1, 11)]
>>> a.yxvsx(array,yxx) # Plot array using specified␣
↪yxx and default template
<vcs.displayplot.Dp ...>
>>> a.clear() # Clear VCS canvas
>>> template=a.gettemplate('hovmuller')
>>> a.yxvsx(array,yxx,template) # Plot array using␣
↪specified yxx and template
<vcs.displayplot.Dp ...>
```

Parameters

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis

- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks

- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **slab** (*array*) – (Ex: [1, 2]) Data at least 1D, last dimension will be plotted

Returns Display Plot object representing the plot.

Return type

vcs.displayplot.Dp

returns A VCS displayplot object.

rtype vcs.displayplot.Dp

1.3 Graphics Methods

Graphics methods are VCS' objects for configuring your visualization's data representation. They allow you to set levels, colors, subset your data, set patterns, and much more.

1.3.1 boxfill

Boxfill (Gfb) module

class vcs.boxfill.**Gfb**(*Gfb_name=None*, *Gfb_name_src='default'*)

The boxfill graphics method (Gfb) displays a two-dimensional data array by surrounding each data value by a colored grid box.

This class is used to define a boxfill table entry used in VCS, or it can be used to change some or all of the attributes in an existing boxfill table entry.

General use of a boxfill:

```
# Constructor
a=vcs.init()
# Show predefined boxfill graphics methods
a.show('boxfill')
# Change the VCS color map
a.setcolormap("AMIP")
# Plot data 's' with boxfill 'b' and 'default' template
a.boxfill(s,b,'default')
```

Updating a boxfill:

```
# Updates the VCS Canvas at user's request
a.update()
# Set VCS Canvas to automatic update mode
a.mode=1
# Use update function to update the VCS Canvas
a.mode=0
```

Create a new instance of boxfill:

```
#  Copies content of 'quick' to 'new'
box=a.createboxfill('new','quick')
#  Copies content of 'default' to 'new'
box=a.createboxfill('new')
```

Modifying an existing boxfill:

```
fill=a.getboxfill('quick')

# Set index using fillarea
box.fillareaindices=(7,fill,4,9,fill,15)
# list fillarea attributes
fill.list()
# change style
fill.style='hatch'
# change color
fill.color=241
# change style index
fill.index=3
```

Overview of boxfill attributes:

•Listing all the boxfill attribute values:

```
box.list()
```

•Setting boxfill attribute values:

```
box.projection='linear'
lon30={-180:'180W',-150:'150W',0:'Eq'}
box.xticlabels1=lon30
box.xticlabels2=lon30
# Will set them both
box.xticlabels(lon30, lon30)
box.xmtics1=''
box.xmtics2=''
# Will set them both
box.xmtics(lon30, lon30)
box.yticlabels1=lat10
box.yticlabels2=lat10
# Will set them both
box.yticlabels(lat10, lat10)
box.ymtics1=''
box.ymtics2=''
# Will set them both
box.ymtics(lat10, lat10)
box.datawc_y1=-90.0
box.datawc_y2=90.0
box.datawc_x1=-180.0
box.datawc_x2=180.0
# Will set them all
box.datawc(-90, 90, -180, 180)
box.xaxisconvert='linear'
box.yaxisconvert='linear'
# Will set them both
box.xyscale('linear', 'area_wt')
box.level_1=1e20
box.level_2=1e20
```

```
box.color_1=16
box.color_2=239
# Will set them both
box.colors(16, 239 )
# 'linear' - compute or specify legend
box.boxfill_type='linear'
# 'log10' - plot using log10
box.boxfill_type='log10'
# 'custom' - use custom values to display legend evenly
box.boxfill_type='custom'
# Hold the legend values
box.legend=None
# Show left overflow arrow
box.ext_1='n'
# Show right overflow arrow
box.ext_2='y'
# Will set them both
box.exts('n', 'y' )
# Color index value range 0 to 255
box.missing=241
```

•Setting the boxfill levels:

```
# Case 1: Levels are all contiguous:
box.levels=([0,20,25,30,35,40],)
box.levels=([0,20,25,30,35,40,45,50])
box.levels=[0,20,25,30,35,40]
box.levels=(0.0,20.0,25.0,30.0,35.0,40.0,50.0)

# Case 2: Levels are not contiguous:
box.levels=([0,20],[30,40],[50,60])
box.levels=([0,20,25,30,35,40],[30,40],[50,60])
```

•Setting the fillarea color indices:

```
# Three different methods for setting color indices:
box.fillareacolors=([22,33,44,55,66,77])
box.fillareacolors=(16,19,33,44)
box.fillareacolors=None
```

•Setting the fillarea style:

```
box.fillareastyle = 'solid'
box.fillareastyle = 'hatch'
box.fillareastyle = 'pattern'
```

•Setting the fillarea hatch or pattern indices:

```
box.fillareaindices=([1,3,5,6,9,20])
box.fillareaindices=(7,1,4,9,6,15)
```

•Using the fillarea secondary object (Ex):

```
f=createfillarea('fill1')
#To Create a new instance of fillarea use:
# Copies 'quick' to 'new'
fill=a.createfillarea('new','quick')
# Copies 'default' to 'new'
fill=a.createfillarea('new')
```

•Attribute descriptions:

boxfill_type (*str*)
Type of boxfill legend. One of 'linear', 'log10', or 'custom'. See examples above for usage.

level_1 (*float*)
Used in conjunction with boxfill_type linear/log10. Sets the value of the legend's first level

level_2 (*float*)
Used in conjunction with boxfill_type linear/log10, sets the value of the legend's end level

color_1 (*float*)
Used in conjunction with boxfill_type linear/log10, sets the legend's color range first value

color_2 (*float*)
Used in conjunction with boxfill_type linear/log10, sets the legend's color range lasst value

levels (*list of floats*)
Used in conjunction for boxfill_type custom, sets the levels range to use, can be either a list of contiguous levels, or list of tuples indicating first and last value of the range.

legend (*{float:str}*)
Used in conjunction with boxfill_type linear/log10, replaces the legend values in the dictionary keys with their associated string.

ext_1 (*str*)
Draws an extension arrow on right side (values less than first range value)

ext_2 (*str*)
Draws an extension arrow on left side (values greater than last range value)

missing (*int*)
Color to use for missing value or values not in defined ranges.

xmtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

xmtics2 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 2nd side of y axis

ymtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

ymtics2 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 2nd side of y axis

xticlabels1 (*str/{float:str}*)
(Ex: '*') values for labels on 1st side of x axis

xticlabels2 (*str/{float:str}*)
(Ex: '*') values for labels on 2nd side of x axis

yticlabels1 (*str/{float:str}*)
(Ex: '*') values for labels on 1st side of y axis

yticlabels2 (*str/{float:str}*)
(Ex: '*') values for labels on 2nd side of y axis

projection (*str/vcs.projection.Proj*)
(Ex: 'default') projection to use, name or object

datawc_x1 (*float*)
(Ex: 1.E20) first value of xaxis on plot

datawc_x2 (*float*)
(Ex: 1.E20) second value of xaxis on plot

datawc_y1 (*float*)
(Ex: 1.E20) first value of yaxis on plot

datawc_y2 (*float*)
(Ex: 1.E20) second value of yaxis on plot

datawc_timeunits (*str*)
(Ex: 'days since 2000') units to use when displaying time dimension auto tick

datawc_calendar (*int*)
(Ex: 135441) calendar to use when displaying time dimension auto tick, default is proleptic gregorian calendar

colors (*color1=16*, *color2=239*)
Sets the color_1 and color_2 properties of the object.

Parameters

- **color1** (*int*) – Sets the *color_1* value on the object
- **color2** (*int*) – Sets the *color_2* value on the object

datawc (*dsp1=1e+20*, *dsp2=1e+20*, *dsp3=1e+20*, *dsp4=1e+20*)
Sets the data world coordinates for object

Parameters

- **dsp1** (*float*) – Sets the *datawc_y1* property of the object.
- **dsp2** (*float*) – Sets the *datawc_y2* property of the object.
- **dsp3** (*float*) – Sets the *datawc_x1* property of the object.
- **dsp4** (*float*) – Sets the *datawc_x2* property of the object.

exts (*ext1='n'*, *ext2='y'*)
Sets the ext_1 and ext_2 values on the object.

Parameters

- **ext1** (*str*) – Sets the *ext_1* value on the object. 'y' sets it to True, 'n' sets it to False.
- **ext2** (*str*) – Sets the *ext_2* value on the object. 'y' sets it to True, 'n' sets it to False.

list ()
Lists the current values of object attributes

rename (*newname*)
Renames the boxfill in the VCS name table.

Note: This function will not rename the 'default' boxfill. If rename is called on the 'default' boxfill, newname is associated with default in the VCS name table, but the boxfill's name will not be changed, and will behave in all ways as a 'default' boxfill.

Example

```
>>> b=vcs.createboxfill()
>>> b.name
'...'
>>> vcs.listelements('boxfill') # list will include the name␣
↪show above
[...]
>>> b.rename('foo')
>>> b.name
'foo'
>>> vcs.listelements('boxfill') # list will include 'foo', but␣
↪not the old name
[...'foo'...]
```

Parameters **newname** – The new name you want given to the boxfill

script (*script_filename*, *mode='a'*)

Saves out a copy of the boxfill graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getboxfill() # Get default boxfill
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

xmtics (*xmt1=''*, *xmt2=''*)

Sets the xmtics1 and xmtics2 values on the object

Parameters

- **xmt1** (*{float:str} or str*) – Value for *xmtics1*. Must be a str, or a dictionary object with float:str mappings.
- **xmt2** (*{float:str} or str*) – Value for *xmtics2*. Must be a str, or a dictionary object with float:str mappings.

xticlabels (*xtl1=''*, *xtl2=''*)

Sets the xticlabels1 and xticlabels2 values on the object

Parameters

- **xtl1** (*{float:str} or str*) – Sets the object's value for *xticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **xtl2** (*{float:str} or str*) – Sets the object's value for *xticlabels2*. Must be a str, or a dictionary object with float:str mappings.

xyscale (*xat='linear'*, *yat='linear'*)

Sets xaxisconvert and yaxisconvert values for the object.

Example

```
>>> a=vcs.init()
>>> ex=a.createboxfill('xyscale_ex') # create a boxfill to␣
↪work with
>>> ex.xyscale(xat='linear', yat='linear') # set xaxisconvert␣
↪and yaxisconvert to 'linear'
```

Parameters

- **xat** (*str*) – Set value for x axis conversion.
- **yat** (*str*) – Set value for y axis conversion.

ymtics (*ymt1=''*, *ymt2=''*)

Sets the ymtics1 and ymtics2 values on the object

Parameters

- **ymt1** (*{float:str} or str*) – Value for *ymtics1*. Must be a str, or a dictionary object with float:str mappings.
- **ymt2** (*{float:str} or str*) – Value for *ymtics2*. Must be a str, or a dictionary object with float:str mappings.

yticlabels (*ytl1=''*, *ytl2=''*)

Sets the yticlabels1 and yticlabels2 values on the object

Parameters

- **ytl1** (*{float:str} or str*) – Sets the object's value for *yticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **ytl2** (*{float:str} or str*) – Sets the object's value for *yticlabels2*. Must be a str, or a dictionary object with float:str mappings.

1.3.2 dv3d

Created on Jun 18, 2014

@author: tpmaxwel

class vcs.dv3d.**Gfdv3d** (*Gfdv3d_name*, *Gfdv3d_name_src='default'*)

1.3.3 isofill

Isofill (Gfi) module

class vcs.isofill.**Gfi** (*Gfi_name*, *Gfi_name_src='default'*)

The Isofill graphics method fills the area between selected isolevels (levels of constant value) of a two-dimensional array with a user-specified color. The example below shows how to display an isofill plot on the VCS Canvas and how to create and remove isofill isolevels.

This class is used to define an isofill table entry used in VCS, or it can be used to change some or all of the isofill attributes in an existing isofill table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined isofill graphics methods
a.show('isofill')
# Show predefined fillarea objects
a.show('fillarea')
# Show predefined template objects
a.show('template')
# Change the VCS color map
a.setcolormap("AMIP")
# Create a template
a.createtemplate('test')
# Create a fillarea
a.createfillarea('fill')
# Get an existing template
a.gettemplate('AMIP')
# Get an existing fillarea
a.getfillarea('def37')
# Plot array 's' with isofill 'i' and template 't'
a.isofill(s,i,t)
# Updates the VCS Canvas at user's request
a.update()
```

Creating an isofill object:

```
#Create a VCS Canvas
a=vcs.init()
#Create a new instance of isofill:
# Copies content of 'quick' to 'new'
iso=a.createisofill('new','quick')
# Copies content of 'default' to 'new'
iso=a.createisofill('new')
```

Modifying an existing isofill:

```
iso=a.getisofill('AMIP_psl')
```

Overview of isofill attributes:

•List all isofill attribute values:

```
iso.list()
```

•Set isofill attributes:

```
iso.projection='linear'
lon30={-180:'180W',-150:'150W',0:'Eq'}
iso.xticlabels1=lon30
iso.xticlabels2=lon30
# Will set them both
iso.xticlabels(lon30, lon30)
iso.xmtics1=''
```

```
iso.xmtics2=''
# Will set them both
iso.xmtics(lon30, lon30)
iso.yticlabels1=lat10
iso.yticlabels2=lat10
# Will set them both
iso.yticlabels(lat10, lat10)
iso.ymtics1=''
iso.ymtics2=''
# Will set them both
iso.ymtics(lat10, lat10)
iso.datawc_y1=-90.0
iso.datawc_y2=90.0
iso.datawc_x1=-180.0
iso.datawc_x2=180.0
# Will set them all
iso.datawc(-90, 90, -180, 180)
iso.xaxisconvert='linear'
iso.yaxisconvert='linear'
# Will set them both
iso.xyscale('linear', 'area_wt')
# Color index value range 0 to 255
iso.missing=241
iso.legend=None
ext_1='n'
ext_2='y'
# Will set them both
iso.exts('n', 'y' )
```

•Setting the isofill levels:

```
# 1) When levels are all contiguous:
    iso.levels=([0,20,25,30,35,40],)
    iso.levels=([0,20,25,30,35,40,45,50])
    iso.levels=[0,20,25,30,35,40]
    iso.levels=(0.0,20.0,25.0,30.0,35.0,40.0,50.0)

# 2) When levels are not contiguous:
    iso.levels=([0,20],[30,40],[50,60])
    iso.levels=([0,20,25,30,35,40],[30,40],[50,60])
```

•Setting the fillarea color indices:

```
iso.fillareacolors=([22,33,44,55,66,77])
iso.fillareacolors=(16,19,33,44)
iso.fillareacolors=None
```

•Setting the fillarea style:

```
iso.fillareastyle = 'solid'
iso.fillareastyle = 'hatch'
iso.fillareastyle = 'pattern'
```

•Setting the fillarea hatch or pattern indices:

```
iso.fillareaindices=([1,3,5,6,9,20])
iso.fillareaindices=(7,1,4,9,6,15)
```

Using the fillarea secondary object (Ex):

•Create a new instance of fillarea:

```
f=createfillarea('fill1')
```

•Create a new isofill:

```
# Copies 'quick' to 'new'
fill=a.createisofill('new','quick')
# Copies 'default' to 'new'
fill=a.createisofill('new')
```

•Modify an existing isofill:

```
fill=a.getisofill('def37')
```

•Set index using fillarea

```
iso.fillareaindices=(7,fill,4,9,fill,15)
# list fillarea attributes
fill.list()
# change style
fill.style='hatch'
# change color
fill.color=241
# change style index
fill.index=3
```

Attribute descriptions:

xmtics1 (*str/{float:str}*)
(Ex: ‘’) dictionary with location of intermediate tics as keys for 1st side of y axis

xmtics2 (*str/{float:str}*)
(Ex: ‘’) dictionary with location of intermediate tics as keys for 2nd side of y axis

ymtics1 (*str/{float:str}*)
(Ex: ‘’) dictionary with location of intermediate tics as keys for 1st side of y axis

ymtics2 (*str/{float:str}*)
(Ex: ‘’) dictionary with location of intermediate tics as keys for 2nd side of y axis

xticlabels1 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 1st side of x axis

xticlabels2 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 2nd side of x axis

yticlabels1 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 1st side of y axis

yticlabels2 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 2nd side of y axis

projection (*str/vcs.projection.Proj*)
(Ex: ‘default’) projection to use, name or object

datawc_x1 (*float*)
(Ex: 1.E20) first value of xaxis on plot

datawc_x2 (*float*)
(Ex: 1.E20) second value of xaxis on plot

datawc_y1 (*float*)
(Ex: 1.E20) first value of yaxis on plot

datawc_y2 (*float*)
(Ex: 1.E20) second value of yaxis on plot

datawc_timeunits (*str*)
(Ex: 'days since 2000') units to use when displaying time dimension auto tick

datawc_calendar (*int*)
(Ex: 135441) calendar to use when displaying time dimension auto tick, default is proleptic gregorian calendar

levels (*[float,...]/[[float,float],...]*)
Sets the levels range to use, can be either a list of contiguous levels, or list of tuples indicating first and last value of the range.

fillareacolors ([*int*, ...])
Colors to use for each level

fillareastyle (*str*)
Style to use for levels filling: solid/pattern/hatch

fillareaindices ([*int*, ...])
List of patterns to use when filling a level and using pattern/hatch

legend (*None/{float:str}*)
Replaces the legend values in the dictionary keys with their associated string

ext_1 (*str*)
Draws an extension arrow on right side (values less than first range value)

ext_2 (*str*)
Draws an extension arrow on left side (values greater than last range value)

missing (*int*)
Color to use for missing value or values not in defined ranges

colors (*color1=16*, *color2=239*)
Sets the color_1 and color_2 properties of the object.
Parameters

- **color1** (*int*) – Sets the `color_1` value on the object
- **color2** (*int*) – Sets the `color_2` value on the object

datawc (*dsp1=1e+20*, *dsp2=1e+20*, *dsp3=1e+20*, *dsp4=1e+20*)
Sets the data world coordinates for object
Parameters

- **dsp1** (*float*) – Sets the *datawc_y1* property of the object.
- **dsp2** (*float*) – Sets the *datawc_y2* property of the object.
- **dsp3** (*float*) – Sets the *datawc_x1* property of the object.
- **dsp4** (*float*) – Sets the *datawc_x2* property of the object.

exts (*ext1='n'*, *ext2='y'*)
Sets the ext_1 and ext_2 values on the object.
Parameters

- **ext1** (*str*) – Sets the *ext_1* value on the object. 'y' sets it to True, 'n' sets it to False.
- **ext2** (*str*) – Sets the *ext_2* value on the object. 'y' sets it to True, 'n' sets it to False.

list()
Lists the current values of object attributes

script(*script_filename*, *mode='a'*)
Saves out a copy of the isofill graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getisofill() # Get default isofill
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

xmtics(*xmt1=''*, *xmt2=''*)
Sets the xmtics1 and xmtics2 values on the object
Parameters

- **xmt1** (*{float:str} or str*) – Value for *xmtics1*. Must be a str, or a dictionary object with float:str mappings.
- **xmt2** (*{float:str} or str*) – Value for *xmtics2*. Must be a str, or a dictionary object with float:str mappings.

xticlabels(*xtl1=''*, *xtl2=''*)
Sets the xticlabels1 and xticlabels2 values on the object
Parameters

- **xtl1** (*{float:str} or str*) – Sets the object's value for *xticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **xtl2** (*{float:str} or str*) – Sets the object's value for *xticlabels2*. Must be a str, or a dictionary object with float:str mappings.

xyscale(*xat=''*, *yat=''*)
Sets xaxisconvert and yaxisconvert values for the object.
Example

```
>>> a=vcs.init()
>>> ex=a.createisofill('xyscale_ex') # create a boxfill to␣
↪work with
>>> ex.xyscale(xat='linear', yat='linear') # set xaxisconvert␣
↪and yaxisconvert to 'linear'
```

Parameters

- **xat** (*str*) – Set value for x axis conversion.
- **yat** (*str*) – Set value for y axis conversion.

ymtics (*ymt1=''*, *ymt2=''*)

Sets the ymtics1 and ymtics2 values on the object

Parameters

- **ymt1** (*{float:str} or str*) – Value for *ymtics1*. Must be a str, or a dictionary object with float:str mappings.
- **ymt2** (*{float:str} or str*) – Value for *ymtics2*. Must be a str, or a dictionary object with float:str mappings.

yticlabels (*ytl1=''*, *ytl2=''*)

Sets the yticlabels1 and yticlabels2 values on the object

Parameters

- **ytl1** (*{float:str} or str*) – Sets the object's value for *yticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **ytl2** (*{float:str} or str*) – Sets the object's value for *yticlabels2*. Must be a str, or a dictionary object with float:str mappings.

1.3.4 isoline

Isoline (Gi) module

class vcs.isoline.**Gi** (*Gi_name*, *Gi_name_src='default'*)

The Isoline graphics method (Gi) draws lines of constant value at specified levels in order to graphically represent a two-dimensional array. It also labels the values of these isolines on the VCS Canvas. The example below shows how to plot isolines of different types at specified levels and how to create isoline labels having user-specified text and line type and color.

This class is used to define an isoline table entry used in VCS, or it can be used to change some or all of the isoline attributes in an existing isoline table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined isoline graphics methods
a.show('isoline')
# Show predefined VCS line objects
a.show('line')
# Change the VCS color map
a.setcolormap("AMIP")
# Plot data 's' with isoline 'i' and 'default' template
a.isoline(s,a,'default')
# Updates the VCS Canvas at user's request
a.update()
```

Create a canvas object:

```
a=vcs.init()
```

Create a new instance of isoline:

```
# Copies content of 'quick' to 'new'
iso=a.createisoline('new','quick')
# Copies content of 'default' to 'new'
iso=a.createisoline('new')
```

Modify an existing isoline:

```
iso=a.getisoline('AMIP_psl')
```

Overview of isoline attributes:

•List all the isoline attribute values

```
iso.list()
```

•Set isoline attribute values:

```
iso.projection='linear'
lon30={-180:'180W',-150:'150W',0:'Eq'}
iso.xticlabels1=lon30
iso.xticlabels2=lon30
# Will set them both
iso.xticlabels(lon30, lon30)
iso.xmtics1=''
iso.xmtics2=''
# Will set them both
iso.xmtics(lon30, lon30)
iso.yticlabels1=lat10
iso.yticlabels2=lat10
# Will set them both
iso.yticlabels(lat10, lat10)
iso.ymtics1=''
iso.ymtics2=''
# Will set them both
iso.ymtics(lat10, lat10)
iso.datawc_y1=-90.0
iso.datawc_y2=90.0
iso.datawc_x1=-180.0
iso.datawc_x2=180.0
# Will set them all
iso.datawc(-90, 90, -180, 180)
xaxisconvert='linear'
yaxisconvert='linear'
# Will set them both
iso.xyscale('linear', 'area_wt')
```

•Setting isoline *level* values:

```
#1) As a list of tuples (Examples):
    iso.level=[(23,32,45,50,76),]
    iso.level=[(22,33,44,55,66)]
    iso.level=[(20,0.0),(30,0),(50,0)]
    iso.level=[(23,32,45,50,76), (35, 45, 55)]
#2) As a tuple of lists (Examples):
```

```
    iso.level=([23,32,45,50,76],)
    iso.level=([22,33,44,55,66])
    iso.level=([23,32,45,50,76],)
    iso.level=([0,20,25,30,35,40],[30,40],[50,60])
#3) As a list of lists (Examples):
    iso.level=[[20,0.0],[30,0],[50,0]]
#4) As a tuple of tuples (Examples):
    iso.level=((20,0.0),(30,0),(50,0),(60,0),(70,0))
```

Note: A combination of a pairs (i.e., (30,0) or [30,0]) represents the isoline value plus its increment value. Thus, to let VCS generate "default" isolines:

```
# Same as iso.level=((0,1e20),)
iso.level=[[0,1e20]]
```

•Displaying isoline labels:

```
# Same as iso.label=1, will display isoline labels
iso.label='y'
# Same as iso.label=0, will turn isoline labels off
iso.label='n'
```

•Specify the isoline line style (or type):

```
# The following two lines of code are equivalent.
iso.line=([0,1,2,3,4])
# Both specify the isoline style
iso.line=(['solid, 'dash', 'dot', 'dash-dot', 'long-dash'])
```

•There are three possibilities for setting the line color indices:

```
# The following two lines of code are equivalent
# Both will set the isoline to a specific color index
iso.linecolors=(22,33,44,55,66,77)
iso.linecolors=([22,33,44,55,66,77])
# Turns off the line color index
iso.linecolors=None
```

•There are three possibilities for setting the line widths:

```
# The following two lines of code are equivalent
iso.linewidths=(1,10,3,4,5,6,7,8)
# Both will set the isoline to a specific width size
iso.linewidths=([1,2,3,4,5,6,7,8])
# Turns off the line width size
iso.linewidths=None
```

Note: If the number of line styles, colors or widths are less than the number of levels, we extend the attribute list using the last attribute value in the attribute list.

•There are three ways to specify the text or font number:

```
# Font numbers are between 1 and 9
iso.text=(1,2,3,4,5,6,7,8,9)
iso.text=[9,8,7,6,5,4,3,2,1]
iso.text=([1,3,5,6,9,2])
```

```
# Removes the text settings
iso.text=None
```

•There are three possibilities for setting the text color indices:

```
iso.textcolors=([22,33,44,55,66,77])
iso.textcolors=(16,19,33,44)
# Turns off the text color index
iso.textcolors=None
```

•Attribute descriptions:

label (*str*)

Turn on/off labels on isolines

labelskipdistance (*float*)

Minimum distance between isoline labels

labelbackgroundcolors ([*float*])

Background color for isoline labels

labelbackgroundopacities ([*float*])

Background opacity for isoline labels

level ([*float*, ...])

Isocountours to display

clockwise ([*int*, ...])

Draw directional arrows +-(0,1,2) Indicate none/clockwise/clokwise on y axis >0. Clockwise on x axis positive negative value invert behaviour

scale ([*float*, ...])

Scales the directional arrow lengths

angle ([*float*, ...])

Directional arrows head angle

spacing ([*float*, ...])

Scales spacing between directional arrows

xmtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

xmtics2 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 2nd side of y axis

ymtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

ymtics2 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 2nd side of y axis

xticlabels1 (*str/{float:str}*)
(Ex: '*') values for labels on 1st side of x axis

xticlabels2 (*str/{float:str}*)
(Ex: '*') values for labels on 2nd side of x axis

yticlabels1 (*str/{float:str}*)
(Ex: '*') values for labels on 1st side of y axis

yticlabels2 (*str/{float:str}*)
(Ex: '*') values for labels on 2nd side of y axis

projection (*str/vcs.projection.Proj*)
(Ex: 'default') projection to use, name or object

datawc_x1 (*float*)
(Ex: 1.E20) first value of xaxis on plot

datawc_x2 (*float*)
(Ex: 1.E20) second value of xaxis on plot

datawc_y1 (*float*)
(Ex: 1.E20) first value of yaxis on plot

datawc_y2 (*float*)
(Ex: 1.E20) second value of yaxis on plot

datawc_timeunits (*str*)
(Ex: 'days since 2000') units to use when displaying time dimension auto tick

datawc_calendar (*int*)
(Ex: 135441) calendar to use when displaying time dimension auto tick, default is proleptic gregorian calendar

line :: ([str,...]/[vcs.line.Tl,...]/[int,...]) (['solid',]) line type to use for each isoline, can also pass a line object or line object name

linecolors :: ([int,...]) ([241]) colors to use for each isoline linewidths :: ([float,...]) ([1.0]) list of width for each isoline

text :: (None/[vcs.textcombined.Tc,...]) (None) text objects or text objects names to use for each countour labels textcolors :: (None/[int,...]) (None) colors to use for each countour labels

datawc (*dsp1=1e+20*, *dsp2=1e+20*, *dsp3=1e+20*, *dsp4=1e+20*)
Sets the data world coordinates for object

Parameters

- **dsp1** (*float*) – Sets the *datawc_y1* property of the object.
- **dsp2** (*float*) – Sets the *datawc_y2* property of the object.
- **dsp3** (*float*) – Sets the *datawc_x1* property of the object.
- **dsp4** (*float*) – Sets the *datawc_x2* property of the object.

list ()
Lists the current values of object attributes

script (*script_filename*, *mode='a'*)
Saves out a copy of the isoline graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getisoline() # Get default isoline
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

xmtics (*xmt1=''*, *xmt2=''*)

Sets the xmtics1 and xmtics2 values on the object

Parameters

- **xmt1** (*{float:str} or str*) – Value for *xmtics1*. Must be a str, or a dictionary object with float:str mappings.
- **xmt2** (*{float:str} or str*) – Value for *xmtics2*. Must be a str, or a dictionary object with float:str mappings.

xticlabels (*xtl1=''*, *xtl2=''*)

Sets the xticlabels1 and xticlabels2 values on the object

Parameters

- **xtl1** (*{float:str} or str*) – Sets the object's value for *xticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **xtl2** (*{float:str} or str*) – Sets the object's value for *xticlabels2*. Must be a str, or a dictionary object with float:str mappings.

xyscale (*xat=''*, *yat=''*)

Sets xaxisconvert and yaxisconvert values for the object.

Example

```
>>> a=vcs.init()
>>> ex=a.createisoline('xyscale_ex') # create a boxfill to␣
↪work with
>>> ex.xyscale(xat='linear', yat='linear') # set xaxisconvert␣
↪and yaxisconvert to 'linear'
```

Parameters

- **xat** (*str*) – Set value for x axis conversion.
- **yat** (*str*) – Set value for y axis conversion.

ymtics (*ymt1=''*, *ymt2=''*)

Sets the ymtics1 and ymtics2 values on the object

Parameters

- **ymt1** (*{float:str} or str*) – Value for *ymtics1*. Must be a str, or a dictionary object with float:str mappings.
- **ymt2** (*{float:str} or str*) – Value for *ymtics2*. Must be a str, or a dictionary object with float:str mappings.

yticlabels (*ytl1=''*, *ytl2=''*)
Sets the yticlabels1 and yticlabels2 values on the object
Parameters

- **ytl1** (*{float:str} or str*) – Sets the object's value for *yticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **ytl2** (*{float:str} or str*) – Sets the object's value for *yticlabels2*. Must be a str, or a dictionary object with float:str mappings.

1.3.5 meshfill

Meshfill (Gfm) module

class vcs.meshfill.**Gfm**(*Gfm_name*, *Gfm_name_src='default'*)
The meshfill graphics method (Gfm) displays a two-dimensional data array by surrounding each data value by a colored grid mesh.

This class is used to define a meshfill table entry used in VCS, or it can be used to change some or all of the attributes in an existing meshfill table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined meshfill graphics methods
a.show('meshfill')
# Change the VCS color map
a.setcolormap("AMIP")
# Plot data 's' with meshfill 'b' and 'default' template
a.meshfill(s,b,'default')
# Updates the VCS Canvas at user's request
a.update()
```

Create a new instance of meshfill:

```
# Copies content of 'quick' to 'new'
mesh=a.createmeshfill('new','quick')
# Copies content of 'default' to 'new'
mesh=a.createmeshfill('new')
```

Modify an existing meshfill:

```
mesh=a.getmeshfill('AMIP_psl')
```

Overview of meshfill object attributes:

```

```

•List all the meshfill attribute values

```
mesh.list()
```

•Setting attributes:

–Setting general attributes:

```
mesh.projection='linear'
lon30={-180:'180W',-150:'150W',0:'Eq'}
mesh.xticlabels1=lon30
mesh.xticlabels2=lon30
# Will set them both
mesh.xticlabels(lon30, lon30)
mesh.xmtics1=''
mesh.xmtics2=''
# Will set them both
mesh.xmtics(lon30, lon30)
mesh.yticlabels1=lat10
mesh.yticlabels2=lat10
# Will set them both
mesh.yticlabels(lat10, lat10)
mesh.ymtics1=''
mesh.ymtics2=''
# Will set them both
mesh.ymtics(lat10, lat10)
mesh.datawc_y1=-90.0
mesh.datawc_y2=90.0
mesh.datawc_x1=-180.0
mesh.datawc_x2=180.0
# Will set them all
mesh.datawc(-90, 90, -180, 180)
mesh.ext_1='n'
mesh.ext_2='y'
# Will set them both
mesh.exts('n', 'y' )
# Color index value range 0 to 255
mesh.missing=241
```

–There are two possibilities for setting meshfill levels:

1.Levels are all contiguous:

```
mesh.levels=([0,20,25,30,35,40],)
mesh.levels=([0,20,25,30,35,40,45,50])
mesh.levels=[0,20,25,30,35,40]
mesh.levels=(0.0,20.0,25.0,30.0,35.0,40.0,50.0)
```

2.Levels are not contiguous (Examples):

```
mesh.levels=([0,20],[30,40],[50,60])
mesh.levels=([0,20,25,30,35,40],[30,40],[50,60])
```

–There are three ways to set fillarea color indices:

```
mesh.fillareacolors=([22,33,44,55,66,77])
mesh.fillareacolors=(16,19,33,44)
mesh.fillareacolors=None
```

–There are three ways to set fillarea style:

```
mesh.fillareastyle = 'solid'
mesh.fillareastyle = 'hatch'
mesh.fillareastyle = 'pattern'
```

–There are two ways to set fillarea hatch or pattern indices:

```
mesh.fillareaindices=([1,3,5,6,9,20])
mesh.fillareaindices=(7,1,4,9,6,15)
```

Using the fillarea secondary object:

•Create a new instance of fillarea:

```
# Copies 'quick' to 'new'
fill=a.createfillarea('new','quick')
# Copies 'default' to 'new'
fill=a.createfillarea('new')
```

•Modify an existing fillarea:

```
fill=a.getmfillarea('def37')
# Set index using fillarea
mesh.fillareaindices=(7,fill,4,9,fill,15)
# list fillarea attributes
fill.list()
# change style
fill.style='hatch'
# change color
fill.color=241
# change style index
fill.index=3
```

xmtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

xmtics2 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 2nd side of y axis

ymtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

ymtics2 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 2nd side of y axis

xticlabels1 (*str/{float:str}*)
(Ex: '*') values for labels on 1st side of x axis

xticlabels2 (*str/{float:str}*)
(Ex: '*') values for labels on 2nd side of x axis

yticlabels1 (*str/{float:str}*)
(Ex: '*') values for labels on 1st side of y axis

yticlabels2 (*str/{float:str}*)
(Ex: '*') values for labels on 2nd side of y axis

projection (*str/vcs.projection.Proj*)
(Ex: 'default') projection to use, name or object

datawc_x1 (*float*)
(Ex: 1.E20) first value of xaxis on plot

datawc_x2 (*float*)
(Ex: 1.E20) second value of xaxis on plot

datawc_y1 (*float*)
(Ex: 1.E20) first value of yaxis on plot

datawc_y2 (*float*)
(Ex: 1.E20) second value of yaxis on plot

datawc_timeunits (*str*)
(Ex: 'days since 2000') units to use when displaying time dimension auto tick

datawc_calendar (*int*)
(Ex: 135441) calendar to use when displaying time dimension auto tick, default is proleptic gregorian calendar

levels (*[float,...]/[[float,float],...]*)
Sets the levels range to use, can be either a list of contiguous levels, or list of tuples indicating first and last value of the range.

fillareacolors ([*int*, ...])
Colors to use for each level

fillareastyle (*str*)
Style to use for levels filling: solid/pattern/hatch

fillareaindices ([*int*, ...])
List of patterns to use when filling a level and using pattern/hatch

legend (*None/{float:str}*)
Replaces the legend values in the dictionary keys with their associated string

ext_1 (*str*)
Draws an extension arrow on right side (values less than first range value)

ext_2 (*str*)
Draws an extension arrow on left side (values greater than last range value)

missing (*int*)
Color to use for missing value or values not in defined ranges

mesh :: (str/int) (0) Draws the mesh wrap :: ([float,float]) ([0.,0.]) Modulo to wrap around on either axis (automatically sets to 360 for longitude axes)

colors (*color1=16*, *color2=239*)
Sets the color_1 and color_2 properties of the object.

Parameters

- **color1** (*int*) – Sets the `color_1` value on the object
- **color2** (*int*) – Sets the `color_2` value on the object

datawc (*dsp1=1e+20*, *dsp2=1e+20*, *dsp3=1e+20*, *dsp4=1e+20*)
Sets the data world coordinates for object

Parameters

- **dsp1** (*float*) – Sets the *datawc_y1* property of the object.
- **dsp2** (*float*) – Sets the *datawc_y2* property of the object.
- **dsp3** (*float*) – Sets the *datawc_x1* property of the object.
- **dsp4** (*float*) – Sets the *datawc_x2* property of the object.

exts (*ext1='n'*, *ext2='y'*)
Sets the ext_1 and ext_2 values on the object.

Parameters

- **ext1** (*str*) – Sets the *ext_1* value on the object. 'y' sets it to True, 'n' sets it to False.

- **ext2** (*str*) – Sets the *ext_2* value on the object. 'y' sets it to True, 'n' sets it to False.

list()
Lists the current values of object attributes

script(*script_filename*, *mode='a'*)
Saves out a copy of the meshfill graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getmeshfill() # Get default meshfill
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

xmtics(*xmt1=''*, *xmt2=''*)
Sets the xmtics1 and xmtics2 values on the object
Parameters

- **xmt1** (*{float:str} or str*) – Value for *xmtics1*. Must be a str, or a dictionary object with float:str mappings.
- **xmt2** (*{float:str} or str*) – Value for *xmtics2*. Must be a str, or a dictionary object with float:str mappings.

xticlabels(*xtl1=''*, *xtl2=''*)
Sets the xticlabels1 and xticlabels2 values on the object
Parameters

- **xtl1** (*{float:str} or str*) – Sets the object's value for *xticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **xtl2** (*{float:str} or str*) – Sets the object's value for *xticlabels2*. Must be a str, or a dictionary object with float:str mappings.

xyscale(*xat=''*, *yat=''*)
Sets xaxisconvert and yaxisconvert values for the object.
Example

```
>>> a=vcs.init()
>>> ex=a.createmeshfill('xyscale_ex') # create a boxfill to␣
↪work with
>>> ex.xyscale(xat='linear', yat='linear') # set xaxisconvert␣
↪and yaxisconvert to 'linear'
```

Parameters

- **xat** (*str*) – Set value for x axis conversion.
- **yat** (*str*) – Set value for y axis conversion.

ymtics (*ymt1=''*, *ymt2=''*)

Sets the ymtics1 and ymtics2 values on the object

Parameters

- **ymt1** (*{float:str} or str*) – Value for *ymtics1*. Must be a str, or a dictionary object with float:str mappings.
- **ymt2** (*{float:str} or str*) – Value for *ymtics2*. Must be a str, or a dictionary object with float:str mappings.

yticlabels (*ytl1=''*, *ytl2=''*)

Sets the yticlabels1 and yticlabels2 values on the object

Parameters

- **ytl1** (*{float:str} or str*) – Sets the object's value for *yticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **ytl2** (*{float:str} or str*) – Sets the object's value for *yticlabels2*. Must be a str, or a dictionary object with float:str mappings.

1.3.6 taylor

class vcs.taylor.**Gtd**(*name*, *source='default'*)

The Taylor Diagram graphics method (Gtd) is used to plot Taylor diagrams on a VCS Canvas. Taylor diagrams provide a way of graphically summarizing how closely a pattern matches observations.

defaultSkillFunction(*s*, *R*)

Provides a default function for determining the skill with which a model predicts observations. This function may be used in the function parameter of *drawSkill()*, although it may be preferable to provide a custom function for determining skill, depending on the application.

Parameters

- **s** (*float*) – A float representing the standard deviation of a model.
- **R** (*float*) – A float representing the correlation of a model.

Returns The skill of a model, computed using this function.

Return type float

drawSkill(*canvas*, *values*, *function=None*)

Draw a skill score. Default skill score provided in *defaultSkillFunction()* from Karl taylor, see PCMDI report series 55 for more information on Taylor diagrams and 'skill'_s.

Note: The function parameter must be provided for drawSkill to work. The *defaultSkillFunction()* provided in this module can be used to provide a default skill score. Be aware that, as stated in PCMDI report series 55 section 5, it is not possible to define a single

skill score that is appropriate for all models. It may be more suitable to create a custom function for determining the skill score of your model.

Parameters

- **canvas** (vcs.Canvas.Canvas) – A VCS Canvas object on which to draw the skill score.
- **values** (*list/tuple*) – A list/tuple used to specify the *levels* of an *isoline* object.
- **function** – A function for determining the skill score of a model.

getArc(*value*, *val1=0.0*, *val2=90.0*, *convert=True*)
Return coordinates to draw an arc from 0 to 90 degrees.

Note: val1 and val2 can be used to limit the arc (in degrees).

Parameters

- **value** (*float*) – The radius of the arc to be calculated.
- **val1** (*float*) – Lower limit of the arc to compute.
- **val2** (*float*) – Upper limit of the arc to compute.
- **convert** (*bool*) – Boolean flag indicating whether

Returns The coordinates for the calculated arc.

Return type tuple

plot(*data*, *template='deftaylor'*, *skill=None*, *bg=0*, *canvas=None*)
Plots an instance of a *Taylor diagram* on the provided VCS Canvas.

Parameters

- **data** –
- **template** (*str/vcs.template.P*) – A *VCS template* or a string name of a VCS template.
- **skill** –
- **bg** (*bool/int*) – A boolean/integer flag indicating whether to plot this object in the background.
- **canvas** (vcs.Canvas.Canvas) – A VCS Canvas object on which the diagram will be plotted.

script(*script_filename*, *mode='a'*)
Saves out a copy of the taylordiagram graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.gettaylordiagram() # Get default taylordiagram
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

class vcs.taylor.**TDMarker**
class

equalize()
Make sure that we have the same amount of everything usage self.equalize() Also updates self.number

1.3.7 unified1D

Unification of all 1D gms

class vcs.unified1D.**G1d**(*name*, *name_src='default'*)
This graphics method displays a line plot from 1D data array (i.e. a plot of Y(x), where y represents the 1D coordinate values, and x can be either Y's axis or another 1D arrays). The example below shows how to change line and marker attributes for the Yxvsx graphics method.

This class is used to define an Yxvsx table entry used in VCS, or it can be used to change some or all of the Yxvsx attributes in an existing Yxvsx table entry.

Make a Canvas object:
You'll need a Canvas object to work with.

```
# VCS Canvas constructor
a=vcs.init()
```

Create a new instance of Yxvsx:

```
# Copies content of 'quick' to 'new'
yxx=a.create1D('new','quick')
# Copies content of 'default' to 'new'
yxx=a.create1D('new')
```

Modify an existing Yxvsx:
•Get a YXvsX object to work with:

```
yxx=a.get1D('AMIP_psl')
```

•Overview of YXvsX attributes:

–To view YXvsX attributes:

```
# Will list all the Yxvsx attribute values
yxx.list()
```

–To set the projection attribute:

```
yxx.projection='linear'
```

Note: YXvsX projection attribute can only be 'linear' i.e. lon30={-180:'180W',-150:'150W',0:'Eq'}

–To set axis attributes:

```
yxx.xticlabels1=lon30
yxx.xticlabels2=lon30
# Will set them both
yxx.xticlabels(lon30, lon30)
yxx.xmtics1=''
yxx.xmtics2=''
# Will set them both
yxx.xmtics(lon30, lon30)
yxx.yticlabels1=lat10
yxx.yticlabels2=lat10
# Will set them both
yxx.yticlabels(lat10, lat10)
yxx.ymtics1=''
yxx.ymtics2=''
# Will set them both
yxx.ymtics(lat10, lat10)
yxx.datawc_y1=-90.0
yxx.datawc_y2=90.0
yxx.datawc_x1=-180.0
yxx.datawc_x2=180.0
# Will set them all
yxx.datawc(-90, 90, -180, 180)
yxx.xaxisconvert='linear'
```

–To specify the Yxvsx line type:

```
# same as yxx.line = 'solid'
yxx.line=0
# same as yxx.line = 'dash'
yxx.line=1
# same as yxx.line = 'dot'
yxx.line=2
# same as yxx.line = 'dash-dot'
yxx.line=3
# same as yxx.line = 'long-dash
yxx.line=4
```

–To specify the Yxvsx line color:

```
# color range: 16 to 230, default color is black
yxx.linecolor=16
# width range: 1 to 100, default color is 1
yxx.linewidth=1
```

–To specify the Yxvsx marker type:

```
# Same as yxx.marker='dot'
yxx.marker=1
```

```
# Same as yxx.marker='plus'
yxx.marker=2
# Same as yxx.marker='star'
yxx.marker=3
# Same as yxx.marker='circle'
yxx.marker=4
# Same as yxx.marker='cross'
yxx.marker=5
# Same as yxx.marker='diamond'
yxx.marker=6
# Same as yxx.marker='triangle_up'
yxx.marker=7
# Same as yxx.marker='triangle_down'
yxx.marker=8
# Same as yxx.marker='triangle_left'
yxx.marker=9
# Same as yxx.marker='triangle_right'
yxx.marker=10
# Same as yxx.marker='square'
yxx.marker=11
# Same as yxx.marker='diamond_fill'
yxx.marker=12
# Same as yxx.marker='triangle_up_fill'
yxx.marker=13
# Same as yxx.marker='triangle_down_fill'
yxx.marker=14
# Same as yxx.marker='triangle_left_fill'
yxx.marker=15
# Same as yxx.marker='triangle_right_fill'
yxx.marker=16
# Same as yxx.marker='square_fill'
yxx.marker=17
# Draw no markers
yxx.marker=None
```

–There are four possibilities for setting the marker color index:

```
# Same as below
yxx.markercolors=22
# Same as below
yxx.markercolors=(22)
# Will set the markers to a specific color index
yxx.markercolors=([22])
# Color index defaults to Black
yxx.markercolors=None
```

–To set the Yxvsx Marker size:

```
yxx.markersize=5
yxx.markersize=55
yxx.markersize=100
yxx.markersize=300
yxx.markersize=None
```

xmtics1 (*str/{float:str}*)
(Ex: '') dictionary with location of intermediate tics as keys for 1st side of y axis

xmtics2 (*str/{float:str}*)

(Ex: ‘’) dictionary with location of intermediate tics as keys for 2nd side of y axis

ymtics1 (*str/{float:str}*)
(Ex: ‘’) dictionary with location of intermediate tics as keys for 1st side of y axis

ymtics2 (*str/{float:str}*)
(Ex: ‘’) dictionary with location of intermediate tics as keys for 2nd side of y axis

xticlabels1 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 1st side of x axis

xticlabels2 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 2nd side of x axis

yticlabels1 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 1st side of y axis

yticlabels2 (*str/{float:str}*)
(Ex: ‘*’) values for labels on 2nd side of y axis

projection (*str/vcs.projection.Proj*)
(Ex: ‘default’) projection to use, name or object

datawc_x1 (*float*)
(Ex: 1.E20) first value of xaxis on plot

datawc_x2 (*float*)
(Ex: 1.E20) second value of xaxis on plot

datawc_y1 (*float*)
(Ex: 1.E20) first value of yaxis on plot

datawc_y2 (*float*)
(Ex: 1.E20) second value of yaxis on plot

datawc_timeunits (*str*)
(Ex: ‘days since 2000’) units to use when displaying time dimension auto tick

datawc_calendar (*int*)
(Ex: 135441) calendar to use when displaying time dimension auto tick, default is proleptic gregorian calendar

Parameters **xaxisconvert** – (Ex: ‘linear’) converting xaxis linear/log/log10/ln/exp/area_wt

linecolor :: (int) (241) colors to use for each isoline linewidth :: (float) (1.0) list of width for each isoline

marker :: (None/int/str/vcs.marker.Tm) (None) markers type to use markercolor :: (None/int) (None) color to use for markers markersize :: (None/int) (None) size of markers

datawc (*dsp1=1e+20*, *dsp2=1e+20*, *dsp3=1e+20*, *dsp4=1e+20*)
Sets the data world coordinates for object

Parameters

- **dsp1** (*float*) – Sets the *datawc_y1* property of the object.
- **dsp2** (*float*) – Sets the *datawc_y2* property of the object.
- **dsp3** (*float*) – Sets the *datawc_x1* property of the object.
- **dsp4** (*float*) – Sets the *datawc_x2* property of the object.

g_type
the 1d graphics method type

list()
Lists the current values of object attributes

script(*script_filename*, *mode='a'*)
Saves out a copy of the yxvsx graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getyxvsx() # Get default yxvsx
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

smooth
beta parameter for kaiser smoothing

xmtics(*xmt1=''*, *xmt2=''*)
Sets the xmtics1 and xmtics2 values on the object
Parameters

- **xmt1** (*{float:str} or str*) – Value for *xmtics1*. Must be a str, or a dictionary object with float:str mappings.
- **xmt2** (*{float:str} or str*) – Value for *xmtics2*. Must be a str, or a dictionary object with float:str mappings.

xticlabels(*xtl1=''*, *xtl2=''*)
Sets the xticlabels1 and xticlabels2 values on the object
Parameters

- **xtl1** (*{float:str} or str*) – Sets the object's value for *xticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **xtl2** (*{float:str} or str*) – Sets the object's value for *xticlabels2*. Must be a str, or a dictionary object with float:str mappings.

ymtics(*ymt1=''*, *ymt2=''*)
Sets the ymtics1 and ymtics2 values on the object
Parameters

- **ymt1** (*{float:str} or str*) – Value for *ymtics1*. Must be a str, or a dictionary object with float:str mappings.

- **ymt2** (*{float:str} or str*) – Value for *ymtics2*. Must be a str, or a dictionary object with float:str mappings.

yticlabels(*ytl1=''*, *ytl2=''*)

Sets the yticlabels1 and yticlabels2 values on the object

Parameters

- **ytl1** (*{float:str} or str*) – Sets the object's value for *yticlabels1*. Must be a str, or a dictionary object with float:str mappings.
- **ytl2** (*{float:str} or str*) – Sets the object's value for *yticlabels2*. Must be a str, or a dictionary object with float:str mappings.

1.3.8 vector

Vector (Gv) module

class vcs.vector.**Gv**(*Gv_name*, *Gv_name_src='default'*)

The vector graphics method displays a vector plot of a 2D vector field. Vectors are located at the coordinate locations and point in the direction of the data vector field. Vector magnitudes are the product of data vector field lengths and a scaling factor. The example below shows how to modify the vector's line, scale, alignment, type, and reference.

This class is used to define an vector table entry used in VCS, or it can be used to change some or all of the vector attributes in an existing vector table entry.

Useful Functions:

```
# Constructor
a=vcs.init()
# Show predefined vector graphics methods
a.show('vector')
# Show predefined VCS line objects
a.show('line')
# Change the VCS color Map
a.setcolormap("AMIP")
# Plot data 's1', and 's2' with vector 'v' and 'default' template
a.vector(s1, s2, v,'default')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of vector:

```
# Copies content of 'quick' to 'new'
vc=a.createvector('new','quick')
# Copies content of 'default' to 'new'
vc=a.createvector('new')
```

Modify an existing vector:

```
vc=a.getvector('AMIP_psl')
```

Overview of vector attributes:

•List all attributes:

```
# Will list all the vector attribute values
vc.list()
```

•Set axis attributes:

```
# Can only be 'linear'
vc.projection='linear'
lon30={-180:'180W',-150:'150W',0:'Eq'}
vc.xticlabels1=lon30
vc.xticlabels2=lon30
# Will set them both
vc.xticlabels(lon30, lon30)
vc.xmtics1=''
vc.xmtics2=''
# Will set them both
vc.xmtics(lon30, lon30)
vc.yticlabels1=lat10
vc.yticlabels2=lat10
# Will set them both
vc.yticlabels(lat10, lat10)
vc.ymtics1=''
vc.ymtics2=''
# Will set them both
vc.ymtics(lat10, lat10)
vc.datawc_y1=-90.0
vc.datawc_y2=90.0
vc.datawc_x1=-180.0
vc.datawc_x2=180.0
# Will set them all
vc.datawc(-90, 90, -180, 180)
xaxisconvert='linear'
yaxisconvert='linear'
# Will set them both
vc.xyscale('linear', 'area_wt')
```

•Specify the line style:

```
# Same as vc.line='solid'
vc.line=0
# Same as vc.line='dash'
vc.line=1
# Same as vc.line='dot'
vc.line=2
# Same as vc.line='dash-dot'
vc.line=3
# Same as vc.line='long-dot'
vc.line=4
```

•Specify the line color of the vectors:

```
# Color range: 16 to 230, default line color is black
vc.linecolor=16
# Width range: 1 to 100, default size is 1
vc.linewidth=1
```

•Specify the vector scale factor:

```
# Can be an integer or float
vc.scale=2.0
```

•Specify the vector alignment:

```
# Same as vc.alignment='head'
vc.alignment=0
# Same as vc.alignment='center'
vc.alignment=1
# Same as vc.alignment='tail'
vc.alignment=2
```

•Specify the vector type:

```
# Same as vc.type='arrow head'
vc.type=0
# Same as vc.type='wind barbs'
vc.type=1
# Same as vc.type='solid arrow head'
vc.type=2
```

•Specify the vector reference:

```
# Can be an integer or float
vc.reference=4
```

script (*script_filename=None*, *mode=None*)

Saves out a copy of the vector graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getvector() # Get default vector
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.4 Templating

Templates define the layout of your visualization. The Template object is used to lay out labels, lines, tick marks, etc.; it uses the P* objects to do so.

1.4.1 template

Template (P) module

class vcs.template.**P**(*Pic_name=None*, *Pic_name_src='default'*)

The template primary method (P) determines the location of each picture segment, the space to be allocated to it, and related properties relevant to its display.

Useful Functions:

```
# Show predefined templates
a.show('template')
# Show predefined text table methods
a.show('texttable')
# Show predefined text orientation methods
a.show('textorientation')
# Show predefined line methods
a.show('line')
# Show templates as a Python list
a.listelements('template')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
# VCS Canvas constructor
a=vcs.init()
```

Create a new instance of template:

```
# Two ways to create a templates:
# Copies content of 'hovmuller' to 'new'
temp=a.createtemplate('new','hovmuller')
# Copies content of 'default' to 'new'
temp=a.createtemplate('new')
```

Modify an existing template:

```
temp=a.gettemplate('hovmuller')
```

blank(*attribute=None*)

This function turns off elements of a template object.

Parameters **attribute** (*None, str, list*) – String or list, indicating the elements of a template which should be turned off. If attribute is left blank, or is None, all elements of the template will be turned off.

move (*p*, *axis*)

Move a template by p% along the axis 'x' or 'y'. Positive values of p mean movement toward right/top Negative values of p mean movement toward left/bottom The reference point is t.data.x1/y1

Example

```
>>> t = vcs.createtemplate('example1', 'default') # Create␣
↪template 'example1', inherits from 'default'
>>> t.move(0.2,'x') # Move everything right by 20%
>>> t.move(0.2,'y') # Move everything up by 20%
```

Parameters

- **p** (*float*) – Float indicating the percentage by which the template should move. i.e. 0.2 = 20%.
- **axis** (*str*) – One of ['x', 'y']. The axis along which the template will move.

moveto (*x*, *y*)

Move a template to point (x,y), adjusting all attributes so data.x1 = x, and data.y1 = y.

Example

```
>>> t = vcs.createtemplate('example1', 'default') # Create␣
↪template 'example1', inherits from 'default'
>>> t.moveto(0.2, 0.2) # Move everything so that data.x1= 0.2␣
↪and data.y1= 0.2
```

Parameters

- **x** (*float*) – Float representing the new coordinate of the template's data.x1 attribute.
- **y** (*float*) – Float representing the new coordinate of the template's data.y1 attribute.

scale (*scale*, *axis='xy'*, *font=-1*)

Scale a template along the axis 'x' or 'y' by scale Positive values of scale mean increase Negative values of scale mean decrease The reference point is t.data.x1/y1

Example

```
>>> t = vcs.createtemplate('example1', 'default') # Create␣
↪template 'example1', inherits from 'default'
>>> t.scale(0.5) # Halves the template size
>>> t.scale(1.2) # Upsize everything to 20% more than the␣
↪original size
>>> t.scale(2,'x') # Double the x axis
```

Parameters

- **scale** (*float*) – Float representing the factor by which to scale the template.
- **axis** (*str*) – One of ['x', 'y', 'xy']. Represents the axis/axes along which the template should be scaled.
- **font** (*int*) – Integer flag indicating what should be done with the template's fonts. One of [-1, 0, 1]. 0: means do not scale the fonts. 1: means scale the fonts. -1: means do not scale the fonts unless axis='xy'

script (*script_filename=None*, *mode=None*)

Saves out a copy of the template graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.gettemplate() # Get default template
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

vcs.template.**epsilon_gte**(*a*, *b*)
a >= b, using floating point epsilon value.

vcs.template.**epsilon_lte**(*a*, *b*)
a <= b, using floating point epsilon value.

1.4.2 Pboxeslines

Template Boxes and Lines (Pbl) module

class vcs.Pboxeslines.**Pbl**(*member*)
The Template text object allows the manipulation of line type, width, and color index.

This class is used to define a line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Example

```
# Basic Usage Overview:

a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()

#For mode:
#   If 1, then automatic update.
#   If 0,use update function to update VCS canvas
a.mode=1

#To Create a new instance of line use:
```

```
#    Copies content of 'red' to 'new'
ln=a.createline('new','red')
#    Copies content of 'default' to 'new'
ln=a.createline('new')

#To Modify an existing line use:
ln=a.getline('red')

# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100

 #Specify the line type:
 # Same as ln.type=0
 ln.type='solid'
 # Same as ln.type=1
 ln.type='dash'
 # Same as ln.type=2
 ln.type='dot'
 # Same as ln.type=3
 ln.type='dash-dot'
 # Same as ln.type=4
 ln.type='long-dash'
```

1.4.3 Pdata

Template Data Space (Pds) module

class vcs.Pdata.**Pds**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
```

```
# Copies content of 'default' to 'new'
ln=a.createeline('new')
```

Modify an existing line:

```
# Get a copy of 'red' line
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.4 Pformat

Template Format (Pf) module

class vcs.Pformat.**Pf**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createeline('new','red')
# Copies content of 'default' to 'new'
ln=a.createeline('new')
```

Modify an existing line:

```
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.5 Plegend

Template Legend Space (Pls) module

class vcs.Plegend.**Pls**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

```
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.6 Ptext

Template Text (Pt) module

class vcs.Ptext.**Pt**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

```
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.7 Pxlabels

Template X - Labels (Pxl) module

class vcs.Pxlabels.**Pxl**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
```

```
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

```
# Get a copy of 'red' line
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.8 Pxtickmarks

Template X - Tick Marks (Pxt) module

class vcs.Pxtickmarks.**Pxt**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

```
# Get a copy of 'red' line
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.9 Pylabels

Template Y - Labels (Pyl) module

class vcs.Pylabels.**Pyl** (*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

```
# Get a copy of 'red' line
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.4.10 Pytickmarks

Template Y - Tick Marks (Pyt) module

class vcs.Pytickmarks.**Pyt**(*member*)

The Template text object allows the manipulation of line type, width, and color index.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

```
# Get a copy of 'red' line
ln=a.getline('red')
```

Overview of line attributes:

•Listing line attributes:

```
# Will list all the line attribute values
ln.list()
# Range from 1 to 256
ln.color=100
# Range from 1 to 300
ln.width=100
```

•Specifying the line type:

```
# Same as ln.type=0
ln.type='solid'
# Same as ln.type=1
ln.type='dash'
# Same as ln.type=2
ln.type='dot'
# Same as ln.type=3
ln.type='dash-dot'
# Same as ln.type=4
ln.type='long-dash'
```

1.5 Secondary Graphics Methods

Secondary graphics methods define primitives that can be used to create arbitrary shapes, lines, glyphs, and labels on your visualization.

1.5.1 fillarea

Fillarea (Tf) module

class `vcs.fillarea.`**`Tf`**(*Tf_name=None*, *Tf_name_src='default'*)

The Fillarea class object allows the user to edit fillarea attributes, including fillarea interior style, style index, and color index.

This class is used to define an fillarea table entry used in VCS, or it can be used to change some or all of the fillarea attributes in an existing fillarea table entry.

`Useful Functions:`

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined fillarea objects
a.show('fillarea')
# Updates the VCS Canvas at user's request
a.update()
```

`Create a fillarea object:`

```
#Create a VCS Canvas object
a=vcs.init()

# Two ways to create a fillarea:

# Copies content of 'def37' to 'new'ea:
fa=a.createfillarea('new','def37')
# Copies content of 'default' to 'new'
fa=a.createfillarea('new')
```

`Modify an existing fillarea:`

```
fa=a.getfillarea('red')
```

•Overview of fillarea attributes:

–List all the fillarea attribute values

```
fa.list()
```

–There are three possibilities for setting the isofill style:

```
fa.style = 'solid'
fa.style = 'hatch'
fa.style = 'pattern'
```

–Setting index, color, opacity:

```
# Range from 1 to 20
fa.index=1
# Range from 1 to 256
fa.color=100
# Range from 0 to 100
fa.opacity=100
```

–Setting the graphics priority viewport, worldcoordinate:

```
fa.priority=1
# FloatType [0,1]x[0,1]
fa.viewport=[0, 1.0, 0,1.0]
# FloatType [#,#]x[#,#]
fa.worldcoordinate=[0,1.0,0,1.0]
```

–Setting x and y values:

```
#List of FloatTypes
fa.x=[[0,.1,.2], [.3,.4,.5]]
# List of FloatTypes
fa.y=[[.5,.4,.3], [.2,.1,0]]
```

script (*script_filename=None*, *mode=None*)

Saves out a copy of the fillarea secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getfillarea() # Get default fillarea
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.5.2 line

Line (Tl) module

class vcs.line.**Tl**(*Tl_name*, *Tl_name_src='default'*)

The Line object allows the manipulation of line type, width, color index, view port, world coordinates, and (x,y) points.

This class is used to define an line table entry used in VCS, or it can be used to change some or all of the line attributes in an existing line table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined line objects
a.show('line')
# Will list all the line attribute values
ln.list()
# Updates the VCS Canvas at user's request
a.update()
```

Create a new instance of line:

```
# Copies content of 'red' to 'new'
ln=a.createline('new','red')
# Copies content of 'default' to 'new'
ln=a.createline('new')
```

Modify an existing line:

•Get a line object 'ln' to manipulate:

```
ln=a.getline('red')
```

•Set line color:

```
# Range from 1 to 256
ln.color=100
```

•Set line width:

```
# Range from 1 to 300
ln.width=100
```

•Specify the line type:

```
# Same as ln.type=0
 ln.type='solid'
 # Same as ln.type=1
 ln.type='dash'
 # Same as ln.type=2
 ln.type='dot'
 # Same as ln.type=3
 ln.type='dash-dot'
 # Same as ln.type=4
 ln.type='long-dash'
```

•Set the graphics priority on the canvas:

```
ln.priority=1
# FloatType [0,1]x[0,1]
ln.viewport=[0, 1.0, 0,1.0]
# FloatType [#,#]x[#,#]
ln.worldcoordinate=[0,1.0,0,1.0]
```

•Set line x and y values:

```
# List of FloatTypes
ln.x=[[0,.1,.2], [.3,.4,.5]]
# List of FloatTypes
ln.y=[[.5,.4,.3], [.2,.1,0]]
```

script (*script_filename=None*, *mode=None*)
Saves out a copy of the line secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getline() # Get default line
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.5.3 marker

class vcs.marker.**Tm**(*Tm_name*, *Tm_name_src='default'*)
The Marker object allows the manipulation of marker type, size, and color index.

This class is used to define an marker table entry used in VCS, or it can be used to change some or all of the marker attributes in an existing marker table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined marker objects
a.show('marker')
# Updates the VCS Canvas at user's request
a.update()
a=vcs.init()
```

Create a new instance of marker:

```
# Copies content of 'red' to 'new'
mk=a.createmarker('new','red')
# Copies content of 'default' to 'new'
mk=a.createmarker('new')
```

Modify an existing marker:

```
mk=a.getmarker('red')
```

Overview of marker attributes:

•List all the marker attribute values:

```
mk.list()
# Range from 1 to 256
mk.color=100
# Range from 1 to 300
mk.size=100
```

•Specify the marker type:

```
# Same as mk.type=1
mk.type='dot'
# Same as mk.type=2
mk.type='plus'
# Same as mk.type=3
mk.type='star'
# Same as mk.type=4
mk.type='circle'
# Same as mk.type=5
mk.type='cross'
# Same as mk.type=6
mk.type='diamond'
# Same as mk.type=7
mk.type='triangle_up'
# Same as mk.type=8
mk.type='triangle_down'
# Same as mk.type=9
mk.type='triangle_left'
# Same as mk.type=10
mk.type='triangle_right'
# Same as mk.type=11
mk.type='square'
# Same as mk.type=12
mk.type='diamond_fill'
# Same as mk.type=13
mk.type='triangle_up_fill'
# Same as mk.type=14
mk.type='triangle_down_fill'
# Same as mk.type=15
mk.type='triangle_left_fill'
# Same as mk.type=16
mk.type='triangle_right_fill'
# Same as mk.type=17
mk.type='square_fill'
```

•Set the graphics priority on the canvas

```
mk.priority=1
# FloatType [0,1]x[0,1]
mk.viewport=[0, 1.0, 0,1.0]
# FloatType [#,#]x[#,#]
mk.worldcoordinate=[0,1.0,0,1.0]
```

•Example x and y coordinates:

```
# List of FloatTypes
mk.x=[[0,.1,.2], [.3,.4,.5]]
# List of FloatTypes
mk.y=[[.5,.4,.3], [.2,.1,0]]
```

script (*script_filename=None*, *mode=None*)

Saves out a copy of the marker secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getmarker() # Get default marker
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.5.4 textcombined

Text Combined (Tc) module

class vcs.textcombined.**Tc**(*Tt_name=None*, *Tt_name_src='default'*, *To_name=None*, *To_name_src='default'*)

The (Tc) Text Combined class will combine a text table class and a text orientation class together. From combining the two classess, the user will be able to set attributes for both classes (i.e., define the font, spacing, expansion, color index, height, angle, path, vertical alignment, and horizontal alignment).

This class is used to define and list a combined text table and text orientation entry used in VCS.

Useful Functions:

```
# Constructor
 a=vcs.init()
 # Show predefined text table objects
 a.show('texttable')
 # Show predefined text orientation objects
 a.show('textorientation')
 # Updates the VCS Canvas at user's request
 a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of text table:

```
# Copies content of 'std' to 'new_tt' and '7left' to 'new_to'
tc=a.createtextcombined('new_tt','std','new_to','7left')
```

Modify an existing texttable:

```
tc=a.gettextcombined('std','7left')
```

Overview of textcombined attributes:

Note: Textcombined attributes are a combination of texttable and textorientation attributes

•Listing the attributes:

```
# Will list all the textcombined attribute values
tc.list()
```

•Specify the text font type:

```
# The font value must be in the range 1 to 9
tc.font=1
```

•Specify the text spacing:

```
# The spacing value must be in the range -50 to 50
tc.spacing=2
```

•Specify the text expansion:

```
# The expansion value ranges from 50 to 150
tc.expansion=100
```

•Specify the text color:

```
# The text color value ranges from 1 to 257
tc.color=241
```

•Specify the graphics text priority on the VCS Canvas:

```
tt.priority = 1
```

•Specify the viewport and world coordinate:

```

```

FloatType [0,1]x[0,1] tt.viewport=[0, 1.0, 0,1.0] # FloatType [#,#]x[#,#] tt.worldcoordinate=[0,1.0,0,1.0]

•Specify the location of the text:

```
# List of FloatTypes
tt.x=[[0,.1,.2], [.3,.4,.5]]
# List of FloatTypes
tt.y=[[.5,.4,.3], [.2,.1,0]]
```

•Specify the text height:

```
# The height value must be an integer
tc.height=20
```

•Specify the text angle:

```
# The angle value ranges from 0 to 360
tc.angle=0
```

•Specify the text path:

```
# Same as tc.path=0
tc.path='right'
# Same as tc.path=1
tc.path='left'
# Same as tc.path=2
tc.path='up'
# Same as tc.path=3
tc.path='down'
```

•Specify the text horizontal alignment:

```
# Same as tc.halign=0
tc.halign='right'
# Same as tc.halign=1
tc.halign='center'
# Same as tc.halign=2
tc.halign='right'
```

•Specify the text vertical alignment:

```
# Same as tcvalign=0
tc.valign='tcp'
# Same as tcvalign=1
tc.valign='cap'
# Same as tcvalign=2
tc.valign='half'
# Same as tcvalign=3
tc.valign='base'
# Same as tcvalign=4
tc.valign='bottom'
```

script (*script_filename=None*, *mode=None*)

Saves out a copy of the text table and text orientation secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto',
 ↪'7left') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> ex=a.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto') # Get␣
 ↪default textcombined
>>> ex.script('filename.py') # Append to a Python script named
 ↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
 ↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.5.5 textorientation

Text Orientation (To) module

class vcs.textorientation.**To**(*To_name*, *To_name_src='default'*)

The (To) Text Orientation lists text attribute set names that define the font, spacing, expansion, and color index.

This class is used to define an text orientation table entry used in VCS, or it can be used to change some or all of the text orientation attributes in an existing text orientation table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined text orientation objects
a.show('textorientation')
# Updates the VCS Canvas at user's request
a.update()
```

Make a canvas object to work with:

```
a=vcs.init()
```

Create a new instance of text orientation:

```
# Copies content of '7left' to 'new'
to=a.createtextorientation('new','7left')
# Copies content of 'default' to 'new'
to=a.createtextorientation('new')
```

Modify an existing textorientation:

```
to=a.gettextorientation('7left')
```

Overview of textorientation attributes:

•Listing the attributes:

```
# Will list all the textorientation attribute values
to.list()
```

•Specify the text height:

```
# The height value must be an integer
to.height=20
```

•Specify the text angle:

```
# The angle value must be in the range 0 to 360
to.angle=0
```

•Specify the text path:

```
# Same as to.path=0
to.path='right'
# Same as to.path=1
to.path='left'
# Same as to.path=2
to.path='up'
# Same as to.path=3
to.path='down'
```

•Specify the text horizontal alignment:

```
# Same as to.halign=0
to.halign='right'
# Same as to.halign=1
to.halign='center'
# Same as to.halign=2
to.halign='right'
```

•Specify the text vertical alignment:

```
# Same as tovalign=0
to.valign='top'
# Same as tovalign=1
to.valign='cap'
# Same as tovalign=2
to.valign='half'
# Same as tovalign=3
to.valign='base'
# Same as tovalign=4
to.valign='bottom'
```

script(*script_filename=None*, *mode=None*)

Saves out a copy of the textorientation secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.gettextorientation() # Get default textorientation
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.5.6 texttable

Text Table (Tt) module

class vcs.texttable.**Tt**(*Tt_name=None*, *Tt_name_src='default'*)

The (Tt) Text Table lists text attribute set names that define the font, spacing, expansion, and color index.

This class is used to define an text table table entry used in VCS, or it can be used to change some or all of the text table attributes in an existing text table table entry.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined text table objects
a.show('texttable')
# Updates the VCS Canvas at user's request
a.update()
```

Make a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of text table:

```
# Copies content of 'std' to 'new'
tt=a.createtexttable('new','std')
```

```
# Copies content of 'default' to 'new'
tt=a.createtexttable('new')
```

Modify an existing texttable:

```
tt=a.gettexttable('std')
```

Overview of texttable attributes:

•Listing attributes:

```
# Will list all the texttable attribute values
tt.list()
```

•Specify the text font type:

```
# The font value must be in the range 1 to 9
tt.font=1
```

•Specify the text spacing:

```
# The spacing value must be in the range -50 to 50
tt.spacing=2
```

•Specify the text expansion:

```
# The expansion value must be in the range 50 to 150
tt.expansion=100
```

•Specify the text color:

```

```

The text color attribute value must be in the range 1 to 257 tt.color=241

•Specify the text background color and opacity:

```
# The text backgroundcolor attribute value must be in the␣
↪range 1 to 257
tt.backgroundcolor=241
# The text backgroundopacity attribute value must be in the␣
↪range 0 to 100
tt.backgroundopacity=0
# Set the graphics priority on the canvas
tt.priority=1
# FloatType [0,1]x[0,1]
tt.viewport=[0, 1.0, 0,1.0]
# FloatType [#,#]x[#,#]
tt.worldcoordinate=[0,1.0,0,1.0]
# List of FloatTypes
tt.x=[[0,.1,.2], [.3,.4,.5]]
# List of FloatTypes
tt.y=[[.5,.4,.3], [.2,.1,0]]
```

script (*script_filename=None*, *mode=None*)

Saves out a copy of the texttable secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.gettexttable() # Get default texttable
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.6 Miscellaneous Modules

These are a variety of modules from VCS that help out with useful functionality.

1.6.1 animate_helper

class vcs.animate_helper.**animate_obj_old**(*vcs_self*)

Animate the contents of the VCS Canvas. The animation can also be controlled from the animation GUI. (See VCDAT for more details.)

See the **'animation GUI documenation'_** .. _animation GUI documenation: http://www-pcmdi.llnl.gov/software/vcs

Example

```
>>> a=vcs.init()
>>> a.plot(array,'default','isofill','quick')
>>> a.animate()
```

1.6.2 projection

Projection (Proj) module

class vcs.projection.**Proj**(*Proj_name=None*, *Proj_name_src='default'*)

The projection secondary method (Proj) is used when plotting 2D data, and define how to project from lon/lat coord to another mapping system (lambert, mercator, mollweide, etc...)

This class is used to define a projection table entry used in VCS, or it can be used to change some or all of the attributes in an existing projection table entry.

Projection Transformation Package Projection Parameters

	Array Element								
Code & Projection Id	1	2	3	4	5	6	7	8	9
0 Geographic									
1 U T M	Lon/Z	Lat/Z							
2 State Plane									
3 Albers Equal Area	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN	
4 Lambert Conformal C	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN	
5 Mercator	SMajor	SMinor			CentMer	TrueScale	FE	FN	
6 Polar Stereographic	SMajor	SMinor			LongPol	TrueScale	FE	FN	
7 Polyconic	SMajor	SMinor			CentMer	OriginLat	FE	FN	
8 Equid. Conic A	SMajor	SMinor	STDPAR		CentMer	OriginLat	FE	FN	zero
Equid. Conic B	SMajor	SMinor	STDPR1	STDPR2	CentMer	OriginLat	FE	FN	one
9 Transverse Mercator	SMajor	SMinor	Factor		CentMer	OriginLat	FE	FN	
10 Stereographic	Sphere				CentLon	CenterLat	FE	FN	
11 Lambert Azimuthal	Sphere				CentLon	CenterLat	FE	FN	
12 Azimuthal	Sphere				CentLon	CenterLat	FE	FN	
13 Gnomonic	Sphere				CentLon	CenterLat	FE	FN	
14 Orthographic	Sphere				CentLon	CenterLat	FE	FN	
15 Gen. Vert. Near Per	Sphere		Height		CentLon	CenterLat	FE	FN	
16 Sinusoidal	Sphere				CentMer		FE	FN	
17 Equirectangular	Sphere				CentMer	TrueScale	FE	FN	
18 Miller Cylindrical	Sphere				CentMer		FE	FN	
19 Van der Grinten	Sphere				CentMer	OriginLat	FE	FN	
20 Hotin Oblique Merc A	SMajor	SMinor	Factor			OriginLat	FE	FN	Long1
Hotin Oblique Merc B	SMajor	SMinor	Factor	AziAng	AzmthPt	OriginLat	FE	FN	
21 Robinson	Sphere				CentMer		FE	FN	
22 Space Oblique Merc A	SMajor	SMinor		IncAng	AscLong		FE	FN	PSRev
Space Oblique Merc B	SMajor	SMinor	Satnum	Path			FE	FN	
23 Alaska Conformal	SMajor	SMinor					FE	FN	
24 Interrupted Goode	Sphere								
25 Mollweide	Sphere				CentMer		FE	FN	
26 Interrupt Mollweide	Sphere								
27 Hammer	Sphere				CentMer		FE	FN	
28 Wagner IV	Sphere				CentMer		FE	FN	
29 Wagner VII	Sphere				CentMer		FE	FN	
30 Oblated Equal Area	Sphere		Shapem	Shapen	CentLon	CenterLat	FE	FN	Angle

	Array Element			
Code & Projection Id	10	11	12	13
20 Hotin Oblique Merc A	Lat1	Long2	Lat2	zero
Hotin Oblique Merc B				one
22 Space Oblique Merc A	LRat	PFlag		zero
Space Oblique Merc B				one

Note: All other projections are blank (containing 0) for elements 10-13

Lon/Z Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
Lat/Z Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
SMajor Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.
SMinor Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.
Sphere Radius of reference sphere. If zero, 6370997 meters is used.

STDPAR Latitude of the standard parallel
STDPR1 Latitude of the first standard parallel
STDPR2 Latitude of the second standard parallel
CentMer Longitude of the central meridian
OriginLat Latitude of the projection origin
FE False easting in the same units as the semi-major axis
FN False northing in the same units as the semi-major axis
TrueScale Latitude of true scale
LongPol Longitude down below pole of map
Factor Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator)
CentLon Longitude of center of projection
CenterLat Latitude of center of projection
Height Height of perspective point
Long1 Longitude of first point on center line (Hotine Oblique Mercator, format A)
Long2 Longitude of second point on center line (Hotine Oblique Mercator, format A)
Lat1 Latitude of first point on center line (Hotine Oblique Mercator, format A)
Lat2 Latitude of second point on center line (Hotine Oblique Mercator, format A)
AziAng Azimuth angle east of north of center line (Hotine Oblique Mercator, format B)
AzmthPt Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B)
IncAng Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A)
AscLong Longitude of ascending orbit at equator (SOM, format A)
PSRev Period of satellite revolution in minutes (SOM, format A)
LRat Landsat ratio to compensate for confusion at northern end of orbit (SOM, format A – use 0.5201613)
PFlag End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, format A)
Satnum Landsat Satellite Number (SOM, format B)
Path Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4, 5 and 6.) (SOM, format B)
Shapem Oblated Equal Area oval shape parameter m
Shapen Oblated Equal Area oval shape parameter n
Angle Oblated Equal Area oval rotation angle

`Array Elements:`

- Array elements 14 and 15 are set to zero
- All array elements with blank fields are set to zero
- **All angles (latitudes, longitudes, azimuths, etc.) are entered in packed** degrees/ minutes/ seconds (DDDMMMSSS.SS) format

`Space Oblique Mercator A projection:`

- **A portion of Landsat rows 1 and 2 may also be seen as parts of rows** 246 or 247. To place these locations at rows 246 or 247, set the end of path flag (parameter 11) to 1–end of path. This flag defaults to zero.
- **When Landsat-1,2,3 orbits are being used, use the following values** for the specified parameters:
  - Parameter 4 099005031.2
  - Parameter 5 128.87 degrees - (360/251 * path number) in packed DMS format
  - Parameter 9 103.2669323
  - Parameter 10 0.5201613

- **When Landsat-4,5 orbits are being used, use the following values** for the specified parameters:
  - Parameter 4 098012000.0
  - Parameter 5 129.30 degrees - (360/233 * path number) in packed DMS format
  - Parameter 9 98.884119
  - Parameter 10 0.5201613

Note: In vcs angles can be entered either in DDDMMMSSS or regular angle format.

Useful Functions:

```
# VCS Canvas Constructor
a=vcs.init()
# Show predefined projection secondary methods
a.show('projection')
```

Create a Canvas object to work with:

```
a=vcs.init()
```

Create a new instance of projection:

```
# Copies content of 'quick' to 'new'
p=a.createprojection('new','quick')
# Copies content of 'default' to 'new'
p=a.createprojection('new')
```

Modify an existing projection:

```
p=a.getprojection('lambert')
# List all the projection attribute values
p.list()
p.type='lambert'
# Fill a list with projection parameter values
params= []
for _ in range(0,14):
   params.append(1.e20)
# params now a list with 1.e20, 15 times
p.parameters= params
iso=x.createisoline('new')
iso.projection=p
# or
iso.projection='lambert'
```

script(*script_filename=None*, *mode=None*)

Saves out a copy of the projection graphics method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's

data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getprojection() # Get default projection
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

1.6.3 colormap

Colormap (Cp) module

class vcs.colormap.**Cp**(*Cp_name*, *Cp_name_src='default'*)

The Colormap object allows the manipulation of the colormap index R,G,B values.

This class is used to define a colormap table entry used in VCS, or it can be used to change some or all of the colormap R,G,B attributes in an existing colormap table entry.

Some Useful Functions:

```
# Constructor
a=vcs.init()
# Show predefined colormap objects
a.show('colormap')
# Updates the VCS Canvas at user's request
a.update()
# If mode=1, automatic update
a.mode=1
#If mode=0, use update function to update the VCS Canvas.
a.mode=0
```

General use of a colormap:

```
# Create a VCS Canvas object
a=vcs.init()
#To Create a new instance of colormap use:
# Copies content of 'red' to 'new'
cp=a.createcolormap('new','quick')
# Copies content of 'default' to 'new'
cp=a.createcolormap('new')
```

Modifying an existing colormap:

```
cp=a.getcolormap('quick')
```

Overview of colormap attributes:

•List all the colormap indices and R,G,B attribute values

```
cp.list()
```

•Setting colormap attribute values:

```
# Index, R, G, B
cp.color=16,100,0,0
# Index range from 0 to 255, but can only modify from 0 to 239
cp.color=16,0,100,0
# R, G, B values range from 0 to 100, where 0 is low intensity␣
↪and 100 is highest intensity
cp.color=17,0,0,100
```

getcolorcell(*index*)

Gets the R,G,B,A values of a colorcell.

Example

```
>>> a=vcs.init() # Create a vcs Canvas
>>> cmap = a.createcolormap('example', 'default') # Create a␣
↪colormap
>>> cmap.getcolorcell(1) # Get RGBA values
[0, 0, 0, 100.0]
```

Parameters **index** (*int*) – Index of a cell in the colormap. Must be an integer from 0-255.

Returns A list containing the red, green, blue, and alpha values (in that order), of the colorcell at the given index.

Return type *list*

script(*script_filename=None*, *mode=None*)

Saves out a copy of the colormap secondary method in JSON, or Python format to a designated file.

Note: If the the filename has a '.py' at the end, it will produce a Python script. If no extension is given, then by default a .json file containing a JSON serialization of the object's data will be produced.

Warning: VCS Scripts Deprecated. SCR script files are no longer generated by this function.

Example

```
>>> a=vcs.init() # Make a Canvas object to work with
>>> ex=a.getcolormap() # Get default colormap
>>> ex.script('filename.py') # Append to a Python script named
↪'filename.py'
>>> ex.script('filename','w') # Create or overwrite a JSON␣
↪file 'filename.json'.
```

Parameters

- **script_filename** (*str*) – Output name of the script file. If no extension is specified, a .json object is created.
- **mode** (*str*) – Either 'w' for replace, or 'a' for append. Defaults to 'a', if not specified.

setcolorcell(*index*, *red*, *green*, *blue*, *alpha=100.0*)

Sets the R,G,B,A values of a colorcell

Example

```
>>> a = vcs.init() # Create a vcs Canvas
>>> cmap = a.createcolormap('example', 'default') # Create a␣
↪colormap
>>> cmap.setcolorcell(40,80,95,1.0) # Set RGBA values
```

Parameters

- **index** (*int*) – Integer from 0-255.
- **red** (*int*) – Integer from 0-255 representing the concentration of red in the colorcell.
- **green** (*int*) – Integer from 0-255 representing the concentration of green in the colorcell.
- **blue** (*int*) – Integer from 0-255 representing the concentration of blue in the colorcell.
- **alpha** (*float*) – Float representing the percentage of opacity in the colorcell.

1.6.4 colors

vcs.colors.**matplotlib2vcs**(*cmap*, *vcs_name=None*)

Convert a matplotlib colormap to a vcs colormap Input can be either the actual matplotlib colormap or its name Optional second argument: vcs_name, name of the resulting vcs colormap

Parameters

- **cmap** (*str , matplotlib colormap*) – A matplotlib colormap or string name of a matplotlib colormap
- **vcs_name** (*str*) – String to set the name of the generated VCS colormap

Returns A VCS colormap object

Return type *vcs.colormap.Cp*

1.6.5 displayplot

Display Plot (Dp) module

class vcs.displayplot.**Dp**(*Dp_name*, *Dp_name_src='default'*, *parent=None*)

The Display plot object allows the manipulation of the plot name, off, priority, template, graphics type, graphics name, and data array(s).

This class is used to define a display plot table entry used in VCS, or it can be used to change some or all of the display plot attributes in an existing display plot table entry.

Useful Functions:

```
# Canvas constructor
a=vcs.init()
# Show display plot objects
a.show('plot')
# Updates the VCS Canvas at user's request
a.update()
```

General display plot usage:

```
#Create a VCS Canvas object
a=vcs.init()
#To Create a new instance of plot:
# Create a plot object
p1=a.plot(s)
#To Modify an existing plot in use:
p1=a.getplot('dpy_plot_1')
```

Display plot object attributes:

```
# Will list all the display plot attributes
p1.list()
# "On" or "Off" status, 1=on, 0=off
p1.off=1
# Priority to place plot in front of other objects
p1.priority=1
# Name of template object
p1.template='quick'
# Graphics method type
p1.g_type='boxfill'
# Graphics method name
p1.g_name='quick'
# List of all the array names
p1.array=['a1']
```

backend
: dictionary of things the backend wants to be able to reuse

1.6.6 error

Error object for vcs module, vcsError

1.6.7 manageElements

vcs.manageElements.**check_name_source**(*name*, *source*, *typ*)
: make sure it is a unique name for this type or generates a name for user

vcs.manageElements.**create3d_dual_scalar**(*name=None*, *source='default'*)
: Create a new dv3d graphics method given the the name and the existing dv3d graphics method to copy the attributes from. If no existing dv3d graphics method is given, then the default dv3d graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('3d_dual_scalar') # show all available 3d_dual_scalar
********************3d_dual_scalar Names List**********************
...
********************End 3d_dual_scalar Names␣
↪List**********************
>>> ex=vcs.create3d_dual_scalar('3d_dual_scalar_ex1') # Create 3d_
↪dual_scalar '3d_dual_scalar_ex1' that inherits from 'default'
>>> vcs.listelements('3d_dual_scalar') # should now contain the '3d_
↪dual_scalar_ex1' 3d_dual_scalar
[...'3d_dual_scalar_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a 3d_dual_scalar or a string name of a 3d_dual_scalar*) – The object to inherit from

Returns A 3d_dual_scalar graphics method object

Return type vcs.dv3d.Gf3DDualScalar

vcs.manageElements.**create3d_scalar**(*name=None*, *source='default'*)

Create a new dv3d graphics method given the the name and the existing dv3d graphics method to copy the attributes from. If no existing dv3d graphics method is given, then the default dv3d graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('3d_scalar') # show all available 3d_scalar
********************3d_scalar Names List**********************
...
********************End 3d_scalar Names List**********************
>>> ex=vcs.create3d_scalar('3d_scalar_ex1') # Create 3d_scalar '3d_
↪scalar_ex1' that inherits from 'default'
>>> vcs.listelements('3d_scalar') # should now contain the '3d_
↪scalar_ex1' 3d_scalar
[...'3d_scalar_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a 3d_scalar or a string name of a 3d_scalar*) – The object to inherit from

Returns A 3d_scalar graphics method object

Return type vcs.dv3d.Gf3Dscalar

vcs.manageElements.**create3d_vector**(*name=None*, *source='default'*)

Create a new dv3d graphics method given the the name and the existing dv3d graphics method to copy the

attributes from. If no existing dv3d graphics method is given, then the default dv3d graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('3d_vector') # show all available 3d_vector
********************3d_vector Names List**********************
...
********************End 3d_vector Names List**********************
>>> ex=vcs.create3d_vector('3d_vector_ex1') # Create 3d_vector '3d_
↪vector_ex1' that inherits from 'default'
>>> vcs.listelements('3d_vector') # should now contain the '3d_
↪vector_ex1' 3d_vector
[...'3d_vector_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a 3d_vector or a string name of a 3d_vector*) – The object to inherit from

Returns A 3d_vector graphics method object

Return type vcs.dv3d.Gf3Dvector

vcs.manageElements.**createboxfill**(*name=None*, *source='default'*)

Create a new boxfill graphics method given the the name and the existing boxfill graphics method to copy the attributes from. If no existing boxfill graphics method is given, then the default boxfill graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('boxfill') # show all available boxfill
********************Boxfill Names List**********************
...
********************End Boxfill Names List**********************
>>> ex=vcs.createboxfill('boxfill_ex1') # Create boxfill 'boxfill_
↪ex1' that inherits from 'default'
>>> vcs.listelements('boxfill') # should now contain the 'boxfill_
↪ex1' boxfill
[...'boxfill_ex1'...]
>>> ex2=vcs.createboxfill('boxfill_ex2','polar') # create 'boxfill_
↪ex2' from 'polar' template
>>> vcs.listelements('boxfill') # should now contain the 'boxfill_
↪ex2' boxfill
[...'boxfill_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a boxfill or a string name of a boxfill*) – The object to inherit from

- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean

- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A boxfill graphics method object

Return type *vcs.boxfill.Gfb*

vcs.manageElements.**createcolormap**(*Cp_name=None*, *Cp_name_src='default'*)

Create a new colormap secondary method given the the name and the existing colormap secondary method to copy the attributes from. If no existing colormap secondary method is given, then the default colormap secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('colormap') # show all available colormap
********************Colormap Names List**********************
...
********************End Colormap Names List**********************
>>> ex=vcs.createcolormap('colormap_ex1') # Create colormap
↪'colormap_ex1' that inherits from 'default'
>>> vcs.listelements('colormap') # should now contain the 'colormap_
↪ex1' colormap
[...'colormap_ex1'...]
>>> ex2=vcs.createcolormap('colormap_ex2','rainbow') # create
↪'colormap_ex2' from 'rainbow' template
>>> vcs.listelements('colormap') # should now contain the 'colormap_
↪ex2' colormap
[...'colormap_ex2'...]
```

Parameters

- **Cp_name** (*str*) – The name of the created object

- **Cp_name_src** (*a colormap or a string name of a colormap*) – The object to inherit

Returns A VCS colormap object

Return type *vcs.colormap.Cp*

vcs.manageElements.**createfillarea**(*name=None*, *source='default'*, *style=None*, *index=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

Create a new fillarea secondary method given the the name and the existing fillarea secondary method to copy the attributes from. If no existing fillarea secondary method is given, then the default fillarea secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('fillarea') # show all available fillarea
********************Fillarea Names List**********************
...
********************End Fillarea Names List**********************
>>> ex=vcs.createfillarea('fillarea_ex1') # Create fillarea
↪'fillarea_ex1' that inherits from 'default'
>>> vcs.listelements('fillarea') # should now contain the 'fillarea_
↪ex1' fillarea
[...'fillarea_ex1'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – a fillarea, or string name of a fillarea
- **style** (*str*) – One of "hatch", "solid", or "pattern".
- **index** – Specifies which pattern to fill with.

Accepts ints from 1-20.

Parameters **color** – A color name from the X11 Color Names list,
or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))

Parameters

- **priority** (*int*) – The layer on which the fillarea will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A fillarea object

Return type *vcs.fillarea.Tf*

vcs.manageElements.**createisofill**(*name=None*, *source='default'*)

Create a new isofill graphics method given the the name and the existing isofill graphics method to copy the

attributes from. If no existing isofill graphics method is given, then the default isofill graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('isofill') # show all available isofill
********************Isofill Names List**********************
...
********************End Isofill Names List**********************
>>> ex=vcs.createisofill('isofill_ex1') # Create isofill 'isofill_
↪ex1' that inherits from 'default'
>>> vcs.listelements('isofill') # should now contain the 'isofill_
↪ex1' isofill
[...'isofill_ex1'...]
>>> ex2=vcs.createisofill('isofill_ex2','polar') # create 'isofill_
↪ex2' from 'polar' template
>>> vcs.listelements('isofill') # should now contain the 'isofill_
↪ex2' isofill
[...'isofill_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*an isofill object, or string name of an isofill object*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot

- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns An isofill graphics method
Return type *vcs.isofill.Gfi*

vcs.manageElements.**createisoline**(*name=None*, *source='default'*)

Create a new isoline graphics method given the the name and the existing isoline graphics method to copy the attributes from. If no existing isoline graphics method is given, then the default isoline graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('isoline') # show all available isoline
********************Isoline Names List**********************
...
********************End Isoline Names List**********************
>>> ex=vcs.createisoline('isoline_ex1') # Create isoline 'isoline_
↪ex1' that inherits from 'default'
>>> vcs.listelements('isoline') # should now contain the 'isoline_
↪ex1' isoline
[...'isoline_ex1'...]
>>> ex2=vcs.createisoline('isoline_ex2','polar') # create 'isoline_
↪ex2' from 'polar' template
>>> vcs.listelements('isoline') # should now contain the 'isoline_
↪ex2' isoline
[...'isoline_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*an isoline object, or string name of an isoline object*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number

- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns An isoline graphics method object
Return type *vcs.isoline.Gi*

vcs.manageElements.**createline**(*name=None*, *source='default'*, *ltype=None*, *width=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *projection=None*)

Create a new line secondary method given the the name and the existing line secondary method to copy the attributes from. If no existing line secondary method is given, then the default line secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('line') # show all available line
********************Line Names List**********************
...
********************End Line Names List**********************
>>> ex=vcs.createline('line_ex1') # Create line 'line_ex1' that␣
↪inherits from 'default'
>>> vcs.listelements('line') # should now contain the 'line_ex1'␣
↪line
[...'line_ex1'...]
>>> ex2=vcs.createline('line_ex2','red') # create 'line_ex2' from
↪'red' template
>>> vcs.listelements('line') # should now contain the 'line_ex2'␣
↪line
[...'line_ex2'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – a line, or string name of a line
- **ltype** (*str*) – One of “dash”, “dash-dot”, “solid”, “dot”, or “long-dash”.
- **width** (*int*) – Thickness of the line to be created
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the line will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS line secondary method object
Return type *vcs.line.Tl*

vcs.manageElements.**createmarker**(*name=None*, *source='default'*, *mtype=None*, *size=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *projection=None*)

Create a new marker secondary method given the the name and the existing marker secondary method to copy the attributes from. If no existing marker secondary method is given, then the default marker secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('marker') # show all available marker
********************Marker Names List**********************
...
********************End Marker Names List**********************
>>> ex=vcs.createmarker('marker_ex1') # Create marker 'marker_ex1'␣
↪that inherits from 'default'
>>> vcs.listelements('marker') # should now contain the 'marker_ex1
↪' marker
[...'marker_ex1'...]
>>> ex2=vcs.createmarker('marker_ex2','red') # create 'marker_ex2'␣
↪from 'red' template
>>> vcs.listelements('marker') # should now contain the 'marker_ex2
↪' marker
[...'marker_ex2'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – A marker, or string name of a marker
- **mtype** (*str*) – Specifies the type of marker, i.e. “dot”, “circle”
- **size** (*int*) –
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A secondary marker method

Return type *vcs.marker.Tm*

vcs.manageElements.**createmeshfill**(*name=None*, *source='default'*)

Create a new meshfill graphics method given the the name and the existing meshfill graphics method to copy the attributes from. If no existing meshfill graphics method is given, then the default meshfill graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('meshfill') # show all available meshfill
********************Meshfill Names List**********************
...
********************End Meshfill Names List**********************
>>> ex=vcs.createmeshfill('meshfill_ex1') # Create meshfill
↪'meshfill_ex1' that inherits from 'default'
>>> vcs.listelements('meshfill') # should now contain the 'meshfill_
↪ex1' meshfill
[...'meshfill_ex1'...]
>>> ex2=vcs.createmeshfill('meshfill_ex2','a_polar_meshfill') #␣
↪create 'meshfill_ex2' from 'a_polar_meshfill' template
>>> vcs.listelements('meshfill') # should now contain the 'meshfill_
↪ex2' meshfill
[...'meshfill_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a meshfill or a string name of a meshfill*) – The object to inherit from

Returns A meshfill graphics method object

Return type *vcs.meshfill.Gfm*

vcs.manageElements.**createprojection**(*name=None*, *source='default'*)

Create a new projection graphics method given the the name and the existing projection graphics method to copy the attributes from. If no existing projection graphics method is given, then the default projection graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('projection') # show all available projection
********************Projection Names List**********************
...
********************End Projection Names List**********************
>>> ex=vcs.createprojection('projection_ex1') # Create projection
↪'projection_ex1' that inherits from 'default'
>>> vcs.listelements('projection') # should now contain the
↪'projection_ex1' projection
[...'projection_ex1'...]
>>> ex2=vcs.createprojection('projection_ex2','polar') # create
↪'projection_ex2' from 'polar' template
>>> vcs.listelements('projection') # should now contain the
↪'projection_ex2' projection
[...'projection_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object

- **source** (*a projection or a string name of a projection*) – The object to inherit from

Returns A projection graphics method object

Return type *vcs.projection.Proj*

vcs.manageElements.**createscatter**(*name=None*, *source='default'*)

Create a new scatter graphics method given the the name and the existing scatter graphics method to copy the attributes from. If no existing scatter graphics method is given, then the default scatter graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('scatter') # show all available scatter
********************Scatter Names List**********************
...
********************End Scatter Names List**********************
>>> ex=vcs.createscatter('scatter_ex1') # Create scatter 'scatter_
↪ex1' that inherits from 'default'
>>> vcs.listelements('scatter') # should now contain the 'scatter_
↪ex1' scatter
[...'scatter_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a scatter or a string name of a scatter*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number

- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A scatter graphics method
Return type *vcs.unified1D.G1d*

vcs.manageElements.**createtaylordiagram**(*name=None*, *source='default'*)
Create a new taylordiagram graphics method given the the name and the existing taylordiagram graphics method to copy the attributes from. If no existing taylordiagram graphics method is given, then the default taylordiagram graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('taylordiagram') # show all available taylordiagram
********************Taylordiagram Names List**********************
...
********************End Taylordiagram Names␣
↪List**********************
>>> ex=vcs.createtaylordiagram('taylordiagram_ex1') # Create␣
↪taylordiagram 'taylordiagram_ex1' that inherits from 'default'
>>> vcs.listelements('taylordiagram') # should now contain the
↪'taylordiagram_ex1' taylordiagram
[...'taylordiagram_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a taylordiagram or a string name of a*) – The object to inherit from

Returns A taylordiagram graphics method object
Return type *vcs.taylor.Gtd*

vcs.manageElements.**createtemplate**(*name=None*, *source='default'*)
Create a new template graphics method given the the name and the existing template graphics method to copy the attributes from. If no existing template graphics method is given, then the default template graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('template') # show all available template
********************Template Names List**********************
...
********************End Template Names List**********************
>>> ex=vcs.createtemplate('template_ex1') # Create template
↪'template_ex1' that inherits from 'default'
>>> vcs.listelements('template') # should now contain the 'template_
↪ex1' template
[...'template_ex1'...]
>>> ex2=vcs.createtemplate('template_ex2','polar') # create
↪'template_ex2' from 'polar' template
>>> vcs.listelements('template') # should now contain the 'template_
↪ex2' template
[...'template_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a template or a string name of a template*) – The object to inherit from

Returns A template

Return type *vcs.template.P*

vcs.manageElements.**createtext**(*Tt_name=None*, *Tt_source='default'*, *To_name=None*, *To_source='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*, *projection=None*)

Create a new textcombined secondary method given the the name and the existing textcombined secondary method to copy the attributes from. If no existing textcombined secondary method is given, then the default textcombined secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('textcombined') # show all available textcombined
********************Textcombined Names List**********************
...
********************End Textcombined Names List**********************
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> vcs.listelements('textcombined') # should now contain the 'qa_
↪tt:::left_tto' textcombined
[...'qa_tt:::left_tto'...]
```

Parameters

- **Tt_name** (*str*) – Name of created object
- **Tt_source** (*str or vcs.texttable.Tt*) – Texttable object to inherit from. Can be a texttable, or a string name of a texttable.
- **To_name** (*str*) – Name of the textcombined's text orientation (to be created)
- **To_source** (*str or vcs.textorientation.To*) – Name of the textorientation to inherit. Can be a textorientation, or a string name of a textorientation.
- **font** (*int or str*) – Which font to use (index or name).
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas

- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*int*) – Angle of the text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of [”left”, “center”, “right”].
- **valign** (*str*) – Vertical alignment of the text. One of [”top”, “center”, “botom”].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS text object
Return type *vcs.textcombined.Tc*

vcs.manageElements.**createtextcombined**(*Tt_name=None*, *Tt_source='default'*, *To_name=None*, *To_source='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*, *projection=None*)

Create a new textcombined secondary method given the the name and the existing textcombined secondary method to copy the attributes from. If no existing textcombined secondary method is given, then the default textcombined secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('textcombined') # show all available textcombined
********************Textcombined Names List**********************
...
********************End Textcombined Names List**********************
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> vcs.listelements('textcombined') # should now contain the 'qa_
↪tt:::left_tto' textcombined
[...'qa_tt:::left_tto'...]
```

Parameters

- **Tt_name** (*str*) – Name of created object
- **Tt_source** (*str or vcs.texttable.Tt*) – Texttable object to inherit from. Can be a texttable, or a string name of a texttable.
- **To_name** (*str*) – Name of the textcombined's text orientation (to be created)

- **To_source** (*str or vcs.textorientation.To*) – Name of the textorientation to inherit. Can be a textorientation, or a string name of a textorientation.
- **font** (*int or str*) – Which font to use (index or name).
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*int*) – Angle of the text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of [”left”, “center”, “right”].
- **valign** (*str*) – Vertical alignment of the text. One of [”top”, “center”, “botom”].
- **projection** (*str or projection object*) – Specify a geographic projection used to convert x/y from spherical coordinates into 2D coordinates.

Returns A VCS text object

Return type *vcs.textcombined.Tc*

vcs.manageElements.**createtextorientation**(*name=None*, *source='default'*)

Create a new textorientation secondary method given the the name and the existing textorientation secondary method to copy the attributes from. If no existing textorientation secondary method is given, then the default textorientation secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('textorientation') # show all available textorientation
********************Textorientation Names List**********************
...
********************End Textorientation Names␣
↪List**********************
>>> ex=vcs.createtextorientation('textorientation_ex1') # Create␣
↪textorientation 'textorientation_ex1' that inherits from 'default'
>>> vcs.listelements('textorientation') # should now contain the
↪'textorientation_ex1' textorientation
[...'textorientation_ex1'...]
```

```
>>> ex2=vcs.createtextorientation('textorientation_ex2','bigger') #␣
↪create 'textorientation_ex2' from 'bigger' template
>>> vcs.listelements('textorientation') # should now contain the
↪'textorientation_ex2' textorientation
[...'textorientation_ex2'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a textorientation or a string name of a textorientation*) – The object to inherit from

Returns A textorientation secondary method

Return type *vcs.textorientation.To*

vcs.manageElements.**createtexttable**(*name=None*, *source='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

Create a new texttable secondary method given the the name and the existing texttable secondary method to copy the attributes from. If no existing texttable secondary method is given, then the default texttable secondary method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. secondary method names must be unique.

Example

```
>>> vcs.show('texttable') # show all available texttable
********************Texttable Names List**********************
...
********************End Texttable Names List**********************
>>> ex=vcs.createtexttable('texttable_ex1') # Create texttable
↪'texttable_ex1' that inherits from 'default'
>>> vcs.listelements('texttable') # should now contain the
↪'texttable_ex1' texttable
[...'texttable_ex1'...]
>>> ex2=vcs.createtexttable('texttable_ex2','bigger') # create
↪'texttable_ex2' from 'bigger' template
>>> vcs.listelements('texttable') # should now contain the
↪'texttable_ex2' texttable
[...'texttable_ex2'...]
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – a texttable, or string name of a texttable
- **font** (*int or string*) – Which font to use (index or name).
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the texttable will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas

- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between world-coordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between world-coordinate[2] and worldcoordinate[3].

Returns A texttable graphics method object
Return type *vcs.texttable.Tt*

vcs.manageElements.**createvector**(*name=None*, *source='default'*)
Create a new vector graphics method given the the name and the existing vector graphics method to copy the attributes from. If no existing vector graphics method is given, then the default vector graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('vector') # show all available vector
********************Vector Names List**********************
...
********************End Vector Names List**********************
>>> ex=vcs.createvector('vector_ex1') # Create vector 'vector_ex1'␣
↪that inherits from 'default'
>>> vcs.listelements('vector') # should now contain the 'vector_ex1
↪' vector
[...'vector_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a vector or a string name of a vector*) – The object to inherit from

Returns A vector graphics method object
Return type *vcs.vector.Gv*

vcs.manageElements.**createxvsy**(*name=None*, *source='default'*)
Create a new xvsy graphics method given the the name and the existing xvsy graphics method to copy the attributes from. If no existing xvsy graphics method is given, then the default xvsy graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('xvsy') # show all available xvsy
********************Xvsy Names List**********************
...
********************End Xvsy Names List**********************
>>> ex=vcs.createxvsy('xvsy_ex1') # Create xvsy 'xvsy_ex1' that␣
↪inherits from 'default'
>>> vcs.listelements('xvsy') # should now contain the 'xvsy_ex1'␣
↪xvsy
```

```
[...'xvsy_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a xvsy or a string name of a xvsy*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)

- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A XvsY graphics method object
Return type *vcs.unified1D.G1d*

vcs.manageElements.**createxyvsy**(*name=None*, *source='default'*)
Create a new xyvsy graphics method given the the name and the existing xyvsy graphics method to copy the attributes from. If no existing xyvsy graphics method is given, then the default xyvsy graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('xyvsy') # show all available xyvsy
********************Xyvsy Names List**********************
...
********************End Xyvsy Names List**********************
>>> ex=vcs.createxyvsy('xyvsy_ex1') # Create xyvsy 'xyvsy_ex1' that␣
↪inherits from 'default'
>>> vcs.listelements('xyvsy') # should now contain the 'xyvsy_ex1'␣
↪xyvsy
[...'xyvsy_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object
- **source** (*a xyvsy or a string name of a xyvsy*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)

- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A XYvsY graphics method object
Return type *vcs.unified1D.G1d*

vcs.manageElements.**createyxvsx**(*name=None*, *source='default'*)

Create a new yxvsx graphics method given the the name and the existing yxvsx graphics method to copy the attributes from. If no existing yxvsx graphics method is given, then the default yxvsx graphics method will be used as the graphics method to which the attributes will be copied from.

Note: If the name provided already exists, then an error will be returned. graphics method names must be unique.

Example

```
>>> vcs.show('yxvsx') # show all available yxvsx
********************Yxvsx Names List**********************
...
********************End Yxvsx Names List**********************
>>> ex=vcs.createyxvsx('yxvsx_ex1') # Create yxvsx 'yxvsx_ex1' that␣
↪inherits from 'default'
>>> vcs.listelements('yxvsx') # should now contain the 'yxvsx_ex1'␣
↪yxvsx
[...'yxvsx_ex1'...]
```

Parameters

- **name** (*str*) – The name of the created object

- **source** (*a yxvsy or a string name of a yxvsy*) – The object to inherit from
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)

- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **new_GM_name** (*str*) – (Ex: 'my_awesome_gm') name of the new graphics method object. If no name is given, then one will be created for use.
- **source_GM_name** – (Ex: 'default') copy the contents of the source object to the newly created one. If no name is given, then the 'default' graphics methond contents is copied over to the new object.

Returns A YXvsX graphics method object

Return type *vcs.unified1D.G1d*

vcs.manageElements.**get3d_dual_scalar**(*Gfdv3d_name_src='default'*)

VCS contains a list of graphics methods. This function will create a dv3d class object from an existing VCS dv3d graphics method. If no dv3d name is given, then dv3d 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.create3d_dual_scalar()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('3d_dual_scalar') # Show all the existing 3d_
↪dual_scalar graphics methods
[...]
>>> ex=vcs.get3d_dual_scalar()  # instance of 'default' 3d_dual_
↪scalar graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.plot(ex, slab1, slab2) # plot using specified 3d_dual_scalar␣
↪object
<vcs.displayplot.Dp ...>
```

Parameters **Gfdv3d_name_src** (*str*) – String name of an existing 3d_dual_scalar VCS object

Returns A pre-existing 3d_dual_scalar VCS object
Return type vcs.dv3d.Gf3DDualScalar

vcs.manageElements.**get3d_scalar**(*Gfdv3d_name_src='default'*)
VCS contains a list of graphics methods. This function will create a dv3d class object from an existing VCS dv3d graphics method. If no dv3d name is given, then dv3d 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.create3d_scalar()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('3d_scalar') # Show all the existing 3d_scalar␣
↪graphics methods
[...]
>>> ex=vcs.get3d_scalar()  # instance of 'default' 3d_scalar␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.plot(ex, slab1) # plot using specified 3d_scalar object
<vcs.displayplot.Dp ...>
```

Parameters **Gfdv3d_name_src** (*str*) – String name of an existing 3d_scalar VCS object.
Returns A pre-existing 3d_scalar VCS object
Return type vcs.dv3d.Gf3Dscalar

vcs.manageElements.**get3d_vector**(*Gfdv3d_name_src='default'*)
VCS contains a list of graphics methods. This function will create a dv3d class object from an existing VCS dv3d graphics method. If no dv3d name is given, then dv3d 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.create3d_vector()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('3d_vector') # Show all the existing 3d_vector␣
↪graphics methods
[...]
>>> ex=vcs.get3d_vector()  # instance of 'default' 3d_vector␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.plot(ex, slab1, slab2) # plot using specified 3d_vector object
<vcs.displayplot.Dp ...>
```

Parameters **Gfdv3d_name_src** (*str*) – String name of an existing 3d_vector VCS object
Returns A pre-existing 3d_vector VCS object

Return type vcs.dv3d.Gf3Dvector

vcs.manageElements.**getboxfill**(*Gfb_name_src='default'*)

VCS contains a list of graphics methods. This function will create a boxfill class object from an existing VCS boxfill graphics method. If no boxfill name is given, then boxfill 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createboxfill()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('boxfill') # Show all the existing boxfill␣
↪graphics methods
[...]
>>> ex=vcs.getboxfill()  # instance of 'default' boxfill graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.boxfill(ex, slab1) # plot using specified boxfill object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getboxfill('polar')  # instance of 'polar' boxfill␣
↪graphics method
>>> a.boxfill(ex2, slab1) # plot using specified boxfill object
<vcs.displayplot.Dp ...>
```

Parameters

- **Gfb_name_src** (*str*) – String name of an existing boxfill VCS object
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D

- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A pre-existing boxfill graphics method
Return type *vcs.boxfill.Gfb*

vcs.manageElements.**getcolormap**(*Cp_name_src='default'*)

VCS contains a list of secondary methods. This function will create a colormap class object from an existing VCS colormap secondary method. If no colormap name is given, then colormap 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createcolormap()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('colormap') # Show all the existing colormap␣
↪secondary methods
[...]
>>> ex=vcs.getcolormap()  # instance of 'default' colormap␣
↪secondary method
>>> ex2=vcs.getcolormap('rainbow')  # instance of 'rainbow'␣
↪colormap secondary method
```

Parameters **Cp_name_src** (*str*) – String name of an existing colormap VCS object
Returns A pre-existing VCS colormap object
Return type *vcs.colormap.Cp*

vcs.manageElements.**getfillarea**(*name='default'*, *style=None*, *index=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a fillarea class object from an existing VCS fillarea secondary method. If no fillarea name is given, then fillarea 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createfillarea()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('fillarea') # Show all the existing fillarea␣
↪secondary methods
[...]
>>> ex=vcs.getfillarea()  # instance of 'default' fillarea␣
↪secondary method
>>> a.fillarea(ex) # plot using specified fillarea object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – String name of an existing fillarea VCS object
- **style** (*str*) – One of "hatch", "solid", or "pattern".
- **index** (*int*) – Specifies which pattern to fill with. Accepts ints from 1-20.
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the texttable will be drawn.

- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A fillarea secondary object

Return type *vcs.fillarea.Tf*

vcs.manageElements.**getisofill**(*Gfi_name_src='default'*)

VCS contains a list of graphics methods. This function will create a isofill class object from an existing VCS isofill graphics method. If no isofill name is given, then isofill 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createisofill()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('isofill') # Show all the existing isofill␣
↪graphics methods
[...]
>>> ex=vcs.getisofill()  # instance of 'default' isofill graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.isofill(ex, slab1) # plot using specified isofill object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getisofill('polar')  # instance of 'polar' isofill␣
↪graphics method
>>> a.isofill(ex2, slab1) # plot using specified isofill object
<vcs.displayplot.Dp ...>
```

Parameters

- **Gfi_name_src** (*str*) – String name of an existing isofill VCS object
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D

- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode

- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns The specified isofill VCS object
Return type *vcs.isofill.Gfi*

vcs.manageElements.**getisoline**(*Gi_name_src='default'*)

VCS contains a list of graphics methods. This function will create a isoline class object from an existing VCS isoline graphics method. If no isoline name is given, then isoline 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createisoline()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('isoline') # Show all the existing isoline␣
↪graphics methods
[...]
>>> ex=vcs.getisoline()  # instance of 'default' isoline graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.isoline(ex, slab1) # plot using specified isoline object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getisoline('polar')  # instance of 'polar' isoline␣
↪graphics method
>>> a.isoline(ex2, slab1) # plot using specified isoline object
<vcs.displayplot.Dp ...>
```

Parameters

- **Gi_name_src** (*str*) – String name of an existing isoline VCS object
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D

- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode

- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns The requested isoline VCS object

Return type *vcs.isoline.Gi*

vcs.manageElements.**getline**(*name='default'*, *ltype=None*, *width=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a line class object from an existing VCS line secondary method. If no line name is given, then line 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createline()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('line') # Show all the existing line secondary
↪methods
[...]
>>> ex=vcs.getline()  # instance of 'default' line secondary method
>>> a.line(ex) # plot using specified line object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getline('red')  # instance of 'red' line secondary
↪method
>>> a.line(ex2) # plot using specified line object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object
- **ltype** (*str*) – One of "dash", "dash-dot", "solid", "dot", or "long-dash".
- **width** (*int*) – Thickness of the line to be created
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].

- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A VCS line object
Return type *vcs.line.Tl*

vcs.manageElements.**getmarker**(*name='default'*, *mtype=None*, *size=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a marker class object from an existing VCS marker secondary method. If no marker name is given, then marker 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createmarker()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('marker') # Show all the existing marker␣
↪secondary methods
[...]
>>> ex=vcs.getmarker()  # instance of 'default' marker secondary␣
↪method
>>> a.marker(ex) # plot using specified marker object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getmarker('red')  # instance of 'red' marker secondary␣
↪method
>>> a.marker(ex2) # plot using specified marker object
<vcs.displayplot.Dp ...>
```

Parameters

- **name** (*str*) – Name of created object
- **source** (*str*) – A marker, or string name of a marker
- **mtype** (*str*) – Specifies the type of marker, i.e. "dot", "circle"
- **size** (*int*) – Size of the marker
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the marker will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A marker graphics method object
Return type *vcs.marker.Tm*

vcs.manageElements.**getmeshfill**(*Gfm_name_src='default'*)

VCS contains a list of graphics methods. This function will create a meshfill class object from an existing VCS meshfill graphics method. If no meshfill name is given, then meshfill 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createmeshfill()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('meshfill') # Show all the existing meshfill␣
↪graphics methods
[...]
>>> ex=vcs.getmeshfill()  # instance of 'default' meshfill graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.meshfill(ex, slab1) # plot using specified meshfill object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getmeshfill('a_polar_meshfill')  # instance of 'a_polar_
↪meshfill' meshfill graphics method
>>> a.meshfill(ex2, slab1) # plot using specified meshfill object
<vcs.displayplot.Dp ...>
```

Parameters **Gfm_name_src** (*str*) – String name of an existing meshfill VCS object
Returns A meshfill VCS object
Return type *vcs.meshfill.Gfm*

vcs.manageElements.**getprojection**(*Proj_name_src='default'*)

VCS contains a list of graphics methods. This function will create a projection class object from an existing VCS projection graphics method. If no projection name is given, then projection 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createprojection()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('projection') # Show all the existing␣
↪projection graphics methods
[...]
>>> ex=vcs.getprojection()  # instance of 'default' projection␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.plot(ex, slab1) # plot using specified projection object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.getprojection('polar')  # instance of 'polar'␣
↪projection graphics method
>>> a.plot(ex2, slab1) # plot using specified projection object
```

```
<vcs.displayplot.Dp ...>
```

Parameters **Proj_name_src** (*str*) – String name of an existing VCS projection object
Returns A VCS projection object
Return type *vcs.projection.Proj*

vcs.manageElements.**getscatter**(*GSp_name_src='default'*)
VCS contains a list of graphics methods. This function will create a scatter class object from an existing VCS scatter graphics method. If no scatter name is given, then scatter ''**default_scatter_**'' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createscatter()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('scatter') # Show all the existing scatter␣
↪graphics methods
[...]
>>> ex=vcs.getscatter('default_scatter_')  # instance of ''default_
↪scatter_'' scatter graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.scatter(ex, slab1, slab2) # plot using specified scatter␣
↪object
<vcs.displayplot.Dp ...>
```

Parameters

- **GSp_name_src** (*str*) – String name of an existing scatter VCS object.
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D

- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A scatter graphics method object

Return type *vcs.unified1D.G1d*

vcs.manageElements.**gettaylordiagram**(*Gtd_name_src='default'*)

VCS contains a list of graphics methods. This function will create a taylordiagram class object from an existing VCS taylordiagram graphics method. If no taylordiagram name is given, then taylordiagram 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtaylordiagram()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('taylordiagram') # Show all the existing␣
↪taylordiagram graphics methods
[...]
>>> ex=vcs.gettaylordiagram()  # instance of 'default'␣
↪taylordiagram graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.taylordiagram(ex, slab1) # plot using specified taylordiagram␣
↪object
<vcs.displayplot.Dp ...>
```

Parameters **Gtd_name_src** (*str*) – String name of an existing taylordiagram VCS object
Returns A taylordiagram VCS object
Return type *vcs.taylor.Gtd*

vcs.manageElements.**gettemplate**(*Pt_name_src='default'*)

VCS contains a list of graphics methods. This function will create a template class object from an existing VCS template graphics method. If no template name is given, then template 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtemplate()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('template') # Show all the existing template␣
↪graphics methods
[...]
>>> ex=vcs.gettemplate()  # instance of 'default' template graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.plot(ex, slab1) # plot using specified template object
<vcs.displayplot.Dp ...>
>>> ex2=vcs.gettemplate('polar')  # instance of 'polar' template␣
↪graphics method
>>> a.plot(ex2, slab1) # plot using specified template object
```

```
<vcs.displayplot.Dp ...>
```

Parameters **Pt_name_src** – String name of an existing template VCS object
Returns A VCS template object
Return type *vcs.template.P*

vcs.manageElements.**gettext**(*Tt_name_src='default'*, *To_name_src=None*, *string=None*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*)

VCS contains a list of secondary methods. This function will create a textcombined class object from an existing VCS textcombined secondary method. If no textcombined name is given, then textcombined 'EXAMPLE_tt:::EXAMPLE_tto' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtextcombined()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('textcombined') # Show all the existing␣
↪textcombined secondary methods
[...]
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> ex=vcs.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto')  # instance␣
↪of 'EXAMPLE_tt:::EXAMPLE_tto' textcombined secondary method
>>> a.textcombined(ex) # plot using specified textcombined object
<vcs.displayplot.Dp ...>
```

Parameters

- **Tt_name_src** (*str*) – Name of created object
- **To_name_src** (*str*) – Name of parent textorientation object
- **string** – Text to render
- **string** – list of str
- **font** (*int or str*) – Which font to use (index or name)
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].

- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*list of int*) – Angle of the rendered text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of ["left", "center", "right"]
- **valign** (*str*) – Vertical alignment of the text. One of ["top", "center", "bottom"]

Returns A textcombined object

Return type *vcs.textcombined.Tc*

vcs.manageElements.**gettextcombined**(*Tt_name_src='default'*, *To_name_src=None*, *string=None*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*, *height=None*, *angle=None*, *path=None*, *halign=None*, *valign=None*)

VCS contains a list of secondary methods. This function will create a textcombined class object from an existing VCS textcombined secondary method. If no textcombined name is given, then textcombined 'EXAMPLE_tt:::EXAMPLE_tto' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtextcombined()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('textcombined') # Show all the existing␣
↪textcombined secondary methods
[...]
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> ex=vcs.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto')  # instance␣
↪of 'EXAMPLE_tt:::EXAMPLE_tto' textcombined secondary method
>>> a.textcombined(ex) # plot using specified textcombined object
<vcs.displayplot.Dp ...>
```

Parameters

- **Tt_name_src** (*str*) – Name of created object
- **To_name_src** (*str*) – Name of parent textorientation object
- **string** – Text to render
- **string** – list of str
- **font** (*int or str*) – Which font to use (index or name)
- **spacing** (*DEPRECATED*) – DEPRECATED
- **expansion** (*DEPRECATED*) – DEPRECATED
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the object will be drawn.

- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].
- **height** (*int*) – Size of the font
- **angle** (*list of int*) – Angle of the rendered text, in degrees
- **path** (*DEPRECATED*) – DEPRECATED
- **halign** (*str*) – Horizontal alignment of the text. One of ["left", "center", "right"]
- **valign** (*str*) – Vertical alignment of the text. One of ["top", "center", "bottom"]

Returns A textcombined object
Return type *vcs.textcombined.Tc*

vcs.manageElements.**gettextorientation**(*To_name_src='default'*)

VCS contains a list of secondary methods. This function will create a textorientation class object from an existing VCS textorientation secondary method. If no textorientation name is given, then textorientation 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtextorientation()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('textorientation') # Show all the existing␣
↪textorientation secondary methods
[...]
>>> ex=vcs.gettextorientation()  # instance of 'default'␣
↪textorientation secondary method
>>> ex2=vcs.gettextorientation('bigger')  # instance of 'bigger'␣
↪textorientation secondary method
```

Parameters **To_name_src** (*str*) – String name of an existing textorientation VCS object
Returns A textorientation VCS object
Return type *vcs.textorientation.To*

vcs.manageElements.**gettexttable**(*name='default'*, *font=None*, *spacing=None*, *expansion=None*, *color=None*, *priority=None*, *viewport=None*, *worldcoordinate=None*, *x=None*, *y=None*)

VCS contains a list of secondary methods. This function will create a texttable class object from an existing VCS texttable secondary method. If no texttable name is given, then texttable 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createtexttable()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('texttable') # Show all the existing texttable␣
↪secondary methods
[...]
>>> ex=vcs.gettexttable()  # instance of 'default' texttable␣
↪secondary method
>>> ex2=vcs.gettexttable('bigger')  # instance of 'bigger'␣
↪texttable secondary method
```

Parameters

- **name** (*str*) – String name of an existing VCS texttable object
- **font** – ???
- **expansion** – ???
- **color** (*str or int*) – A color name from the X11 Color Names list, or an integer value from 0-255, or an RGB/RGBA tuple/list (e.g. (0,100,0), (100,100,0,50))
- **priority** (*int*) – The layer on which the texttable will be drawn.
- **viewport** (*list of floats*) – 4 floats between 0 and 1. These specify the area that the X/Y values are mapped to inside of the canvas
- **worldcoordinate** (*list of floats*) – List of 4 floats (xmin, xmax, ymin, ymax)
- **x** (*list of floats*) – List of lists of x coordinates. Values must be between worldcoordinate[0] and worldcoordinate[1].
- **y** (*list of floats*) – List of lists of y coordinates. Values must be between worldcoordinate[2] and worldcoordinate[3].

Returns A texttable graphics method object

Return type *vcs.texttable.Tt*

vcs.manageElements.**getvector**(*Gv_name_src='default'*)

VCS contains a list of graphics methods. This function will create a vector class object from an existing VCS vector graphics method. If no vector name is given, then vector 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createvector()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('vector') # Show all the existing vector␣
↪graphics methods
[...]
>>> ex=vcs.getvector()  # instance of 'default' vector graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
```

```
>>> a.vector(ex, slab1, slab2) # plot using specified vector object
<vcs.displayplot.Dp ...>
```

Parameters **Gv_name_src** (*str*) – String name of an existing vector VCS object
Returns A vector graphics method object
Return type *vcs.vector.Gv*

vcs.manageElements.**getxvsy**(*GXY_name_src='default'*)
VCS contains a list of graphics methods. This function will create a xvsy class object from an existing VCS xvsy graphics method. If no xvsy name is given, then xvsy '**default_xvsy_**' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createxvsy()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('xvsy') # Show all the existing xvsy graphics␣
↪methods
[...]
>>> ex=vcs.getxvsy()  # instance of 'default_xvsy_' xvsy graphics␣
↪method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> slab2 = f('v') # need 2 slabs, so get another
>>> a.xvsy(ex, slab1, slab2) # plot using specified xvsy object
<vcs.displayplot.Dp ...>
```

Parameters

- **GXY_name_src** (*str*) – String name of a 1d graphics method
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D

- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A XvsY graphics method object

Return type *vcs.unified1D.G1d*

vcs.manageElements.**getxyvsy**(*GXy_name_src='default'*)

VCS contains a list of graphics methods. This function will create a xyvsy class object from an existing VCS xyvsy graphics method. If no xyvsy name is given, then xyvsy ''**default_xyvsy_**'' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createxyvsy()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('xyvsy') # Show all the existing xyvsy␣
↪graphics methods
[...]
>>> ex=vcs.getxyvsy('default_xyvsy_')  # instance of ''default_
↪xyvsy_'' xyvsy graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.xyvsy(ex, slab1) # plot using specified xyvsy object
<vcs.displayplot.Dp ...>
```

Parameters

- **GXy_name_src** (*str*) – String name of an existing Xyvsy graphics method
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot

- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot
- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns An XYvsY graphics method object
Return type *vcs.unified1D.G1d*

vcs.manageElements.**getyxvsx**(*GYx_name_src='default'*)
VCS contains a list of graphics methods. This function will create a yxvsx class object from an existing VCS yxvsx graphics method. If no yxvsx name is given, then yxvsx '**default_yxvsx_**' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the *vcs.manageElements.createyxvsx()* function.)

Example

```
>>> a=vcs.init()
>>> vcs.listelements('yxvsx') # Show all the existing yxvsx␣
↪graphics methods
[...]
>>> ex=vcs.getyxvsx()  # instance of 'default_yxvsx_' yxvsx␣
↪graphics method
>>> import cdms2 # Need cdms2 to create a slab
>>> f = cdms2.open(vcs.sample_data+'/clt.nc') # use cdms2 to open a␣
↪data file
>>> slab1 = f('u') # use the data file to create a cdms2 slab
>>> a.yxvsx(ex, slab1) # plot using specified yxvsx object
<vcs.displayplot.Dp ...>
```

Parameters

- **GYx_name_src** (*str*) – String name of an existing Yxvsx graphics method
- **xaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -1 dim axis
- **yaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -2 dim axis, only if slab has more than 1D
- **zaxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -3 dim axis, only if slab has more than 2D
- **taxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -4 dim axis, only if slab has more than 3D
- **waxis** (*cdms2.axis.TransientAxis*) – Axis object to replace the slab -5 dim axis, only if slab has more than 4D
- **xrev** (*bool*) – reverse x axis
- **yrev** (*bool*) – reverse y axis, only if slab has more than 1D
- **xarray** (*array*) – Values to use instead of x axis
- **yarray** (*array*) – Values to use instead of y axis, only if var has more than 1D
- **zarray** (*array*) – Values to use instead of z axis, only if var has more than 2D
- **tarray** (*array*) – Values to use instead of t axis, only if var has more than 3D
- **warray** (*array*) – Values to use instead of w axis, only if var has more than 4D
- **continents** (*int*) – continents type number
- **name** (*str*) – replaces variable name on plot
- **time** (*A cdtime object*) – replaces time name on plot
- **units** (*str*) – replaces units value on plot
- **ymd** (*str*) – replaces year/month/day on plot
- **hms** (*str*) – replaces hh/mm/ss on plot

- **file_comment** (*str*) – replaces file_comment on plot
- **xbounds** (*array*) – Values to use instead of x axis bounds values
- **ybounds** (*array*) – Values to use instead of y axis bounds values (if exist)
- **xname** (*str*) – replace xaxis name on plot
- **yname** (*str*) – replace yaxis name on plot (if exists)
- **zname** (*str*) – replace zaxis name on plot (if exists)
- **tname** (*str*) – replace taxis name on plot (if exists)
- **wname** (*str*) – replace waxis name on plot (if exists)
- **xunits** (*str*) – replace xaxis units on plot
- **yunits** (*str*) – replace yaxis units on plot (if exists)
- **zunits** (*str*) – replace zaxis units on plot (if exists)
- **tunits** (*str*) – replace taxis units on plot (if exists)
- **wunits** (*str*) – replace waxis units on plot (if exists)
- **xweights** (*array*) – replace xaxis weights used for computing mean
- **yweights** (*array*) – replace xaxis weights used for computing mean
- **comment1** (*str*) – replaces comment1 on plot
- **comment2** (*str*) – replaces comment2 on plot
- **comment3** (*str*) – replaces comment3 on plot
- **comment4** (*str*) – replaces comment4 on plot
- **long_name** (*str*) – replaces long_name on plot
- **grid** (*cdms2.grid.TransientRectGrid*) – replaces array grid (if exists)
- **bg** (*bool/int*) – plots in background mode
- **ratio** () – sets the y/x ratio ,if passed as a string with 't' at the end, will aslo moves the ticks
- **xaxisconvert** (*str*) – (Ex: 'linear') converting xaxis linear/log/log10/ln/exp/area_wt
- **yaxisconvert** (*str*) – (Ex: 'linear') converting yaxis linear/log/log10/ln/exp/area_wt
- **GM_name** – (Ex: 'default') retrieve the graphics method object of the given name. If no name is given, then retrieve the 'default' graphics method.

Returns A Yxvsx graphics method object
Return type *vcs.unified1D.G1d*

vcs.manageElements.**removeobject**(*obj*)

The user has the ability to create primary and secondary class objects. The function allows the user to remove these objects from the appropriate class list.

Note, To remove the object completely from Python, remember to use the "del" function.

Also note, The user is not allowed to remove a "default" class object.

Example

```
>>> a=vcs.init()
>>> line=a.getline('red') # To Modify an existing line object
>>> iso=a.createisoline('dean') # Create an instance of an isoline␣
↪object
>>> a.removeobject(line) # Removes line object from VCS list
'Removed line object red'
>>> a.removeobject(iso) # Remove isoline object from VCS list
'Removed isoline object dean'
```

Parameters **obj** (*VCS object*) – Any VCS primary or secondary object
Returns String indicating the specified object was removed
Return type str

1.6.8 queries

vcs.queries.**graphicsmethodlist**()
List available graphics methods.
Example

```
>>> a=vcs.init()
>>> vcs.graphicsmethodlist() # Return graphics method list
[...]
```

Returns A list of available grapics methods (i.e., boxfill, isofill, isoline, outfill, scatter, vector, xvsy, xyvsy, yxvsx, taylordiagram).
Return type *list*

vcs.queries.**graphicsmethodtype**(*gobj*)
Check the type of a graphics object.

Returns None if the object is not a graphics method.
Example

```
>>> a=vcs.init()
>>> box=a.getboxfill() # Get default boxfill graphics method
>>> iso=a.getisofill() # Get default isofill graphics method
>>> ln=a.getline()# Get default line element
>>> vcs.graphicsmethodtype(box)
'boxfill'
>>> vcs.graphicsmethodtype(iso)
'isofill'
>>> vcs.graphicsmethodtype(ln)
Traceback (most recent call last):
...
vcsError: The object passed is not a graphics method object.
```

returns If gobj is a graphics method object, returns its type: 'boxfill', 'isofill', 'isoline', 'scatter', 'vector', 'xvsy', 'xyvsy', or 'yxvsx', 'taylordiagram'. If gobj is not a graphics method object, raises an exception and prints a vcsError message.

rtype str or None

vcs.queries.**is1d**(*obj*)
Check to see if this object is a VCS 1d graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('1d') # Show all available 1d
********************1d Names List**********************
...
********************End 1d Names List**********************
>>> ex = a.get1d('default') # To  test an existing 1d object
>>> vcs.queries.is1d(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a 1d graphics method (1), or not (0).
Return type int

vcs.queries.**is3d_dual_scalar**(*obj*)
Check to see if this object is a VCS 3d_dual_scalar graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('3d_dual_scalar') # Show all available 3d_dual_scalar
********************3d_dual_scalar Names List**********************
...
********************End 3d_dual_scalar Names␣
↪List**********************
>>> ex = a.get3d_dual_scalar() # To  test an existing 3d_dual_
↪scalar object
>>> vcs.queries.is3d_dual_scalar(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a 3d_dual_scalar graphics method (1), or not (0).
Return type int

vcs.queries.**is3d_scalar**(*obj*)
Check to see if this object is a VCS 3d_scalar graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('3d_scalar') # Show all available 3d_scalar
********************3d_scalar Names List**********************
...
********************End 3d_scalar Names List**********************
>>> ex = a.get3d_scalar() # To  test an existing 3d_scalar object
>>> vcs.queries.is3d_scalar(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a 3d_scalar graphics method (1), or not (0).
Return type int

vcs.queries.**is3d_vector**(*obj*)
Check to see if this object is a VCS 3d_vector graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('3d_vector') # Show all available 3d_vector
********************3d_vector Names List**********************
...
********************End 3d_vector Names List**********************
```

```
>>> ex = a.get3d_vector() # To  test an existing 3d_vector object
>>> vcs.queries.is3d_vector(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a 3d_vector graphics method (1), or not (0).
Return type int

vcs.queries.**isboxfill**(*obj*)
Check to see if this object is a VCS boxfill graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('boxfill') # Show all available boxfill
******************Boxfill Names List**********************
...
******************End Boxfill Names List**********************
>>> ex = a.getboxfill() # To  test an existing boxfill object
>>> vcs.queries.isboxfill(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a boxfill graphics method (1), or not (0).
Return type int

vcs.queries.**iscolormap**(*obj*)
Check to see if this object is a VCS colormap secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('colormap') # Show all available colormap
******************Colormap Names List**********************
...
******************End Colormap Names List**********************
>>> ex = a.getcolormap() # To  test an existing colormap object
>>> vcs.queries.iscolormap(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a colormap secondary method (1), or not (0).
Return type int

vcs.queries.**isfillarea**(*obj*)
Check to see if this object is a VCS fillarea secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('fillarea') # Show all available fillarea
******************Fillarea Names List**********************
...
******************End Fillarea Names List**********************
>>> ex = a.getfillarea() # To  test an existing fillarea object
>>> vcs.queries.isfillarea(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a fillarea secondary method (1), or not (0).
Return type int

vcs.queries.**isgraphicsmethod**(*gobj*)

Indicates if the entered argument is one of the following graphics methods: boxfill, isofill, isoline, scatter, vector, xvsy, xyvsy, yxvsx.

Example

```
>>> a=vcs.init()
>>> box=a.getboxfill() # get default boxfill object
>>> vcs.isgraphicsmethod(box)
1
```

Parameters **gobj** (*A VCS graphics object*) – A graphics object

Returns Integer reperesenting whether gobj is one of the above graphics methods. 1 indicates true, 0 indicates false.

Return type int

vcs.queries.**isisofill**(*obj*)

Check to see if this object is a VCS isofill graphics method.

Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('isofill') # Show all available isofill
********************Isofill Names List**********************
...
********************End Isofill Names List**********************
>>> ex = a.getisofill() # To  test an existing isofill object
>>> vcs.queries.isisofill(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object

Returns An integer indicating whether the object is a isofill graphics method (1), or not (0).

Return type int

vcs.queries.**isisoline**(*obj*)

Check to see if this object is a VCS isoline graphics method.

Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('isoline') # Show all available isoline
********************Isoline Names List**********************
...
********************End Isoline Names List**********************
>>> ex = a.getisoline() # To  test an existing isoline object
>>> vcs.queries.isisoline(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object

Returns An integer indicating whether the object is a isoline graphics method (1), or not (0).

Return type int

vcs.queries.**isline**(*obj*)

Check to see if this object is a VCS line secondary method.

Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('line') # Show all available line
********************Line Names List**********************
...
********************End Line Names List**********************
>>> ex = a.getline() # To  test an existing line object
>>> vcs.queries.isline(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a line secondary method (1), or not (0).
Return type int

vcs.queries.**ismarker**(*obj*)
Check to see if this object is a VCS marker secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('marker') # Show all available marker
********************Marker Names List**********************
...
********************End Marker Names List**********************
>>> ex = a.getmarker() # To  test an existing marker object
>>> vcs.queries.ismarker(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a marker secondary method (1), or not (0).
Return type int

vcs.queries.**ismeshfill**(*obj*)
Check to see if this object is a VCS meshfill graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('meshfill') # Show all available meshfill
********************Meshfill Names List**********************
...
********************End Meshfill Names List**********************
>>> ex = a.getmeshfill() # To  test an existing meshfill object
>>> vcs.queries.ismeshfill(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a meshfill graphics method (1), or not (0).
Return type int

vcs.queries.**isplot**(*pobj*)
Check to see if this object is a VCS secondary display plot.
Example

```
>>> a=vcs.init()
>>> import cdms2 # need this to make a slab for a boxfill plot
>>> f = cdms2.open(vcs.sample_data + '/clt.nc') # open a variable␣
↪file
>>> v = f('v') # create a slab from the variable file
>>> dsp_plot=(a.getboxfill(), v) # plot a boxfill. Should return␣
↪vcs.displayplot.Dp.
>>> vcs.queries.isplot(dsp_plot)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a display plot (1), or not (0).
Return type int

vcs.queries.**isprojection**(*obj*)
Check to see if this object is a VCS projection graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('projection') # Show all available projection
********************Projection Names List**********************
...
********************End Projection Names List**********************
>>> ex = a.getprojection() # To  test an existing projection object
>>> vcs.queries.isprojection(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a projection graphics method (1), or not (0).
Return type int

vcs.queries.**isscatter**(*obj*)
Check to see if this object is a VCS scatter graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('scatter') # Show all available scatter
********************Scatter Names List**********************
...
********************End Scatter Names List**********************
>>> ex = a.getscatter('default_scatter_') # To  test an existing␣
↪scatter object
>>> vcs.queries.isscatter(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a scatter graphics method (1), or not (0).
Return type int

vcs.queries.**issecondaryobject**(*sobj*)
Check to see if this object is a VCS secondary object

Note: Secondary objects will be one of the following: 1.) colormap: specification of combinations of 256 available
colors
2.) fill area: style, style index, and color index 3.) format: specifications for converting numbers to display
strings
4.) line: line type, width, and color index 5.) list: a sequence of pairs of numerical and character values 6.) marker: marker type, size, and color index 7.) text table: text font type, character spacing, expansion, and
color index

8.) text orientation: character height, angle, path, and horizontal/vertical alignment

9.) projections

Example

```
>>> a=vcs.init()
>>> a.show('line') # Show all available lines
********************Line Names List**********************
...
********************End Line Names List**********************
>>> ex = a.getprojection('default') # To test an existing line␣
↪object
```

```
>>> vcs.issecondaryobject(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a projection graphics object (1), or not (0).
Return type int

vcs.queries.**istaylordiagram**(*obj*)
Check to see if this object is a VCS taylordiagram graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('taylordiagram') # Show all available taylordiagram
********************Taylordiagram Names List**********************
...
********************End Taylordiagram Names␣
↪List**********************
>>> ex = a.gettaylordiagram() # To  test an existing taylordiagram␣
↪object
>>> vcs.queries.istaylordiagram(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a taylordiagram graphics method (1), or not (0).
Return type int

vcs.queries.**istemplate**(*gobj*)
Check to see if this object is a VCS template graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('template') # Show all available template
********************Template Names List**********************
...
********************End Template Names List**********************
>>> ex = a.gettemplate() # To  test an existing template object
>>> vcs.queries.istemplate(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a template graphics method (1), or not (0).
Return type int

vcs.queries.**istext**(*obj*)
Check to see if this object is a VCS textcombined secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> a.show('textcombined') # Show all available textcombined
********************Textcombined Names List**********************
...
********************End Textcombined Names List**********************
>>> ex = a.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto') # To  test␣
↪an existing textcombined object
>>> vcs.queries.istextcombined(ex)
```

```
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a textcombined secondary method (1), or not (0).
Return type int

vcs.queries.**istextcombined**(*obj*)
Check to see if this object is a VCS textcombined secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.createtextcombined('EXAMPLE_tt', 'qa', 'EXAMPLE_tto', '7left
↪') # Create 'EXAMPLE_tt' and 'EXAMPLE_tto'
<vcs.textcombined.Tc ...>
>>> a.show('textcombined') # Show all available textcombined
********************Textcombined Names List**********************
...
********************End Textcombined Names List**********************
>>> ex = a.gettextcombined('EXAMPLE_tt', 'EXAMPLE_tto') # To  test␣
↪an existing textcombined object
>>> vcs.queries.istextcombined(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a textcombined secondary method (1), or not (0).
Return type int

vcs.queries.**istextorientation**(*obj*)
Check to see if this object is a VCS textorientation secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('textorientation') # Show all available textorientation
********************Textorientation Names List**********************
...
********************End Textorientation Names␣
↪List**********************
>>> ex = a.gettextorientation() # To  test an existing␣
↪textorientation object
>>> vcs.queries.istextorientation(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a textorientation secondary method (1), or not (0).
Return type int

vcs.queries.**istexttable**(*obj*)
Check to see if this object is a VCS texttable secondary method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('texttable') # Show all available texttable
********************Texttable Names List**********************
...
********************End Texttable Names List**********************
>>> ex = a.gettexttable() # To  test an existing texttable object
```

```
>>> vcs.queries.istexttable(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a texttable secondary method (1), or not (0).
Return type int

vcs.queries.**isvector**(*obj*)
Check to see if this object is a VCS 1d graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('1d') # Show all available 1d
********************1d Names List**********************
...
********************End 1d Names List**********************
>>> ex = a.get1d('default') # To  test an existing 1d object
>>> vcs.queries.is1d(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a 1d graphics method (1), or not (0).
Return type int

vcs.queries.**isxvsy**(*obj*)
Check to see if this object is a VCS xvsy graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('xvsy') # Show all available xvsy
********************Xvsy Names List**********************
...
********************End Xvsy Names List**********************
>>> ex = a.getxvsy() # To  test an existing xvsy object
>>> vcs.queries.isxvsy(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a xvsy graphics method (1), or not (0).
Return type int

vcs.queries.**isxyvsy**(*obj*)
Check to see if this object is a VCS xyvsy graphics method.
Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('xyvsy') # Show all available xyvsy
********************Xyvsy Names List**********************
...
********************End Xyvsy Names List**********************
>>> ex = a.getxyvsy('default_xyvsy_') # To  test an existing xyvsy␣
↪object
>>> vcs.queries.isxyvsy(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object
Returns An integer indicating whether the object is a xyvsy graphics method (1), or not (0).
Return type int

vcs.queries.**isyxvsx**(*obj*)

Check to see if this object is a VCS yxvsx graphics method.

Example

```
>>> a=vcs.init() # Make a VCS Canvas object to work with:
>>> a.show('yxvsx') # Show all available yxvsx
********************Yxvsx Names List**********************
...
********************End Yxvsx Names List**********************
>>> ex = a.getyxvsx() # To  test an existing yxvsx object
>>> vcs.queries.isyxvsx(ex)
1
```

Parameters **obj** (*VCS Object*) – A VCS object

Returns An integer indicating whether the object is a yxvsx graphics method (1), or not (0).

Return type int

1.6.9 utils

vcs.utils.**generate_time_labels**(*d1*, *d2*, *units*, *calendar=135441*)

Generates a dictionary of time labels for an interval of time, in a user defined units system.

Example

```
# Two ways to generate a dictionary of time labels
>>> lbls = generate_time_labels(cdtime.reltime(0,'months since 2000
↪'),
...     cdtime.reltime(12,'months since 2000'),
...     'days since 1800',) # for the year 2000 in units of 'days␣
↪since 1800'
>>> lbls = generate_time_labels(cdtime.reltime(0,'months since 2000
↪'),
...     cdtime.comptime(2001),
...     'days since 1800',) # for the year 2000 in units of 'days␣
↪since 1800'
>>> lbls = generate_time_labels(0, 12, 'months since 2000', ) #␣
↪Generate a dictionary of time labels
                                                              # for␣
↪year 2000, units of 'months since 2000'
```

Parameters

- **d1** (*cdtime object, int, long, float*) – The beginning of the time interval to be labelled. Expects a cdtime object. Can also take int, long, or float, which will be used to create a cdtime object with the given units parameter.
- **d2** (*cdtime object, int, long, float*) – The end of the time interval to be labelled. Expects a cdtime object. Can also take int, long, or float, which will be used to create a cdtime object with the given units parameter.
- **units** (*str*) – String with the format '[time_unit] since [date]'.
- **calendar** – A cdtime calendar,

Returns Dictionary of time labels over the given time interval

Return type dict

vcs.utils.**getcolorcell**(*cell*, *obj=None*)

Gets the colorcell of the provided object's colormap at the specified cell index. If no object is provided, or if the provided object has no colormap, the default colormap is used.

Example

```
>>> a=vcs.init()
>>> b=vcs.createboxfill()
>>> b.colormap='rainbow'
>>> a.getcolorcell(2,b)
[85, 85, 85, 100.0]
```

Parameters

- **cell** (*int*) – An integer value indicating the index of the desired colorcell.
- **obj** (*Any VCS object capable of containing a colormap*) – Optional parameter containing the object to extract a colormap from.

Returns The RGBA values of the colormap at the specified cell index.

Return type *list*

vcs.utils.**getcolormap**(*Cp_name_src='default'*)

VCS contains a list of secondary methods. This function will create a colormap class object from an existing VCS colormap secondary method. If no colormap name is given, then colormap 'default' will be used.

Note: VCS does not allow the modification of 'default' attribute sets. However, a 'default' attribute set that has been copied under a different name can be modified. (See the createcolormap function.)

Example

```
>>> a=vcs.init()
>>> a.show('colormap') # Show all the existing colormap secondary␣
↪methods
********************Colormap Names List**********************
...
********************End Colormap Names List**********************
>>> cp=a.getcolormap() # cp instance of 'default' colormap␣
↪secondary method
>>> cp2=a.getcolormap('rainbow') # cp2 instance of existing 'rainbow
↪' colormap secondary method
```

Parameters **Cp_name_src** (*str*) – String name of an existing colormap VCS object

Returns A pre-existing VCS colormap object

Return type *vcs.colormap.Cp*

vcs.utils.**getcolors**(*levs, colors=[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239], split=1, white=240*)

For isofill/boxfill purposes Given a list of levels this function returns the colors that would best spread a list of "user-defined" colors (default is 16 to 239, i.e 224 colors), always using the first and last color. Optionally the color range can be split into 2 equal domain to represent <0 and >0 values. If the colors are split an interval goes from <0 to >0 then this is assigned the "white" color

Example

```
>>> a=[0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]
>>> vcs.getcolors (a)
[16, 41, 66, 90, 115, 140, 165, 189, 214, 239]
>>> vcs.getcolors (a,colors=range(16,200))
[16, 36, 57, 77, 97, 118, 138, 158, 179, 199]
>>> vcs.getcolors(a,colors=[16,25,15,56,35,234,12,11,19,32,132,17])
[16, 25, 15, 35, 234, 12, 11, 32, 132, 17]
>>> a=[-6.0, -2.0, 2.0, 6.0, 10.0, 14.0, 18.0, 22.0, 26.0]
>>> vcs.getcolors (a,white=241)
[72, 241, 128, 150, 172, 195, 217, 239]
>>> vcs.getcolors (a,white=241,split=0)
[16, 48, 80, 112, 143, 175, 207, 239]
```

Parameters

- **levs** (*list, tuple*) – levels defining the color ranges
- **colors** (*list*) – A list/tuple of the of colors you wish to use
- **split** (*int*) – Integer flag to split colors between two equal domains. 0 : no split 1 : split if the levels go from <0 to >0 2 : split even if all the values are positive or negative
- **white** (*int*) – If split is on and an interval goes from <0 to >0 this color number will be used within this interval (240 is white in the default VCS palette color). Integer must be between 0 and 255.

Returns List of colors

Return type *list*

vcs.utils.**getfontname**(*number*)

Retrieve a font name for a given font index.

Parameters **number** (*int*) – Index of the font to get the name of.

vcs.utils.**getfontnumber**(*name*)

Retrieve a font index for a given font name.

Parameters **name** (*str*) – Name of the font to get the index of.

vcs.utils.**getworldcoordinates**(*gm*, *X*, *Y*)

Given a graphics method and two axes figures out correct world coordinates.

Parameters

- **gm** (*graphics method object*) – A VCS graphics method object to get world-coordinates for.
- **X** (*cdms2 transient axis*) – A cdms2 transient axs
- **Y** (*cdms2 transient axis*) – A cdms2 transient axs

Returns

Return type

vcs.utils.**match_color**(*color*, *colormap=None*)

Returns the color in the colormap that is closest to the required color.

Example

```
>>> a=vcs.init()
>>> print vcs.match_color('salmon')
>>> print vcs.match_color('red')
>>> print vcs.match_color([0,0,100],'default') # closest color from␣
↪blue
```

Parameters

- **color** (*str, int*) – Either a string name, or a rgb value between 0 and 100.

- **colormap** (vcs.colormap.Cp) – A VCS colormap object. If not specified, the default colormap is used.

Returns Integer value representing a matching rgb color
Return type int

vcs.utils.**minmax**(**data*)
Return the minimum and maximum of a series of array/list/tuples (or combination of these) You can combine list/tuples/arrays pretty much any combination is allowed
Example

```
>>> s=range(7)
>>> vcs.minmax(s)
(0.0, 6.0)
>>> vcs.minmax([s,s])
(0.0, 6.0)
>>> vcs.minmax([[s,s*2],4.,[6.,7.,s]],[5.,-7.,8,(6.,1.)])
(-7.0, 8.0)
```

Parameters **data** (list) – A comma-separated list of lists/arrays/tuples
Returns A tuple in the form (min, max)
Return type tuple

vcs.utils.**mkevenlevels**(*n1*, *n2*, *nlev=10*)
Return a series of evenly spaced levels going from n1 to n2. By default 10 intervals will be produced.
Example

```
>>> vcs.mkevenlevels(0,100)
[0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
>>> vcs.mkevenlevels(0,100,nlev=5)
[0.0, 20.0, 40.0, 60.0, 80.0, 100.0]
>>> vcs.mkevenlevels(100,0,nlev=5)
[100.0, 80.0, 60.0, 40.0, 20.0, 0.0]
```

Parameters

- **n1** (*int, float*) – Beginning of range. Int or float.
- **n2** (*int, float*) – End of range. Int or float.
- **nlev** (*int*) – Number of levels by which to split the given range.

Returns List of floats, splitting range evenly between n1 and n2
Return type *list*

vcs.utils.**mklabels**(*vals*, *output='dict'*)
This function gets levels and output strings for nice display of the levels values.
Examples

```
>>> a=vcs.mkscale(2,20,zero=2)
>>> vcs.mklabels (a)
{20.0: '20', 18.0: '18', 16.0: '16', 14.0: '14', 12.0: '12',
    10.0: '10', 8.0: '8', 6.0: '6', 4.0: '4', 2.0: '2', 0.0: '0'}
>>> vcs.mklabels ( [5,.005])
{0.0050000000000000001: '0.005', 5.0: '5.000'}
>>> vcs.mklabels ( [.00002,.00005])
{2.0000000000000002e-05: '2E-5', 5.0000000000000002e-05: '5E-5'}
>>> vcs.mklabels ( [.00002,.00005],output='list')
['2E-5', '5E-5']
```

Parameters

- **vals** (*list, tuple*) – List or tuple of float values

- **output** (*str*) – Specifies the desired output type. One of ['dict', 'list'].

Returns Dictionary or list of labels for the given values.
Return type dict, list

vcs.utils.**mkscale**(*n1*, *n2*, *nc=12*, *zero=1*, *ends=False*)
This function return a nice scale given a min and a max

> **Warning:** Not all functionality for the 'zero' parameter has been implemented. zero=0 is intended to let the function decide what should be done with zeros, but it has yet to be defined. Do not use zero=0.

Examples

```
>>> vcs.mkscale(0,100)
[0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
>>> vcs.mkscale(0,100,nc=5)
[0.0, 20.0, 40.0, 60.0, 80.0, 100.0]
>>> vcs.mkscale(-10,100,nc=5)
[-25.0, 0.0, 25.0, 50.0, 75.0, 100.0]
>>> vcs.mkscale(-10,100,nc=5,zero=-1)
[-20.0, 20.0, 60.0, 100.0]
>>> vcs.mkscale(2,20)
[2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]
>>> vcs.mkscale(2,20,zero=2)
[0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]
```

Parameters

- **n1** (*float*) – Minimum number in range.
- **n2** (*float*) – Maximum number in range.
- **nc** (*int*) – Maximum number of intervals
- **zero** (*int*) – Integer flag to indicate how zero should be handled. Flags are as follows -1: zero MUST NOT be a contour

  0: let the function decide # NOT IMPLEMENTED 1: zero CAN be a contour (default) 2: zero MUST be a contour
- **end** (*bool*) – Boolean value indicating whether n1 and n2 should be part of the returned labels. Defaults to False.

Returns List of floats split into nc intervals
Return type *list*

vcs.utils.**rgba_color**(*color*, *colormap*)
Try all of the various syntaxes of colors and return 0-100 RGBA values.

Example

```
>>> cp = vcs.getcolormap() # Get a copy of the default colormap
>>> vcs.rgba_color('black', cp) # Find the rgba equivalent for black
[0.0, 0.0, 0.0, 100]
```

Parameters

- **color** (*int, str*) – The color to get the rgba value for. Can be an integer from 0-255, or a string name of a color.
- **colormap** (vcs.colormap.Cp) – A VCS colormap

Returns List of 4 floats; the R, G, B, and A values associated with the given color.
Return type *list*

vcs.utils.**setTicksandLabels**(*gm*, *copy_gm*, *datawc_x1*, *datawc_x2*, *datawc_y1*, *datawc_y2*, *x=None*, *y=None*)

Sets the labels and ticks for a graphics method made in python

Example

Parameters

- **gm** (*VCS graphics method*) – A VCS graphics method to alter
- **copy_gm** (*VCS graphics method*) – A VCS graphics method object
- **datawc_x1** (*float*) – Float value to set the graphics method's datawc_x1 property to.
- **datawc_x2** (*float*) – Float value to set the graphics method's datawc_x2 property to.
- **datawc_y1** (*float*) – Float value to set the graphics method's datawc_y1 property to.
- **datawc_y2** (*float*) – Float value to set the graphics method's datawc_y2 property to.
- **x** (*str*) – If provided, must be the string 'longitude'
- **y** (*str*) – If provided, must be the string 'latitude'

Returns A VCS graphics method object

Return type A VCS graphics method object

vcs.utils.**setcolorcell**(*obj*, *num*, *r*, *g*, *b*, *a=100*)

Set a individual color cell in the active colormap. If default is the active colormap, then return an error string.

Note: If the the visual display is 16-bit, 24-bit, or 32-bit TrueColor, then a redrawing of the VCS Canvas is made every time the color cell is changed.

Example

```
>>> vcs.setcolorcell("AMIP",11,0,0,0)
>>> vcs.setcolorcell("AMIP",21,100,0,0)
>>> vcs.setcolorcell("AMIP",31,0,100,0)
>>> vcs.setcolorcell("AMIP",41,0,0,100)
>>> vcs.setcolorcell("AMIP",51,100,100,100)
>>> vcs.setcolorcell("AMIP",61,70,70,70)
```

Parameters

- **obj** (*str or VCS object*) – String name of a colormap, or a VCS object
- **num** (*int*) – Integer specifying which color cell to change. Must be from 0-239.
- **r** (*int*) – Integer specifying the red value for the colorcell
- **g** (*int*) – Integer specifying the green value for the colorcell
- **b** (*int*) – Integer specifying the blue value for the colorcell
- **a** (*int*) – Integer specifying the opacity value for the colorcell. Must be from 0-100.

vcs.utils.**show**(**args*)

Show the list of VCS primary and secondary class objects.

Example

```
>>> a=vcs.init() # Create a VCS Canvas instance, named 'a'
>>> a.show('boxfill') # List boxfill objects on Canvas 'a'
>>> a.show('isofill') # List isofill objects on Canvas 'a'
>>> a.show('line') # List line objects on Canvas 'a'
>>> a.show('marker') # List marker objects on Canvas 'a'
>>> a.show('text') # List text objects on Canvas 'a'
```

1.6.10 vcshelp

VCS help module

vcs.vcshelp.**objecthelp**(**arg*)

Print the documentation of each object in the argument list. Prints a blank line if no documentation.

Example

```
>>> objects = [ vcs.get3d_scalar(), vcs.getcolormap(), vcs.
↪getboxfill() ]
>>> for object in objects:
...     vcs.objecthelp(object)
```

Parameters **arg** (*VCS object, or list of vcs objects*) – Instance(s) of VCS object(s) to display the documentation for. Multiple objects should be comma-delimited.

V