User's Guide to
PlotIt: Plotting Overture data,
GenericGraphicsInterface: A generic graphics interface,
GUIState: A graphical user interface,
GraphicsParameters: Setting graphics parameters,
NameList: Inputting parameters.

Version 1.19

Bill Henshaw & Anders Petersson[1]

Centre for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, 94551

henshaw@llnl.gov
http://www.llnl.gov/casc/people/henshaw

andersp@llnl.gov
http://www.llnl.gov/casc/people/petersson

http://www.llnl.gov/casc/Overture

May 20, 2011 UCRL-MA-132238 rev. 2

Abstract: The classes PlotIt, GenericGraphicsInterface, GUIState and GraphicsParameters can be used to setup graphics windows and a graphical user interface for plotting and interacting with various stuff from Overture, such as Mappings, MappedGrids or GridCollections, as well as plotting one-dimensional line plots from A++ arrays.

While the current implementation of the graphics interface is based on the graphics library OpenGL and the windowing system Motif/X11, the implementation details are confined to the class **GL_GraphicsInterface**, which is derived from the abstract base-class **GenericGraphicsInterface**. Therefore, application codes using graphics only need to know of the **GenericGraphicsInterface** class, which make them independent of the underlying implementation. A hierachical graphical user interface (GUI) containing dialog windows with pulldown, option and popup menues, push, toggle and radio buttons, as well as informative labels and textboxes for editable text, can be setup using the **GUIState** class.

Higher level functionality, such as plotting grids, contours, surfaces, and streamlines is provided in the class **PlotIt**, which only is a collection of functions and does not contain any data in itself. Parameters controlling the graphics are set using objects from the **GraphicsParameters** class.

Contents

16 List of parameters in GraphicsParameters

17 Writing your own plotting routines with OpenGL

18 Using the NameList Class for interactively changing parameters

List of Figures

1   Introduction

The classes **PlotIt**, **GenericGraphicsInterface**, **GUIState** and **GraphicsParameters** can be used to setup graphics windows and a graphical user interface for plotting and interacting with various stuff from Overture, such as Mappings, MappedGrids or GridCollections, as well as plotting one-dimensional line plots from A++ arrays. These classes are used in many places throughout the Overture library, for example to interactively make grids in **ogen**, manipulating geometry in **rap**, to setup parameters in the solver **overBlown**, and to postprocess results in **plotStuff**.

   The abstract base-class **GenericGraphicsInterface** defines an interface for using graphics from an application code. In itself, it implements text-based I/O as well as the reading and writing of command files, but it does not contain any code that is specific to a particular window or graphics system. Hence, application codes using **GenericGraphicsInterface** are independent of the underlying graphics and windowing system. Interactions with the underlying graphics system are done through virtual functions. The class **GL_GraphicsInterface**, which is derived from **GenericGraphicsInterface**, provides an implementation for systems with OpenGL graphics. The interaction with the underlying windowing system is confined to routines in the file mogl.C, which implements it using Motif/X11. Hence, to port **GL_GraphicsInterface** to another windowing system with OpenGL, it would not be necessary to change any other files in the distribution.

   The class **GUIState**, which is derived from **DialogData**, can be used to setup a hierachical graphical user interface (GUI). The GUI can contain dialog windows with pulldown, option and popup menues, push, toggle and radio buttons, as well as informative labels and textboxes for editable text.

   Higher level functionality, such as plotting contours, surfaces, and streamlines is implemented in the separate class **PlotIt**, which only contains static member functions and no data.

   It is quite simple to use **GenericGraphicsInterface** to plot stuff from Overture. Here is an example

```
  Overture::start();              // Initialize the Overture library
  CompositeGrid cg = ... ;        // define a CompositeGrid somehow
  Range all;
  realCompositeGridFunction u(cg,all,all,all,2);  // create a grid function
  u=...                           // give values to u

// get a reference to the graphics interface
  GenericGraphicsInterface & gi = *Overture::getGraphicsInterface();

  PlotIt::plot(gi,cg);                       // interactively plot a CompositeGrid
  PlotIt::contour(gi,u)                      // interactively plot contours and/or shaded surface of u
  PlotIt::streamLines(gi,u)                  // interactively plot streamlines
  ...
  Overture::finish();             // cleanup after the Overture library
```

   The GenericGraphicsInterface / GUIState graphical user interface is used by the applications 'ogen', 'rap', 'plotStuff', and most other applications in the Overture library. An example of the GenericGraphicsInterface / GUIState graphical user interface is shown in figure 1. This figure was produced by the short example code given in section 2, which will be used throughout the manual to describe the functionality in this class. The figure shows some of the features of the GUI, which include

- Plot a Mapping, MappedGrid or GridCollection (CompositeGrid).

- Plot contour, surface and streamlines of a 2D MappedGridFunction or CompositeGridFunction

- Plot contours on cutting planes, isosurfaces and streamlines of a 3D MappedGridFunction or CompositeGridFunction

- Plot 1D line plots of 1D grid functions.

- User defined dialog windows that can contain pulldown menus with push or toggle buttons, option menus, text labels (for inputting strings), push buttons, and toggle buttons.

- Multiple graphics windows and a single command window with a scrollable sub-window for outputting text, a command line, and a scrollable list of previous commands.
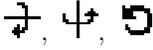
- Rotation buttons $\leftrightsquigarrow$, $\downarrow$, $\circlearrowleft$,... which rotate the object on the screen about fixed x, y, and z axes (the x axis is to the right, the y-axis is up and the z-axis is out of the screen).
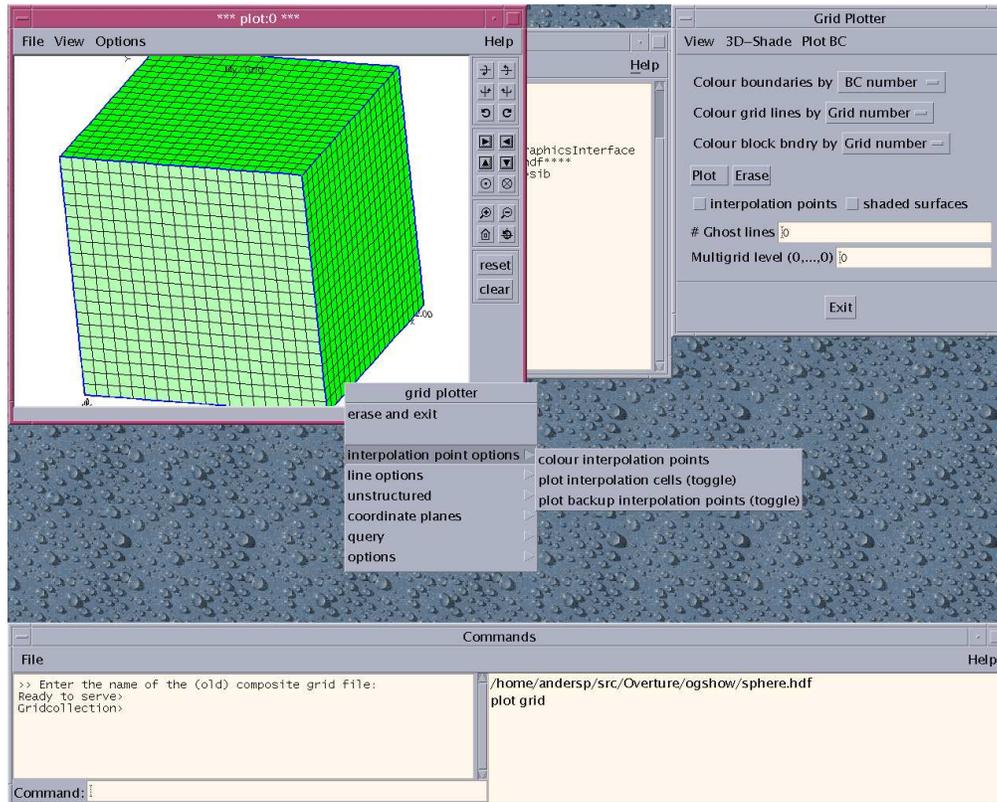
Figure 1: A screen shot from the test program

- Translation buttons ▣ ▣ ⊗, ⊙,... which shift the object on the screen along a given axis. The two last buttons shift the object in and out of the screen, respectively. Since an othographic projection method is used in the graphics interface, these buttons only change the appearance when clipping planes are used.

- Push buttons for making the objects bigger: ⊕, or smaller: ⊖, and a reset button: ⌂ to reset the view point, and a **clear** button to erase all objects on the screen.

- A push button ✛ to set the rotation center.

- A **rubber band zoom** feature.

- Mouse driven **translate, rotate and zoom**.

- A pop-up menu that is active on the command and graphics windows. This menu is defined by the application (= user program).

- Static pull-down menus (**file**, **view**, and **help**) on the graphics windows. Here, the screen can be saved in different formats, clipping planes and viewing characteristics can be set, annotations can be made (not fully implemented), and some help can be found.

- Static pull-down menus (**file** and **help**) on the command window. Here command files can be read/saved, new graphics windows can be opened, the window focus can be set, and the application can be aborted.

- An optional pull-down menu (**My menu** in this case) that is defined by the application.

- Pushbuttons (**Pick 3D** and **Plot** in this case) that are defined by the application.

- A file-selection dialog box (not shown in the figure).

- The option of typing any command on the command line or reading any command from a command file. All commands can be entered in this fashion, including any pop-up or pull-down menu item or any of the buttons, **x+r:0**, **y-r:0**, **x+:0**, **y+:0**, **bigger:0**, etc. For the buttons, the :0 refers to the window number where the view should be modified, which in this case is window #0. Furthermore, when typing a command, only the first distinguishing characters need to be entered.

- Recording or retrieving a command sequence in a command file.

2 A small application

The following program was used to generate the plot in figure 1. The source code is in the `tests` directory of the Overture distribution and is called `small-GI-test.C`. Instructions on how to run the application follows in next section.

3 Running the test application

After starting the **small-GI-test** application from a UNIX window, you will be queried "Enter the name of the (old) composite grid file:" and a file selection dialog box will appear. A proper input to the program is a database file containing an overlapping grid. This file can, for example, be generated with the program 'ogen'. Usually these files have the extension .hdf and you can look for such files by setting the filter text to be *.hdf. For example, when I run the program I set the text in the filterbox to be "/home/andersp/src/Overture/ogshow/*.hdf". To open a file, you click on one of the files in the list and then click on the OK button, or you can double click on the file name. After the program has succeeded in opening the file it will print "Ready to serve>" in the output sub-window in the command window.

At this point, there is one graphics window, one command window, and one dialog window on the screen. If you press the "Grid" push button, you will enter the grid plotter and the grid that is stored in the file will get plotted. The grid plotter changes the push buttons and the popup menu and puts up another dialog window with a lot of options. By pressing the toggle buttons on the dialog window, you can toggle different features on or off. The popup menu gives you further opportunity to change the appearance of the plot, for instance by changing the colours. When you are done exploring the grid plotter, you can exit by clicking on the "exit" button on the dialog window. The dialog window will then disappear and the push buttons will be restored to the original setting.

If you press the "Contour" push button in the left window, you will enter the contour plotter and a contour plot on the composite grid will appear on the screen. Similar to the grid plotter, a number of push buttons will appear and the popup menu will change to accomodate the needs of the contour plotter. When you are done exploring the contour plotter, you can exit by clicking on the "exit" push button. The push buttons will then get restored to the original setting.

To explore the more advanced (and perhaps less well-tested) features of the graphics interface, you can try plotting the grids and contours in different windows. To open a new graphics window, you select "new window" from the "File" menu on the command window (on the bottom of the screen). You can plot in the new window using the same technique as above. The new graphics window will have its title surrounded by asterisks to indicate that this is the active graphics window. To switch back to the original window, you need to first activate it. This is done by choosing the "figure" item from the "File" menu on the command window and typing a "0" in the text window following "Command:" (below the prompt). Once you have activated the first window, you will see that the title in this window now is surrounded by asterisks indicating that it is the active window.

3.1 Rotating and scaling the graphics window

Rotations are performed with respect to axes that are fixed relative to the screen. The x-axis points to the right, the y-axis points upward and the z-axis points out of the screen. Rotations can either be performed about the centre of the window, or about a user defined point. This point can either be set by first pressing the **set rotation point** icon on the graphics window and then clicking on the screen with the left mouse button. The rotation point can also be set by opening the "Set View Characteristics" dialog from the "View" pull-down menu (see section 3.3 for details). Note that the centre of the window is changed with the translation commands **x+**, **x-**, **y+**, etc.

Typing a rotation command with an argument (on the command line), such as **x+r:1 45**, will cause the view in window number 1 to rotate by 45 degrees about the x-axis.

Typing a translation command with an argument, such as **x+:0 .25** will cause the view in window number 0 to move to the right .25 units (in normalized screen coordinates; the screen goes from -1 to 1).

Rubber band zoom: The middle mouse button is used to ZOOM in. Press the middle button at one corner of a square, drag the mouse to another corner and lift the button. The view will magnify to the square that was marked. Use 'reset' to reset the view.

Mouse driven translate, rotate and zoom: All these operations are performed with the SHIFT key down. To translate, you hold the SHIFT key down, press the left mouse button and drag the cursor; the plotted objects will translate in the same direction as the mouse is moved. To rotate the view, you hold the SHIFT key down and press the middle mouse button and drag the cursor. Moving the cursor left or right will rotate about the y-axis (the vertical screen direction) and moving up or down will rotate about the x-axis (horizontal screen direction). To zoom in or out, you hold the SHIFT key down, press the right mouse button and drag the cursor vertically. To rotate about the z-axis, you press the left mouse buttons and drag the cursor horizontally.

Picking (aka selecting): While clicking the left mouse button, you select an object and get the (x, y, z) coordinate of the point on the object where you clicked. The object that was selected is reported in the selectionInfo data structure, which holds the global ID number, the front and back z-buffer coordinates and the window and 3–D coordinates. The global ID number can be used by the application to identify the selected object.

It is also possible to select several objects on the screen by specifying a rectangular region. To do this, you press the left mouse button in one corner of the rectangle and drag it to the diagonally opposite corner of the rectangle, where the mouse-button is released. The program will draw a rectangular frame to indicate the selected region. When you are happy with the selected region, you release the mouse button. An imaginary viewing volume is defined by translating the rectangular region into the screen and all objects that intersect the viewing volume are selected. However, only the 3–D coordinate of the closest object is computed. We remark that the object with the lowest front z-buffer value is the closest to the viewer. Also note:

1. Objects that are hidden by another object are also selected by this method.

2. The selection takes clipping planes (see below) into account, so it is not possible to select an object that has been removed by a clipping plane.

The mouse driven features are summarized in table 1.

Modifier	Mouse button	Function
	left	picking
	middle	rubber band zoom
	right	pop-up menu
<SHIFT>	left	translate
<SHIFT>	middle	rotate around the x & y axes
<SHIFT>	right	zoom (up & down) and z-rotation (left & right)

Table 1: Mouse driven features.

3.2   Using clip planes

The clip plane dialog for each graphics window is opened from the 'View' pull-down menu on the menu bar in the graphics window. Figure 2 shows an example from the test program. The first clip plane is activated by clicking on the toggle button in the top left corner. After it is activated, we can look inside the cube and see the sphere. By dragging the slider bar for the clipping plane, the clipping plane is moved closer or further away from the eye. The direction of the normal of the clipping plane can also be changed by editing the numbers in the 'Normal' box.

3.3   Setting the view characteristics

The view characteristics dialog for each graphics window is opened from the 'View' pull-down menu on the menu bar in the graphics window. Figure 3 shows an example from the test program. NOTE: When entering numerical values into a text box, it is necessary to hit <RETURN> before the changes take effect.

Here follows a brief description of the functionality in this window:

Background colour: Select a background colour from the menu by clicking on the label with the current colour. In this example, the background colour is white. Changing the background colour will take effect immediately.
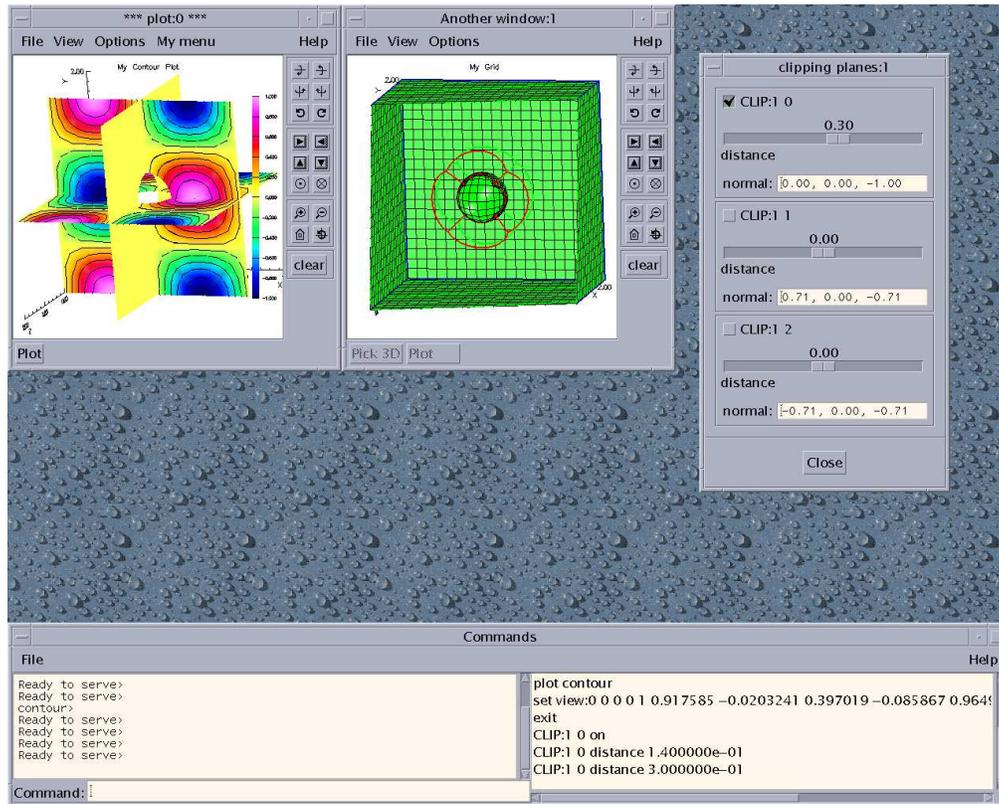
Figure 2: The clip plane dialog window

Text colour: Select a text (foreground) colour from the menu by clicking on the label with the current colour. In this example, the text colour is steel blue. Note that the text colour is used to colour the axes, the labels, and sometimes also the grid lines. However, only the axes will change colour immediately after a new colour is chosen. To update the colour of the labels and the grid lines, it is necessary to replot the object on the screen.

Axes origin: Click on the radio buttons to set the origin of the coordinate axes either at the default location (lower, left, back corner of the bounding box), or at the rotation point.

Rotation point: Enter the (X, Y, Z) coordinates of the rotation point. The rotation point will remain fixed to the screen during both interactive rotation with <SHIFT>+middle mouse button, and during rotation with the buttons **x+r**, **y+r**, etc.

Pick rotation point: After clicking on this button, set the rotation point by clicking with the left mouse button on a point on an object in the graphics window. NOTE:

1. Picking will only work in the window from which the view characteristics dialog was opened. If you are unsure which window to pick in, you can press the right mouse button and read the window number from the title of the popup menu.

Lighting: Activate/deactivate lighting in the graphics window.

Light #i: Turn on/off light source number i, $i = 0, 1, 2$. Turning off all light sources is the same as deactivating lighting with the above function.

   Position (X, Y, Z): The location (X, Y, Z) of light source number i.

   Ambient (R, G, B, A): The ambient colour (R, G, B, A) of light source number i.

   Diffusive (R, G, B, A): The diffusive colour (R, G, B, A) of light source number i.

   Specular (R, G, B, A): The specular colour (R, G, B, A) of light source number i.
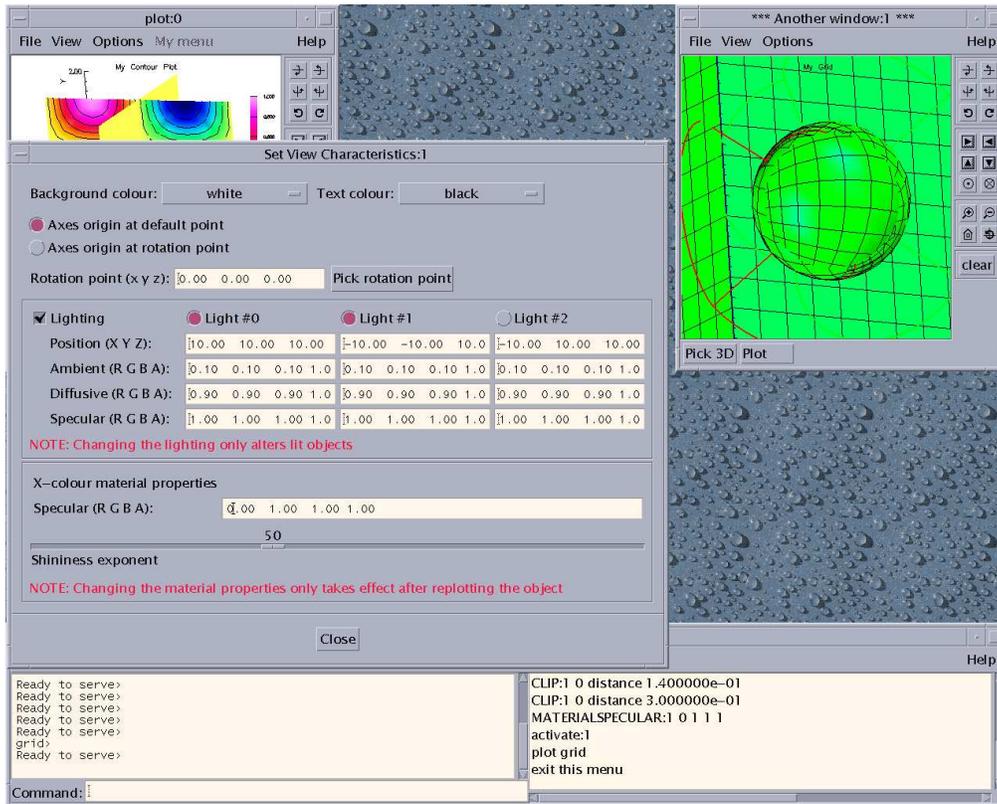
Figure 3: The view characteristics dialog window

X-colour material properties: The following properties characterize the default material, which is used when a X-colour is chosen for a lit object. The X-colours are distinguished from the predefined "special" materials in that only their ambient and diffuse reflections are defined, but not their specular and shininess properties. (For those fluent in OpenGL, this functionality is obtained by calling the function glColorMaterial with the argument GL_AMBIENT_AND_DIFFUSE.) Note that the reflective properties only influence objects that are lit. Also note that the objects need to be re-plotted in order for the changes to take effect.

The predefined materials are listed in table 2. Hence, any colour that is not in the table is considered to be an X-colour.

emerald	jade	obsidian	pearl
ruby	turquoise	brass	bronze
chrome	copper	gold	silver
blackPlastic	cyanPlastic	greenPlastic	redPlastic
whitePlastic	yellowPlastic	blackRubber	cyanRubber
greenRubber	redRubber	whiteRubber	yellowRubber

Table 2: Predefined materials.

Specular (R, G, B, A): The specular reflective property of the X-colour material. For example, by setting the first number to zero, you will get a bluish reflection on the green sphere in the example application. Note that the same effect could have been obtained by changing the specular colour of the light sources.

Shininess exponent: A number between 0 and 128 that describes how "narrow" the specular reflection of the X-colour material will be. A lower number gives a wider reflection.

3.4   Opening additional graphics windows

Additional graphics windows can be opened both interactively and from within the code. To open a new graphics window from the test application, you select the "new window" item from the "File" menu in the command window. Once the new window is opened, it will become the active window. This window will have the same push buttons as the original one, and you plot in the same way as before.

The following example code opens a new graphics window.

```
...
GenericGraphicsInterface & gi = *Overture::getGraphicsInterface("window title");
int win0 = 0; // the default graphics window has number 0
...
int win1 = gi.createWindow("Another window");
...
```

3.5   Saving postscript files and including them in TeX documents

To save the contents of the graphics window in a postscript, encapsulated postscript, or ppm file, you can use the `Hardcopy` dialog which is opened from the `file` pull-down menu on the graphics window. From within a C++ program, you can use the `hardCopy` function to save a post-script file.

Note that the image is rendered off-screen in a buffer with default resolution of 1024×1024. This resolution can be changed in the `Hardcopy` dialog (but see the note below). The format of the post-script file can also be changed between 8-bit colour (i.e. 2^8 colours, but actually only 255 colours) , 24-bit colour (i.e. 2^{24} colours), black-and-white or gray-scale. See the comments for the `hardCopy` function for more details. Also see the comments in the section describing the `GraphicsParameters`, reference `GI_OUTPUT_FORMAT`, `GI_RASTER_RESOLUTION`, and `GI_HARD_COPY_TYPE`.

If you have saved an image as a postscript file you can use the `epsfig` command in a LaTeX or TeX file to embed the figure as in the following example:

```
\begin{figure} \label{fig:CompositeGrid}
  \begin{center}
   \epsfig{file=figure.ps,height=7.0in}
  \caption{Plot of a 2D CompositeGrid}
  \end{center}
\end{figure}
```

Note that the raster resolution is independent of the height and width values that you give to `epsfig`. If you make the figure too small, however, some thin lines may be lost on the plot depending on the resolution of the output device. Thus a plot may print ok even though it may look bad on the screen since a laser printer has more resolution than the typical screen.

3.5.1   Saving high resolution figures

If you are using OpenGL Mesa you may increase the maximum resolution of hard-copies. Mesa is compiled with certain default maximum values and the highest resolution depends on the parameters that were used to compile Mesa. To increase the Mesa resolution you should change the definitions for `MAX_WIDTH` and `MAX_HEIGHT` in `Mesa/src/config.h`. For example you may set these to be 2048 or more. Mesa must be re-compiled after these changes are made.

There is a problem with saving an image at a high resolution since the width of lines and size of points are specified in number of pixels in OpenGL (!). Thus lines become thinner as the resolution increases. To fix this you can increase the width of all lines by setting the `Line width scale factor` in `View Characteristics dialog` (found under the `View` button on the top of the main graphics window). The default scale factor is 1.0 and I suggest that you change this to 3.0 if your output resolution is 2048. (You may even want to set the scale factor to 2.0 for the default resolution of 1024.) You will then need to replot your figure. The lines will now appear very thick on the screen but they will look better on the hardcopy.

4   Embedding Perl statements into command files

Perl statements can be used inside command files to define variables that can later be used in commands to Overture functions. Overture creates an instance of a perl interpreter (see OvertureParser.C if you are interested in the details)

and thus any valid perl statement can be used in a command file. Here is a simple example showing how to compute the number of lines that are later used as input to some Overture function:

```
$nx=5*7; $ny=$nx+5;
lines
  $nx $ny
```

This example shows the two cases when the perl interpreter is invoked:

- Any non-comment line containing a semi-colon, ';', is considering to contain perl statements and is sent to the perl interpreter.

- Any line containing a dollar, '$', but no semi-colon, is first sent to the perl interpreter for *variable evaluation only*, and the result is returned to the Overture function that is reading the command file.

As a second example we show how to define and use a perl subroutine,

```
* scale number of grid points in each direction by the following factor
$factor=2.**(1./3.); printf(" factor=$factor\n");
*
* Define a subroutine to convert the number of grid points
sub getGridPoints\
{ local($n1,$n2,$n3)=@_; \
  $nx=int(($n1-1)*$factor+1.5); \
  $ny=int(($n2-1)*$factor+1.5); \
  $nz=int(($n3-1)*$factor+1.5);\
}
* call the perl subroutine to compute $nx,$ny,$nz
getGridPoints(21,11,31);
lines
  $nx $ny $nz
```

Note the use of the back-slash for continuation lines.

Perl conditional statements and loops can be used in an indirect way within command files. Within a conditional or loop, a string can be generated that contains multiple commands; these commands can then be used as command file statements as illustrated in the following example:

```
if( $nd eq "2" ){  $commands = "plot grid \n plot streamLines \n plot contour";} \
else{ $commands = "plot grid \n plot contour"; }
* Now execute the commands
$commands
```

Note that multiple commands can be placed in the $commands perl variable using the newline character '\n'.

5   ppm2mpeg: Making mpeg movies

The `plotStuff` post processor can be used to make mpeg movies. The `movie and save` option allows one to create a sequence of `ppm` files that can be encoded into an mpeg movie using the perl script `Overture/bin/ppm2mpeg`.

You can also make frames for a movie directly in the GUI using the Movie dialog from the File menu. This functionality enables you to save a specified number of frames while moving the view point from one place to another.

You can also make your own `ppm` files by directly calling the `hardCopy` function with the `ppm` file option or by using the `Hardcopy` to save frames in the ppm format.

6   Creating gif files to add to your www home page

To create a `gif` file (that can be included on one's www home page) one can first create a `ppm` file by calling the `hardCopy` function with the `ppm` file option or by using the `Hardcopy` dialog. You then need to convert the `ppm` file to a `gif` file. I use the `convert` utility which is now available on many machines. In principle you could also convert a postscript file to a gif file but this doesn't usually work very well because the resolution gets messed up.

7 ps2ppm: Converting Post-script files to PPM files

There is a conversion routine, `Overture/bin/ps2ppm`, (which you may have to make by going to the Overture/bin directory and typing "make ps2ppm), that can convert a post-script file generated by the GenericGraphicsInterface into a raster file (.ppm file). The later can be converted to gif using some utility routine. Most conversion routines that I am aware of will botch the direct conversion of the postscript file to a gif file.

8 Changing the default appearance of the windows

Default settings for some parameters can be changed throught a file named .overturerc in your HOME directory. An example of an .overturerc file is

```
commandwindow*width: 800
commandwindow*height: 150
graphicswindow*width:  650
graphicswindow*height: 500
backgroundcolour: mediumgoldenrod
foregroundcolour: steelblue
```

The window sizes is specified in pixels. It is not necessary to specify both the width and height, and the default size is obtained either by omitting the command completely, or by setting the size to -1. The default foreground colour is black and the default background colour is white. If you change them, you must use one of the following colours:

black	white	red	blue	green
orange	yellow	darkgreen	seagreen	skyblue
navyblue	violet	pink	turquoise	gold
coral	violetred	darkturquoise	steelblue	orchid
salmon	aquamarine	mediumgoldenrod	wheat	khaki
maroon	slateblue	darkorchid	plum	

9 Examples from PlotIt

In figures 4-15, we present assorted plots generated by PlotIt functions.
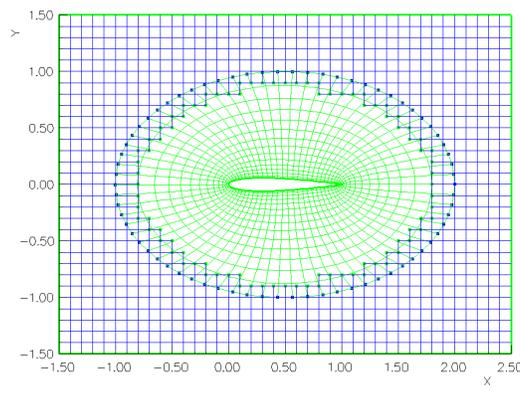
Figure 4: Plot of a 2D CompositeGrid around a NACA0012 airfoil.
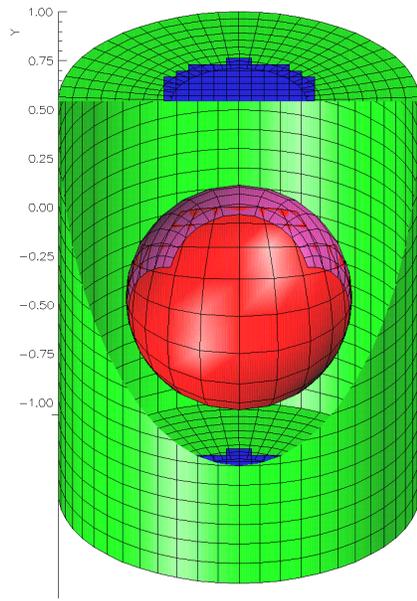
Figure 5: Plot of a 3D CompositeGrid, the sphere-in-a-tube grid.
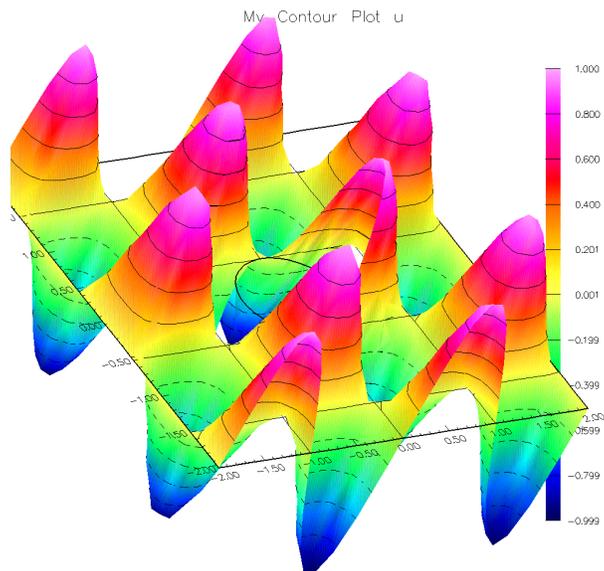
Figure 6: Contour lines and surface plot of a two dimensional grid function.
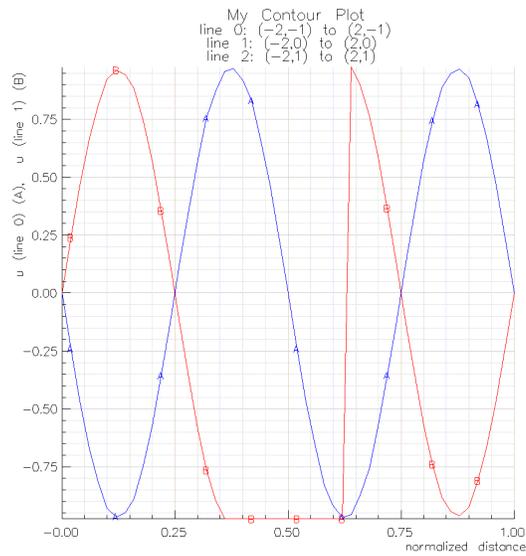
Figure 7: The solution plotted along lines that pass through an overlapping grid (from figure (6)). The solution values are set to the minimum value where the line crosses the hole in the middle of the grid.
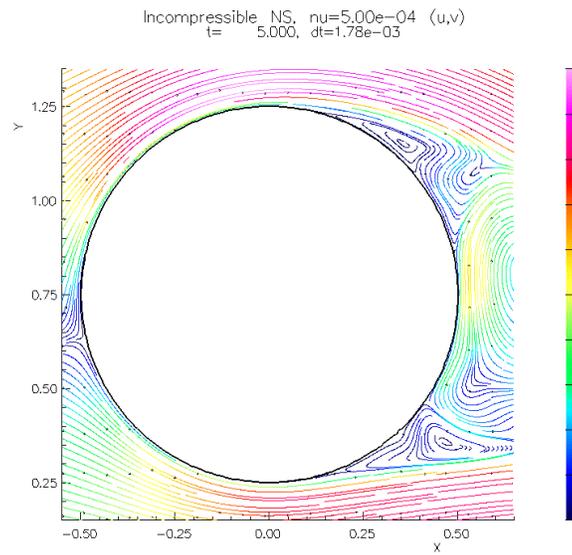
Figure 8: Stream lines around a circular cylinder.

17
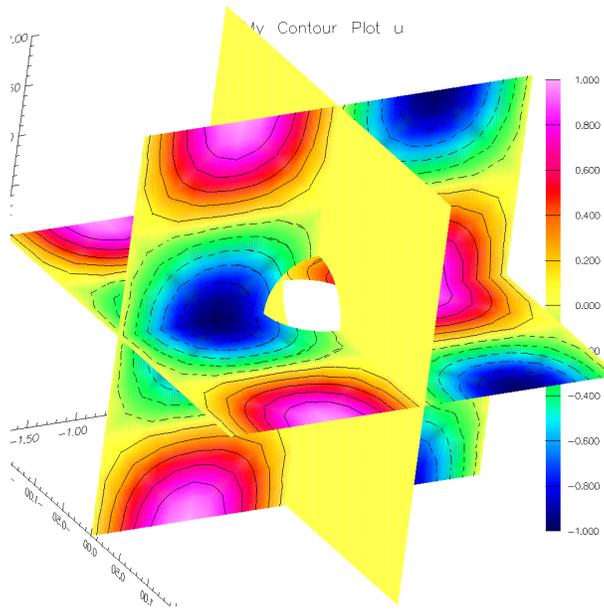
Figure 9: Contour planes drawn on the sphere-in-a-box grid. The planes cut across the component grids.
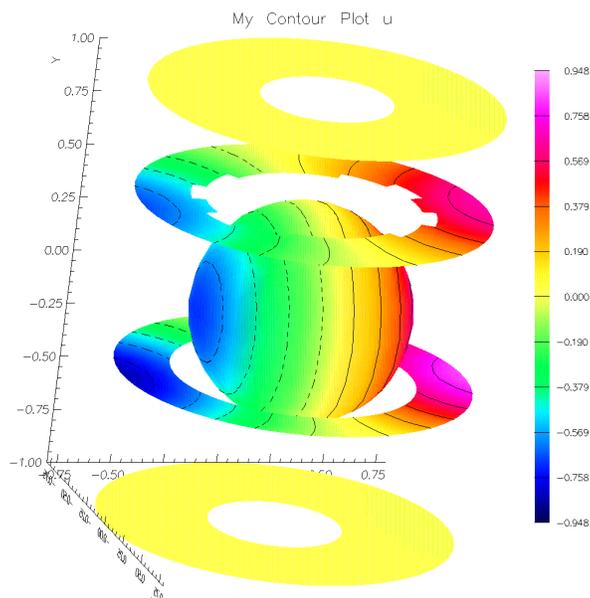
Figure 10: Contour plot of some specified coordinate planes for the sphere-in-a-tube grid.

18
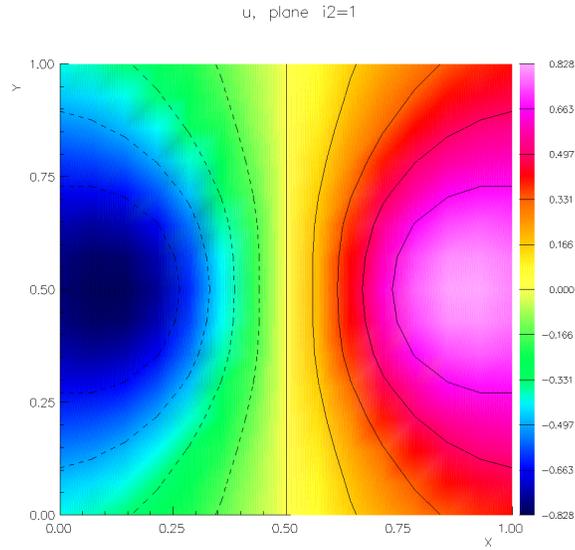
Figure 11: Contour plot of a coordinate plane of a 3d grid (for one of the component grids covering the sphere from the sphere-in-a-box grid) projected onto a plane. The different coordinate planes can be selected from the `component` menu option.
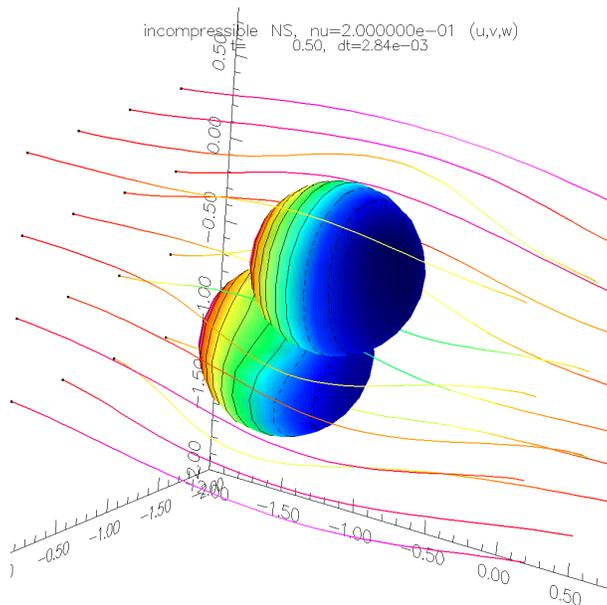
Figure 12: Stream lines in 3D for flow past two spheres, the pressure is plotted on surface of the the spheres.
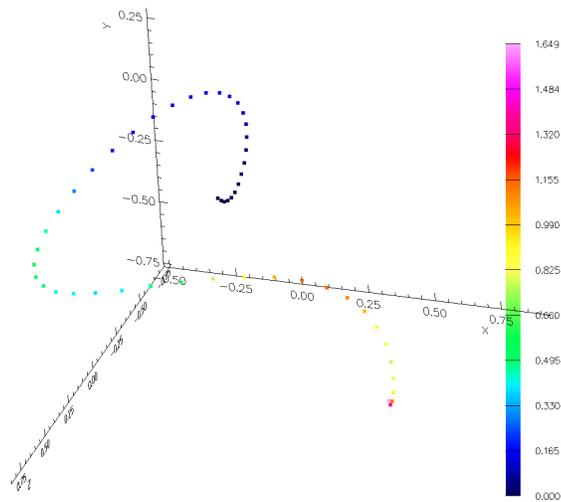
Figure 13: Use the `plotPoints` function to plot points in space and optionally colour each point based on a value.
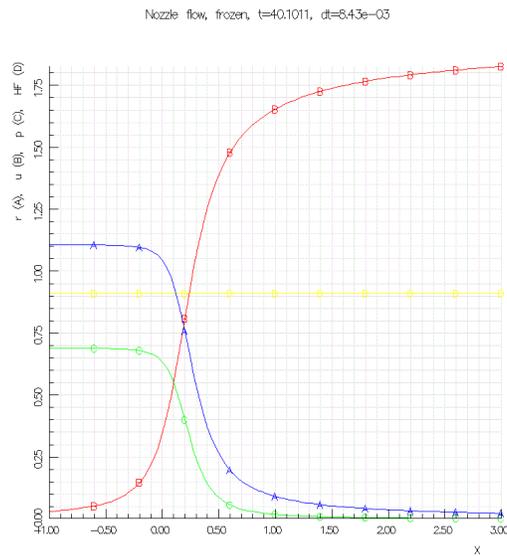
Figure 14: Line plots can be made with the contour function for 1D grid functions or the plot function for arrays
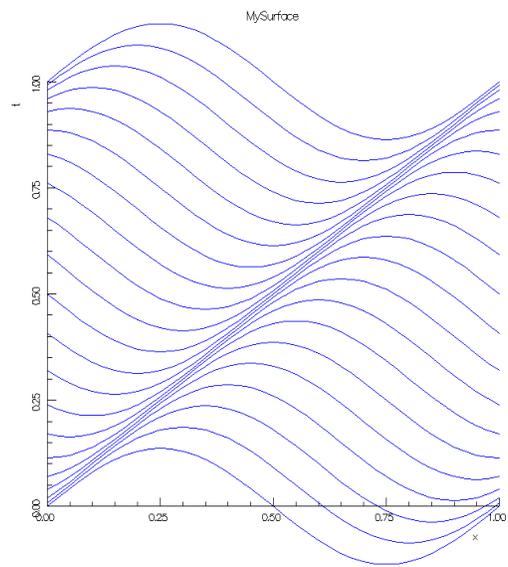
Figure 15: A time sequence of line plots.

10 PlotIt function descriptions

PlotIt contains a collection of functions for making various high level plotting of Overture objects, such as contour plots or streamlines. PlotIt does not contain any data and all functions are static. Hence, you don't have to create a PlotIt object to access these functions. Instead you just prepend the function name with "PlotIt::" in the calling sequence.

10.1 Plot a MappedGrid

void
plot(GenericGraphicsInterface &gi, MappedGrid & mg,
GraphicsParameters & parameters = Overture::defaultGraphicsParameters())

Description: Plot a MappedGrid and optionally supply parameters that define the plot characteristics. In two-dimensions, grid-lines are plotted. In three dimensions, by default only the block boundaries of the grid are plotted. You may also plot grid lines on boundaries and/or plot shaded boundary surfaces.

Grids and boundary conditions are plotted with different colours. Grids are numbered and boundary conditions are numbered. For each number there corresponds a colour. The colour associated with each number is plotted in the lower right corner.

mg (input): MappedGrid to plot.

parameters (input/output): Supply optional parameters to alter plot characteristics.

Author: WDH

10.2 Plot a GridCollection (CompositeGrid)

void
plot(GenericGraphicsInterface &gi, GridCollection & gc0,
GraphicsParameters & parameters = Overture::defaultGraphicsParameters())

Description: Plot a CompositeGrid and optionally supply parameters that define the plot characteristics. In two dimensions, grid-lines are plotted. Interpolation points can be plotted with small circles. In three dimensions, by default only the block boundaries of the grid are plotted. You may also plot grid lines on boundaries and/or plot shaded boundary surfaces.

Grids and boundary conditions are plotted with different colours. Grids are numbered and boundary conditions are numbered. For each number there corresponds a colour. The colour associated with each number is plotted in the lower right corner.

gc (input): CompositeGrid to plot.

parameters (input/output): Supply optional parameters to alter plot characteristics.

Author: WDH & AP

10.3 Plot the displacement for a solid mechanics problem

void
displacement(GenericGraphicsInterface &gi, const realGridCollectionFunction & u,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters())

Description: Plot the "displacement" for a solid mechanics problem, d = x + u, where x are the grid vertices and u is the provided solution.

u (input): the pertubation to the grid points.

parameters (input/output): Supply optional parameters to alter plot characteristics.

Author: WDH & AP

10.4   plot: 1D line plots

void
plot(GenericGraphicsInterface &gi,
  const realArray & t,
  const realArray & x,
  const aString & title = nullString,
  const aString & tName = nullString,
  const aString *xName =NULL,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters())

Description: Make some 1D line plots.

t (input) : t(0:n-1) - values along the horizontal axis

x (input) : x(0:n-1,0:nv-1) - values to plot, nv components

title (input):

tName (input): name for horizontal axis

xName[nv] (input): names of components/

10.5   plot: surface plots

void
plot(GenericGraphicsInterface &gi,
  const realArray & x,
  const realArray & t,
  const realArray & u_,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters())

Description: Make a surface plot of a sequence of 1D functions

x (input) : t(0:n-1) - values along the horizontal axis ("x-axis")

t (input) : t(0:nv-1) - times corresponding to the different components of u ("y-axis")

u (input) : u(0:n-1,0:nv-1) - values to plot, nv components ("z-axis")

parameters: Use these parameters to set the title, sub-titles and axis labels etc.

10.6   Plot a Mapping

void
plot(GenericGraphicsInterface &gi, Mapping& map,
  GraphicsParameters & parameters = nullGraphicsParameters,
  int dList = 0 */, bool lit /* = 0)

Description: Plot a mapping. Plot curves, surfaces and volumes in 1,2 or 3 space dimensions. Plot grid lines. In 3D plot shaded surfaces and grid lines.

map (input): Mapping to plot.

parameters (input/output): supply optional parameters to change plotting characteristics.

Return Values: none.

Author: WDH & AP

10.7   Plot an AdvancingFront

void
plot(GenericGraphicsInterface & gi, AdvancingFront & front,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters())

Description: Plot an AdvancingFront.

front (input): AdvancingFront to plot

parameters (input/output): supply optional parameters to change plotting characteristics.

Return Values: none.

Author: WDH & AP & KKC

10.8   Contour a realMappedGridFunction

void
contour(GenericGraphicsInterface &gi, const realMappedGridFunction & u,
GraphicsParameters & parameters = Overture::defaultGraphicsParameters())

Description: Plot contours of a realMappedGridFunction in 2D or 3D. Optionally supply parameters that define
the plot characteristics. In two dimensions, plotting options include

- plot shaded surface
- plot (colour) contour lines
- plot wire mesh surface (hidden lines are not supported here due to limitations in OpenGL).
- choose which component to plot
- plot the solution along one or more lines that passes through the grid.

In 3D options include

- plot shaded surface contours on arbitrary planes that cut through the grid
- plot shaded surface contours on boundaries.
- plot (colour) contour lines on the planes or boundaries
- plot contours on specified coordinates planes.
- plot 2D contours for specified coordinates planes.
- plot the solution along one or more lines that pass through the grid.

u (input): function to plot contours of

parameters (input): supply optional parameters

Author: WDH

10.9   Contour a CompositeGridFunction

void
contour(GenericGraphicsInterface &gi, const realGridCollectionFunction & u,
GraphicsParameters & parameters = Overture::defaultGraphicsParameters())

Description: Plot contours of a realMappedGridFunction in 2D or 3D. Optionally supply parameters that define
the plot characteristics. In two dimensions, plotting options include

- plot shaded surface
- plot (colour) contour lines
- plot wire mesh surface (hidden lines are not supported here due to limitations in OpenGL).
- choose which component to plot

- plot the solution along one or more lines that passes through the grid.

In 3D options include

- plot shaded surface contours on arbitrary planes that cut through the grid
- plot shaded surface contours on boundaries.
- plot (colour) contour lines on the planes or boundaries
- plot contours on specified coordinates planes.
- plot 2D contours for specified coordinates planes.
- plot the solution along one or more lines that pass through the grid.

u (input): function to plot contours of

parameters (input): supply optional parameters

Author: WDH

10.10   StreamLines of a realMappedGridFunction

void
streamLines(GenericGraphicsInterface &gi, const realMappedGridFunction & uv,
GraphicsParameters & parameters = Overture::defaultGraphicsParameters())

Description: Plot stream lines of a two-dimensional vector field. Optionally supply parameters that define the plot characteristics. This routine draws lines that are parallel to a vector field defined by two components of the grid function uv. By default component values 0 and 1 of the first component of uv are used for "u" and "v". Plotting options include

- choose the components to use for "u" and "v".
- choose new plot bounds (to zoom in on a particular region). In this case new streamlines are drawn on the new region as opposed to the plot being simply magnified.

uv (input): function to plot streamlines of.

parameters (input): supply optional parameters

Remarks:

- The streamlines are coloured by the relative value of $u^2 + v^2$
- Streamlines that move too slowly are stopped
- There is a maximum number of steps used to integrate any streamline.
- To plot streamlines to cover a CompositeGrid, a rectangular background grid is made that covers some region (this region could be smaller than the entire grid if we are zooming). The number of points on this grid is nxg*nyg. The IntegerArray ig(nxg,nyg) is used to mark cells in the background grid. Streamlines are drawn starting at the midpoints of the background grid. Whenever a streamline passes through a cell of the background grid, the value of ig(i,j) is increased by one. Only two streamlines are allowed per cell or else the streamline is stopped (or never started). In this way streamlines cover the domain in a reasonably uniform manner.

Author: WDH & AP

10.11   StreamLines of a CompositeGridFunction

void
streamLines(GenericGraphicsInterface &gi, const realGridCollectionFunction & uv0,
**   GraphicsParameters & parameters)**

Description: Streamline plots of the "velocity pair" (u,v) (where u and v are both grid collection functions).

uv (input): function to plot streamlines of.

parameters (input): supply optional parameters

Remarks: 1. The streamlines are coloured by the relative value of u**2+v**2

2. Streamlines that move too slowly are stopped

3. There is a maximum number of steps used to integrate any streamline.

4. To plot streamlines to cover a GridCollection, I make a rectangular background grid that covers some region (this region could be smaller than the entire grid if we are zooming). The number of points on this grid is nxg*nyg. The IntegerArray maskForStreamLines(nxg,nyg) is used to mark cells in the background grid. I draw streamlines starting at the midpoints of the background grid. Whenever a streamline pass through a cell of the background grid I increase the value of maskForStreamLines(i,j) by one. Only two streamlines are allowed per cell or else the streamline is stopped (or never started). In this way streamlines cover the domain in a reasonably uniform manner.

Author: WDH & AP

11 GenericGraphicsInterface function descriptions

GenericGraphicsInterface contains functions for handling text IO. For example, it knows how to display a text menu and getting a response, input and output strings, and how to read and write command files. The class GenericGraphicsInterface does not implement any of the plotting or graphical user interface (GUI) functions, but it provides pure virtual functions to perform such tasks. This enables applications to be written independently of the actual graphics and windowing system. The class GL_GraphicsInterface is derived from GenericGraphicsInterface and implementes the virtual functions using OpenGL and Motif.

The GraphicsInterface object is allocated by the first call to the function getGraphicsInterface in the Overture class. There can only be one such object and subsequent calls to that function will just return a pointer to the original object. Note that you should NOT build your own GL_GraphicsInterface object in your application.

We proceed by describing the functionality provided in GenericGraphicsInterface.

11.1 turnOffGraphics

void
turnOffGraphics()

Description: Turn off graphics plotting (grid, contour, streamline... plots). We may turn off plotting in batch mode or parallel to avoid some computations (in parallel we avoid building a copy of the grid on one processor)

return value: true or false.

Author: WDH

11.2 getValues (IntegerArray)

int
getValues(const aString & prompt,
   IntegerArray & values,
   const int minimumValue =INT_MIN,
   const int maximumValue =INT_MAX,
   const int sort = 0)

Description: Read in a set of integer values.

prompt (input) : use this prompt

values (output) : return values in this array, dimensioned to the number of values.

minimumValue (input) : specify an optional minimum value. All returned values will be at least this value.

maximumValue (input) : specify an optional maximum value. All returned values will be at no greater than this value.

sort (input) : optional indicator. If $sort > 0$ sort the values to be in increasing order, if $sort < 0$, sort the values to be in decreasing order. If $sort == 0$, no sorting is done.

return value: Number of values read.

Author: WDH

11.3 getValues (RealArray)

int
getValues(const aString & prompt,
   RealArray & values,
   const real minimumValue =-REAL_MAX,
   const real maximumValue =REAL_MAX,
   const int sort = 0)

Description: Read in a set of real values.

prompt (input) : use this prompt

values (output) : return values in this array, dimensioned to the number of values.

minimumValue (input) : specify an optional minimum value. All returned values will be at least this value.

maximumValue (input) : specify an optional maximum value. All returned values will be at no greater than this value.

sort (input) : optional indicator. If $sort > 0$ sort the values to be in increasing order, if $sort < 0$, sort the values to be in decreasing order. If $sort == 0$, no sorting is done.

return value: Number of values read.

Author: WDH

11.4   getDefaultPrompt

const aString &
getDefaultPrompt()

Description: Return the current defaultPrompt

Author: WDH

11.5   setDefaultPrompt

int
setDefaultPrompt(const aString & prompt)

Description: Set the deafult prompt and clear the stack of default prompts

Author: WDH

11.6   pushDefaultPrompt

int
pushDefaultPrompt(const aString & prompt)

Description: Push a default prompt onto a stack and make it the current prompt

Author: WDH

11.7   popDefaultPrompt

int
popDefaultPrompt()

Description: pop a default prompt off the stack and make the next prompt the new default

Author: WDH

11.8   appendToTheDefaultPrompt

int
appendToTheDefaultPrompt(const aString & appendage)

Description: Append a aString to the defaultPrompt and push this new prompt onto the stack. Also increase the amount of indentation used when writing to command files.

appendage (input): append this string to the default prompt.

Author: WDH

11.9   unAppendTheDefaultPrompt

int
unAppendTheDefaultPrompt()

Description: Remove the last string appended to the default prompt by popping the stack. Also decrease the amount of indentation used when writing to command files.

Author: WDH

11.10   outputString (base class)

void
outputString(const aString & message, int messageLevel =2)

Description: Output a string to standard output. If the echo file is open, also output the string in that file.

message (input): the string to be output.

messageLevel (input) : output the string if messageLevel is less than or equal to the current value for infoLevel. Values for infoLevel are 0=expert, 1=intermediate, 2=novice.

Return Values: none.

Author: WDH

11.11   readCommandFile

FILE*
readCommandFile(const aString & commandFileName =nullString)

Description: Start reading a command file.

commandFileName (input): If `commandFileName` is specified then this should be the name of the command file to read. This routine will automatically add a ".cmd" to the file name if the file named `commandFileName` is not found. If `commandFileName` is not given then you will be prompted to enter the name of the file.

Errors: Unable to open the file.

Return Values: Pointer to the opened file or NULL if able to open the file.

Author: WDH

11.12   getReadCommandFile

FILE*
getReadCommandFile() const

Return a file pointer to the current command file we are reading

11.13   getSaveCommandFile

FILE*
getSaveCommandFile() const

Return a file pointer to the current command file we are saving

11.14   readCommandsFromStrings

int
readCommandsFromStrings(const aString *commands)

Description: Start reading commands from an array of Strings, commands terminated by the aString=""

commands (input): A list of Strings (commands). There must be a null string, "", to indicate the end of the list.

Errors: unexpected results will occur if there is no null string to terminate the array.

Return Values: 0

Author: WDH

11.15   readingFromCommandFile

bool
readingFromCommandFile() const

Description: Return true of we are reading from a command file.

11.16   saveCommandFile

FILE*
saveCommandFile(const aString & commandFileName =nullString)

Description: Start saving a command file.

commandFileName (input): If `commandFileName` is specified then this should be the name of the command file to save commands in. It will first be opened. If `commandFileName` is not given then you will be prompted to enter the name of the file.

Errors: Unable to open the file.

Return Values: Pointer to the opened file or NULL if able to open the file.

Author: WDH & AP

11.17   abortIfCommandFileEnds

void
abortIfCommandFileEnds(bool trueOrFalse =true)

Specify whether to abort the program if we stop reading a command file. This option is used by automated scripts to prevent a program hanging while waiting for input.

trueOrFalse (input): the explanation is in the name itself!

Return Values: None.

Author: WDH

11.18   outputToCommandFile

void
outputToCommandFile(const aString & line)

Description: Output a line to the command file if there is one open.

line (input) : save this string, NOTE: you should include a newline character if you want one.

Author: WDH

11.19   saveEchoFile

FILE*
saveEchoFile(const aString & fileName /*=nullString*/)

Description: Start saving an echo file (if no file name is given, prompt for one)

Author: AP

11.20   stopSavingEchoFile

void
stopSavingEchoFile()

Description: Stop saving the echo file (and close the file)

Author: AP

11.21   savePickCommands

int
savePickCommands(bool trueOrFalse =TRUE)

Description: Set wether picking commands should be logged in the command file in their raw form.

trueOrFalse(input): the description is in the name!

Author: AP

11.22   setIgnorePause

void
setIgnorePause(bool trueOrFalse =true)

Description: If true, ignore the "pause" statement in command files.

Author: WDH

11.23   stopReadingCommandFile

void
stopReadingCommandFile()

Description: Stop reading the command file (and close the file)

Author: WDH

11.24   stopSavingCommandFile

void
stopSavingCommandFile()

Description: Stop saving the command file (and close the file)

Author: WDH

11.25   buildCascadingMenu

int
buildCascadingMenu(aString *&menu,
**   int startCascade,**
**   int endCascade) const**

Description: Take a menu that (might) have a long list of items and cascade these items so that they will appear nicely on the screen.

menu (input/output) : On input an array of strings termined with a "" (null) string. On output a new cascading menu.

startCascade,endCascade (input) : these specify the set of menu items to cascade.

Author: WDH

11.26   indexInCascadingMenu

int
indexInCascadingMenu(int & index,
**   const int startCascade,**
**   const int endCascade) const**

Description: Used in conjuction with buildCascadingMenu to convert an index into the cascading menu into an index into the original menu.

index (input/ouput) : on input this is a index into the cascading menu, built by buildCascadingMenu On output this is an index into the original menu.

return value: is the same as index.

Author: WDH

11.27   inputString (base class)

void
inputString(aString &answer,
**   const aString & prompt =nullString)**

Description: Input a string after displaying an optional prompt

answer (output): the string that was read

prompt (input): display an optional prompt.

Return Values: none.

Author: WDH

11.28   getMatch

int
getMatch(const aString *menu, aString & answer)

Description: Find the menu item that "best" matches answer – answer can be a truncated version of the menu item but it must be a unique match.

menu(input): array of strings terminated by empty string.

answer(input): string to be matched.

Return values: Index in the menu array of the unique matching entry, or -1 if no unique entry was found or other error occured.

Author: WDH

11.29 readLineFromCommandFile

int
readLineFromCommandFile(aString & answer)

Purpose: Read a line from the command file. Lines beginning with a "*" or "#" are treated as comments.

Return values: Number of characters read. A return value of zero means that an end-of-file was reached.

Author: WDH

11.30 createWindow

int
createWindow(const aString & windowTitle = nullString,
          int argc =0,
          char *argv[] = NULL)

Description: On the first call (usually made by the constructor), open a command and a graphics window. On subsequent calls, open another graphics window.

windowTitle (input): Title to appear on the graphics window

argc (input/output): The argument count to main.

argv (input/output): The arguments to main.

Return Value: The number of the graphics window that was created. The graphics windows are numbered 0,1,2,... This number needs to be passed to `setCurrentWindow`, for example. The window number is also used when typing viewing commands on the command line, such as **x+r:0**.

Author: AP

11.31 setCurrentWindow

void
setCurrentWindow(const int & w)

Description: Set the active graphics window to 'w'. Subsequent plots will appear in this window.

w (input) : the number of the window to activate

Return value: none.

Author: AP

11.32 getCurrentWindow

int
getCurrentWindow()

Description: Return the number of the active graphics window.

Return value: The number of the active graphics window.

Author: AP

11.33 displayHelp

bool
displayHelp(const aString & topic)

Description: display help on a topic that appears in the "help" pulldown menu

topic (input) : the topic that help is requested for.

Return value: TRUE if help was found for the topic, false if no help found

Author: WDH

11.34 destroyWindow

int
destroyWindow(int win_number)

Description: Destroy one graphics window.

Note: NOT implemented yet.

Author: WDH

11.35 generateNewDisplayList

int
**generateNewDisplayList(bool lit = false */, bool plotIt /* = true */, bool hideable /* = false,
bool interactive = true)**

Description: Use this function to allocate an available display list to use in the currentWindow. The display lists
allocated here will be rotated and scaled when buttons like x+r, "bigger", etc. are chosen.

lit (input): The lighting is turned OFF if lit == 0 and ON if lit != 0. This setting can be changed by calling the
function setLighting after the list is allocated.

Return Value: The number of the new display list.

Remark: Lighting is OFF by default.

Author: WDH & AP

11.36 getNewLabelList

int
getNewLabelList(int win = -1)

Description: Use this function to get an unused list for labels (non-rotatable).

Return Value: The number of the new display list.

Author: WDH

11.37 deleteList

void
deleteList(int dList)

Description: Delete one display list in the current window

Return Value: None.

Author: AP

11.38 getGlobalBound

RealArray
getGlobalBound() const

Description: return a copy of the global bounds in the current graphics window.

return value: globalBound(0:1,0:2): current global bounds.

author: WDH

11.39 hardCopy (save a Postscript File)

int
hardCopy(const aString & fileName =nullString,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
**   int win_number =-1)**

Description: This routine saves the contents of one graphics window in hard-copy form. If off-screen rendering is available (MESA) then use it, unless the hardCopyType is set to GraphicsParameters::postScriptRaster.

fileName (input): Optional name for the file to save the plot in. If no name is given then the user is prompted for a name.

hardCopyType (input): GraphicsParameters::postScriptRaster, or GraphicsParameters::postScript.

win_number (input): The number of the window to save. If that argument is omitted, the contents of the current window is saved .

Return Values: 1: unable to open the file.

Author: WDH & AP

11.40 outputString

void
outputString(const aString & message, int messageLevel =2)

Description: Output a string in the prompt sub-window in the command window. If the echo file is open, also output the string in that file.

message (input): the string to be output.

messageLevel (input) : output the string if messageLevel is less than or equal to the current value for infoLevel. Values for infoLevel are 0=expert, 1=intermediate, 2=novice.

Return Value: none.

Author: AP

11.41 erase

void
erase()

Description: Erase the current graphics window. Shorthand for erase(getCurrentWindow(), false);

Author: AP

11.42 erase (win_number)

void
erase(const int win_number, bool forceDelete = false)

Description: Erase the contents in one graphics window. win_number(input): window number forceDelete(input): If true, delete all display lists associated with this window. If false, delete the display lists that are not hidable and don't plot the hidable lists.

Author: WDH & AP

11.43   erase (IntegerArray)

void
erase(const IntegerArray & displayList)

Description: Erase some display lists in the current graphics window.

displayList (input) : an array of display lists to delete, all values should be non-negative.

Author: WDH

11.44   inputFileName

void
inputFileName(aString & fileName, const aString & prompt, const aString &extension =nullString)

Description: Open a Motif file selection dialog window and prompt for a file name.

fileName (output): an aString with the selected file name.

prompt (input): an aString with the prompt that will be displayed in the text sub-window of the command window.

extension (input): an aString with the extension of the files that will be displayed when the file selection dialog is opened.

Return Value: none.

Author: AP

11.45   inputString

void
inputString(aString & answer, const aString & prompt)

Description: Output a prompt and wait for an answer.

answer (output): a aString with the answer.

prompt (input): a aString with the prompt that will be displayed in the text sub-window of the command window (if the GUI is active).

Return Value: none.

Author: WDH

11.46   redraw

void
redraw(bool immediate)

Description: Redraw all graphics display lists in the current window.

immediate(input): If true, force an immediate redraw. Otherwise, post a redraw event, in which case the window will be redrawn next time the application asks for a user input.

11.47   resetGlobalBound

void
resetGlobalBound(const int win_number)

Description: Reset the global bounds to represent no bounds at all.

11.48 setGlobalBound

void
setGlobalBound(const RealArray & xBound)

Description: Set the global bounds for plotting. These values will only increase the size of the current bounds.

xBound(0: 1,0:2) (input) : global bounds should be at least this large.

11.49 setKeepAspectRatio

int
setKeepAspectRatio(bool trueOrFalse =true)

Description: If "true", keep the aspect ratio of plots.

Author: WDH & AP

11.50 getWindowShape

void
getWindowShape(int window, real & leftSide_ , real & rightSide_ , real & top_ , real & bottom_)
const

Description: Return the shape of the window.

11.51 getLineWidthScaleFactor

real
getLineWidthScaleFactor(int window = -1)

Description: Return the scale factor for line widths.

11.52 displayColourBar

void
displayColourBar(const int & numberOfContourLevels,
**   RealArray & contourLevels,**
**   real uMin,**
**   real uMax,**
**   GraphicsParameters & parameters)**

Description: Display the colour bar.

11.53 updateColourBar

void
updateColourBar(GraphicsParameters & parameters, int window =0)

update the colour bar.

11.54 setLighting

void
setLighting(int list, bool lit)

Description: Use this function to turn on or off lighting in a display list in the currentWindow. Note that each display list can only be completely lit or unlit. If you need to display both lit and unlit objects, you need to split the plotting into two display lists.

list (input): The number of the existing display list. $1 <=$ list $<$ getMaxNOfDL(currentWindow). This number is for example returned by the function generateNewDisplayList.

lit (input): The lighting is turned OFF if lit $==0$ and ON if lit $!=0$.

Author: AP

11.55   setPlotDL

void
setPlotDL(int list, bool plot)

Description: Use this function to turn on or off plotting of a display list in the currentWindow.

list (input): The number of the existing display list. getFirstUserRotableDL(currentWindow) $<=$ list $<$ get-MaxNOfDL(currentWindow). The number of the display list is for example returned by the function generateNewDisplayList.

plot (input): The plotting of the display list is turned OFF if plot $==$ false and ON if plot $==$ true .

Author: AP

11.56   setInteractiveDL

void
setInteractiveDL(int list, bool interactive)

Description: Use this function to turn on or off interactive plotting of this display list during rotations in the currentWindow.

list (input): The number of the existing display list. getFirstUserRotableDL(currentWindow) $<=$ list $<$ get-MaxNOfDL(currentWindow). The number of the display list is for example returned by the function generateNewDisplayList.

interactive (input): Interactive plotting of the display list is turned OFF if plot $==$ false and ON if plot $==$ true
.

Author: AP

11.57   initView

void
initView(int win_number/*=-1*/)

Initialize the view and rotation point to default values

win_number(optional input): window number

Return value: None

Author: AP

11.58   resetView

void
resetView(int win_number /*=-1*/)

Reset the view point (but not the rotation point)

win_number(optional input): window number

Return value: None

Author: AP

11.59   chooseAColour

aString
chooseAColour()

Description: Choose a colour from a menu.

Return value: The name of the colour.

Author: AP

11.60   setColour

int
setColour(const aString & nameIn)

set the colour for subsequent objects that are plotted

Return value: 0 means success, 1 means failure

11.61   setColour

int
setColour(ItemColourEnum item)

Description: Set colour to default for a given type of item

11.62   getColour

aString
getColour(ItemColourEnum item)

Description: Get the name of the colour for backGroundColour, textColour, ...

11.63   setColourName

void
setColourName(int i, aString newColourName) const

Description: Assign the name of colour i in the list of colours.

11.64   getColourName

aString
getColourName(int i) const

Description: Return the name of colour i in the list of colours.

11.65   setAxesLabels

int
setAxesLabels(const aString & xAxisLabel_ = blankString,
**   const aString & yAxisLabel_ = blankString,**
**   const aString & zAxisLabel_ = blankString)**

Description: Set labels on the coordinate axes. The labels will be plotted in the currentWindow next time the screen is updated.

xAxisLabel_: The label on the x-axis.

yAxisLabel_: The label on the y-axis.

zAxisLabel_: The label on the z-axis.

Return values: none.

Author: WDH

11.66 setLineWidthScaleFactor

void
setLineWidthScaleFactor(const real & scaleFactor =1. */, int win_number /* = -1)

Description: Set scale factor for line widths (this can be used to increase the line widths for high-res off screen rendering.

Author: WDH & AP

11.67 normalizedToWorldCoordinates

int
normalizedToWorldCoordinates(const RealArray & r, RealArray & x) const

Description: Convert normalized coordinates [-1,+1] to world coordinates

r(i,0: 2) (input) : points to convert.

x(i,0: 2) (output) : converted points. (x and r can be the same array).

11.68 worldToNormalizedCoordinates

int
worldToNormalizedCoordinates(const RealArray & x, RealArray & r) const

Description: Convert world coordinate to normalized coordinates [-1,+1]

x(i,0: 2) (input) : points to convert.

r(i,0: 1) (output) : converted points. (x and r can be the same array).

11.69 setView

void
setView(const ViewParameters & viewParameter, const real & value)

Description: set some view parameters. The change will take effect the next time the view is updated (with a call to redraw for example).

viewParmeter (input): indicate which parameter to change:

```
enum ViewParameters
  {
    xAxisAngle,      // angle to rotate about x-axis (absolute value, not incremental)
    yAxisAngle,
    zAxisAngle,
    xTranslation,
    yTranslation,
    zTranslation,
    magnification
  }
```

value (input) : change the parameter to this value.

Note: setting one of the angle parameters (xAxisAngle, yAxisAngle or zAxisAngle) will cause the current rotation matrix to be reset to the identity. One or more angle parameters can be changed and the changes will take effect the next time the view is updated.

Author: WDH & AP

11.70   pollEvents

void
pollEvents()

Description: Process all current events. Exit when there are no more events. Note that this function is very similar to the internal event loop in mogl.C, except that this function is non-blocking. Note that this routine can be called from anywhere in an application code to update the windows, parse any pending commands, etc. This might for instance be useful during a long computation.

Return values: none.

Author: AP

11.71   setUserButtonSensitive

void
setUserButtonSensitive(int btn, int trueOrFalse)

Set the sensitivity (on/off) of push button number "btn" on the bottom of the graphics window.

btn(input): Set the sensitivity of this button

trueOrFalse(input): Turn the button on or off (grayed out).

Author: AP

11.72   pushGUI

void
pushGUI(GUIState &newState)

Description: Push newState onto the top of the internal GUIState stack and change the menus, buttons and dialog window according to newState.

newState(input): The description of the context and layout of the new menus, buttons and dialog window. See the GUIState function descriptions for an explaination of how to set the context of a GUIState object.

Author: AP

11.73   popGUI

void
popGUI()

Description: Pop the internal GUIState stack and restore menus, buttons and the dialog window according to the previous state.

Author: AP

11.74   createMessageDialog

void
createMessageDialog(aString msg, MessageTypeEnum type)

Description: Open a dialog window with a message and a close button. The dialog window only appears if the graphical user interface is opened and commands are NOT beeing read from a command file.

msg(input): The text string with the message. Newline characters '\n' indicate line breaks.

type(input): The type of dialog window to open. The type determines the symbol and the title of the dialog window. Can have the following values:

```
enum MessageTypeEnum
{
  errorDialog,
  warningDialog,
  informationDialog,
  messageDialog // No symbol
};
```

Return values: None

Author: AP

11.75   appendCommandHistory

void
appendCommandHistory(const aString &answer)

Description: Write a string in the command history window.

answer(input): String to be written.

Return values: None

Author: AP

11.76   beginRecordDisplayLists

int
beginRecordDisplayLists(IntegerArray & displayLists)

Description: Record the display list numbers that are allocated from now until a call to endRecordDisplayLists. This list could be used, for example, to selectively delete items that were drawn.

displayLists (input) : save display list numbers in this array. The array will be automatically redimensioned to hold the numbers.

11.77   endRecordDisplayLists

int
endRecordDisplayLists(IntegerArray & displayLists)

Description: Stop recording the display list numbers.

displayLists (output) : The same array passed when calling beginRecordDisplayLists. On output this array will hold the display list numbers, it will be exactly the correct size.

11.78   pause

int
pause()

Description: Pause and wait for a response: "continue" or "break"

Author: WDH

11.79   drawColouredSquares

void
drawColouredSquares(const IntegerArray & numberList AP changed to a reference,
GraphicsParameters & parameters,
const int & numberOfColourNames_ = -1,
aString *colourNames_ = NULL)

Description: Draw a coloured square with the number inside it for each of the colours shown on the plot

Input - numberList : a list of numbers that should be labeled. The numbers may appear more than once in the list and they need not be ordered

Author: WDH

11.80   label: plot a aString in normalized coordinates

void
label(const aString & string,
real xPosition,
real yPosition,
real size =.1,
int centering =0 ,
real angle =0.,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters,
const aString & colour =nullString,
real zOffset =.99)

Description: This routine plots a label in the normalized coordinate system where the screen has dimensions [-1,1]x[-1,1]. This label does NOT rotate or scale with the plot.

string (input): aString to draw.

xPosition (input): x coordinate of the string in normalized coordinates, [-1,1]. (See the centering argument).

yPosition (input): y coordinate of the string in normalized coordinates, [-1,1]. (See the centering argument).

size (input): Size of the characters in normalized coordinates (size=2.0 would fill the whole view).

centering (input): centering=0 means put the centre of the string at (xPosition,yPosition). centering=-1 means put the left end of the string at (xPosition,yPosition). centering=+1 means put the right end of the string at (xPosition,yPosition).

angle (input): Angle in degrees to rotate the string.

colour (input): optionally specify a colour for the text.

zOffset (input): by default raise the label so that it is not covered by the plot (NOTE: the front clip plane is at z=1)

Errors: none (Ha).

Return Values: none.

Author: WDH

11.81   xlabel: plot a aString in 2D world coordinates

void
xLabel(const aString & string,
const real xPosition,
const real yPosition,
const real size =.1,

```
        const int centering =0 ,
        const real angle =0.,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
        int win_number = -1)
```

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xLabel plots the label in the $z = 0$ plane.

string (input): aString to draw.

xPosition (input): x coordinate of the string in world coordinates. (See the centering argument).

yPosition (input): y coordinate of the string in world coordinates. (See the centering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): centering=0 means put the centre of the string at (xPosition,yPosition). centering=-1 means put the left end of the string at (xPosition,yPosition). centering=+1 means put the right end of the string at (xPosition,yPosition).

angle (input): Angle in degrees to rotate the string.

Errors: none.

Return Values: none.

Author: WDH

11.82   xlabel: plot a aString in 3D world coordinates

```
void
xLabel(const aString & string,
        const real x[3],
        const real size =.1,
        const int centering =0 ,
        const real angle =0.,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
        int win_number = -1)
```

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xLabel plots the label in the $z = 0$ plane.

string (input): aString to draw.

x (input): x(0:2) 3D coordinates of the string in world coordinates. (See the centering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): centering=0 means put the centre of the string at (xPosition,yPosition). centering=-1 means put the left end of the string at (xPosition,yPosition). centering=+1 means put the right end of the string at (xPosition,yPosition).

angle (input): Angle in degrees to rotate the string.

Errors: none.

Return Values: none.

Author: WDH

11.83   xlabel: plot a aString in 3D world coordinates

void
xLabel(const aString & string,
         const RealArray & x,
         const real size =.1,
         const int centering =0 ,
         const real angle =0.,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
         int win_number = -1)

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xLabel plots the label in the $z = 0$ plane.

string (input): aString to draw.

x (input): x(0:2) 3D coordinates of the string in world coordinates. (See the centering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): `centering=0` means put the centre of the string at (xPosition,yPosition). `centering=-1` means put the left end of the string at (xPosition,yPosition). `centering=+1` means put the right end of the string at (xPosition,yPosition).

angle (input): Angle in degrees to rotate the string.

Errors: none.

Return Values: none.

Author: WDH

11.84   xlabel: plot a aString in 3D world coordinates

void
xLabel(const aString & string,
         const RealArray & x,
         const real size,
         const int centering,
         const RealArray & rightVector,
         const RealArray & upVector,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
         int win_number = -1)

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xLabel plots the string in the plane formed by the vectors `rightVector` and `upVector`.

string (input): aString to draw.

x(0: 2) (input): x,y,z coordinates of the string in world coordinates. (see the centering entering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): `centering=0` means put the centre of the string at `x`, `centering=-1` means put the left end of the string at `x`. `centering=+1` means put the right end of the string at `x`.

rightVector(0: 2) (input): The string is drawn to lie parallel to this vector.

upVector(0: 2) (input): This vector defines the "up" direction for the characters. The characters are drawn in the plane defined by the rightVector and the upVector.

Errors: none.

Return Values: none.

Author: WDH

11.85   xlabel: plot a aString in 3D world coordinates

void
xLabel(const aString & string,
  const real x[3],
  const real size,
  const int centering,
  const real rightVector[3],
  const real upVector[3],
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
  int win_number = -1)

Description: This routine plots a label with position and size in World coordinates, This label DOES rotate and scale with the plot. This version of xLabel plots the string in the plane formed by the vectors `rightVector` and `upVector`.

string (input): aString to draw.

x(0: 2) (input): x,y,z coordinates of the string in world coordinates. (see the centering entering argument).

size (input): Size of the characters in NORMALIZED coordinates.

centering (input): `centering=0` means put the centre of the string at `x`, `centering=-1` means put the left end of the string at `x`. `centering=+1` means put the right end of the string at `x`.

rightVector(0: 2) (input): The string is drawn to lie parallel to this vector.

upVector(0: 2) (input): This vector defines the "up" direction for the characters. The characters are drawn in the plane defined by the rightVector and the upVector.

Errors: none.

Return Values: none.

Author: WDH

11.86   plotLabels

void
plotLabels(GraphicsParameters & parameters,
  const real & labelSize =-1.,
  const real & topLabelHeight =.925,
  const real & bottomLabelHeight =-.925,
  int win_number = -1)

Description: Plot labels.

labelSize (input) : if ¡= 0 use default in parameters Utility routine used to plot labels from a GraphicParameters

Return Values: none.

Author: WDH

11.87   eraseLabels

void
eraseLabels(GraphicsParameters & parameters, int win_number /* = -1*/)

Description: Erase the labels. Utility routine used to erase title labels.

Return Values: none.

Author: WDH

11.88   plotAxes

void
plotAxes(const RealArray & xBound_,
          const int numberOfDimensions,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
          int win_number = -1)

Description: This routines generates the display list for plotting axes along the specified bounds, for the given number of space dimensions. The actual plotting is done by the display call-back function, which is activated by calling the function redraw().

xBound(0: 1,0:2) (input): Bounds to use for the axes. xBound(Start,axis1), xBound(End,axis1), ...

numberOfDimensions (input): Number of space dimensions. This determines how many axes to draw.

parameters (input): Specification of the graphics parameters (line width, etc.)

win_number (input): The number of the window where the axes should be plotted. If win_number==-1, or it is omitted, the axes will be plotted in the currentWindow.

Return Values: none.

Author: WDH & AP

11.89   eraseAxes

void
eraseAxes(int win_number)

Description: Erase the display lists holding the axes in window 'win_number'.

Author: WDH & AP

11.90   setCurrentWindow

void
setColourFromTable(const real value, GraphicsParameters & parameters)

Set the colour from a table defined in the GraphicsParameters.

value (input) : real value in the range [0,1]

parameters (input) : holds the colour table to use

Return value: none.

Author: WDH

11.91   psToRaster

int
psToRaster(const aString & fileName,
          const aString & ppmFileName)

Description: Convert a RLE compressed .ps file from PlotStuff into a ppm raster

Author: WDH

11.92 plotPoints

void
plotPoints(const realArray & points, GraphicsParameters & parameters =
defaultGraphicsParameters(),
          int dList = 0)

Description: Plot points. Plot an array of points in 2D or 3D.

points (input) : an array of the form: points(0:n-1,0:r-1) where n is the number of points and r is the range
    dimension.

parameters (input/output): supply optional parameters to change plotting characteristics.

Errors: There are no known bugs...

Return Values: none.

Author: WDH

11.93 plot points or spheres with individual colours (or radii)

void
plotPoints(const realArray & points_,
          const realArray & value,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
          int dList = 0)

Description: Plot points (or spheres) and colour each point a different colour based on the value array.

points (input) : an array of the form: points(0:n-1,0:r-1) where n is the number of points and r is the range
    dimension.

value (input) : an array of values, value(0:n-1,0), that will determine the colour (and optionally the sphere radius)
    for each point (or sphere). The colour will be taken from a colour table with the colour table value for point i
    based on the scaled quantity v(i) = (value(i,0)-min(value))/(max(value)-min(value)).

    If the value array is dimensioned value(0:n-1,0:1) then value(i,1) will denote the radius of the sphere that should
    be plotted with center given by the points array.

parameters (input/output): supply optional parameters to change plotting characteristics.

Errors: Some...

Return Values: none.

Author: WDH

11.94 plotLines

void
plotLines(const realArray & arrows,
GraphicsParameters & parameters =Overture::defaultGraphicsParameters(),
          int dList /*= 0*/)

Description: Plot line segments

arrows(input): array holding the coordinates of the start and end points for each line segment. It should be
    dimensioned array(0:Npoints-1, 0:rangeDimension-1, 0:1), where the last index is 0 for the start point and 1
    for the end point.

parameters(input): Graphics parameters controlling the plot.

dList(optional input): If provided, put the drawing command in this display list.

Author: WDH

11.95   eraseColourBar

void
eraseColourBar()

Description: Erase the colour bar.

11.96   drawColourBar

void
drawColourBar(const int & numberOfContourLevels,
　　　　　　RealArray & contourLevels,
　　　　　　real uMin,
　　　　　　real uMax,
　　　　　　GraphicsParameters & parameters,
　　　　　　real xLeft =.775.8,
　　　　　　real xRight =.825.85,
　　　　　　real yBottom =-.75,
　　　　　　real yTop =.75)

Description: Draw the colour Bar *** this is the old version ***

numberOfContourLevels (input): put this many labels on the colour bar

contourLevels (input) : if not null, this array species the contour levels uMin,uMax : these values determine the labels

xLeft, xRight, xBotton, xTop (input): position of colour bar in normalized coordinates, [-1,1]

11.97   getKeepAspectRatio

virtual bool
getKeepAspectRatio()return keepAspectRatio[currentWindow];;

Description: Return true if the aspect ratio is preserved in the current window, otherwise return false.

Author: AP

11.98   getAnswer

int
getAnswer(aString & answer, const aString & prompt)

Description: Wait for an answer to be issued by the user. If the program is reading commands from the GUI, this routine will return after the user has issued a command from the popup menu, the pulldown menu or the push buttons on the graphics window, or from any of the buttons or menus in the dialog window. If the program is reading commands from a file, the next line of the file not starting with '*' or a '#' is returned.

See the routines pushGUI and popGUI as well as the functions in the GUIState class for instructions on how to setup the graphical user interface (GUI).

answer(output): The string issued by the GUI or read from the command file.

prompt(input): A prompt used by the GUI.

Return Values: On return, "answer" is set equal to the menu item chosen. The function return value is set equal to the number of the item chosen, starting from zero. The items are the union of the popup menus, pulldown menus, buttons on graphics windows, and all items on the current dialog window.

Author: AP

11.99   getAnswer with selection

int
getAnswer(aString & answer, const aString & prompt, SelectionInfo &selection)

Description: In addition to the functionality in the basic getAnswer() function, this routine can also return a selection that is made by the user, or read from the command file.

The SelectionInfo object contains the following information:

```
class SelectionInfo
{
public:
IntegerArray selection;
int nSelect;
int active;
real r[4];
real zbMin;
real x[3];
int globalID;
int winNumber;
};
```

If the user picks a point or a region anywhere in a graphics window (with the left mouse button while holding down the CONTROL key), `active` will be set to 1 and the window coordinates will be saved in `r[4]` according to r[0]: rMin (horizontal window coordinate), r[1]: rMax, r[2]: sMin (vertical window coordinate), r[3]: sMax.

If the pick was made on one or several objects, the closest z-buffer value is stored in `zbMin` and the corresponding 3–D coordinates are saved in `x[3]`. The global ID number of the closest object is saved in `globalID` and the window number where the picking occured is saved in `winNumber`.

Furthermore, `nSelect` contains the number of objects that were selected and the array selection(nSelect,3) will contain information about what was selected:

selection(i,0): globalID of object # i,

selection(i,1): front z-buffer value of object # i,

selection(i,2): back z-buffer value of object # i.

Note that `active` will be 1 and `nSelect`=0 if the user picks outside all objects in the graphics window.

Author: AP

11.100   getAnswerNoBlock

int
getAnswerNoBlock(aString & answer, const aString & prompt)

Description: This is a non-blocking version of getAnswer(), i.e., if no events are pending, it will return without any answer. Note that if a command file is open, this routine works in the same way as the standard (blocking) getAnswer(), except when the file ends. In that case this routine will only return an answer if an event was pending before the file ended.

See the routines pushGUI and popGUI as well as the functions in the GUIState class for instructions on how to setup the graphical user interface (GUI).

answer(output): The string issued by the GUI or read from the command file.

prompt(input): A prompt used by the GUI.

Return Values: If no button, menu, popup, etc., was chosen since last time a getAnswer routines was called, **answer** will be set to "" and the return value will be 0. Otherwise, "answer" is set equal to the return string assigned by the callback function. The function return value is set equal to the number of the item chosen, starting from zero. The items are the union of the popup menus, pulldown menus, buttons on graphics windows, and all items on the current dialog window.

Author: AP

11.101   pickPoints

int
pickPoints(RealArray & x,
            bool plotPoints = TRUE,
            int win_number = -1)

Description: Pick points in 2D or 3D by clicking the mouse. In 3D one should click on an object. If multiple objects are 'hits' then the closest one is chosen.

x (input/output) : On input x should be dimensioned x(a:b,0:1) for 2D picks or x(a:b.,0:2) for 3d picks. At most (b-a+1) points will be chosen. The actual number chosen is the return value.

plotPoints (input): Specifies whether the picked points should be plotted on the screen.

win_number (input): The window number in which the picking should occur. If omitted, the currentWindow is used.

Return value: the number of points chosen.

Author: WDH & AP

11.102   setPlotTheAxes

void
setPlotTheAxes(bool newState, int win_number =-1)

Description: Toggle plotting of coordinate axes on or off.

newState(input): Toggle on (true) or off (false).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.103   getPlotTheAxes

bool
getPlotTheAxes(int win_number =-1)

win_number(optional input): window number. If absent, use the current window.

Return value: true if axes are plotted, otherwise false.

Author: AP

11.104   setAxesDimension

void
setAxesDimension(int dim, int win_number =-1)

Description: Set the dimensionality of the plotted coordinate axes.

dim(input): The dimensionality (1-3).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.105   setPlotTheLabels

void
setPlotTheLabels(bool newState, int win_number = -1)

Description: Toggle plotting of the label on the graphics window.

newState(input): Toggle on (true) or off (false).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.106   getPlotTheLabels

bool
getPlotTheLabels(int win_number =-1)

win_number(optional input): window number. If absent, use the current window.

Return value: true if labels are plotted, otherwise false.

Author: AP

11.107   setPlotTheRotationPoint

void
setPlotTheRotationPoint(bool newState, int win_number = -1)

Description: Toggle plotting of the rotation point in the graphics window.

newState(input): Toggle on (true) or off (false).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.108   getPlotTheRotationPoint

bool
getPlotTheRotationPoint(int win_number =-1)

win_number(optional input): window number. If absent, use the current window.

Return value: true if the rotation point is plotted, otherwise false.

Author: AP

11.109 setPlotTheColourBar

void
setPlotTheColourBar(bool newState, int win_number = -1)

Description: Toggle plotting of the colour bar in the graphics window.

newState(input): Toggle on (true) or off (false).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.110 getPlotTheColourBar

bool
getPlotTheColourBar(int win_number =-1)

win_number(optional input): window number. If absent, use the current window.

Return value: true if the colour bar is plotted, otherwise false.

Author: AP

11.111 setPlotTheColouredSquares

void
setPlotTheColouredSquares(bool newState, int win_number = -1)

Description: Toggle plotting of the coloured squares (grid labels) in the graphics window.

newState(input): Toggle on (true) or off (false).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.112 getPlotTheColouredSquares

bool
getPlotTheColouredSquares(int win_number =-1)

win_number(optional input): window number. If absent, use the current window.

Return value: true if the coloured squares (grid labels) are plotted, otherwise false.

Author: AP

11.113 setPlotTheBackgroundGrid

void
setPlotTheBackgroundGrid(bool newState, int win_number = -1)

Description: Toggle plotting of the background grid in the graphics window. Currently only implemented for one and two dimensional plots.

newState(input): Toggle on (true) or off (false).

win_number(optional input): window number. If absent, use the current window.

Return value: None

Author: AP

11.114   getPlotTheBackgroundGrid

bool
getPlotTheBackgroundGrid(int win_number =-1)

win_number(optional input): window number. If absent, use the current window.

Return value: true if the background grid is plotted, otherwise false.

Author: AP

11.115   getMenuItem

int
getMenuItem(const aString *menu, aString & answer, const aString & prompt /*=nullString*/)

OBSOLETE function used before all old calls to getMenuItem have been replaced by getAnswer() OBSOLETE function used before all old calls to getMenuItem have been replaced by getAnswer() OBSOLETE function used before all old calls to getMenuItem have been replaced by getAnswer()

Description: Setup a popup menu and wait for a reply.

menu (input): The `menu` is an array of Strings (the menu choices) with an empty aString indicating the end of the menu choices. Optionally, a title can be put on top of the menu by starting the first aString with an '!'. For example,

```
GL_GraphicsInterface ps;
aString menu[] = { "!MenuTitle",
                   "plot",
                   "erase",
                   "exit",
                   "" };
aString menuItem;
int i=ps.getMenuItem(menu,menuItem);
```

To create a cascading menu, begin the string with an '>'. To end the cascade begin the string with an '<'. To end a cascade and start a new cascade, begin the string with '<' followed by '>'. Here is an example:

```
char *menu1[] = {  "!my title",
                   "plot",
                   ">component",
                           "u",
                           "v",
                           "w",
                   "<erase",
                   ">stuff",
                           "s1",
                           ">more stuff",
                                   "more1",
                                   "more2",
                           "<s2",
                   "<>apples",
                           "apple1",
                   "<exit",
                   NULL };
```

answer (output): Return the chosen item.

prompt (input): display the optional prompt message

Return Values: On return "answer" is set equal to the menu item chosen. The function return value is set equal to the number of the item chosen, starting from zero. Thus, for example, in the above menu if the user picked "erase" the return value would be 2, if the user picked "plot" the return value would be 1, since the title also counts.

Author: AP

12 GUIState Function Descriptions

Objects of the class GUIState describe the context and layout of the Graphical User Interface (GUI). The GUI can have a popup menu, pulldown menus and push buttons in the graphics windows as well as several dialog windows. The class GUIState is derived from the class DialogData, which handles dialog windows. Each GUIState object has one main dialog window, but can also have several sibling dialog windows. The sibling windows can, for example, be used when there are more options than what fits in the main dialog window.

   The GenericGraphicsInterface stores the GUIStates on a stack to help support a hierarchical GUI, which is commonly used throughout the Overture class library. A hierachical GUI is organized in a tree-like structure which can change appearance as commands are given. For example, the GUI changes in the grid generator 'ogen' when the create mappings command is given. When the user is done creating mappings, the GUI changes back to the original appearance.

   The following example illustrates how the GUIState class can be used.

```
  ...
  GenericGraphicsInterface & ps = *Overture::getGraphicsInterface("GUI test program");
  GUIState interface;// create a GUI object
  interface.setWindowTitle("DIA test dialog"); // window title
  interface.setExitCommand("exit", "Exit"); // set the exit command

// specify toggle buttons
  aString tbCommands[] = {"plot the object and exit", "render directly", ""};
  aString tbLabels[] = {"Plot and exit", "render directly", ""};
  int tbState[] = {1,0}; // 1 means on, 0 off
  interface.setToggleButtons(tbCommands, tbLabels, tbState, 2); // organize in 2 columns

// first option menu
  aString opCommand0[10]; aString opLabel0[10];
// assign opCommand0 and opLabel0...
  interface.addOptionMenu( "Mapping to plot", opCommand0, opLabel0, 0);

// make a popup menu
  aString menu[] = { "!DIA test program", "plot points", "file output", "" };
  interface.buildPopup(menu);

// make window buttons
  aString wbuttons[][2] = {{"plot points with colour", "Colour pnt"}, {"", ""}  };
  interface.setUserButtons(wbuttons);

// make a window pulldown menu
  aString pulldown[] = {"enter 2D points", ""};
  interface.setUserMenu(pulldown, "DIA test");

// bring up the new GUI on the screen and disable any previous GUI
  ps.pushGUI( interface );

  aString answer;
  for(;;)
  {
    ps.getAnswer(answer, "Dia test>");
// process answer...
  }
// remove the GUI from the screen and restore any previous GUI
  ps.popGUI();
```

12.1 Constructor

GUIState()

Description: Default constructor.

Author: AP

12.2   setUserMenu

void
setUserMenu(const aString menu[], const aString & menuTitle)

Description: Sets up a user defined pulldown menu in the graphics windows. The menu will appear after the routine pushGUI() has been called.

menu (input): The `menu` is an array of aStrings (the menu choices) with an empty aString indicating the end of the menu choices. If menu == NULL, any existing user defined menu will be removed.

menuTitle (input): A aString with the menu title that will appear on the menu bar. Note that a menuTitle must be provided even when menu == NULL.

Return value: none.

Author: AP

12.3   setUserButtons

void
setUserButtons(const aString buttons[][2])

Description: This function builds user defined push buttons in the graphics windows. The buttons will appear after the routine pushGUI() has been called.

buttons (input): A two-dimensional array of Strings, terminated by an empty aString. For example,

```
aString buttons[][2] = {{"plot shaded surfaces", "Shade"},
                        {"erase",                "Erase"},
                        {"exit",                 "Exit"},
                        {"",                     ""}};
```

The first entry in each row is the aString that will be passed as a command when the button is pressed. The second aString in each row is the name of the button that will appear on the graphics window. There can be at most MAX_BUTTONS buttons, where MAX_BUTTONS is defined in mogl.h, currently to 15.

If buttons == NULL, any existing buttons will be removed from the current window.

Return value: none.

Author: AP

12.4   buildPopup

void
buildPopup(const aString menu[])

Description: Sets up a user defined popup menu in all graphics windows and in the command window. The menu will appear after the routine pushGUI() has been called.

menu (input): The `menu` is an array of Strings (the menu choices) with an empty aString indicating the end of the menu choices. Optionally, a title can be put on top of the menu by starting the first aString with an '!'. For example,

```
        PlotStuff ps;
        aString menu[] = { "!MenuTitle",
                           "plot",
                           "erase",
                           "exit",
                           "" };
        aString menuItem;
        int i=ps.getMenuItem(menu,menuItem);
```

To create a cascading menu, begin the string with an '>'. To end the cascade begin the string with an '<'. To end a cascade and start a new cascade, begin the string with '<' followed by '>'. Here is an example:

```
 char *menu1[] = {  "!my title",
                    "plot",
                    ">component",
                              "u",
                              "v",
                              "w",
                    "<erase",
                    ">stuff",
                            "s1",
                            ">more stuff",
                                      "more1",
                                      "more2",
                            "<s2",
                    "<>apples",
                            "apple1",
                    "<exit",
                    NULL };
```

Return value: none.

Author: AP

12.5   getDialogSibling

DialogData &
getDialogSibling(int number =-1)

Description: If number==-1 (default), allocate a sibling (dialog) window, otherwise return sibling # 'number'. Note that the sibling window will appear on the screen after pushGUI() has been called for this (GUIState) object and showSibling() has been called for the DialogData object returned from this function. See the DialogData function description for an example.

number (input): by default return a new sibling (if number==-1), otherwise return the sibling specified by number.

Returnvalues: The function returns an alias to the DialogData object. There is currently space for 10 siblings (0,...,9) for each GUIState object.

Author: AP

13 DialogData Function Descriptions

The class DialogData handles the layout and creation of dialog windows. A dialog window can contain push buttons, toggle buttons, text labels (for inputting strings), option menus, radio boxes and pulldown menus. The pulldown menus can consist of either push buttons or toggle buttons.
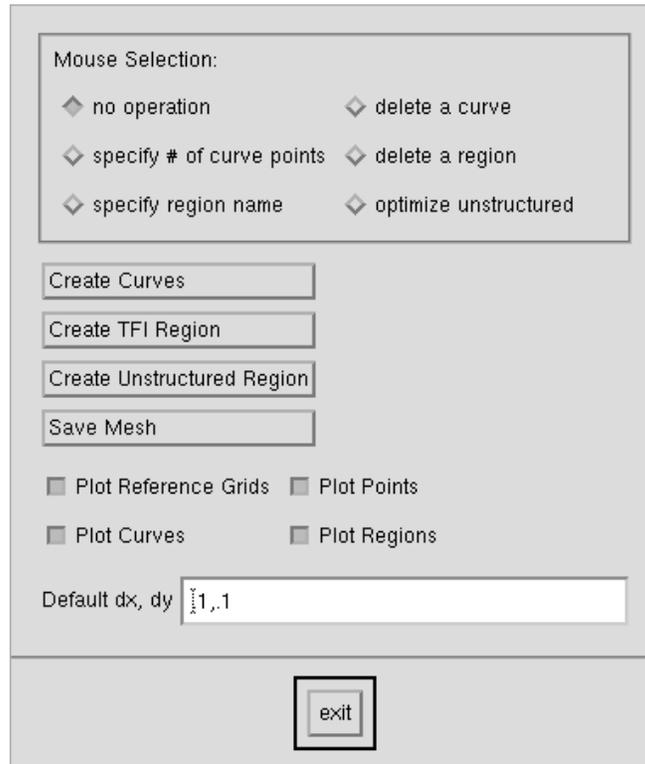
Figure 16: A dialog window.

DialogData is the base class for GUIState. Objects from DialogData can only be made through access functions in the GUIState class.

The widgets in the DialogData class are handled by small classes that only provide some basic functionality, typically for setting the sensitivity or the state of the widget. These classes are described in sections 14.1-14.16. We mention in passing that a widget is insensitive when it is grayed out and cannot be interacted with. The state of a ToggleButton determines whether a checkmark is drawn in the square to the left of the ToggleButton.

The following example shows how to set up the main dialog window in a GUIState object as well as initiating one sibling window.

```
  ...
  GenericGraphicsInterface & ps = *Overture::getGraphicsInterface("GUI test program");
  GUIState interface;// create a GUI object
  interface.setWindowTitle("DIA test dialog"); // window title
  interface.setExitCommand("exit", "Exit"); // set the exit command

// specify toggle buttons in the main dialog window
  aString tbCommands[] = {"plot the object and exit", "render directly", "show sibling 1", ""};
  aString tbLabels[] = {"Plot and exit", "render directly", "more options", ""};
  int tbState[] = {1, 0, 0}; // initial state of the toggle buttons
  int siblingToggle=2;  // remember the index of the sibling toggle button
  interface.setToggleButtons(tbCommands, tbLabels, tbState, 2); // organize in 2 columns

// Set up the popup menu, the pulldown menu, push buttons and the rest
// of the main dialog window...
```

```
// make a dialog sibling
  DialogData &ds = interface.getDialogSibling();
  ds.setWindowTitle("Sibling 1"); // Sibling window title
  ds.setExitCommand("close sibling 1", "Close"); // Sibling exit command
// define push buttons in the Sibling window
  aString pbCommands[] = {"clear points", "spline", ""};
  aString pbLabels[] = {"clear points", "Spline", ""};
  ds.setPushButtons( pbCommands, pbLabels, 1 ); // organize buttons in 1 row

// bring up the interface on the screen. The sibling will be hidden initially.
  ps.pushGUI( interface );

  aString answer;
  for(;;){
    ps.getAnswer(answer, "Sibling test");
    if (answer(0,13) == "show sibling 1"){
      int onOff=1;
      sScanF(&answer[14],"%i", &onOff); // read the toggle state
      if (onOff)
        ds.showSibling(); // show the sibling
      else
        ds.hideSibling(); // hide the sibling
    }
    else if (answer == "close sibling 1"){
      ds.hideSibling(); // hide the sibling
// unset the toggle button on the main dialog
      interface.setToggleState( siblingToggle , 0);
    }
// parse all other commands...
  }
  ps.popGUI();
```

13.1   setExitCommand

int
setExitCommand(const aString &exitC, const aString &exitL)

Description: Set the exit command on the dialog window in the GUIState. Note that the dialog window will apear after pushGUI has been called.

exitC(input): The command hat will be issued when the exit button is pressed.

exitL(input): The text label that will appear on the exit button.

Return values: The function returns 1 on a successful completion and 0 if an error occured.

Author: AP

13.2   setToggleButtons

int
setToggleButtons(const aString tbCommands[], const aString tbLabels[], const int initState[],
**   int numberOfColumns = 2)**

Description: Set the toggle buttons of the dialog window in the GUIState. The buttons will appear on the dialog window after pushGUI has been called.

tbCommands(input): Array of strings containing the commands for the toggle buttons. The array must be terminated by an empty string ("").

tbLabels(input): Array of strings containing the text labels that will be put on the toggle buttons. The array must be terminated by an empty string ("").

initState(input): Array that describes the initial state of each toggle button.

numberOfColumns(input): Optional argument that specifies the number of columns in which the toggle buttons shall be organized in the dialog window.

Return values: The function returns 1 on a successful completion and 0 if an error occured.

Author: AP

13.3   deleteToggleButtons

int
deleteToggleButtons()

Description: Delete the toggle buttons.

Author: WDH

13.4   setPushButtons

int
setPushButtons(const aString pbCommands[], const aString pbLabels[], int numberOfRows = 2)

Description: Set the push buttons of the dialog window in the GUIState. The buttons will appear on the dialog window after pushGUI has been called.

pbCommands(input): Array of strings containing the commands for the push buttons. The array must be terminated by an empty string ("").

pbLabels(input): Array of strings containing the text labels that will be put on the push buttons. The array must be terminated by an empty string ("").

numberOfRows(input): Optional argument that specifies the number of rows in which the push buttons shall be organized in the dialog window.

Return values: The function returns 1 on a successful completion and 0 if an error occured.

Author: AP

13.5   setTextBoxes

int
setTextBoxes(const aString textCommands[], const aString textLabels[], const aString initString[])

Description: Set the text boxes of the dialog window in the GUIState. The boxes will appear on the dialog window after pushGUI has been called.

textCommands(input): Array of strings containing the commands for the text boxes. The array must be terminated by an empty string ("").

textLabelsLabels(input): Array of strings containing the text labels that will be put in front of the text boxes. The array must be terminated by an empty string ("").

initString(input): Array of strings containing the initial text that will be put in each text box. The array must be terminated by an empty string ("").

Return values: The function returns 1 on a successful completion and 0 if an error occured.

Author: AP

13.6   addInfoLabel

int
addInfoLabel(const aString & textLabel)

Description: Add a new info label to the dialog window.

textLabel(input): The new text string.

Return code: The number of the new info label in the GUI, or -1 if there was no space left. (There is only space for MAX_INFO_LABELS (=10 by default) in each dialog window.)

Author: AP

13.7   addInfoLabel

int
deleteInfoLabels()

Description: Delete the existing info labels.

Return code: 0

Author: WDH

13.8   setTextLabel

int
setTextLabel(const aString & textLabel, const aString &buff)

Description: Set the text string textlabel with label "textLabel" in the currently active GUIState.

textLabel(input): The label of the text label in the array given to setTextBoxes during setup.

buff(input): The new text string.

Author: WDH

13.9   addOptionMenu

int
addOptionMenu(const aString &opMainLabel, const aString opCommands[], const aString opLabels[], int initCommand)

Description: Add an option menu to the dialog window. The option menu will appear when the dialog window is displayed, i.e., after pushGUI has been called.

opMainLabel(input): The descriptive label that will appear to the left of the option menu on the dialog window.

opCommands(input): An array of strings with the command that will be issued when each menu item is selected. The array must be terminated by an empty string ("").

opLabels(input): An array of strings with the label that will be put on each menu item. The array must be terminated by an empty string ("").

initCommand(input): The index of the initial selection in the opLabels array. This label will appear on top of the option menu to indicate the initial setting.

Return values: The function returns 1 on a successful completion and 0 if an error occured.

Author: AP

13.10   deleteOptionMenus

int
deleteOptionMenus()

Description: Delete all option menus.

Author: WDH

13.11   addRadioBox

bool
addRadioBox(const aString &rbMainLabel, const aString rbCommands[], const aString rbLabels[], int initCommand,
     int columns = 1)

Description: Add a radio box to the dialog window. The radio buttons will appear when the dialog window is displayed, i.e., after pushGUI has been called.

rbCommands(input): An array of strings with the command that will be issued when the radio button is pressed. The array must be terminated by an empty string (""").

rbLabels(input): An array of strings with the label that will be put on each radio button. The array must be terminated by an empty string (""").

initCommand(input): The index of the initial selection in the rbLabels array. This radio button will be marked initially.

Return values: The function returns true on a successful completion and false if an error occured.

Author: AP

13.12   addPulldownMenu

int
addPulldownMenu(const aString &pdMainLabel, const aString commands[], const aString labels[], button_type bt,
     int *initState = NULL)

Description: Add a pulldown menu to the dialog window. The pulldown menu will appear when the dialog window is displayed, i.e., after pushGUI has been called. Successive pulldown menus will be stacked from left to right on the menu bar.

pdMainLabel(input): The label that will appear on the menu bar.

commands(input): An array of strings with the command that will be issued when each menu item is selected. The array must be terminated by an empty string (""").

labels(input): An array of strings with the label that will be put on each menu item. The array must be terminated by an empty string (""").

bt(input): The type of buttons in the menu. Can be either GI_PUSHBUTTON or GI_TOGGLEBUTTON.

initState(input): Optional argument that only is used when bt == GI_TOGGLEBUTTON. This argument is an array that specifies the initial state of each toggle buttons. If this argument is absent when bt == GI_TOGGLEBUTTON, no menu items are marked as beeing selected.

Return values: The function returns 1 on a successful completion and 0 if an error occured.

Author: AP

13.13   changeOptionMenu

bool
changeOptionMenu(const aString & opMainLabel, const aString opCommands[], const aString opLabels[], int initCommand)

Description: Change an option menu with a given name

opMainLabel(input): name of the option menu

// /opCommands, opLabels (input);: // /opCommands, opLabels (input);

initCommand (input) :

Return values: None.

Author: wdh

13.14   setWindowTitle

void
setWindowTitle(const aString &title)

Description: Set the title of the dialog window in the GUIState. The title will appear on the dialog window after pushGUI has been called.

title(input): The new title.

Return values: None.

Author: AP

13.15   setOptionMenuColumns

void
setOptionMenuColumns(int columns)

Description: Set the number of columns in which the option menus should be organized on the dialog window.

columns(input): The number of columns.

Return values: None.

Author: AP

13.16   setLastPullDownIsHelp

void
setLastPullDownIsHelp(int trueFalse)

Description: Specify whether the last pulldown menu should appear in the right end of the menu bar on the dialog window, where the help menu often is located.

trueFalse(input): 1 if the last pulldown menu should be placed in the right end. Otherwise, the last pulldown menu is placed just to the right of the second last pulldown menu

Return value: None.

Author: AP

13.17 getPulldownMenu

PullDownMenu&
getPulldownMenu(int n)

Description: return the n'th pull-down menu, $0 \le n < n_{pullDownMenu}$.

Author: WDH

13.18 getPulldownMenu

PullDownMenu&
getPulldownMenu(const aString & label)

Description: Find the pulldown menu with the given main label.

label (input) : the label given to the pulldown

Return values: the pulldown menu with the given main label, return PulldownMenu 0 if the label was not found.

Author: WDH

13.19 getOptionMenu

OptionMenu&
getOptionMenu(int n)

Description: return the n'th option menu, $0 \le n < n_{optionMenu}$.

Author: WDH

13.20 getOptionMenu

OptionMenu&
getOptionMenu(const aString & opMainLabel)

Description: Find the option menu with the given main label.

opMainLabel (input) : the label given to an option menu.

Return values: the OptionMenu with the given main label, return OptionMenu 0 if the label was not found.

Author: WDH

13.21 getRadioBox

RadioBox&
getRadioBox(int n)

Description: return the n'th radio box, $0 \le n < n_{radioBoxes}$.

Author: AP

13.22 getRadioBox

RadioBox&
getRadioBox(const aString & radioLabel)

Description: Find the radio box menu with the given main label.

radioLabell (input) : the label given to an radio box

Return values: the RadioBox with the given label, return RadioBox 0 if the label was not found.

Author: WDH

13.23   getToggleValue

bool
getToggleValue(const aString & answer, const aString & label, bool & target)

Description: If 'answer' requests a change in a toggle state then set 'target' and adjust the toggle state.

Return values: true if answer requested a change in a toggle state, return false oterwise.

Author: WDH

13.24   getTextValue(real)

bool
getTextValue(const aString & answer, const aString & label, const aString & format, real & target)

Description: If 'answer' requests a change in a real text value with label='label' then set 'target' and adjust the text label.

answer (input) : check this answer

label (input): check if answer is of the form "label ..."

target (output) : fill in this value if answer begins with "label"

format (input) : use this format to reset the text label field with a new value.

return value: true if found, false otherwise

Author: WDH

13.25   getTextValue(int)

bool
getTextValue(const aString & answer, const aString & label, const aString & format, int & target)

Description: If 'answer' requests a change in a int text value with label='label' then set 'target' and adjust the text label.

answer (input) : check this answer

label (input): check if answer is of the form "label ..."

target (output) : fill in this value if answer begins with "label"

format (input) : use this format to reset the text label field with a new value.

return value: true if found, false otherwise

Author: WDH

13.26   getTextValue(string)

bool
getTextValue(const aString & answer, const aString & label, const aString & format, aString & target)

Description: If 'answer' requests a change in a string text value with label='label' then set 'target' and adjust the text label.

answer (input) : check this answer

label (input): check if answer is of the form "label ..."

target (output) : fill in this value if answer begins with "label"

format (input) : use this format to reset the text label field with a new value.

return value: true if found, false otherwise

Author: WDH

13.27   constructor

PullDownMenu()

Description: default constructor

13.28   setPullDownMenu

bool
setPullDownMenu(const aString &pdMainLabel, const aString commands[], const aString labels[], button_type bt,
                int *initState = NULL)

Description: Fill in all fields of a pulldown menu object except menupane which will be set to NULL and sensitive which will be set to true. This function can for example be used to setup the optionMenu argument to makeGraphicsWindow.

pdMainLabel(input): The label that will appear on the menu bar.

commands(input): An array of strings with the command that will be issued when each menu item is selected. The array must be terminated by an empty string (""").

labels(input): An array of strings with the label that will be put on each menu item. The array must be terminated by an empty string (""").

bt(input): The type of buttons in the menu. Can be either GI_PUSHBUTTON or GI_TOGGLEBUTTON.

initState(input): Optional argument that only is used when bt == GI_TOGGLEBUTTON. This argument is an array that specifies the initial state of each toggle buttons. If this argument is absent when bt == GI_TOGGLEBUTTON, no menu items are marked as beeing selected.

Return values: The function returns true on a successful completion and false if an error occured.

Author: AP

13.29   setSensitive

void
setSensitive(int trueFalse)

Description: Set the sensitivity of a DialogData object.

trueOrFalse: The new state of the DialogData widget

Return valuse: None

Author: AP & WDH

13.30   setSensitive

void
setSensitive(bool trueOrFalse, WidgetTypeEnum widgetType, int number)

Description: Set the sensitivity of a widget in the DialogData

trueOrFalse (input): set senstive or not

widgetType (input): choose a widget type to assign. One of

```
    enum WidgetTypeEnum
    {
       optionMenuWidget,
       pushButtonWidget,
       pullDownWidget,
       toggleButtonWidget,
       textBoxWidget,
       radioBoxWidget
    };
```

number (input) : set sensitivity for this widget.

13.31   setSensitive

void
setSensitive(bool trueOrFalse, WidgetTypeEnum widgetType, const aString & label)

Description: Set the sensitivity of a widget in the DialogData

trueOrFalse (input): set senstive or not

widgetType (input): choose a widget type to assign. One of

```
    enum WidgetTypeEnum
    {
       optionMenuWidget,
       pushButtonWidget,
       pullDownWidget,
       toggleButtonWidget,
       textBoxWidget,
       radioBoxWidget
    };
```

label (input) : set sensitivity for the widget with this label

13.32   changeOptionMenu

bool
changeOptionMenu(int nOption, const aString opCommands[], const aString opLabels[], int initCommand)

Description: Change the menu items in an option menu after it has been created (by pushGUI)

nOption(input): Change option menu # nOption.

opCommands(input): An array of strings with the command that will be issued when each menu item is selected. The array must be terminated by an empty string (""").

opLabels(input): An array of strings with the label that will be put on each menu item. The array must be terminated by an empty string (""").

initCommand(input): The index of the initial selection in the opLabels array. This label will appear on top of the option menu to indicate the initial setting.

Return values: The function returns true on a successful completion and false if an error occured.

Author: AP

13.33   showSibling

int
showSibling()

Description: Show a sibling (dialog) window that previously was allocated with getDialogSibling() and created with pushGUI().

Returnvalues: The function returns 1 if the sibling could be shown, otherwise 0 (in which case it doesn't exist or already is shown).

Author: AP

13.34   hideSibling

int
hideSibling()

Description: Hide a sibling (dialog) window that previously was allocated with getDialogSibling(), created with pushGUI() and shown with showSibling().

Returnvalues: The function returns 1 if the sibling could be hidden, otherwise 0 (in which case it doesn't exist or already is hidden).

Author: AP

13.35   setTextLabel

int
setTextLabel(int n, const aString &buff)

Description: Set the text string in textlabel # n in the currently active GUIState.

n(input): The index of the text label in the array given to setTextBoxes during setup.

buff(input): The new text string.

Author: AP

13.36   setInfoLabel

bool
setInfoLabel(int n, const aString &buff)

Description: Set the text string in info label # n in the currently active GUIState.

n(input): The index of the text label returned by addInfoLabel during the setup.

buff(input): The new text string.

Return code: true if the label could be changed successfully, otherwise false

Author: AP

13.37   setToggleState

int
setToggleState(int n, int trueFalse)

Description: Set the state of toggle button # n in the currently active GUIState.

n(input): The index of the toggle button in the array given to setToggleButtons during setup.

trueFalse(input): trueFalse==1 turns the toggle button on, all other values turn it off.

Author: AP

13.38 setToggleState

int
setToggleState(const aString & toggleButtonLabel, int trueOrFalse)

Description: Set the toggle state for the toggle button with the given label.

toggleButtonLabel(input): The label of the toggle button to set.

trueOrFalse(input): The new state.

Author: wdh

14 Helper classes for the widgets in a dialog window

14.1 The PushButton Class

```
struct PushButton
{
// buttonCommand holds the name of the command and
// buttonLabel holds the label that will appear on the button.
  PushButton(){pb=NULL;sensitive=true;}
  void setSensitive(bool trueOrFalse);

  aString buttonCommand;
  aString buttonLabel;
  bool sensitive;
  void *pb; // widget
};
```

14.2 setSensitive

void
setSensitive(bool trueOrFalse)

Description: Set the sensitivity of a PushButton object.

trueOrFalse: The new state of the PushButton widget

Return valuse: None

Author: AP & WDH

14.3 The ToggleButton Class

```
struct ToggleButton
{
  ToggleButton(){tb=NULL;sensitive=true;}
  int setState(bool trueOrFalse );
  void setSensitive(bool trueOrFalse);

// buttonCommand holds the name of the command and
// buttonLabel holds the label that will appear on the button.
  aString buttonCommand;
  aString buttonLabel;
  int state;
  bool sensitive;
  void *tb; // widget
};
```

14.4 setSensitive

void
setSensitive(bool trueOrFalse)

Description: Set the sensitivity of a ToggleButton object.

trueOrFalse: The new state of the ToggleButton widget

Return valuse: None

Author: AP & WDH

14.5   setState

int
setState(bool trueOrFalse)

Description: Set the state of a Toggle button

trueOrFalse:

14.6   The TextLabel Class

```
struct TextLabel
{
// textCommand holds the name of the command and
// textLabel holds the label that will appear in front of the editable text.
// string holds the editable string.
// textWidget holds a pointer to the editable text widget, which makes it possible to change the text
// without typing in the dialog window. This might be useful for correcting typos, for example.
  TextLabel(){textWidget=NULL;sensitive=true; labelWidget=NULL;} // *wdh*
  void setSensitive(bool trueOrFalse);  // *wdh*

  aString textCommand;
  aString textLabel;
  aString string;
  bool sensitive;       // *wdh*
  void *textWidget; // Widget
  void *labelWidget; // Widget
};
```

14.7   setSensitive

void
setSensitive(bool trueOrFalse)

Description: Set the sensitivity of a TextLabel object.

trueOrFalse: The new state of the textlabel widget

Return valuse: None

Author: AP & WDH

14.8   The OptionMenu Class

```
struct OptionMenu
{
  OptionMenu(){menupane=NULL;sensitive=true; menuframe=NULL;}
  int setCurrentChoice(int command);
  void setSensitive(bool trueOrFalse);

  aString optionLabel;
  int n_options;
  PushButton *optionList;
  aString currentChoice;
  bool sensitive;
  void *menupane; // widget
  void *menuframe; // widget
};
```

14.9 setSensitive

void
setSensitive(int btn, bool trueOrFalse)

Description: Set the sensitivity of one button in a OptionMenu object.

btn: The button number in the option menu.

trueOrFalse: The new state of the push button widget

Return valuse: None

Author: AP

14.10 setCurrentChoice

int
setCurrentChoice(int command)

Description: Set the current choice for an option menu.

14.11 setCurrentChoice

int
setCurrentChoice(const aString & label)

Description: Set the current choice for an option menu.

14.12 The RadioBox Class

```
class RadioBox
{
public:
  RadioBox(){sensitive=true; radioBox=NULL; columns=1;}
  bool setCurrentChoice(int command);
  void setSensitive(bool trueOrFalse); // set the sensitivity of the entire radio box widget
  void setSensitive(int btn, bool trueOrFalse);  // set the sensitivity of one toggle button

  aString radioLabel;
  int n_options;
  ToggleButton *optionList;
  aString currentChoice;
  int currentIndex;
  bool sensitive;
  int columns;
  void *radioBox; // widget
};
```

14.13 setCurrentChoice

bool
setCurrentChoice(int command)

Description: Set the current choice for a radio box.

command(input): The command to be chosen

Return value: true if the command could be chosen, otherwise false. A command cannot be chosen if it is insensitive or out of bounds.

Author: AP

14.14 setSensitive

void
setSensitive(bool trueOrFalse)

Description: Set the sensitivity of a RadioBox object.

trueOrFalse: The new state of the RadioBox widget

Return value: None

Author: AP

14.15 setSensitive

void
setSensitive(int btn, bool trueOrFalse)

Description: Set the sensitivity of one button in a RadioBox object.

btn: The button number in the radio box.

trueOrFalse: The new state of the toggle button widget

Return value: None

Author: AP

14.16 The PullDownMenu Class

```
class PullDownMenu
{
public:
PullDownMenu(){menupane=NULL;sensitive=true;n_button=0;} // default constructor
// set all fields except the menupane in a PullDownMenu
int setPullDownMenu(const aString &pdMainLabel, aString commands[], aString labels[], button_type bt,
    int *initState /* = NULL */);
// set the state of a toggle button
int setToggleState(int n, bool trueOrFalse ); // *wdh*
void setSensitive(bool trueOrFalse);  // *wdh*

aString menuTitle;
int n_button;
button_type type;
PushButton *pbList;
ToggleButton *tbList;
bool sensitive;       // *wdh*
void *menupane; // widget
};
```

14.17 setSensitive

void
setSensitive(bool trueOrFalse)

Description: Set the sensitivity of a PullDownMenu object.

trueOrFalse: The new state of the PullDownMenu widget

Return value: None

Author: AP & WDH

14.18   setToggleState

int
setToggleState(int n, bool trueOrFalse)

Description: Set the state of a Toggle button in a pulldown menu.

trueOrFalse:

15   Setting Parameters: GraphicsParameters

Optional parameters can be passed to the various plotting functions by using the GraphicsParameters Class. These parameters can be used to set titles on the plots, or to turn off the plotting of the axes, for example. To pass optional parameters you create an object of type GraphicsParameters, set parameter values for this object and then pass the object to the plotting function. Here is an example of setting the title on a contour plot and turning off the plotting of the colour bar:

```
  ...
  GenericGraphicsInterface & ps = *Overture::getGraphicsInterface();
  GraphicsParameters gp; // create an object that is used to pass parameters

  gp.set(GI_TOP_LABEL,"My Title");   // Set the title
  gp.set(GI_PLOT_COLOUR_BAR,FALSE);  // Do not plot the colour bar
  PlotIt::contour(ps, u, gp);        // plot contours and pass parameters
  ...
```

Observe that the GraphicsParameters object, `gp`, will be changed within `contour` if parameters are changed interactively in the contour window. This means that you can set parameters interactively the first time you call a plotting function and then on subsequent calls the parameters will be remembered. A list of all variables that can be set is given in section 16.

NOTE: There used to be a derived class called PlotStuffParameters, which provided alternative ways to access the information in the GraphicsParameters class. This is no longer the case and all new development should use the class GraphicsParameters. To ensure backwards compatibility, a typedef is supplied in the include file PlotStuffParameters.h that defines the name PlotStuffParameters to be equivalent to GraphicsParameters.

Also note that the parameter GI_PLOT_THE_AXES has been removed completely. To turn on or off the axes, you instead call the routine setPlotTheAxes in the GenericGraphicsInterface class.

The class GraphicsParameters contains the following member functions:

15.1   Constructor

GraphicsParameters(bool default0)

Description: Constructor

15.2   isDefault

bool
isDefault()

Description: Return true if this object is a default object. This routine can be used to tell whether a GraphicsParameter object is equal to the static object `Overture::defaultGraphicsParameters()` which can be used as a default argument in a function call.

15.3   getObjectWasPlotted

int
getObjectWasPlotted() const

Description: Determine if the object was plotted in the last plotting routine that was called.

Return value: true if an object was plotted, false otherwise.

15.4   get(aString)

aString &
get(const GraphicsOptions & option, aString & label) const

Description: Return the aString associated with a GraphicsParameter option.

option (input) : Return the aString associated with this option (if any).

label (output) : Return the string in this variable

Return value: the return value is also equal to label.

15.5   get(int)

int &
get(const GraphicsOptions & option, int & value) const

Description: Return the int associated with a GraphicsParameter option.

option (input) : Return the int value associated with this option (if any).

value (output) : Return the value in this variable

Return value: the return value is also equal to value.

15.6   get(real)

real &
get(const GraphicsOptions & option, real & value) const

Description: Return the real associated with a GraphicsParameter option.

option (input) : Return the real value associated with this option (if any).

value (output) : Return the value in this variable

Return value: the return value is also equal to value.

15.7   get(IntegerArray)

IntegerArray &
get(const GraphicsOptions & option, IntegerArray & values) const

Description: Return the IntegerArray associated with a GraphicsParameter option.

option (input) : Return the IntegerArray value associated with this option (if any).

value (output) : Return the value in this variable

Return value: the return value is also equal to value.

15.8   get(RealArray)

RealArray &
get(const GraphicsOptions & option, RealArray & values) const

Description: Return the RealArray associated with a GraphicsParameter option.

option (input) : Return the RealArray value associated with this option (if any).

value (output) : Return the value in this variable

Return value: the return value is also equal to value.

15.9   get(Sizes)

real &
get(const Sizes & option, real & value) const

Description: Deterimine the value of a *size* parameter

option (input) : determine the value for this size.

value (output) : the value.

Return value: the return value is also equal to value.

15.10   set(GraphicsOptions, int/real)

int
set(const GraphicsOptions & option, real value)

Description: Assign a parameter with an int or real value

15.11   set(GraphicsOptions, IntegerArray)

int
set(const GraphicsOptions & option, const IntegerArray & values)

Description: Assign a parameter with that requires an array of int's

15.12   set(GraphicsOptions, RealArray)

int
set(const GraphicsOptions & option, const RealArray & values)

Description: Assign a parameter with that requires an array of real's

15.13   set(GraphicsOptions, aString)

int
set(const GraphicsOptions & option, const aString & label)

Description: Assign a parameter with a aString

15.14   set(Sizes)

int
set(const Sizes & option, real value) set a size

Description: Assign a *size* parameter

15.15   setColourTable

int
setColourTable(ColourTableFunctionPointer ctf)

Description: Provide a function to use for a colour table. This function will then be subsequently used for the
colour table. The colour table can be reset to one of the provided colour tables using the GI_SET_COLOUR_TABLE
option. (The function provided here corresponds to the userDefined colour table).

ctf (input) : a pointer to a function of the form shown below.

Here is an example of a function that defines a colour table

```
void
defaultColourTableFunction(const real & value, real & red, real & green, real & blue)
// =================================================================================
// Description: Convert a value from [0,1] into (red,green,blue) values, each in the range [0,1]
// value (input) :  0 <= value <= 1
// red, green, blue (output) : values in the range [0,1]
// =================================================================================
{ // a sample user defined colour table function:
  red=0.;
  green=value;
  blue=(1.-value);
}
```

16   List of parameters in GraphicsParameters

Here is a list of the various parameters that can be set in the class GraphicsParameters. Note that there used to be aliases for some of these names in the class PlotStuffParameters. In particular, the name PS_TOP_LABEL was an alias for GI_TOP_LABEL.

Table 3: Graphics Parameter Options

option name	value to supply
GI_AXES_ORIGIN	realArray axesOrigin(0:2)
GI_BACK_GROUND_GRID_FOR_STREAM_LINES	intArray dimension(0:1)
GI_BLOCK_BOUNDARY_COLOUR_OPTION	enum ColourOptions
GI_BOUNDARY_COLOUR_OPTION	enum ColourOptions
GI_BOTTOM_LABEL	String or character array
GI_BOTTOM_LABEL_SUP_1	String or character array
GI_BOTTOM_LABEL_SUP_2	String or character array
GI_BOTTOM_LABEL_SUP_3	String or character array
GI_COLOUR_INTERPOLATION_POINTS	bool, TRUE or FALSE
GI_COLOUR_LINE_CONTOURS	bool, TRUE or FALSE
GI_CONTOUR_ON_GRID_FACE	intArray cgf(0:1,0:2,0:?)
GI_CONTOUR_SURFACE_VERTICAL_SCALE_FACTOR	real
GI_COLOUR_TABLE	enum ColourTables
GI_COMPONENT_FOR_CONTOURS	int
GI_COMPONENT_FOR_SURFACE_CONTOURS	int
GI_COMPONENTS_TO_PLOT	intArray
GI_CONTOUR_LEVELS	realArray
GI_COORDINATE_PLANES	IntegerArray
GI_GRID_COORDINATE_PLANES	IntegerArray
GI_GRID_LINE_COLOUR_OPTION	enum ColourOptions
GI_GRIDS_TO_PLOT	boolArray grids(0:?)
GI_HARD_COPY_TYPE	HardCopyType
GI_ISO_SURFACE_VALUES	realArray
GI_KEEP_ASPECT_RATIO	bool
GI_LABEL_GRIDS_AND_BOUNDARIES	bool
GI_LINE_COLOUR	String or character array
GI_MAPPING_COLOUR	String or character array
GI_MAPPING_OFFSET	real (default=-3)
GI_MINIMUM_CONTOUR_SPACING	real
GI_MIN_AND_MAX_CONTOUR_LEVELS	realArray
GI_MIN_AND_MAX_STREAM_LINES	realArray minMax(0:1)
GI_MULTIGRID_LEVEL_TO_PLOT	int
GI_NORMAL_AXIS_FOR_2D_CONTOURS– _ON_COORDINATE_PLANES	int (0,1, or 2)
GI_NUMBER_OF_CONTOUR_LEVELS	int
GI_NUMBER_OF_GHOST_LINES_TO_PLOT	int
GI_REFINEMENT_LEVEL_TO_PLOT	int
GI_OUTPUT_FORMAT	OutputFormat
GI_PLOT_2D_CONTOURS_ON_COORDINATE_PLANES	bool
GI_PLOT_BACKUP_INTERPOLATION_POINTS	bool
GI_PLOT_BLOCK_BOUNDARIES	bool
GI_PLOT_GRID_BOUNDARIES_ON_CONTOUR_PLOTS	bool
GI_PLOT_COLOUR_BAR	bool

continued from previous page

option name	value to supply
GI_PLOT_CONTOUR_LINES	bool
GI_PLOT_GRID_LINES	bit mask
GI_PLOT_LABELS	bool
GI_PLOT_LINES_ON_GRID_BOUNDARIES	bool
GI_PLOT_LINES_ON_MAPPING_BOUNDARIES	bool
GI_PLOT_GRID_POINTS_ON_CURVES	bool
GI_PLOT_END_POINTS_ON_CURVES	bool
GI_PLOT_INTERPOLATION_POINTS	bool
GI_PLOT_THE_OBJECT_AND_EXIT	bool
GI_PLOT_REFINEMENT_GRIDS	bool
GI_PLOT_SHADED_MAPPING_BOUNDARIES	bool
GI_PLOT_SHADED_SURFACE	bool
GI_PLOT_SHADED_SURFACE _GRIDS	bool
GI_PLOT_THE_OBJECT	bool
GI_PLOT_WIRE_FRAME	bool
GI_POINT_COLOUR	String
GI_POINT_SIZE	real
GI_POINT_SYMBOL	int
GI_PLOT_BOUNDS	realArray bounds(0:1,0:2)
GI_USE_PLOT_BOUNDS,	bool
GI_USE_PLOT_BOUNDS_OR_LARGER,	bool
GI_RASTER_RESOLUTION	int
GI_STREAM_LINE_TOLERANCE	real
GI_U_COMPONENT_FOR_STREAM_LINES	int
GI_V_COMPONENT_FOR_STREAM_LINES	int
GI_TOP_LABEL	String or character array
GI_TOP_LABEL_SUB_1	String or character array
GI_TOP_LABEL_SUB_2	String or character array
GI_TOP_LABEL_SUB_3	String or character array
GI_X_SCALE_FACTOR	real
GI_Y_SCALE_FACTOR	real
GI_Z_SCALE_FACTOR	real
GI_Y_LEVEL_FOR_1D_GRIDS	real
GI_Z_LEVEL_FOR_2D_GRIDS	real

Here are explanations of some of the less obvious options:

- **GI_PLOT_GRID_LINES:** Set to 1 to turn on grid lines in 2D, set to 2 to turn on grid lines in 3D (set to 1+2=3 to turn on grid lines in 2D and 3D).

- **GI_PLOT_THE_OBJECT_AND_EXIT:** Plot the object and exit without staying in the local menu.

- **GI_CONTOUR_ON_GRID_FACE:** Supply an `intArray cgf(0:1,0:2,0:numberOfGrids-1)` with `cgf(side,axis,grid)=TRUE` if 3D contours should be plotted on that face of the grid.

- **GI_PLOT_SHADED_SURFACE:** plot shaded surfaces on Mappings.

- **GI_PLOT_SHADED_SURFACE_GRIDS:** plot shaded surfaces on grids.

- **GI_MINIMUM_CONTOUR_SPACING:** Force the contour spacing to be larger than this amount. This option is useful when the solution is nearly constant except for small variations caused by round-off errors, and you do not want to see the small variaions.

- **GI_PLOT_BOUNDS:** Set the plot bounds to be used. The plot bounds are set equal to the given realArray "bounds(0:1,0:2)". One must also set GI_USE_PLOT_BOUNDS to TRUE if you want these bounds to be used.

- **GI_USE_PLOT_BOUNDS:** Use the current plot bounds found with the GraphicsParameters object.

- **GI_USE_PLOT_BOUNDS_OR_LARGER:** Use plot bounds that are least as large as the current plot bounds – increase the plot bounds if the current object requires it.

- **GI_CONTOUR_LEVELS:** Specify the contour levels in an array. Levels should be in increasing order. The dimension of the array determines the number of levels. If the array is a null array, the number of contour levels is reset to the default value.

- **GI_COMPONENTS_TO_PLOT** Specify which components to plot for 1D contour plots.

- **GI_COLOUR_TABLE** The available colour tables are defined in the enum `ColourTables` found in the class `GraphicsParameters`:

  ```
  enum ColourTables
  {
    rainbow,
    gray,
    red,
    green,
    blue,
    userDefined,
    numberOfColourTables
  } colourTable;
  ```

- **GI_COORDINATE_PLANES** Specify the coordinate planes to plot contours on. The given array should be dimensioned, `IntegerArray coordinatePlane(3,numberOfCoordinatePlanes)` where

  ```
  coordinatePlane(0,plane) = grid number
  coordinatePlane(1,plane) = coordinate axis (0,1,2)
  coordinatePlane(2,plane) = grid index number
  ```

- **GI_GRID_COORDINATE_PLANES** Specify the coordinate planes to plot grid lines on. The given array should be dimensioned, `IntegerArray coordinatePlane(3,numberOfCoordinatePlanes)` where

  ```
  gridCoordinatePlane(0,plane) = grid number
  gridCoordinatePlane(1,plane) = coordinate axis (0,1,2)
  gridCoordinatePlane(2,plane) = grid index number
  ```

- **GI_GRID_LINE_COLOUR_OPTION** The available colour schemes are defined in the enum `ColourOptions` found in the class `GraphicsParameters`:

  ```
  enum ColourOptions  // options for colouring boundaries, grids lines and block boundaries
  {
    defaultColour,   // default
    colourByGrid,
    colourByRefinementLevel,
    colourByBoundaryCondition,
    colourByShare,
    colourByValue,
    colourBlack
  };
  ```

- **GI_HARD_COPY_TYPE** Specify the file type, postScript, encapsulated postScript, or ppm, to be saved in a call to the `hardCopy` function. The enum HardCopy type defines the types:

  ```
  enum HardCopyType
  {
    postScript,
    encapsulatedPostScript,
    ppm           // portable pixmap format (P6 binary format)
  };
  ```

81

- **GI_MAPPING_OFFSET** By default the mapping is offset by -3 "units" behind grid lines (so the grid lines appear correctly on the surface). You can change this value in order to plot two mapping surfaces that are right on top of one another.

- **GI_MIN_AND_MAX_STREAM_LINES** Supply a `realArray minMax(0:1)` to indicate the the minimum (`minMax(0)`) and maximum (`minMax(1)`) values for the speed, $\sqrt{u^2 + v^2}$, of the stream lines. These values are only used to determine how the stream lines are coloured. Specifying a minimum value greater than the maximum value will reset all values to the default.

- **GI_OUTPUT_FORMAT** Set the output format for files saved with a call to the `hardCopy` function. The choices can be taken from the enum OutputFormat:

```
enum OutputFormat    // formats for outputing postscript files
{
  colour8Bit,        // compressed colour file with 225 colours
  colour24Bit,       // 24 bits of colour (2^24 colours)
  blackAndWhite,     // black and white
  grayScale          // 8 bit gray scale (2^8 shades of gray)
};
```

  The default is `colour8Bit` which results in a much smaller postscript file than `colour24Bit`.

- **GI_RASTER_RESOLUTION** Set the raster resolution for files saved with a call to the `hardCopy` function. This resolution is by default 1024 (ie. the figure saved is rendered as 1024×1024 pixels. Setting the resolution to 0 (zero) will cause the highest resolution available to be used. Mesa can be compiled with increased resolution by changing the macros `MAX_WIDTH` and `MAX_HEIGHT` in the file `''Mesa''/src/config.h`. I change both these values to 2048, for example.

- **GI_[X/Y/Z]_SCALE_FACTOR:** scale the plots by this factor in the [x/y/z]-direction. This can be used to change the aspect ratio of the plot (if plotting a very "thin" geometry, for example).

17 Writing your own plotting routines with OpenGL

You can easily write your own plotting routines if you know OpenGL. Take a look at the source code for the grid plotter or contour plotter for some examples.

OpenGL is a widely available package for three dimensional plotting. Versions of OpenGL exist for almost all workstations and PC's. Currently on Sun worktstations, we use both the (faster) native OpenGL library and Brian Paul's Mesa library – a public domain implementation of OpengL that runs under X windows. See the OpenGL site on the WWW for further details, `http://www.sgi.com/Technology/openGL`. OpenGl is described in the `OpenGL Programming Guide` and the `OpenGL Reference Manual`, published by Addison Wesley.

Here is an example section of code that shows how the colour bar is drawn.

```
if( plotColourBar )
{
  // =========Draw the colour Bar=================
  glPolygonMode(GL_FRONT_AND_BACK,GL_FILL); // filled polygons
  glShadeModel(GL_SMOOTH);                   // interpolate colours between vertices

  // The colour bar is drawn in a way that is unaffected by rotations and scalings
  setNormalizedCoordinates();                // sets view to bounds [-1,1]x[-1,1]

  const int numberOfIntervals=50;
  real xLeft=.8, xRight=.85;                 // place the bar down the right side
  real yBottom=-.75, yTop=.75,y;
  glBegin(GL_QUAD_STRIP);                    // begin drawing a ''quad-strip''
  for( int i=0; i<numberOfIntervals; i++ )
  {
    y=yBottom+i*(yTop-yBottom)/(numberOfIntervals-1);
    // the next routine chooses a colour from a colour table. It then calls glColor3f(red,green,blue)
    setColourFromTable((y-yBottom)/(yTop-yBottom));
    glVertex2f(xLeft ,y);
    glVertex2f(xRight,y);
  }
```

```
  glEnd();                                          // end quad-strip colour bar
  // ---label colour bar---
  int numberOfLabels=numberOfContourLevels;
  glColor3f(0.,0.,0.);  // label colour is black
  real size=.01;
  for( i=0; i<numberOfLabels; i++ )
  {
    y=yBottom+i*(yTop-yBottom)/(numberOfLabels-1);
    real alpha=(y-yBottom)/(yTop-yBottom);
    if( max(fabs(uMax),fabs(uMin)) < 10. )
      label(sprintf(buff," %6.3f",uMin+alpha*(uMax-uMin)),xRight,y,size,-1);  // flush left
    else if( max(fabs(uMax),fabs(uMin)) < 100. )
      label(sprintf(buff," %7.2f",uMin+alpha*(uMax-uMin)),xRight,y,size,-1);  // flush left
    else
      label(sprintf(buff," %6e",uMin+alpha*(uMax-uMin)),xRight,y,size,-1);  // flush left
  }
  // ---draw lines on colour bar corresponding to the contour levels
  glBegin(GL_LINES);
  for( i=0; i<numberOfContourLevels; i++ )
  {
    y=yBottom+i*(yTop-yBottom)/(numberOfContourLevels-1);
    glVertex3f(xLeft ,y,.1);  // raise the lines so we see them
    glVertex3f(xRight,y,.1);
  }
  glEnd();

  unsetNormalizedCoordinates();
}
```

18 Using the NameList Class for interactively changing parameters

This class defines routines for inputing parameters by name in a similar fashion to the namelist facility in Fortran.
    Suppose that an application has a list of variables that can be changed by the user. Also suppose that the variables have default values so that the user may only want to selectively change the values of some variables. Then when the code is run the user would like to input changes to the parameters by name such as by typing

```
cfl=.75
machNumber=.2
boundaryCondition(0,1)=slipWall
```

Here is a program that demonstrates how an application could prompt for changes using the NameList class:

```
1    #include "NameList.h"
2
3    int
4    main()
5    {
6      ios::sync_with_stdio();
7
8      NameList nl;  // create a NameList object
9      // define some parameters:
10     int itest;
11     real a;
12     int array[10], matrix[5][5][5];
13     int value,i0,i1,i2,i3;
14     intArray c(3,3,3,3);
15     realArray d(3,3,3,3);
16     // the array num will take values from the enum "Numbers" defined next
17     intArray num(2,2); num=-1;
18     enum Numbers{ zero, one, two } number;
19     aString enumName[] = { "zero", "one", "two", "" }; // here are the names of the enum
20     intArray n(3,3,3,3);
21
22     printf(
23      "Make changes to the following variables: \n"
24      "itest            (int)\n"
25      "a                (real)\n"
26      "array[10]        (int)\n"
27      "matrix[5][5][5]  (int)\n"
28      "c(3,3,3,3)       (intArray)\n"
29      "d(3,3,3,3)       (realArray)\n"
30      "num(2,2) :       (enumArray, 'zero', 'one', 'two')\n"
31      "n(enum)=value    (enum one of 'zero', 'one', 'two')\n"
32      );
33     aString answer,name;
34     for( ;; )
35     {
36       cout << "Enter changes to variables, exit to continue" << endl;
37       cin >> answer;
38       if( answer=="exit" ) break;
39
40       nl.getVariableName( answer, name );   // parse the answer
41       if( name=="itest" )
42       {
43         itest=nl.intValue(answer);
44         cout << "itest=" << itest << endl;
45       }
46       else if( name=="a" )
47       {
48         a=nl.realValue(answer);
49         cout << "a=" << a << endl;
50       }
51       else if( name=="array" )
52       {
53         nl.intArrayValue(answer,value,i0);
54         cout << "value = " << value << ", i0 = " << i0 << endl;
55         array[i0]=value;
56       }
57       else if( name=="matrix" )
58       {
59         if( nl.intArrayValue(answer,value,i0,i1,i2) )
```

```
60         {
61            matrix[i0][i1][i2]=value;
62            printf("matrix[%i][%i][%i]=%i \n",i0,i1,i2, matrix[i0][i1][i2]);
63         }
64       }
65       else if( name=="c" )
66         nl.getIntArray( answer,c );
67       else if( name=="d" )
68         nl.getRealArray( answer,d );
69       else if( name=="num" )
70       {
71         nl.arrayEqualsName( answer,enumName,num,i0,i1,i2,i3 );
72         printf(" num(%i,%i) = %i (=%s) \n",i0,i1,num(i0,i1),(const char*)enumName[num(i0,i1)]);
73       }
74       else if( name=="n" )
75       {
76         nl.arrayOfNameEqualsValue( answer,enumName,n,i0 );
77         printf(" n(%i) = %i (n(%s)=%i) \n",i0,n(i0),(const char*)enumName[i0],n(i0));
78       }
79       else
80         cout << "unknown response: [" << name << "]" << endl;
81     }
82     return 0;
83   }
```

Here is some possible input when this program is run

```
a=3.
itest=2
array(2)=7
matrix(2,3,1)=8
num(0,1)=one
c(2,2)=6
d(1,2,1,2)=3
n(two)=55
```

Note that "c" style arrays are assigned with "()" rather than "[]"

18.1   NameList Function definitions

18.1.1   getVariableName

void
getVariableName(aString & answer, aString & name)

Description: Parse the aString "answer" and return the variable name:

```
            answer: "method=4"          -> name="method"
                  : "array(5,6)=56.7"   -> name="array"
```

answer (input) : string to parse.

name (output) : string before the first "=" sign

18.1.2   intValue

int
intValue(aString & answer)

Description: Return an int for a string of the form "name=int"

answer (input) : a aString of the form "name=int"

Return value: The value of the rhs in "name=int"

18.1.3 realValue

real
realValue(aString & answer)

Description: Return a real for a string of the form "name=real"

answer (input) : a aString of the form "name=real"

Return value: The value of the rhs in "name=real"

18.1.4 getIntArray

int
getIntArray(aString & answer, IntegerArray & a)

int
getIntArray(aString & answer, IntegerArray & a, int & i0)

int
getIntArray(aString & answer, IntegerArray & a, int & i0, int & i1)

int
getIntArray(aString & answer, IntegerArray & a, int & i0, int & i1, int & i2)

int
getIntArray(aString & answer, IntegerArray & a, int & i0, int & i1, int & i2, int & i3)

Description: Assign the value in an IntegerArray from a string of one of the following forms

```
name=value
name(i0)=value
name(i0,i1)=value
name(i0,i1,i2)=value
name(i0,i1,i2,i3)=value
```

answer (input) : a aString of one of the above forms

a (output) : an array that is to be assigned

i0,i1,i2,i3 (output) : Return the values for the indices used in evaluating the array.

Return value: Return TRUE if successful

18.1.5 getRealArray

int
getRealArray(aString & answer, RealArray & a)

int
getRealArray(aString & answer, RealArray & a, int & i0)

int
getRealArray(aString & answer, RealArray & a, int & i0, int & i1)

int
getRealArray(aString & answer, RealArray & a, int & i0, int & i1, int & i2)

Description: Assign values of a RealArray. see the documentation for getIntArray.

int
getRealArray(aString & answer, RealArray & a, int & i0, int & i1, int & i2, int & i3)

Description: Assign values of a RealArray. see the documentation for getIntArray.

18.1.6   intArrayValue

int
intArrayValue(aString & answer, int & value, int & i0)

int
intArrayValue(aString & answer, int & value, int & i0 , int & i1)

int
intArrayValue(aString & answer, int & value, int & i0, int & i1, int & i2)

int
intArrayValue(aString & answer, int & value, int & i0, int & i1, int & i2, int & i3)

Description: Return value and indices i0,i1,i2,i3 from a string of the form

```
name(i0)         =value  : intArrayValue(answer, value, i0 )
name(i0,i1)      =value  : intArrayValue(answer, value, i0, i1)
name(i0,i1,i2)   =value  : intArrayValue(answer, value, i0, i1, i2)
name(i0,i1,i2,i3)=value  : intArrayValue(answer, value, i0, i1, i2, i3)
```

answer (input) : string to parse.

value (output) : value found on the rhs of the string

i0,i1,i2,i3 (output) : index values

Return value : Return TRUE if successful

18.1.7   realArrayValue

int
realArrayValue(aString & answer, real & value, int & i0)

int
realArrayValue(aString & answer, real & value, int & i0 , int & i1)

int
realArrayValue(aString & answer, real & value, int & i0, int & i1, int & i2)

int
realArrayValue(aString & answer, real & value, int & i0, int & i1, int & i2, int & i3)

Description: Return value and indices i0,i1,i2,i3 from a string of the form

```
name(i0)         =value  : realArrayValue(answer, value, i0 )
name(i0,i1)      =value  : realArrayValue(answer, value, i0, i1)
name(i0,i1,i2)   =value  : realArrayValue(answer, value, i0, i1, i2)
name(i0,i1,i2,i3)=value  : realArrayValue(answer, value, i0, i1, i2, i3)
```

answer (input) : string to parse.

value (output) : value found on the rhs of the string

i0,i1,i2,i3 (output) : index values

Return value : Return TRUE if successful

18.1.8 arrayEqualsName

int
arrayEqualsName(aString & answer,
const aString nameList[],
IntegerArray & a,
int & i0 optional argument,
int & i1 optional argument,
int & i2 optional argument,
int & i3 optional argument)

Description: The aString answer should be of the form of one of

- arrayName(i0)=name
- arrayName(i0,i1)=name
- arrayName(i0,i1,i2)=name
- arrayName(i0,i1,i2,i3)=name

and the result of this function will be to set

- a(i0)=value where nameList[value]==name
- a(i0,i1)=value where nameList[value]==name
- a(i0,i1,i2)=value where nameList[value]==name
- a(i0,i1,i2,i3)=value where nameList[value]==name

answer (input) : a aString that should be of the form shown above.

nameList (input) : a null terminated array of names. These names will appear on the right hand side of the equals sign.

a (output): assign a value into this array.

i0,i1,i2,i3 (ouput) : optional arguments, return the values used in assigning a.

Return values: return TRUE if successful

18.1.9 arrayOfNameEqualsValue

int
arrayOfNameEqualsValue(aString & answer,
const aString nameList[],
IntegerArray & a,
int & i0 optional argument,
int & i1 optional argument,
int & i2 optional argument,
int & i3 optional argument)

Description: The aString answer should be of the form of one of

- arrayName(name0)=value
- arrayName(name0,name1)=value
- arrayName(name0,name1,name2)=value
- arrayName(name0,name1,name2,name3)=value

and the result of this function will be to set

- a(i0)=value where nameList[i0]==name0
- a(i0,i1)=value where nameList[i0]==name0, nameList[i1]==name1
- a(i0,i1,i2)=value where nameList[i0]==name0, nameList[i1]==name1,...

- a(i0,i1,i2,i3)=value where nameList[i0]==name0, nameList[i1]==name1,...

answer (input) : a aString that should be of the form shown above.

nameList (input) : a null terminated array of names. These names will appear as array arguements in answer.

a (output): assign a value into this array.

i0,i1,i2,i3 (ouput) : optional arguments, return the values used in assigning a.

Return values: return TRUE if successful

18.2   arrayOfNameEqualsValue

int
arrayOfNameEqualsValue(aString & answer,
                        const aString nameList[],
                        RealArray & a,
                        int & i0,
                        int & i1,
                        int & i2,
                        int & i3)

Description: The aString answer should be of the form of one of

- arrayName(name0)=value
- arrayName(name0,name1)=value
- arrayName(name0,name1,name2)=value
- arrayName(name0,name1,name2,name3)=value

and the result of this function will be to set

- a(i0)=value where nameList[i0]==name0
- a(i0,i1)=value where nameList[i0]==name0, nameList[i1]==name1
- a(i0,i1,i2)=value where nameList[i0]==name0, nameList[i1]==name1,...
- a(i0,i1,i2,i3)=value where nameList[i0]==name0, nameList[i1]==name1,...

answer (input) : a aString that should be of the form shown above.

nameList (input) : a null terminated array of names. These names will appear as array arguements in answer.

a (output): assign a value into this array.

i0,i1,i2,i3 (ouput) : optional arguments, return the values used in assigning a.

Return values: return TRUE if successful

Index