1.3 viewer - Module

Viewing Images and *Glish* arrays
include "viewer.g"

Description

The `viewer` module provides display services. It can produce false color image, contour and vector map representations of AIPS++ images and *Glish* arrays, rendered either on-screen or to PostScript files. Extensive control over the display of data is available from both graphical user and programming (command-line) interfaces.
The most important `tool` in this module is the Viewer `tool`, which combines a number of the other `tools` in the module to produce a cohesive application, operable from the command-line or from multiple windows in a graphical user interface. Most display services can be accessed directly from the Viewer `tool`.
An example on how to operate the Viewer `tool` from the command-line can be found here.
In general, there should be little need to read documentation on the other `tools` in this module. However, for reference purposes and if you want to integrate the Viewer `tool` into your own application/script take a closer look at the description of the `tools` which are part of this module.
Most of the `tools` in this module can be driven by graphical user interfaces. This documentation starts with a general description of how to use these GUIs to interact with the Viewer `tool`. You really only need to know about the GUIs which the Viewer `tool` provides access to:

1. The GUI which controls the import of data, the Viewerdatamanager `tool`
2. The GUI which controls your display

A list of aipsrc setting to customize the Viewer can be found in the Resources section.
Ok - here a little sitemap for this module:

- **Toolkit approach** go here
- **GUI functionality** look here for the Viewerdisplaypanel `tool` and the Viewerdatamanager `tool`.
- **Customization** look here
- **Viewer - behind the scenes** look here
- **CLI example** look here.

Fundamentals

Displaydatas

The main task of the Viewer `tool` is to produce visual representations of AIPS++ data. Presently, the data itself may be an AIPS++ Image stored on disk, or a *Glish* array, while the visual representation can be a false color image (hereafter a "raster"), or a contour map (hereafter a "contour"). One of the two fundamental `tools` that the Viewer `tool` is built upon is the Viewerdisplaydata `tool`. This `tool` combines a data source and a representation method into a single unit which can be passed around the various `tools` which are managed by a single Viewer `tool`.
A Viewerdisplaydata `tool` is completely self-contained, and has "state". Its state consists of various attributes, some of which will not change during the lifetime of the Viewerdisplaydata `tool`, and many others which probably will. Examples of unchangeable state include the dimensionality of the Image or *Glish* array, and the type of visual representation requested: a raster or a contour. Examples of changeable state include the many attributes which describe how the data should be translated into the requested represetation, such as the "colormap" (see below) to use for a raster, or the contour levels to use for a contour Viewerdisplaydata `tool`. The changeable state of a Viewerdisplaydata `tool` can be modified from the command-line or in a graphical user interface. **Many Viewerdisplaydata `tools` can be created and managed by a single Viewer `tool`.**

Displaypanels

The Viewerdisplaypanel `tool` provides one ore more "canvas(es)" on which one or more Viewerdisplaydata `tools` can draw themselves, and is the second fundamental `tool` upon which the Viewer `tool` is built. This `tool` provides facilities for Viewerdisplaydata `tools` to register and unregister themselves, that is, to join and leave the list of Viewerdisplaydata `tools` that will get asked to draw themselves whenever the canvas is redrawn. Furthermore, it provides a "controlbox" of operations which can be applied to the canvas, for example, zooming (magnifying) the display. Finally, each Viewerdisplaypanel `tool` has two "managers" associated with it: a Viewercanvasmanager `tool` which provides the facility to modify some of the canvas-specific settings (for example, background color, margin widths, and the number of panels), and a Viewercanvasprintmanager `tool` which can be used to generate PostScript and X11 Pixmap output from the contents of the Viewerdisplaypanel `tool`. **Many Viewerdisplaypanel `tools` can be created and managed by a single Viewer `tool`.**

Options One of the most important concepts of the Viewer `tool` is the way

states and attributes are set. `tools` like Viewerdisplaydata `tool` and Viewerdisplaypanel `tool` have functions *getoptions* and *setoptions* to control these "options". For example, the state of a Viewerdisplaydata `tool` is held in

an option record. The documentation of each `tool` summarizes it's options which can be controlled by the user.
Let's say you have made a Viewerdisplaydata `tool` as a raster and you want to know the options on this displaydata, you would to something like this:

```
dd := dv.loaddata('myfantasticimage','raster');
opts := dd.getoptions();
print field_names(opts);
```

These options are usually context dependent, i.e. an option record for a "raster" Viewerdisplaydata `tool` is different from a "contour" Viewerdisplaydata `tool` option record.

Colormaps

The display of rasters requires that the pixel values in a data source be mapped to colors selected from a pre-defined set of colors, or a "colormap." The Viewer `tool` provides colormaps via the Viewercolormapmanager `tool`, and there is one and only one Viewercolormapmanager `tool` per Viewer `tool`. Sixteen unique colormaps are available. Whilst generation of new colormaps by the user or *Glish* programmer is not available yet, the existing colormaps can be modified in several ways by directly using the Viewercolormapmanager `tool`, or its graphical user interface. The usual "slope-shift" colormap fiddling is provided, together with "contrast-brightness" fiddling. Both of these are also available from the controlbox of Viewerdisplaypanel `tools`. Further capabilities (available via only the Viewercolormapmanager `tool`) include the independent inversion of the red, green and blue components of the colors in the colormap.
Viewer `tool` colormaps are entities which can be used by on many Viewerdisplaypanel `tools` concurrently, and which share color resources when possible. They automatically resize themselves to use as many colors as they can, based both on the number of colors that the various Viewerdisplaypanel `tools` are set up to use, and on the number (and weightings) of other Viewer `tool` colormaps that are in use. For this reason, when multiple Viewerdisplaypanel `tools` are in use, registering or unregistering a raster-type Viewerdisplaydata `tool` on one Viewerdisplaypanel `tool` may well cause other Viewerdisplaypanel `tools` to refresh, as more or less colors become available for other colormaps in use. The programmer can reduce the resulting "flickering" by "holding" and "releasing" the Viewer `tools` that are in use. The user can control the number of colors that the various Viewerdisplaypanel `tools` are setup to use, and can modify the Viewer `tool` colormaps themselves, by fiddling and inverting components.

> Advanced note: The Viewer `tool` inserts a software layer above the hardware colormap of the screen in use. This is so that the user can choose to have as many colors as possible for the Viewerdisplaypanel `tools`, knowing they will get color flashing, or instead to use the system colormap, which invariably will limit the number of colors that are used

by the Viewer `tool`, but will prevent color flashing. Indeed, for a single Viewer `tool`, multiple Viewerdisplaypanel `tools` can be used: some of these may use "private" colormaps, and others may use the system colormap.

Once a Viewerdisplaypanel `tool` is on the screen, you cannot change whether it is using the system colormap or a private colormap. You *can*, however, change the number of colors it is actually making use of in its colormap. This can be useful in two ways: firstly, sometimes rasters look better if you use fewer colors and sometimes look better with more colors: so with this feature, you can dynamically reduce or increase the "smoothness" (in colorspace) of raster images. Secondly, let's say we made a Viewerdisplaypanel `tool` and asked it to use the system colormap: that is, we were happy to have fewer colors so that we could avoid color flashing. Sometime later, we start up a (non-AIPS++) image viewing program on the same screen, and because the system colormap is nearly over-subscribed, partly by the Viewerdisplaypanel `tool`, this external program decides to use a private colormap. This might not be satisfactory, so you can actually close that program, reduce the number of colors being used by your Viewerdisplaypanel `tool` (via a graphical user interface perhaps), and then re-start the external program. *You did not have to close the Viewerdisplaypanel* `tool` *to accomplish this!*

While a single Viewer `tool` can have many Viewerdisplaypanel `tools` operating, it only ever has one Viewercolormapmanager `tool`, whose colormaps are shared by all the Viewerdisplaypanel `tools`. Viewerdisplaypanel `tools` which are using the system X colormap (ie. that aren't causing color flashing) actually all share one contiguous section of the system X colormap. Whatever Viewer `tool` colormaps are needed are placed in this block of color cells, usually uniformly distributed. For example, suppose I have two Viewerdisplaypanel `tools` on screen, both using the same 80 cells of the system X colormap. If they both have raster Viewerdisplaydata `tools` registered for display, each of which has its own (different) Viewer `tool` colormap (say one is using "Rainbow 4" and the other "Hot Metal 2"), then each of the Viewer `tool` colormaps will get 40 cells each. The "Rainbow 4" colormap can be fiddled and inverted to your hearts content without ever affecting the raster drawn with the "Hot Metal 2" colormap. If I close one of the Viewerdisplaypanel `tools`, the other one will notice this, and stretch the colormap it is using back out to fill the 80 cells. Conversely, if I open another Viewerdisplaypanel `tool` using the system X colormap, and display yet another raster using yet another Viewer `tool` colormap, the three required colormaps will be packed into the 80 cells (having lengths 27, 27 and 26).

Let's say, however, that with the two system X colormap Viewerdisplaypanel `tools` still open, I make a third Viewerdisplaypanel `tool`, but ask that it use a *private* colormap. I'm going to get color flashing, but I'm also going to get way more than 80 colors: I'll probably get around 250 on an 8-bit system, and around 1000 on a 10-bit system. If I go and register a third raster Viewerdisplaydata `tool` in this Viewerdisplaypanel `tool`, and ask it to use a different Viewer `tool` colormap, eg. "Greyscale 1", then the two existing Viewerdisplaypanel `tools` are *not* affected, and the "Greyscale 1" colormap will fill the 250 (or more) colors available in the third Viewerdisplaypanel `tool`. But this "Greyscale 1" colormap can be mapped into more than one X colormap! So if I choose this colormap to be used for one of the first two Viewerdisplaydata `tools`, then it will replace, say, "Rainbow 4", in the 80 cells of the system X colormap we are using, but it will remain occupying 250 cells in the private colormap Viewerdisplaypanel `tool`. Yet they are the same Viewer `tool` colormap, so modifying the "Greyscale 1" colormap in any way, either via the Viewercolormapmanager `tool`, its graphical user interface, or a Viewerdisplaypanel `tool` controlbox, will cause equivalent changes in the display of both rasters that are being drawn with "Greyscale 1".

Resources

The Viewer `tool` looks for and recognizes a number of variables in the system and user `.aipsrc` files. They are listed in aipsrcdata.

`Tools` in this module

- The Vieweranimator `tool`, providing services to (manually or automatically) step through multi-dimensional images.
- The Viewercanvasmanager `tool`, which provides services to control the appearance of the canvases owned by Viewerdisplaypanel `tools`.
- The Viewercanvasprintmanager `tool`, which provides printing services to for the canvases of Viewerdisplaypanel `tools`.
- The Viewercolormapmanager `tool`, which is used to generate and modify colormaps for use by other `tools` in this module.
- The Viewerdatamanager `tool`, which provides a simple graphical user interface for constructing and destroying Viewerdisplaydata `tools`from files and `tools`.
- The Viewerdisplaydata `tool`, which is used to render false color images or contour maps from various data sources (AIPS++ Images or *Glish* arrays). To become visible to the user, Viewerdisplaydata `tools` must be registered with one or more Viewerdisplaypanel `tools`, but they can be created, manipulated and destroyed without ever being registered. More generally, the Viewerdisplaydata `tool` embodies some data source, and some method for drawing that data.
- The Viewerdisplaypanel `tool`, whose fundamental purpose is to provide a "canvas" on which Viewerdisplaydata `tools` can draw themselves. This `tool` is where users should expect to see their data displayed.
- The Viewerannotations tool which can be used to annotate display panels.

- The Viewerslicepanel `tool`, which provides 3D slicing capabilities. Apart from this the interface is very similar to the Viewerslicepanel `tool`.

Example

Here is a trivial example to give you the flavour of using a Viewer `tool`. Suppose we have an AIPS++ image on disk and wish to view it on the screen then generate a "quick" PostScript file of the image, with axis labels:

```
- include 'viewer.g';                                                   # 1
- mdd := dv.loaddata('myaips++image', 'raster');                        # 2
- mdd.setoptions([axislabelswitch=T]);                                  # 3
- mdp := dv.newdisplaypanel();                                          # 4
- mdp.register(mdd);                                                    # 5
- mdp.canvasprintmanager().writeps('mydiskimage.ps', dpi=300);          # 6
- mdd.done();                                                           # 7
- mdp.done();                                                           # 8
```

1. This gives us access to the Viewer `tool` code, and in particular, the **defaultviewer** `tool`.
2. Create a Viewerdisplaydata `tool` which is a raster representation of the AIPS++ Image with the given name.
3. Turn axis labelling on.
4. Create a Viewerdisplaypanel `tool` for rendering.
5. Register the **mdd** Viewerdisplaydata `tool` on the **mdp** Viewerdisplaypanel `tool`. At this point, the raster should now be visible somewhere on your screen.
6. Print a copy of the raster on the screen to the named PostScript file, at a nice high resolution of 300 dots per inch.
7. Destroy the Viewerdisplaydata `tool`.
8. Destroy the Viewerdisplaypanel `tool`.

Tools

skycatalog	Tool to create and manipulate Viewer `tool` readable skycatalog tables
viewer	Tool for viewing AIPS++ images and arrays
viewerannotations	Tool for drawing annotations onto a displaypanel.
viewercanvasmanager	Tool for managing the canvases of Viewerdisplaypanel `tools`
viewercanvasprintmanager	Tool for managing the printing of canvases of Viewerdisplaypanel `tools`
viewercolormapmanager	Tool for managing colormaps used by the Viewer tool
viewerdatamanager	Tool for importing data into the Viewer

viewerdisplaydata	Tool for managing the drawing of AIPS++ data
viewerdisplaypanel	Tool for providing drawing windows (canvases)
viewerslicedp	Tool for providing 3D slice canvases
viewerstandardgui	Interface for operating Viewer `tool` graphical user interfaces

Functions

is_viewer	Is the argument a viewer tool?

1.3.1 viewer - Tool

Package display

Module viewer

Tool for viewing AIPS++ images and arrays
include "viewer.g"

Constructors

viewer	Create a Viewer `tool`

Functions

alldisplaydatas	Return a list of all Viewerdisplaydata `tools` known to this Viewer `tool`
alldisplaypanels	Return a list of all Viewerdisplaypanel `tools` known to this Viewer `tool`
colormapmanager	Return the Viewercolormapmanager `tool` for this Viewer `tool`
deleteall	Delete all Viewerdisplaydata `tools`
deletedata	Delete a Viewerdisplaydata `tool`
disable	Disable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager `tools`
done	Close down (destroy) this Viewer `tool`
emittoolkitlist	Emit an event describing the toolkitlist for this Viewer `tool`
enable	Enable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager `tools`
gui	Create and show a new Viewerdatamanager `tool` and/or Viewerdisplaypanel `tool` on the screen
hold	Prevent refreshes occuring on all Viewerdisplaypanel `tools`
loaddata	Create a Viewerdisplaydata `tool` for future use
newcolormapmanagergui	Create and optionally display a new GUI to control this Viewer `tool`'s Viewercolormapmanager `tool`
newdatamanager	Create and show a new Viewerdatamanager `tool` on the screen
newdisplaypanel	Create and show a new Viewerdisplaypanel `tool` on the screen
release	Allow refreshes to occur on all Viewerdisplaypanel `tools`
title	Return the (construction) title of this `tool`
toolkit	Returns the toolkit used by this `tool`
type	Return the type of this `tool`
updatedatasets	Update the list of global images and arrays kept by the Viewer `tool`
widgetset	Return the Widgetserver `tool` used by this `tool`

Description

The Viewer `tool` is an AIPS++ tool which provides data display services. It can produce false color image and contour map representations of AIPS++ images or *Glish* arrays, rendered either on-screen or to PostScript files. Extensive control over the display of data is available from both graphical and programming interfaces. The Viewer `tool` is the primary interface to the viewer module of the Display package.

viewer - Constructor

Package	**display**
Module	**viewer**
Tool	**viewer**

Create a Viewer `tool`

Synopsis
viewer(title, deleteatexit, widgetset)

Arguments

title	Title of this Viewer `tool` which appears in windows Allowed:String Default: 'viewer'
deleteatexit	Delete this Viewer `tool` when *Glish* exits? Allowed:Boolean Default: T
widgetset	Widgetserver `tool` to use for the GUIs Allowed:Widgetserver `tool` Default: dws

Returns
Viewer `tool` or fail

Description

This is the only constructor for Viewer `tools`. It constructs (creates) a Viewer `tool` which can subsequently be used to construct and manage the various other component `tools` which might make up a data viewing application or the display services for an existing application.

---

viewer.alldisplaydatas - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return a list of all Viewerdisplaydata `tools` known to this Viewer `tool`

Synopsis
alldisplaydatas()

Returns
list of Viewerdisplaydata `tools`

Description

This function simply returns a record containing all of the Viewerdisplaydata `tools` known to (managed by) this Viewer `tool`. The record contains as many top-level fields as there are Viewerdisplaydata `tools`, and each field is a Viewerdisplaydata `tool`. The fields are named by the identifying name of the corresponding Viewerdisplaydata `tool`, ie. the return value of the viewer:viewerdisplaydata.name function.

---

viewer.alldisplaypanels - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return a list of all Viewerdisplaypanel `tools` known to this Viewer `tool`

Synopsis
alldisplaypanels()

Returns
list of Viewerdisplaypanel `tools`

Description

This function simply returns a record containing all of the Viewerdisplaypanel `tools` known to (managed by) this Viewer `tool`. The record contains as many top-level fields as there are Viewerdisplaypanel `tools`, and each field is a Viewerdisplaypanel `tool`. The fields are named by increasing integers.

---

viewer.colormapmanager - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the Viewercolormapmanager `tool` for this Viewer `tool`

Synopsis
colormapmanager()

Returns
Viewercolormapmanager `tool`

Description

Every Viewer `tool` has one and only one Viewercolormapmanager `tool` associated with it. It is constructed at construction time of the Viewer `tool`. This function returns the Viewercolormapmanager `tool` for this Viewer `tool`. More details on the functionality of a Viewercolormapmanager `tool` can be found in the documentation for Viewercolormapmanager `tool`.

---

viewer.deleteall - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Delete all Viewerdisplaydata `tools`

Synopsis
deleteall(doneit, quiet)

Arguments

doneit	truly delete the Viewerdisplaydata `tools`? Allowed:Boolean Default: T
quiet	prevent emission of updated list of Viewerdisplaydata `tools`? Allowed:Boolean Default: F

Returns

None

Description

This function simply calls the `deletedata` function for every Viewerdisplaydata `tool` managed by (known to) this Viewer `tool`. It is normally called when the Viewer `tool` is dying.

---

viewer.deletedata - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Delete a Viewerdisplaydata `tool`

Synopsis

deletedata(displaydata, doneit, quiet)

Arguments

displaydata	Viewerdisplaydata `tool` to delete Allowed:Viewerdisplaydata `tool`
doneit	truly delete the Viewerdisplaydata `tool`? Allowed:Boolean Default: T
quiet	prevent emission of updated list of Viewerdisplaydata `tools`? Allowed:Boolean Default: F

Returns
None

Description

This function should be used to delete Viewerdisplaydata `tools` that were created by this Viewer `tool`. It will ensure that the Viewerdisplaydata `tool` is unregistered from all Viewerdisplaypanel `tools`, and then if `doneit=T` (the default), will actually free the resources used by the Viewerdisplaydata `tool` by calling its `done` function. The `quiet` parameter should only be `T` when the entire Viewer `tool` is closing done via its own `done` function.

---

viewer.disable - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Disable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager `tools`

Synopsis
disable()

Returns
none

Description

This function is used to disable the GUIs of all the Viewerdisplaypanel `tools` and all the Viewerdatamanager `tools` that are managed by this Viewer `tool`. This differs from the `hold` function in that refreshes of the displays will still be allowed, but the user will not be able to interact with the GUIs themselves. The GUIs will be re-enabled when an equal number of calls are made to the `enable` function. Actually, the GUIs can only effectively be disabled (greyed out) when they created their own top-level frames, so for applications where Viewerdisplaypanel `tools` for example are embedded in externally "owned" frames, this function does nothing.

---

viewer.done - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Close down (destroy) this Viewer `tool`

Synopsis
done()

Returns
T

Description

Calling this function frees up the resources used by this Viewer `tool`, and renders it useless. This function should be called when you are finished with a particular Viewer `tool`, and is also called when *Glish* exits, if the `deleteatexit` argument to the Viewer `tool` constructor was T. All GUIs (including Viewerdisplaypanel `tools`) created by this Viewer `tool` will be removed from the display, and all Viewerdisplaydata `tools` managed by this Viewer `tool` will be destroyed. Any Images used to construct those Viewerdisplaydata `tools` will of course remain on disk. Depending on how imaginatively you have used the Viewer `tool`, this function can take a few seconds to complete.

---

viewer.emittoolkitlist - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Emit an event describing the toolkitlist for this Viewer `tool`

Synopsis
emittoolkitlist()

Returns
None

Description

This function simply makes the Viewer `tool` emit an event with the name 'toolkit', with the event value being the return value of a call to the `toolkitlist` function. Typically this is used internally to synchronise all GUIs which display a list of the available tools and their mappings.
This function is normally used internally only.

---

viewer.enable - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Enable the GUIs of all Viewerdisplaypanel `tools` and Viewerdatamanager `tools`

Synopsis
enable()

Returns
none

Description

This function is used to enable the GUIs of all the Viewerdisplaypanel `tools` and all the Viewerdatamanager `tools` that are managed by this Viewer `tool`. This differs from the `release` function in that it does not affect refreshes of the display, but does affect whether users can interact with the GUIs themselves. The GUIs will only be re-enabled if the number of calls made to this function is equal to that made to the `disable` function.

---

viewer.hold - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Prevent refreshes occuring on all Viewerdisplaypanel `tools`

Synopsis
hold()

Returns
none

Description

This function is used to prevent refreshing occuring on all Viewerdisplaypanel `tools` that are managed by this Viewer `tool`. Refreshes will be re-enabled when an equal number of calls have been made to the `release` function. Typically, this is used internally when Viewerdisplaydata `tools` are being deleted or registered or unregistered from Viewerdisplaypanel `tools`. Programmers should make use of the `hold` and `release` functions when performing multiple operations before a refresh is desired.

---

viewer.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and show a new Viewerdatamanager `tool` and/or Viewerdisplaypanel `tool` on the screen

Synopsis
gui(datamanager, displaypanel)

Arguments

datamanager	Show a new Viewerdatamanager `tool`? Allowed:Boolean Default: T
displaypanel	Show a new Viewerdisplaypanel `tool`? Allowed:Boolean Default: T

Returns
Viewerdatamanager `tool`

Description

This function is a simple front-end to the newdatamanager and newdisplaypanel functions. If the `datamanager` argument is T, then a new Viewerdatamanager `tool` graphical user interface will be placed on the screen in its own frame. Likewise, if the `displaypanel` argument is T, then a new Viewerdisplaypanel `tool` will be put on screen in a frame of its own.
This function is provided partly for convenience, but mostly to suit the requirements of the Toolmanager `tool`. See the individual functions newdatamanager and newdisplaypanel for further information.

---

viewer.loaddata - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create a Viewerdisplaydata `tool` for future use

Synopsis

loaddata(data, drawtype, autoregister)

Arguments

data	data to load Allowed:array or image object, string filename of image or measurement set
drawtype	type of Viewerdisplaydata `tool` to create Allowed:'raster', 'contour', 'vector', 'marker', or 'skycatalog'
autoregister	auto-register this on any auto-register Viewerdisplaypanel `tools`? Allowed:Boolean Default: F

Returns

Viewerdisplaydata `tool`

Description

This function provides the main route to creating Viewerdisplaydata `tools`. Any Viewerdisplaydata `tool` created using this function will be managed by this Viewer `tool`. Supported file types besides image tools and arrays are:

- AIPS++ images (for maximum performance use this)
- FITS images
- Miriad images
- Gipsy images (via FITS conversion;you need to have GIPSY available to be able to convert it to FITS)
- AIPS++ measurement sets (only the 'raster' drawtype is allowed at present)

The arguments are straightforward, so for more information on what you can do with your Viewerdisplaydata `tool` once you've made it, see the documentation on Viewerdisplaydata `tools`. **Internal use:**This function uses the `createdata` function.

---

viewer.newcolormapmanagergui - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and optionally display a new GUI to control this Viewer `tool`'s Viewercolormapmanager `tool`

Synopsis
newcolormapmanagergui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame in which to place the Viewerdatamanager `tool` Allowed:agent or F Default: F
show	show the Viewerdatamanager `tool`? Allowed:Boolean Default: T
hasdismiss	add a "Dismiss" button to the GUI? Allowed:Boolean Default: F
hasdone	add a "Done" button to the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use to build the GUI Allowed:Widgetserver `tool` or unset Default: unset

Returns
`tool` having Viewerstandardgui interface

Description

This function can be used to place on the screen a new GUI for controlling this Viewer `tool`'s Viewercolormapmanager `tool`. Careful use of the `hasdismiss, hasdone` and `parent` arguments can provide quite flexible control over how the user builds their GUI.
See the Viewercolormapmanager `tool` documentation for more information.

---

viewer.newdatamanager - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and show a new Viewerdatamanager `tool` on the screen

Synopsis
newdatamanager(parent, show, showload, showedit, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame in which to place the Viewerdatamanager `tool` Allowed:agent or F Default: F
show	show the Viewerdatamanager `tool`? Allowed:Boolean Default: T
showload	show the "Load data" rollup? Allowed:Boolean Default: F
showedit	show the "Adjust data" rollup? Allowed:Boolean Default: F
hasdismiss	add a "Dismiss" button to the GUI? Allowed:Boolean Default: F
hasdone	add a "Done" button to the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use to build the GUI Allowed:Widgetserver `tool` or unset Default: unset

Returns
Viewerdatamanager `tool`

Description

This function is generally used to place a new Viewerdatamanager `tool` on the screen. By judicious use of the `parent, hasdismiss` and `hasdone` arguments the appearance of the Viewerdatamanager `tool` can be controlled quite carefully. The Viewerdatamanager `tool` is a relatively simple graphical user interface which simplifies the construction and editing of Viewerdisplaydata `tools`.

---

viewer.newdisplaypanel - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Create and show a new Viewerdisplaypanel `tool` on the screen

Synopsis

newdisplaypanel(parent, width, height, nx, nx, maptype, newcmap, mincolors, maxcolors, autoregister, holdsdata, show, hasgui, guihasmenubar, guihascontrolbox, guihasanimator, guihasbuttons, hasdismiss, hasdone, isolationmode, widgetset, slicepanel)

Arguments

parent	parent frame in which to place the Viewerdisplaypanel `tool` Allowed:agent or F Default: F
width	width of the drawing area on screen Allowed:Integer Default: 400
height	height of the drawing area on screen Allowed:Integer Default: 300
nx	number of panels in x Allowed:Integer Default: 1
nx	number of panels in y Allowed:Integer Default: 1
maptype	how colors will be addressed on this Viewerdisplaypanel `tool` Allowed:'index' 'rgb' 'hsv' Default: 'index'
newcmap	install a private colormap? Allowed:Boolean or unset Default: unset
mincolors	minimum number of colors to allocate Allowed:Integer or unset Default: unset
maxcolors	maximum number of colors to allocate Allowed:Integer or unset Default: unset
autoregister	accept auto-register data on this Viewerdisplaypanel `tool`? Allowed:Boolean Default: F
holdsdata	is this Viewerdisplaypanel `tool` going to hold data? Allowed:Boolean Default: T
show	show the Viewerdisplaypanel `tool`? Allowed:Boolean Default: T
hasgui	adorn the Viewerdisplaypanel `tool` with elements of a graphical user interface? Allowed:unset or Boolean Default: unset
guihasmenubar	add the menubar if a GUI is added? Allowed:Boolean Default: T
guihascontrolbox	add the controlbox if a GUI is added? Allowed:Boolean Default: T
guihasanimator	add the animator tapedeck if a GUI is added? Allowed:Boolean Default: T
guihasbuttons	add the button bar if a GUI is added? Allowed:Boolean Default: T
hasdismiss	add a "Dismiss" button to the GUI? Allowed:unset or boolean Default: unset
hasdone	add a "Done" button to the GUI? Allowed:unset or boolean Default: unset
isolationmode	prevent user accessing other components of this Viewer `tool` via the GUI? Allowed:boolean Default: F
widgetset	Widgetserver `tool` to use to build the GUI Allowed:Widgetserver `tool` or unset Default: unset
slicepanel	create a 3d slice panel? Allowed:Boolean Default: F

Returns
Viewerdisplaypanel `tool`

Description

This function is generally used to place a new Viewerdisplaypanel `tool` on the screen. By judicious use of the `parent, hasdismiss` and `hasdone` arguments the appearance of the Viewerdisplaypanel `tool` can be contolled quite carefully. The `guihas*` arguments control the adornment of the Viewerdisplaypanel `tool` with various graphical user interface components. The `hasdismiss` and `hasdone` arguments can be used to specify what type of "Dismiss" button is placed on the Viewerdisplaypanel `tool`, if any. For further information, see the documentation on the Viewerdisplaypanel `tool`constructor, and its addgui function.
The `newcmap` argument is particular to this function, and parsed before it is passed on to the Viewerdisplaypanel `tool` constructor. If it is `unset`, and a parent frame is *not* given, (ie. `parent=F`), then this function will first attempt to construct a Viewerdisplaypanel `tool` using ("sharing") the system colormap, and if that fails, it will try again with a private colormap. If the `newcmap` argument is a boolean, then that is passed directly to the Viewerdisplaypanel `tool` constructor. The last argument `slicepanel` controls if the new tool should be a slicepanel to enable 3D slicing.

---

viewer.release - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Allow refreshes to occur on all Viewerdisplaypanel `tools`

Synopsis
release()

Returns
none

Description

This function is used to re-establish refreshing occuring on all Viewerdisplaypanel `tools` that are managed by this Viewer `tool`. Refreshes will be re-enabled when an equal number of calls have been made to this function as have been made to the `hold` function. Typically, this is used internally when Viewerdisplaydata `tools` are being deleted or registered or unregistered from Viewerdisplaypanel `tools`. Programmers should make use of the `hold` and `release` functions when performing multiple operations before a refresh is desired.

---

viewer.title - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the (construction) title of this `tool`

Synopsis
title()

Returns
string

Description

This function returns the title that was specified at construction of this Viewer `tool`. It is ordinarily used only to generate title text for graphical user interfaces for this Viewer `tool`.

---

viewer.toolkit - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Returns the toolkit used by this `tool`

Synopsis
toolkit()

Returns

Viewer toolkit

Description

This function returns the Viewer toolkit of the current Viewer `tool`, which defines which tools (eg. Zoomer tool) are assigned to which mouse buttons and which tools appear in the controlbox. *This function is normally used internally only.*

---

viewer.type - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the type of this `tool`

Synopsis
type()

Returns
'viewer'

Description

This function simply returns the string `'viewer'`, which can be used to identify this `tool` as a Viewer `tool`.

---

viewer.updatedatasets - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Update the list of global images and arrays kept by the Viewer `tool`

Synopsis
updatedatasets()

Returns
none

Description

This function is quite specific in its use. Its purpose is to search for global *Glish* symbols which are either *Glish* arrays or Image `tools`, and to store them in an internal list. This list is used by other Viewer `tool` functions. After creating the updated list, it emits an event called 'datasets_updated' so that any interested parties are informed that new data might be available for their own lists. For example, the Viewerdatamanager `tool` listens for this event so that its list of data remains current. This function might also be used by a programmer after a global Image `tool` is explicitly constructed in their code.

---

viewer.widgetset - Function

Package	**display**
Module	**viewer**
Tool	**viewer**

Return the Widgetserver `tool` used by this `tool`

Synopsis
widgetset()

Returns
Widgetserver `tool`

Description

This function simply returns the Widgetserver `tool` that is used by this Viewer `tool` to build GUIs. It is generally reserved for internal use.

1.3.2 viewercanvasmanager - Tool

Package	**display**
Module	**viewer**

Tool for managing the canvases of Viewerdisplaypanel `tools`
include "viewercanvasmanager.g"

Constructors

viewercanvasmanager	Create a Viewercanvasmanager `tool`

Functions

done	Close down (destroy) this Viewercanvasmanager `tool`
getoptions	Get the current canvas settings
gui	Create a new graphical user interface for this `tool`
restoreoptions	Restore a previously saved state of this `tool`
saveoptions	Save the state of this `tool`
setoptions	Install new canvas settings
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which manages this `tool`

Description

This `tool` provides services to manage and manipulate the drawing canvases of Viewerdisplaypanel `tools`. At present, this means adjusting margin spacing (to make room for axis labels, for example), selecting the number of panels for channel maps, selecting screen or paper colors (that is, white-on-black or black-on-white), and adjusting the number of colors used by colormaps on the Viewerdisplaypanel `tools`.
The graphical user interface provided by the Viewercanvasmanager `tool` is shown in Figure 1.1. A Viewercanvasmanager `tool` GUI for a Viewerdisplaypanel `tool` can be placed on the screen by selecting Canvas manager from the File menu of the Viewerdisplaypanel `tool`, or by invoking the newcanvasmanagergui function of the Viewerdisplaypanel `tool`. A single Viewercanvasmanager `tool` can have multiple GUIs on the screen at any one time, but this can become confusing to the user. The Viewercanvasmanager `tool` GUI makes use of an autogui to allow modification of the various canvas parameters.

Graphical User Interface

The graphical user interface consists of three rollups:

- Geometry This rollup controls the margins around the image. These are reserved for axis labels. If you don't use labelling you can set them to 0.
- Layout This rollup is only of interest to you if you want to create channel maps, i.e. draw frames of consecutive channels of a data cube on one display. These will be arranged from top-left to right-bottom, starting at

the frame selected in the animator. The sliders control the number of displays in x and y as well as the space between them. Note that this space is on top of the margins in Geometry. To use these appropriately set all margins to zero. This rollup won't show up if a viewercanvasmanager tool is associated with a viewerslicedp tool.

- Basic settings Basic settings control the numer of colors (index mode) and the background color of the display. The background color by default is black, but the options `Use paper colors` can be set to true so that it would look like a PS print out. The orientation of Color Wedges can also be selected here.

Canvas settings can be saved to and restored from the AIPS++ "inputs table" using the Save and Restore buttons.
The viewercanvasmanager tool is context sensitive to some degree. A Viewerdisplaypanel in RGB mode for example has a slightly different interface and on a True Color display in RGB mode you won't get an option to modify the number of colors.

viewercanvasmanager - Constructor

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Create a Viewercanvasmanager `tool`

Synopsis
viewercanvasmanager(displaypanel)

Arguments

displaypanel	The Viewerdisplaypanel `tool` which will use this Viewercanvasmanager `tool` Allowed:Viewerdisplaypanel `tool`

Returns
Viewercanvasmanager `tool` or fail

Description

This is the only constructor for Viewercanvasmanager `tools`. It constructs (creates) a Viewercanvasmanager `tool` which can subsequently be used to provide canvas management to a Viewer `tool`.
This constructor should not be called by users: it is called during construction of Viewerdisplaypanel `tools`. The Viewercanvasmanager `tool` for a given Viewerdisplaypanel `tool` can be retrieved by calling its canvasmanager function.

---

viewercanvasmanager.type - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Return the type of this `tool`

Synopsis
type()

Returns
'viewercanvasmanager'

Description

This function simply returns the string `'viewercanvasmanager'`, which can be used to identify this `tool` as a Viewercanvasmanager `tool`. It is generally reserved for internal use.

---

viewercanvasmanager.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Return the Viewer `tool` which manages this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which owns the Viewerdisplaypanel `tool` that constructed and uses this Viewercanvasmanager `tool`. It is generally reserved for internal use.

---

viewercanvasmanager.done - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Close down (destroy) this Viewercanvasmanager `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewercanvasmanager `tool`, calling this function will free up its resources. Any graphical user interfacesassociated with this `tool` will be removed from the screen.
It is unwise to close down a Viewercanvasmanager `tool` by an explicit call to this function. It is generally reserved for internal use by the Viewerdisplaypanel `tool` which uses this Viewercanvasmanager `tool`.

---

viewercanvasmanager.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Create a new graphical user interface for this `tool`

Synopsis
gui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: dws

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to view and edit the canvas settings that are managed by this Viewercanvasmanager `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.

---

viewercanvasmanager.getoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Get the current canvas settings

Synopsis
getoptions()

Returns
autogui full-form parameter record

Description

This function can be used to determine the current canvas settings according to this Viewercanvasmanager `tool`. The returned record is simply a full-form autogui parameter record (see the autogui documentation).
The parameters that make up the returned record are described in the documentation for the getoptions function of the Viewerdisplaypanel `tool`.

---

viewercanvasmanager.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Install new canvas settings

Synopsis
setoptions(options)

Arguments

options	autogui short-form parameter record describing the requested state Allowed:record

Returns
T

Description

This function should be used to set the state of the canvas managed by this `tool`, according to the field names of the record returned by the getoptions `tool function`. Installing new options on a Viewercanvasmanager `tool` will generally cause the refresh (that is, a total re-drawing) of the Viewerdisplaypanel `tool` which it manages.

---

viewercanvasmanager.saveoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasmanager**

Save the state of this `tool`

Synopsis
saveoptions(setname)

Arguments

setname	name under which to store this state Allowed:string

Returns
T (success) or F (failure)

Description

This function saves the current state (options) of this Viewercanvasmanager `tool` in the AIPS++ `Inputs Table`. The options are stored as inputs for the Viewercanvasmanager `tool`, with the "method name" set to the `setname` argument given in the call to this function.

Example
An example should show how this can be used:

```
- mv := viewer();
- mdp := mv.newdisplaypanel();
- mcm := mdp.canvasmanager();
- mcm.setoptions([leftmarginspacepg=0]);
- mcm.saveoptions('my canvas options');
```

---

viewercanvasmanager.restoreoptions - Function

Package **display**
Module **viewer**
Tool **viewercanvasmanager**

Restore a previously saved state of this `tool`

Synopsis
restoreoptions(setname)

Arguments

setname	name of state to retrieve Allowed:string

Returns
T (success) or F (failure)

Description

This function tries to retrieve a previously saved state (options) of the given name, and if successful, applies them to this Viewercanvasmanager `tool`. The `setname` argument must correspond to a previously saved set of options, otherwise this function will fail.

1.3.3 viewercanvasprintmanager - Tool

Package **display**
Module **viewer**

Tool for managing the printing of canvases of Viewerdisplaypanel `tools`
include "viewer.g"

Constructors

viewercanvasprintmanager Create a Viewercanvasprintmanager `tool`

Functions

done	Close down (destroy) this Viewercanvasprintmanager `tool`
gui	Create a new graphical user interface for this `tool`
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which owns this `tool`
writeps	Write a PostScript file
writexpm	Write an X11 Pixmap file

Description

This `tool` provides services to produce copies of the drawing canvases of Viewerdisplaypanel `tools`. PostScript and bitmap output is available at present.

Graphical User Interface This tool controls two forms of hardcopy output:

PostScript for printing, and X-Windows pixmap format which can be easily converted to GIF, JPEG or other bitmap formats. To initiate printing of what is visible on the Viewerdisplaypanel `tool` simply press the Print... button at the bottom of the window, or select Print manager... from the File menu. This will bring up the GUI for this tool which allows you to set some basic output parameters, most of which are only important for PostScript output (as indicated by the *[PS]* prefix:

- Output file

  This can be left `<unset>`, and the Viewer will automatically create a unique filename for your output. Otherwise, just type in a specific name for your output file.

- [PS] Output media

  You may select to print to A4 or US Letter paper.

- [PS] Orientation

  Selection from portrait or landscape to orient the output on the selected paper.

- [PS] Resolution

  You may choose the resolution of your output here. Bear in mind that the size of the output file for raster images will generally grow as the square of this number, and likewise the speed with which the hardcopy will be generated and printed. For test output, generally leave the resolution quite low, and only increase it to your printer resolution for final hardcopy output. This setting enables you to tune the output to the exact resolution of your hardcopy device.

- [PS] Magnification

  You can scale your output by any factor between 0 and 1.

- [PS] Write EPS format?

  Set this option to True if you want encapsulated PostScript output.

Once you have filled in the desired options, you need to hit one of three buttons at the bottom of the Print manager GUI:

- Save XPM

  This button will write an X-Windows Pixmap copy of the Viewerdisplaypanel `tool` contents to a file on disk. The output will be an *exact* copy of what is displayed on-screen, and then a utility like 'xv' can be used to convert the output file to a GIF file, or a JPEG file, or whatever takes your fancy.

- Save PS

  This button will write a PostScript file which aims to duplicate what is seen on-screen in a higher-resolution mode suitable for printing. Every attempt is made to preserve the aspect ratio of the on-screen display.

- Print

  This button will first write a PostScript file as if the Save PS button had been pressed, and will then bring up a simple window which allows you to direct the output to a printer, to a previewer, or to cancel the operation.

Press Dismiss to cancel printing.

viewercanvasprintmanager - Constructor

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Create a Viewercanvasprintmanager `tool`

Synopsis
viewercanvasprintmanager(displaypanel)

Arguments

displaypanel	The Viewerdisplaypanel `tool` which will use this Viewercanvasprintmanager `tool` Allowed:Viewerdisplaypanel `tool`

Returns
Viewercanvasprintmanager `tool` or fail

Description

This is the only constructor for Viewercanvasprintmanager `tools`. It constructs (creates) a Viewercanvasprintmanager `tool` which can subsequently be used to provide canvas printing facilities to a Viewerdisplaypanel `tool`. **This constructor should not be called by users:** it is called during construction of Viewerdisplaypanel `tools`. The Viewercanvasprintmanager `tool` for a given Viewerdisplaypanel `tool` can be retrieved by calling its canvasprintmanager function.

---

viewercanvasprintmanager.type - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Return the type of this `tool`

Synopsis
type()

Returns
'viewercanvasprintmanager'

Description

This function simply returns the string `'viewercanvasprintmanager'`, which can be used to identify this `tool` as a Viewercanvasprintmanager `tool`. It is generally reserved for internal use.

---

viewercanvasprintmanager.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Return the Viewer `tool` which owns this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which owns the Viewerdisplaypanel `tool` that constructed and uses this Viewercanvasprintmanager `tool`. It is generally reserved for internal use.

---

viewercanvasprintmanager.done - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Close down (destroy) this Viewercanvasprintmanager `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewercanvasprintmanager `tool`, calling this function will free up its resources. Any graphical user interfacesassociated with this `tool` will be removed from the screen.
It is unwise to close down a Viewercanvasprintmanager `tool` by an explicit call to this function. It is generally reserved for internal use by the Viewerdisplaypanel `tool` which uses this Viewercanvasprintmanager `tool`.

---

viewercanvasprintmanager.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Create a new graphical user interface for this `tool`

Synopsis
gui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: unset

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to view and edit the printing settings of this Viewercanvasprintmanager `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.

---

viewercanvasprintmanager.writeps - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Write a PostScript file

Synopsis
writeps(filename, media, landscape, dpi, zoom, eps)

Arguments

filename	name of file to write Allowed:string or unset Default: unset
media	paper type Allowed:'A4' 'LETTER' Default: 'A4'
landscape	print in landscape mode? Allowed:boolean Default: F
dpi	resolution of output in dots per inch Allowed:integer Default: 100
zoom	scaling of output Allowed:float in (0,1] Default: 1.0
eps	generate encapsulated PostScript output Allowed:boolean Default: F

Returns
fail or name of file written

Description

This function writes a PostScript version of the current view on display in the Viewerdisplaypanel `tool` for which this `tool` provides canvas print management facilities. The PostScript output will have the same aspect ratio as that of the Viewerdisplaypanel `tool` displayed on screen (provided the screen pixel aspect ratio is close to 1.0), but may differ slightly in color use and resampling resolution for raster images. If the `filename` argument is left `unset`, then a sensible filename is generated by the Viewer `tool`. The `dpi` argument allows the caller to specify the resolution of the output.

---

viewercanvasprintmanager.writexpm - Function

Package	**display**
Module	**viewer**
Tool	**viewercanvasprintmanager**

Write an X11 Pixmap file

Synopsis
writexpm(filename)

Arguments

filename	name of file to write Allowed:string or unset Default: unset

Returns
name of file written

Description

This function writes an X11 Pixmap version of the current view on display in the Viewerdisplaypanel `tool` for which this `tool` provides canvas print management facilities. The resultant file is an *exact* copy of the screen display: it is actually written directly from the internal copy of the Viewerdisplaypanel `tool` contents. If the `filename` argument is left `unset`, then a sensible filename is generated by the Viewer `tool`.

1.3.4 viewercolormapmanager - Tool

Package	**display**
Module	**viewer**

Tool for managing colormaps used by the Viewer tool
include "viewercolormapmanager.g"

Constructors

viewercolormapmanager	Create a Viewercolormapmanager `tool`

Functions

colormap	Return a handle to a named colormap
colormapnames	Return the names of the available colormaps
done	Close down (destroy) this Viewercolormapmanager `tool`
getbrightness	Return the brightness setting of a named colormap
getcontrast	Return the contrast setting of a named colormap
getinvertflags	Get the current red, green and blue invert state
gui	Create a new graphical user interface for this `tool`
nmaps	Return the number of colormaps available
reset	reset the named colormap to its default state
resetall	reset all colormaps to initial state
setbrightness	Set the brightness of a named colormap
setcontrast	Set the contrast of a named colormap
setinvertflags	Set the red, green and blue invert state of a colormap
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which manages this `tool`

Description

This `tool` provides services to manage and manipulate colormaps for use by the Viewer `tool`. A colormap, or color look-up table, provides a set of colors which are normally used to represent varying intensity values in false color images (rasters). The Viewercolormapmanager `tool` provides sixteen basic colormaps, amongst them two greyscale-type colormaps and four rainbow-type colormaps. Each Viewer `tool` has one and only one Viewercolormapmanager `tool`, from which it obtains the names of the available colormaps, and also the actual colormaps when they are needed by Viewerdisplaydata `tools`.

Colormaps can be modified in a number of simple ways to achieve quite significant effects. Foremost, colormaps can be "fiddled:" a colormap can be stretched and shifted so that only a fraction of the colormap is actually used for drawing. Secondly, the contrast and brightness of the colors in the map can be modified. Thirdly, the individual red, green and blue components of the map can be independently inverted, leading to completely different colormaps.

Graphical User Interface

The graphical user interface provided by the Viewercolormapmanager `tool` is shown in Figure 1.2.
The left of Viewercolormapmanager `tool` GUI displays a listbox showing primary names of the colormaps known to the Viewer `tool` and therefore to the Viewercolormapmanager `tool`. Once a colormap is selected the sliders on the right-hand side can be used to modify various "fiddle" parameters of the selected colormap. The next three fields underneath can be used to invert the red, green and/or blue components in the selected colormap.
Finally, the `Dismiss` button is provided to remove this GUI from the screen.

viewercolormapmanager - Constructor

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Create a Viewercolormapmanager `tool`

Synopsis
viewercolormapmanager(viewer)

Arguments

viewer	The Viewer `tool` which will use this Viewercolormapmanager `tool` Allowed:Viewer `tool`

Returns
Viewercolormapmanager `tool` or fail

Description

This is the only constructor for Viewercolormapmanager `tools`. It constructs (creates) a Viewercolormapmanager `tool` which can subsequently be used to provide colormaps and colormap editing services to a Viewer `tool`.
This constructor should not be called by users: it is called during construction of the Viewer `tool`. The Viewercolormapmanager `tool` for a given Viewer `tool` can be retrieved by calling its colormapmanager function.

---

viewercolormapmanager.type - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the type of this `tool`

Synopsis
type()

Returns
'viewercolormapmanager'

Description

This function simply returns the string `'viewercolormapmanager'`, which can be used to identify this `tool` as a Viewercolormapmanager `tool`. It is generally reserved for internal use.

---

viewercolormapmanager.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the Viewer `tool` which manages this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which uses this Viewercolormapmanager `tool`, and probably constructed it. It is generally reserved for internal use.

---

viewercolormapmanager.done - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Close down (destroy) this Viewercolormapmanager `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewercolormapmanager `tool`, calling this function will free up its resources. Any graphical user interfacesassociated with this `tool` will be removed from the screen.
It is unwise to close down a Viewercolormapmanager `tool` by an explicit call to this function. It is generally reserved for internal use by the Viewer `tool` which uses this Viewercolormapmanager `tool`.

---

viewercolormapmanager.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Create a new graphical user interface for this `tool`

Synopsis
gui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: dws

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to view and edit the colormaps that are provided by this Viewercolormapmanager `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.

---

viewercolormapmanager.colormapnames - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the names of the available colormaps

Synopsis
colormapnames()

Returns
vector of strings

Description

This function should be used to obtain a list of the colormaps that are available for use. The names can then be used to actually get hold of the colormaps themselves.

---

viewercolormapmanager.colormap - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return a handle to a named colormap

Synopsis
colormap(name)

Arguments

name	name of requested colormap Allowed:string

Returns
colormap agent or fail

Description

This function can be used to get hold of a specific colormap agent. Once again, this is generally only used internally.

---

viewercolormapmanager.nmaps - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the number of colormaps available

Synopsis
nmaps()

Returns
integer

Description

This function simply returns the number of colormaps that it is presently managing.

---

viewercolormapmanager.getbrightness - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the brightness setting of a named colormap

Synopsis
getbrightness(name)

Arguments

name	name of colormap to query Allowed:string

Returns
float or fail

Description

This function returns the current brightness setting (in the range 0 to 1) of the named colormap.

---

viewercolormapmanager.setbrightness - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Set the brightness of a named colormap

Synopsis
setbrightness(name, value)

Arguments

name	name of colormap to modify Allowed:string
value	new brightness value Allowed:float in range 0 to 1

Returns
T or fail

Description

This function sets the brightness of the named colormap to the specified value.

---

viewercolormapmanager.getcontrast - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Return the contrast setting of a named colormap

Synopsis
getcontrast(name)

Arguments

name	name of colormap to query Allowed:string

Returns
float or fail

Description

This function returns the current contrast setting (in the range 0 to 1) of the named colormap.

---

viewercolormapmanager.setcontrast - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Set the contrast of a named colormap

Synopsis
setcontrast(name, value)

Arguments

name	name of colormap to modify Allowed:string
value	new contrast value Allowed:float in range 0 to 1

Returns
T or fail

Description

This function sets the contrast of the named colormap to the specified value.

---

viewercolormapmanager.getinvertflags - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Get the current red, green and blue invert state

Synopsis
getinvertflags(name)

Arguments

name	name of colormap to query Allowed:string

Returns
vector of booleans or fail

Description

This function returns a length 3 vector describing whether the red, green and blue components of the named colormap are inverted (T) or not (F).

---

viewercolormapmanager.setinvertflags - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

Set the red, green and blue invert state of a colormap

Synopsis
setinvertflags(name, value)

Arguments

name	name of colormap to modify Allowed:string
value	length 3 vector describing invert state Allowed:vector of booleans

Returns
T or fail

Description

This function sets the red, green and blue component invert states for the named colormap to those specified in the given length 3 vector.

---

viewercolormapmanager.reset - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

reset the named colormap to its default state

Synopsis
reset(name)

Arguments

name	name of colormap to restore to initial state Allowed:string

Returns
T or fail

Description

This function returns the contrast and brightness settings of the named colormap to 0.5, and sets each of the invert ”flags” to F.

---

viewercolormapmanager.resetall - Function

Package	**display**
Module	**viewer**
Tool	**viewercolormapmanager**

reset all colormaps to initial state

Synopsis
resetall()

Returns
T

Description

This function simply calls the `reset` function for each colormap known to this Viewercolormapmanager `tool`.

1.3.5 viewerdisplaydata - Tool

Package display
Module viewer

Tool for managing the drawing of AIPS++ data
include "viewerdisplaydata.g"

Constructors

viewerdisplaydata	Create a Viewerdisplaydata `tool`

Functions

addtouserchoicelist	Add a new possibility to an option which is of type 'userchoice'
classtype	Return the class type of this Viewerdisplaydata `tool`
ddd	Return the drawingdisplaydata agent
ddproxy	Return the true displaydata agent
displaytype	Return the displaytype
done	Close down (destroy) this Viewerdisplaydata `tool`
filename	Return a handle to the underlying image/array
getoptions	Return a record describing the state of this `tool`
hasbeam	has this Viewerdisplaydata `tool`a restoring beam defined
name	Return the (unique) name of this `tool`
newdisplaydatagui	Create a new graphical user interface for this `tool`
pixeltype	Return the data type of the pixels
restoreoptions	Restore a previously saved state of this `tool`
saveoptions	Save the state of this `tool`
setoptions	Set the state of this `tool`
type	Return the type of this `tool`
viewer	Return the Viewer `tool` which manages this `tool`
wedgedd	Return the Wedge displaydata agent
zlength	Return the length of the movie axes

Description

This `tool` is a fundamental component of the Viewer `tool`. It bundles up some piece of AIPS++ data together with functionality to draw that data, into a self-contained object which can be shuffled around by a Viewer `tool`, and manipulated by the user and/or programmer. For example, a specific Viewerdisplaydata `tool` may be a `tool` that knows how to draw a contour map (hereafter a 'contour') of a *Glish* array, or it may be a `tool` that can draw a false color image (hereafter a 'raster') of an AIPS++ Image.
The Viewerdisplaydata `tool` carries around instructions on how to draw itself, but it doesn't carry around a "canvas" on which to draw itself. Thus, to become visible to the user, a Viewerdisplaydata `tool` must be registered with one or more Viewerdisplaypanel `tools`. However, a Viewerdisplaydata `tool` may be created, manipulated and destroyed without ever being registered (and therefore displayed on-screen); why exactly anyone would like to do this is beyond me though! A Viewerdisplaydata `tool` is constructed from some AIPS++ data source, and a string defining the type of drawing, that is, 'raster' or 'contour'.
A Viewerdisplaydata `tool` is totally self-contained, and has "state". Its state consists of various attributes, some of which will not change during the lifetime of the `tool`, and many others which probably will. Examples of unchangeable state include the dimensionality of the AIPS++ Image or *Glish* array, and the type of drawing requested—a raster or a contour. Examples of changeable state include the many attributes which describe how the data should be translated into the requested drawing, such as the colormap to use for a raster, or the contour levels to use for a contour Viewerdisplaydata `tool`. The state of a Viewerdisplaydata `tool` can be modified from the command-line, or from a graphical user interface.
Normally, Viewerdisplaydata `tools` should be created with an appropriate call to a Viewer `tool`; many Viewerdisplaydata `tools` can be created and managed by a single Viewer `tool`.

Events

viewerdisplaydata - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Create a Viewerdisplaydata `tool`

Synopsis
viewerdisplaydata(viewer, name, displaytype, data)

Arguments

viewer	the Viewer `tool` which will manage this `tool` Allowed:Viewer `tool`
name	a unique name for this Viewerdisplaydata `tool` Allowed:string
displaytype	a string describing the type of drawing Allowed:'raster', 'contour', 'simpleaxes' or 'skycatalog'
data	The data source to draw, either an AIPS++ Image name, or a *Glish* array Allowed:string or array

Returns
Viewerdisplaydata `tool` or fail

Description

This is the only constructor for Viewerdisplaydata `tools`. It constructs (creates) a Viewerdisplaydata `tool` which can subsequently be registered on (and hence displayed on) Viewerdisplaypanel `tools`.
The `displaytype` argument can be:

- `'raster'`, for which `data` can be a (float or double) *Glish* array having two or more dimensions, or a string giving the name of an AIPS++ Image on disk.
- `'contour'`, for which `data` can be a (float or double) *Glish* array having two or more dimensions, or a string giving the name of an AIPS++ Image on disk.
- `'simpleaxes'`, for which `data` must be tt '¡none¿'.
- `'skycatalog'`, for which `data` must be a string giving the name of an AIPS++ Table on disk.

This constructor should not be called by users: it is called by the loaddata function of the Viewer `tool`, which returns a Viewerdisplaydata `tool`.

---

viewerdisplaydata.addtouserchoicelist - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Add a new possibility to an option which is of type 'userchoice'

Synopsis
addtouserchoicelist(vl)

Arguments

vl	record with fields 'param' which is the string identifier of the option, and 'newitem' which is the new option to add to the list. Allowed:record

Returns
T or fail

Description

This function is used to add a possibility to a choice list for display options which are of type 'userchoice'.

viewerdisplaydata.classtype - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the class type of this Viewerdisplaydata `tool`

Synopsis
classtype()

Returns
integer

Description

This function returns an integer indicating the class type of this Viewerdisplaydata `tool`. The possible return values are: `Display::Raster`, `Display::Vector`, `Display::Annotation`, and `Display::CanvasAnnotation`. The return value is used by the Viewer `tool` to decide whether position tracks should be shown for this Viewerdisplaydata `tool`.

---

viewerdisplaydata.ddd - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the drawingdisplaydata agent

Synopsis
ddd()

Returns
agent

Description
This function retruns a DrawingDisplayData agent. This for example is used to draw the beam if it is defined.

viewerdisplaydata.ddproxy - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the true displaydata agent

Synopsis
ddproxy()

Returns
agent

Description

This function returns a handle to the true displaydata agent underlying this Viewerdisplaydata `tool`. **This function is for gurus only.**

viewerdisplaydata.displaytype - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the displaytype

Synopsis
displaytype()

Returns
'raster', 'contour', 'vector', 'marker', or 'skycatalog'

Description

This function returns the displaytype chosen to construct this Viewerdisplaydata `tool`.

viewerdisplaydata.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Close down (destroy) this Viewerdisplaydata `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewerdisplaydata `tool`, calling this function will free up its resources. Any Viewerdisplaypanel `tools` on which this Viewerdisplaydata `tool` is registered will automatically unregister this `tool` and refresh their displays. Any graphical user interfacesassociated with this `tool` will be removed from the screen.

---

viewerdisplaydata.filename - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return a handle to the underlying image/array

Synopsis
filename()

Returns
string or array

Description

This function returns the hadle to the underlying data structure which can be the filename of the underlying image/table or an array. Viewerdisplaydata `tool`.

viewerdisplaydata.getoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return a record describing the state of this `tool`

Synopsis
getoptions()

Returns
record

Description

This function returns a record which describes the internal state of this Viewerdisplaydata `tool`. The record has a format that is compatible with the format used by the Autogui `tool`, and indeed is used to build an Autogui `tool` in the `newdisplaydatagui` function of this `tool`. The record is also described in a bit more detail and from an Autogui `tool` point of view here.
These options on the type of the Viewerdisplaydata `tool`. The record for examplee is slightly different for 'raster' and 'contour' Viewerdisplaydata `tool`.
A brief description of the fields which *may* exist in the returned record follows:

- `xaxis`: the axis name in the data to map to the X (horizontal) direction.
- `yaxis`: the axis name in the data to map to the Y (vertical) direction.
- `zaxis`: the axis name in the data to map to the Z (in-to-the-screen) direction. This axis is the "movie" axis.
- `haxisN`: the position along the Nth axis in arrays or Images having more than three axes.
- `aspect`: rule on how to control the aspect ratio of data pixels when they are mapped to display device pixels.
- `pixeltreatment`: whether pixels should be aligned with the edge of the canvas at their centres or edges.
- `axislabelswitch`: global switch to turn axis labelling on or off for this Viewerdisplaydata `tool`. Note that axis labels will only ever be drawn for the *first* registered Viewerdisplaydata `tool` for any given Viewerdisplaypanel `tool`.
- `titletext`: title text if axis labels are drawn.
- `titletextcolor`: color of the title text.
- `xaxistext`: label to appear along X axis.
- `xaxistextcolor`: color of X axis label text.
- `yaxistext`: label to appear along Y axis.
- `yaxistextcolor`: color of Y axis label text.
- `xgridtype`: rule for how to draw tickmarks or gridlines along the X axis.
- `xgridcolor`: color of X axis tickmarks and gridlines.
- `ygridtype`: rule for how to draw tickmarks or gridlines along the Y axis.
- `ygridcolor`: color of Y axis tickmarks and gridlines.

- `ticklength`: length of tickmarks (in mm?).
- `plotoutline`: draw a border around the plot?
- `plotoutlinecolor`: color of plot border.
- `labelcharsize`: size of characters for axis labelling.
- `labelposition`: the position of the world axes labels.
- `labelcharfont`: font for axis labelling.
- `labellinewidth`: line width for axis labelling.
- `axislabelpixelworld`: use pixel or world axis labels
- `axislabelabsrel`: relative or absolute coordinate labels
- `axislabeldirectionsystem`: the direction reference frame (e.g. J2000)
- `axislabeldirectionunit`: the unit to label the direction coordinate axes with
- `axislabelspectralunit`: the unit to label the spectral axis with
- `axislabelvelocitytype`: the velocity refernce frame (e.g. radio)
- `resample`: how to resample data pixels to screen pixels.
- `region`: the region (subimage) to display.
- `datamin`: the minimum data value to map to the colormap.
- `datamax`: the maximum data value to map to the colormap.
- `power`: scaling power for mapping data values to colors.
- `colormap`: which colormap to use.
- `wedge:` display a color wedge.
- `wedgeyaxistext`: the wedge axis label text, unset uses brighntes unit.
- `wedgeyaxistextcolor`: the wedge axis label text color.
- `wedgelabelcharfont`: font for wedge axis labelling.
- `wedgelabelcharsize`: size of characters for wedge axis labelling.
- `beam:` display the restoring beam.
- `beamcolor:` set the color of the beam.
- `beampos:` position the beam

- beamlinewidth: the linewidth for the beam
- beamoutline: draw the beam as an outline or filled
- levels: contour levels.
- scale: scale factor for the contour levels.
- type: whether the contour levels are absolute (in data units) or fractional (in range 0.0 to 1.0).
- line: linewidth of contours.
- dashpos: whether to dash negative-valued contours.
- dashneg: whether to dash positive-valued contours.
- color: contour color.
- complexmode: how to produce real data for display from a complex source.

Example
The best way to see the structure of the options record is to get hold of one and view it with the recordbrowserwidget. Alternatively, using the CLI only the misc module provides a function listfields

```
- include 'viewer.g';
- include 'recordbrowserwidget.g';
- mv := viewer();
- ar := array(1:137, 13, 17);
- mdd := mv.loaddata(ar, 'raster');
- opts := mdd.getoptions();
- mrb := recordbrowserwidget(therecord=opts);
# or
- dm.listnames(opts,F)
```

---

viewerdisplaydata.hasbeam - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

has this Viewerdisplaydata toola restoring beam defined

Synopsis
hasbeam()

Returns
T or F

Description

This function indicates if the underlying image has a restoring beam.

---

viewerdisplaydata.name - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the (unique) name of this `tool`

Synopsis
name()

Returns
string

Description

This function simply returns the (unique) name of this `tool` that was a construction argument.

---

viewerdisplaydata.newdisplaydatagui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Create a new graphical user interface for this `tool`

Synopsis
newdisplaydatagui(parent, uneditable, show, hasdismiss, hasdone, widgetset)

Arguments

parent	parent frame for GUI Allowed:agent or F Default: F
uneditable	parameters which should not be editable Allowed:vector of strings Default: “”
show	map the GUI to the screen immediately? Allowed:Boolean Default: T
hasdismiss	place a 'Dismiss' button in the GUI? Allowed:Boolean Default: F
hasdone	place a 'Done' button in the GUI? Allowed:Boolean Default: F
widgetset	Widgetserver `tool` to use for the GUI Allowed:Widgetserver `tool` Default: dws

Returns
`tool` having Viewerstandardgui interface

Description

Calling this function will create and return a new GUI which can be used to adjust the state of this Viewerdisplaydata `tool`. By judicious use of the function arguments, the graphical user interface can optionally be placed within existing frames, and left off-screen for later use. The interface of the return value conforms to the Viewerstandardgui interface.
The `uneditable` argument should be a vector of strings listing the internal names of the parameters which should not be editable by the user in the graphical user interface. eg. `uneditable=''xaxis yaxis colormap''` would prevent the user from modifying the displayed X and Y axes and the colormap of this Viewerdisplaydata `tool` in the GUI: they could still modify these parameters from the command-line via the setoptions function.

---

viewerdisplaydata.pixeltype - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the data type of the pixels

Synopsis
pixeltype()

Returns
float or complex

Description

This function simply returns the data type of the pixels in the data. At the moment this can be complex or float.

---

viewerdisplaydata.restoreoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Restore a previously saved state of this `tool`

Synopsis
restoreoptions(setname)

Arguments

setname	name of state to retrieve Allowed:string

Returns
T (success) or F (failure)

Description

This function tries to retrieve a previously saved state (options) of the given name, and if successful, applies them to this Viewerdisplaydata `tool`. It is perfectly acceptable to save options from a contour `tool` and sooner or later apply them to a raster `tool`: only the options relevant to a raster will be used. The `setname` argument must correspond to a previously saved set of options.

---

viewerdisplaydata.saveoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Save the state of this `tool`

Synopsis
saveoptions(setname)

Arguments

setname	name under which to store this state Allowed:string

Returns
T (success) or F (failure)

Description

This function saves the current state (options) of this Viewerdisplaydata `tool` in the AIPS++ `Inputs Table`. The options are stored as inputs for the Viewerdisplaydata `tool`, with the "method name" set to the `setname` argument given in the call to this function.

Example
An example should show how this can be used:

```
- mv := viewer();
- mdd := mv.loaddata('myimage', 'raster');
- mdd.setoptions([power=1.6, aspect='flexible']);
- mdd.saveoptions('trial options');
```

viewerdisplaydata.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Set the state of this `tool`

Synopsis
setoptions(newopts, emit, id)

Arguments

newopts	options record describing requested state Allowed:record
emit	emit the options that have been installed Allowed:Boolean Default: T
id	option request identification tag Allowed:Numeric Default: -1

Returns
T

Description

This function should be used to set the internal state of a Viewerdisplaydata `tool`. The format of the `newopts` record is a short form of the record returned by the `getoptions` function, and simply consists of one or more top-level fields, whose values are the values that should be set for each option. The `emit` argument should rarely be anything other than `T`, since this is how other interested parties can find out when the Viewerdisplaydata `tool` has been adjusted. The `id` argument is reserved for internal use, and should *never* be explicitly listed in a call to this function, other than from calls from within the Viewer `tool` which manages this `tool`.
Installing new options in a Viewerdisplaydata `tool` will generally cause the refresh (that is, a total re-drawing) of any Viewerdisplaypanel `tools` on which this `tool` is registered. This is because modifying the options will modify the appearance of the drawing of this data. Furthermore, there are some options

(for example, `region` and `complexmode`) which, when adjusted, will cause *internal* changes to other options, normally `datamin` and `datamax` for raster `tools`.

Example
A simple example will help clarify things:

```
- mv := viewer();
- mdd := mv.loaddata('myimage', 'raster');
- mdd.setoptions([power=1.6, aspect='flexible']);
```

---

viewerdisplaydata.type - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the type of this `tool`

Synopsis
type()

Returns
'viewerdisplaydata'

Description

This function simply returns the string `'viewerdisplaydata'`, which can be used to identify this `tool` as a Viewerdisplaydata `tool`. It is generally reserved for internal use.

---

viewerdisplaydata.wedgedd - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the Wedge displaydata agent

Synopsis
wedgedd()

Returns
proxy agent

Description
This function is for gurus only. So there won't be anby description.

---

viewerdisplaydata.viewer - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the Viewer `tool` which manages this `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` which manages this Viewerdisplaydata `tool`, and probably constructed it. It is generally reserved for internal use.

---

viewerdisplaydata.zlength - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaydata**

Return the length of the movie axes

Synopsis
zlength()

Returns
Integer

Description

This function simply returns the length of the movie (third) axis on this Viewerdisplaydata `tool`.

Option records

Here is a detailed description of the option record elements. This is based on how they appear in the Adjust.. GUI sorted by context.
These are the options available to *raster*, *contour* and *vector* Viewerdisplaydatas. Jump to display type specific options:

- raster
- contour
- vector
- marker
- skycatalog
- measurement set
- Display axes

  In this roll-up, adjustments can be made to select how the data is sliced for display, i.e. which axes of the data are mapped to X and Y on the screen. As well, a third axis (where appropriate) can be assigned to the Z (or movie) axis, which is the data axis along which progressive frames will be shown when a Viewerdisplaypanel is playing a sequence of frames. The new selection will not take place until the Apply button at the bottom of the adjustment window is pressed. If you have accidentally selected the same axis for more than one display axis, then this will be indicated by red crosses, and you should correct your error, and press Apply again.
- Hidden axes

  This roll-up is only present when the image or array has more than three axes, and is used to set the location in the data along all of the axes which are not mapped to either X, Y or Z (movie) axes in the Display axes roll-up. Simply sliding any of the scales should result in immediate updates to any Viewerdisplaypanel on which this Viewerdisplaydata is registered.

Note: For measurement sets, these sliders are incorporated into the Display axes rollup.

- Position tracking

  In this roll-up, adjustments can be made to how the coordinates under the pointer (mouse cursor) are displayed.

  By default, position tracking is enabled, so that as your move the pointer over the display, the data value and coordinate are shown in the status line beneath the display. This information will be displayed on any Viewerdisplaypanel on which this Viewerdisplaydata is registered.

  You can turn position tracking off with the switch called Position Tracking? at the bottom of the rollup.

  Position tracking can also be turned on for all Viewerdisplaydatas registered on a single Viewerdisplaypanel by pressing the *t* key, and turned off by pressing *Shift-t*, while the pointer is hovering over the particular Viewerdisplaypanel.

  Other options in this rollup are

  – Position tracking - Absolute or relative

    You may display absolute or relative (to the reference pixel) coordinates. This switch toggles between the two.

  – Position tracking - World or pixel coordinates

    You may display world or pixel coordinates. This switch toggles between the two.

  – Position tracking - Fractional or integral pixel coordinates

    A data lattice is discrete, but each pixel can be considered to extend over the range $i \pm 0.5$. E.g. pixel [20,30] in a 2D images covers the range [19.5,29.5] to [20.5,30.5]. By default pixel coordinates are presented integrally. If you like, this optional allows you to have them presented fractionally.

  – Position tracking - Spectral unit

    If the data contain a spectral axis, this menu will be included. It allows you to select the units for the presentation of the spectral coordinates (e.g. km/s or Hz etc). Conversion to km/s is only possible if the Spectral coordinate contains the rest frequency. In its absence, no conversion to km/s will be offered. You can set the rest frequency with the coordsys.setrestfrequency function.

– Position tracking - Velocity type

  If the data contain a spectral axis, this menu will be included. If you ask to see the spectral coordinate as a velocity, then this allows you to select the type of velocity definition (e.g. optical, radio etc.).

- Axis labels

  The parameters in this roll-up offer basic control over the axis labels (which by default are not visible). To turn them on, select True for Axis labelling & annotation. The other parameters in this roll-up are pretty much self explanatory, and include text entries for the axis labels (`unset` will give reasonable defaults in most cases), and control of whether tick marks or grid lines are drawn.

- Axis label properties

  In this roll-up, finer control over the axis label properties is provided.

  Here you have control over whether the axis labels are shown as world coordinates or image pixel coordinates (1-relative). You may select whether the labels are relative (to the reference pixel) or absolute.

  For Direction coordinates, you may select the units of the labels if they are relative. For Spectral coordinates you may select the units of the labels, and the velocity type. For other Coordinate types, you don't yet have further control over the units.

  You can also choose the Direction coordinate reference type of the following J2000, B1950, GALACTIC, ECLIPTIC and SUPERGAL. Your coordinate grid will reflect these types.

  You can also choose where to place your axis labels. The default *Auto* is usually sufficient, but other placements are available.

  Colors can be selected for various elements of the labelling, and the character font and size can be chosen. It is generally recommended to first select character font and size, and then pop on over to the Canvas Manager (available under the File menu of the Viewerdisplaypanel) to finely tune the positioning of the displayed image on the Viewerdisplaypanel. In a future release of AIPS++, this geometry adjustment will be automatic.

- Basic Settings This roll-up is rolled down by default! It has the parameters that most dramatically alter the generation of the raster map itself from the image or array data. The elements of this roll-up are:

  – Basic Settings - Aspect ratio

    This option controls the dimensions with which data pixels are drawn to the screen. Fixed lattice means data pixels will be mapped to square pixels on the screen, in as much as the screen pixels themselves are square. Fixed world means that the aspect ratio of the pixels according to the coordinate system of the image will be honoured. Finally, selecting flexible will allow the raster map to stretch independently in each direction to fill as much of the Viewerdisplaypanel as possible. The default setting is fixed world.

– Basic Settings - Pixel treatment

  This option controls how individual data pixels are drawn to the screen. Selecting center means that pixels at the edge of the display will be drawn only from the centres inwards. That is, down the left hand border of the raster map, only the right-hand half of the pixels will be visible, and along the bottom of the raster map, only the upper half of the pixels will be visible. So for center, the raster map is drawn from the center of the bottom left pixel in the selected data to the center of the top right pixel in the selected data.

  Selecting edge (the default) will change this behaviour such that all data pixels will be drawn fully on the screen. This would be useful for cases where only one of the display axes only has length one pixel.

– Basic Settings - Resampling mode

  This setting controls how the data are resampled to the resolution of the screen. The default, nearest, means nearest neighbour resampling, whereby each pixel on the screen is colored according to the intensity of the nearest corresponding data pixel. The alternative, bilinear, applies a bilinear interpolation to produce smooth-looking images independent of the screen pixel dimensions of the data pixels. While bilinear resampling normally produces a better looking image, it can be significantly slower than nearest neighbour resampling, so be aware of this.

– Basic Settings - Complex mode

  If the data source (image or array) is Complex, then this menu allows you to select whether you wish to see the magnitude, phase, real or imaginary part.

– Basic Settings - Image region

  This region entry is only available for raster maps of images, and allows the selection of a particular part of the image for display. A **region** can be placed in the region entry, and the easiest way to do this is to bring up the Regionmanager using the spanner menu to the right of the entry. Simply create a **region** with the Regionmanager, and send it to this control by pressing Send or Send & dismiss in the Regionmanager.

– Basic Settings - Mask expression

  This entry is only available for raster maps of images, and allows you to specify a Boolean expression mask which is applied On-The-Fly (OTF). Thus you can apply an extra mask to the image in addition to its current persistent default mask.

The syntax is that of the Lattice Expression Language and use of the mask expression entry widget is equivalent to use of the `mask` keyword that is available in many Image `tool` functions and provides an OTF mask capability. See note 223 for details on LEL. Also see the Image tool and the Image Analysis chapter of this document for extra information.

For example, let us say you have an image on disk called *3c273.im* and you have made a raster display of it with the Viewer. You then decide to mask the image as bad where all pixels are negative. You could do this with the Image tool function calcmask which would calculate and store a new mask for you. But perhaps you want to try several different masks before making one persistent. In this case, in the entry widget, you would enter the same expression that you would have provided to calcmask.

```
'3c273.im' > 0
```

and the display would be updated so that all non-negative pixels are good and all negative pixels are bad (masked). Thus, where the expression is True, the pixels are displayed. (Note: use the image filename, rather than the name of any image tool you have created, in these expressions).

When you use the Mask expression in combination with an Image region, the mask expression is applied first, and then the region. This means the shape of the mask expression must match that of the image being displayed.

Also note that if the image you are displaying has an underlying pixel mask (see the Images module), then the final mask that is applied is the union of that mask and the mask expression.

Finally, There is one important difference between using the expression mask entry widget and the `mask` keyword in Image `tool` functions. With the Image tool, you can embed Image `tool` names in your expressions. However, when you use the mask expression entry widget in the adjustment GUI, you must **always** refer to the image disk file name. You cannot use the Image `tool` name.

Now to the *raster* specific options.

– Color Wedge - Color Wedge

  Display a color wedge (bar). **Note: In RGB mode the wedge isn't supported yet.** The wedge will align itself to the image only if you set Basic Settings - Aspect Ratio to *flexible*, or the displaypanel is resized to fir the new aspect. The orientation of the wedge can be changed in the `viewercanvasmanager`.

– Color Wedge - Y axis label

  The axis label to show on the wedge. The default *unset* shows the brightness unit if available.

– Color Wedge - Y axis label color

  Color of the axis label text.

– Color Wedge - Character size

  Size of the axis label text in terms of pgplot character height.

– Color Wedge - Character font

  The (pgplot) font to use for the label.

– Basic settings - Color mode

  This option is specific to raster maps, and controls whether the pixel intensities for this Viewerdisplaydata are mapped to colors selected from a colormap (chosen elsewhere in the adjustment panel), or are instead mapped to one of the following color channels: red, green, blue, hue (color), saturation (color strength) or value (brightness). Use of any color mode requires that the Viewerdisplaydata be registered on a suitable Viewerdisplaypanel. That is, color mode *colormap* rasters must be registered on an *Index* Viewerdisplaypanels, color mode *red, green* or *blue* rasters will only draw when registered on *RGB* Viewerdisplaypanels, and color mode *hue, saturation* and *value* rasters will only draw when registered on *HSV* Viewerdisplaypanels.

– Basic Settings - Data Range

  You can use the type in entry box provided to set the minimum and or maximum data values being mapped to the colormap. Additionally, the limits can be set graphically by using the histogramgui window. This can be opened by selecting 'Histogram' from the spanner menu. For very high dynamic range images, you will need to type in a number much less than the maximum to see the low-level brightness pixels.

– Basic settings - Histogram equalisation

  The default value for this option is False. When enabled for raster maps, some internal optimisation of colors from the colormap (or color channel) is made. The number of colors allocated to a given range of data is proportional to the fraction of pixels whose pixel intensities lie in that range. Thus color cells are allocated where they are needed most.

– Basic settings - Scaling power cycles

  This option offers further control over the mapping of data values to color cells from the colormap. The overall process in selecting a color for a particular pixel is as follows: the data value of the pixel is clipped lie between the data minimum and maximum as specified elsewhere in this Basic settings roll-up. Then, according to this option which is described below, this clipped pixel intensity is mapped to an index between 0 and the number of colors available in the selected colormap. The color which is in the colormap at this position, which itself is controlled by the current stretch, shift, contrast, brightness and invert settings of the colormap, is then drawn for this particular pixel.

So, the scaling power cycles option controls the mapping of clipped pixel intensities to color look-up values. When this is set to zero, a straight line connects `(domainMin, rangeMin)` and `(domainMax, rangeMax)` on a plot where the domain is the input data values constrained to the range given by the data min and max sliders, and the range runs from 1 to number of colors in the selected colormap. See Figure 1.3 for sample curves.

As the scaling power decreases from zero, the curve deviates from the straight line, rising *above* it, such that an increasing fraction of the available colormap is used for data values close to the data minimum. This curve is calculated using a log function.

As the scaling power increases from zero, the curve deviates from this straight line, falling *below* it, such that an increasing fraction of the available colormap is used for data values closer to the maximum. The curve is calculated using an exponential function. [1]

– Basic settings - Colormap

You can select a range of different colormaps here. Selections here only take effect if the color mode option is set to *colormap*.

– Basic settings - Colorbar

You can choose to have a color bar (wedge) displayed next to your data. At the moment the wedge can be placed to the right-hand side of the image. **Note: In RGB mode the wedge isn't supported yet.** The wedge will align itself to the image only if you set Aspect Ratio to *flexible*.

Following are the *contour* specific options.

– Basic settings - Contour levels

This option is simply where you enter the levels at which you would like contours to be drawn. There are two main ways to enter contour levels:

1. Simply place an integer vector in Contour levels, eg. `1:9`, set Level type to frac, and then Contour scale factor to 0.1. This

[1]The actual functions are computed as follows:

For negative scaling values (say -p), the data is scaled linearly from the range (dataMin – dataMax) to the range $(1 - 10^p)$. Then the program takes the log (base 10) of that value (arriving at a number from 0 to p) and scales that linearly to the number of available colors. Thus the data is treated as if it had p decades of range, with an equal number of colors assigned to each decade.

For positive scaling values, the inverse (exponential) functions are used. If p is the (positive) value chosen, The data value is scaled linearly to lie between 0 and p, and 10 is raised to this power, yielding a value in the range $(1 - 10^p)$. Finally, that value is scaled linearly to the number of available colors.

way, contours are drawn at 10, 20, . . . , 90 per cent, where 0 per cent corresponds to the lowest pixel intensity in the data, and 100 per cent to the highest.

2. Set Level type to abs, Contour scale factor to 1, and then enter the absolute levels in the natural units of the data in Contour levels, separated by commas.

– Basic settings - Contour scale factor

  The contour levels given will be multiplied by the factor entered in this option before being drawn.

– Basic settings - Level type

  This option indicates whether the vector of contour levels, specified in the Contour levels box, multiplied by the scale factor, are absolute levels (that is, in the native units of the image), or are instead fractional levels, where 0 is mapped to the lowest pixel intensity in the image, and 1 is mapped to the highest.

– Basic settings - Line width

  This option controls the thickness of the contour lines. While only integer values will show any effect on-screen, PostScript output will be affected by the fractional part of the line width.

– Basic settings - Dash negative contours?

  Set this option to True if you would like contours of negative values in your data to be dashed rather than solid lines.

– Basic settings - Dash positive contours?

  Set this option to True if you would like contours of positive values in your data to be dashed rather than solid lines.

– Basic settings - Line color

  Select a color for the contours.

Now to the *vector* specific options

Vector map adjustments are based on those Basic settings. The following options are not available: resampling mode, complex mode and data minimum & maximum. In their place, the following options are added in the Basic settings section:

– Basic settings - Phase Type

  If the data source is Complex ($z = R + iI$) this option is available. It is a menu with two values: values normal and polarimetric mean the phase (position angle) is computed from $atan2(I, R)$ and $atan2(I, R)/2$, respectively. The latter is appropriate when the Complex data holds, say, the Complex linear polarization $z = Q + iU$ which has a two-fold symmetry. The default is polarimetric.

– Basic settings - Constant amplitude ?

  If the data source is Complex, this option allows you to discard the amplitude information and show all vectors with equal length.

– Basic settings - Debias amplitude ?

  If the data source is Complex ($z = R + iI$) this option is available. The default is no debiasing. The amplitude is computed from $\sqrt{(R^2 + I^2)}$. Now if the real and imaginary parts are drawn from independent gaussian variables (as is the case for the Complex polarization $z = Q + iU$), then even in the absence of signal, the mean amplitide is positive. This is noise bias. This option attempts to remove it to first order via $amp = \sqrt{(R^2 + I^2 - variance)}$.

  To do this, you need the variance of the noise (assumed the same for real and imaginary). See the next option.

– Basic settings - Variance for debiasing

  If the data source is Complex, this option is available. It is only used if the debias option is selected (see above). The debiasing is accomplished via $amp = \sqrt{(R^2 + I^2 - variance)}$. By default, the variance is worked out for you from the image (the real and imaginary 10-sigma clipped variances are worked out separately and averaged). However, even though a clipping algorithm is used, the variance may not be accurate. If you know the variance of the noise accurately, enter it here.

– Basic settings - Amplitude scale factor

  By default, the amplitudes of the vectors are scaled so that the longest vector is of length 5% the length of the smallest display window dimension. This is an extra scale factor that you can apply to all the vectors to make them longer or shorter.

– Basic settings - X and Y increment

  By default, vectors are displayed for every third pixel in x and y. You can change that with these options.

– Basic settings - Extra rotation

  You can add an extra rotation (degrees) to the position angles here (e.g. to show a magnetic field position angle from electric field vectors).

– Basic settings - Show arrow heads ?

  This option allows you to put arrow heads on the vectors. However, if the Phase type is Polarimetric arrow heads are not displayed (two-fold symmetry makes them meaningless).

– Basic settings - Arrow head shape

  This option allows you to change the arrow head shape from a triangular head (value 0) to an open head. In between, you get 'barbed' shaped heads.

– Basic settings - Line width

  This option controls the thickness of the vector lines. While only integer values will show any effect on-screen, PostScript output will be affected by the fractional part of the line width.

– Basic settings - Line color

  Select a color for the vectors.

Now to the *marker* specific options

Marker map adjustments are based on those Basic settings. The following options are not available: resampling mode, complex mode and data minimum & maximum. In their place, the following options are added in the Basic settings section:

– Basic settings - Marker Shape

  By default, the marker shape is a square. At present, this is the only marker shape available.

– Basic settings - Scale factor

  By default, the amplitudes of the shapes are scaled so that the width of the shape 5% the length of the smallest display window dimension. This is an extra scale factor that you can apply to all the shapes to make them bigger or smaller.

– Basic settings - X and Y increment

  By default, markers are displayed for every third pixel in x and y. You can change that with these options.

– Basic settings - Line width

  This option controls the thickness of the marker outline. While only integer values will show any effect on-screen, PostScript output will be affected by the fractional part of the line width.

– Basic settings - Line color

  Select a color for the marker.

Skycatalog Viewerdisplaydatas are different from the others since they don't carry a coordinate system. Therefore the option record looks quite different.

- Advanced At the moment this roll-up only contains cache size control.

  – Advanced - Cache size

    This option should rarely need to be adjusted, especially for catalog overlay Viewerdisplaydatas. The value of this option specifies how many different views of the catalog overlay are stored in a cache, and can therefore be redrawn quickly on demand. A different view is generated when one or more options are altered.

- Label properties

  This roll-up provides options for selecting which column is used to label the sources, and for setting the properties of the labels themselves.

  - Label properties - Column listing names

    This option must be changed from `<none>` if you want labels to appear next to the source markers. Select the column which contains the names; the options offered are *only* the columns which contain strings.

  - Label properties - Line width

    Select the line width for the text labels; this effectively controls how bold the labels appear.

  - Label properties - Character font

    This option offers the choice of font for the labels.

  - Label properties - Character size

    Adjust the size of the text labels.

  - Label properties - Character color

    This option enables the user to select a color for the labels, which can of course be different to the color of the markers themselves.

  - Label properties - Label angle

    The labels can be drawn at any rotation, so this option allows the setting of that angle in integer degrees.

  - Label properties - Label X offset (char units) & Label properties - Label Y offset (char units)

    Labels can be offset vertically and horizontally from the markers. Careful use of these options and the label angle can be used to avoid most conflicts between labels and markers.

- Marker properties

  This roll-up provides options for controlling the appearance of the markers used to mark the positions of the sources.

  - Marker properties - Marker type

    This option is a poor but functional way to choose a marker for your sources from the catalog.

  - Marker properties - Marker size

    This option controls the size of the markers.

  - Marker properties - Marker color

    Markers can be drawn in any of the available colors selected from this menu.

- Basic settings

  In this roll-up, some general properties of the catalog overlay can be set.

  - Basic settings - "Where" query The catalog in use can be "pruned" by using this option. A valid table query language (TaQL - see AIPS++ Note 199) **WHERE** clause must be provided. The default value is `unset` so that all sources in the catalog are plotted. For example, you want to plot all sources in the Longitude range 0 ¡ long ¡ 1 degrees and the longitude column in the table is named "Long", the line to enter would be:

    ```
    Long > 0 && Long < 1
    ```

    Two nice features of TaQL are:

    * **Pattern matching** say all Galaxies which start with NGC. Specify the column name (here: "Name") and the pattern:
      `Name == pattern('NGC*')`
    * **regular expressions** for the same example:
      `Name == regex('^NGC.+')`

Raster measurement set Viewerdisplaydatas have some inputs similar to those for raster image display, with some additional interface for selecting and editing (flagging) the MS data.

- MS and Visibility selection

  - Visibility Type
  - Visibility Component
  - Moving Average Size

  This roll-up provides choice boxes for Visibility Type (Observed, Corrected, Model, Residual) and Component (Amplitude, Phase, Real, or Imaginary).

  Changes to Visibility Type or Component (changing from Phase to Amplitude, for example) require the data to be retrieved again from the disk into memory, which can be a lengthy process. When a large MS is first selected for viewing, the user must trigger this retrieval manually by pressing the 'Apply' button, after selecting the data to be viewed (*Field IDs* and *Spectral Windows*, below).

  Changing visibility type between 'Observed' and 'Corrected' can also be used to assure that data and flags are reloaded from disk. You should do this if you're using another flagging tool simultaneously (such as autoflag or msplot), so that the viewer sees the other tool's new edits and doesn't overwrite them with obsolete flags. The 'Apply' button alone won't reload unless something within the viewer itself requires it; in the future, a button will be provided to reload flags from the disk unconditionally.

You can also choose to view the difference from a running mean or the local RMS deviation of either Phase or Amplitude (this will not require a disk reload). There is a slider for choosing the nominal number of time slots in the 'local neighborhood' for these displays.

(Note: **Insufficient Data** is shown in the tracking area during these displays when there is no other unflagged data in the local neighborhood to compare to the point in question. The moving time windows will not extend across changes in either field ID or scan number, so you may see this message if your scan numbers change with every time stamp. An option will be added soon to ignore scan boundaries).

– Field IDs

– Spectral Windows

You can retrieve and edit a selected portion of the MS data by entering the desired Spectral Window and Field ID numbers into these boxes. Especially with large MSs, it makes sense to edit only one or two fields at a time, and to select spectral windows which all have the same number of channels. Doing this will also greatly reduce data retrieval times and memory requirements.

You can separate the ID numbers with spaces or commas; you do not need to enter enclosing brackets. Changes to either entry box will normally cause the selected MS data to be reloaded immediately from disk. If you wish, you can turn off 'autoapply', in order to change *both* spectral window and field selections, without reloading data after each change. In that case, press 'Apply' after entering all your selections.

If you select, say, spectral windows 7, 8, 23, and 24, the animator, slice position sliders, and axis labelling will show these as 1, 2, 3, and 4 (the 'slice positions' or 'pixel coordinates' of the chosen spectral windows). Looking at the position tracking display is the best way to avoid confusion in such cases. It will show something like: **Sp Win 23 (s 3)** when you are viewing spectral window 23, the third selected spectral window.

Changes to MS selections will not be allowed until you have saved (or discarded) any previous edits you have made (see *Flagging Options – Save Edits*, below).

Initially, all fields and spectral windows are selected. To revert to this 'unselected' state, choose 'Original' under the wrench icons next to the entry boxes.

- Display Axes

This rollup is very similar to that for images: it allows the user to choose which axes (from Time, Baseline, Polarization, Channel, and Spectral Window) are are on the display and the animator. There are also sliders here for choosing positions on the remaining axes.

Changing the choice of axis on one control will automatically swap axes, maintaining different axes on each control. Changing axes or slider/animator positions does not normally require pressing 'Apply'–the new slice is shown immediately. However, the display may be partially or completely grey in areas where the required data is not currently in memory, either because no data has been loaded yet, or because not all the selected data will fit into the allowed memory (see *Max. Visibility Memory*, below). When the cursor is over such an area, the following message shows in the position tracking area:

press 'Apply' on Adjust panel to load data

Pressing 'Apply' will reload the memory buffer so that it includes the slice you are trying to view.

The message **No Data** has a different meaning; in that case, there simply *is* no data in the selected MS at the indicated position.

For large measurement sets, loading visibility data into memory is the most time-consuming step. At present it is not possible to show the usual progress meter window, but similar feedback is provided in the console window. Again, careful selection of the data to be viewed can greatly speed up retrieval.

- Flagging Options

  These options allow you to edit (flag or unflag) MS data. The crosshair and rectangle region mouse controls are used on the display to pick the data region to edit; the options below determine how the edits will be applied.

  - Flagging Options - Show Flagged Regions...

    You have the option to display flagged regions in the background color (as in msplot or TVFLG) or to highlight them with color. In the former case, flagged regions look just like regions of no data. With the color option, flags are shown in shades of blue: darker blue for flags already saved to disk, lighter blue for new flags not yet saved; regions with no data will be shown in black.

  - Flagging Options - Flag or Unflag

    This setting determines whether selected regions will be flagged or unflagged. In contrast to msplot, this does *not* affect previous edits; it only determines the effect which later edits will have. Both flagging and unflagging edits can be accumulated and then saved in one pass through the MS.

  - Flagging Options - Flag/Unflag All...

    These flagging extent checkboxes have a similar operation to those in msplot. They allow you to extend your edit over any of the five data axes. For example, to flag *all* the data in a given time range, you would check all the axes *except* Time, and then select the desired time range with the region mouse control. Such edits will extend along the corresponding axes over the entire selected MS (whether loaded into memory or not) and optionally over unselected portions of the MS as well (*Use Entire MS*, below). Use care in selecting edit extents to assure that you're editing all the data you wish to edit.

– Flagging Options - Flag/Unflag Entire Antenna?

  This control can be used to extend subsequent edits to all baselines which include the desired antenna[s]. For example, if you set this item to 'Yes' and then click the crosshair on a visibility point with baseline 3-19, the edit would extend over baselines 1-3, 2-3, 3-3, 3-4, 3-5, 3-6, ... 3-¡nAntennas¿. Note that the second antenna of the selection (19) is irrelevant here–you can click anywhere on the display where the first antenna number is 3, to select all baselines which include antenna 3.

  This item controls the edit extent only along the baseline axis. If you wish to flag *all* the data for a given antenna, you must still check the boxes to flag all Times, Channels, Polarizations and Spectral Windows. There would be no point, however, in activating *both* this item and the 'Flag All Baselines' checkbox. You can flag an antenna in a limited range of times, etc., by using the appropriate checkboxes and selecting a rectangular region of visibilities with the mouse.

– Flagging Options - Undo Last Edit

– Flagging Options - Undo All Edits

  The 'Undo' buttons do the expected thing: completely undo the effect of the last edit (or all unsaved edits). Please note, however, that only unsaved edits can be undone. In contrast to msplot, there is no ability to revert to the flagging state at the start of the session, once flags have been saved to disk.

– Flagging Options - Use Entire MS When Saving Edits?

  "Yes" means that saving the edits will flag/unflag over the entire MS, *including* fields (and possibly spectral windows) which are not currently selected for viewing. Specifically, data within time range[s] you swept out with the mouse (even for unselected fields) will be edited.

  In addition, if "Flag/Unflag All..." boxes were checked, such edits will extend throughout the MS. Note that only unselected *times* (fields) can be edited *without* checking extent boxes for the edits as well. Unselected spectral windows, e.g., will *not* be edited unless the edit also has "Flag/Unflag All Spectral Windows" checked.

  Warning: Beware of checking "All Spectral Windows" unless you have also checked "All Channels" or turned "Entire MS" off; channel edits appropriate to the selected spectral windows may not be appropriate to unselected ones. Set "Use Entire MS" to"No" if your edits need to apply only to the portion of the MS you have selected for viewing. *Edits can often be saved significantly faster this way as well.*

Also note that checkboxes apply to individual edits, and must be checked before making the edit with the mouse. "Use Entire MS", on the other hand, applies to all the edits saved at one time, and must be set as desired before pressing "Save Edits".

– Flagging Options - Save Edits

The measurement set Viewerdisplaydata works like a text editor in that you see all of your edits immediately, but nothing is committed to disk until you press 'Save Edits'. Feel free to experiment with all the other controls; nothing but 'Save Edits' will alter your MS on disk. As mentioned previously, however, there is no way to undo your edits once they are saved, except by manually entering the reverse edits.

Also, *you must save* (or discard) *your edits before changing the MS selections.* If edits are pending, the selection change will not be allowed, and a warning will appear on the console.

If you close the tool, *unsaved edits are simply discarded*, without prior warning. It's important, therefore, to remember to save them yourself. You can distinguish unsaved flags (when using the 'Flags In Color' option), because they are in a lighter shade of blue.

The program must make a pass through the MS on disk to save the edits. This can take a little time; progress is shown in the console window.

- Axis Drawing and Labels
- Position tracking
- Color Wedge
- Basic settings

These four rollups contain entries similar to those for image Viewerdisplaydatas. Some of those entries were unnecessary and were omitted for simplicity here; others will be added in the future.

Together with the brightness/contrast and colormap adjustment mouse controls on the Viewerdisplaypanel, the following Basic settings are especially important for adjusting the color display of your data:

– Basic settings - Data minimum/maximum

At present, these are only sliders, without the histogram capabilities of the image Viewerdisplaydatas. Lowering the data maximum will help brighten weaker data values.

– Basic settings - Scaling power cycles

  This has exactly the same usage as for image Viewerdisplaydatas. Again, lowering this value often helps make weaker data visible. If you want to view several fields with very different amplitudes simultaneously, this is probably the first adjustment you should make.

– Basic settings - Colormap

  Greyscale or Hot Metal colormaps are generally good choices for MS data.

Adding a Color Wedge to the right of your display helps to show the effect of your other color scale adjustments. Its displayed scale and units are not always up-to-date at present (a known problem, to be fixed soon). In the meantime, moving the slider for Data minimum or maximum will cause the correct scale to be displayed.

- Advanced

  These settings can help optimize your memory usage, especially for large MSs, if you feel comfortable tweaking such things. A rule of thumb is that they can be increased until response becomes sluggish, when they should be backed down again.

  You can run the unix 'top' program and hit 'M' in it (to sort by memory usage) in order to examine the effects of these settings. Look at the amount of RSS (main memory) and SWAP used by the X server and 'glishtk' processes. If that sounds familiar and easy, then fiddling with these settings is for you. Otherwise, the default settings should provide reasonable performance in most cases.

  – Advanced - Cache size

    As with the Skycatalog, the value of this option specifies the maximum number of different views of the data to save so that they can be redrawn quickly. If you run an animation or scroll around zoomed data, you will notice that the data displays noticably faster the second time through because of this feature. Often, setting this value to the number of animation frames is ideal Note, however, that on multi-panel displays, each panel counts as one cached image.

    Large images naturally take more room than small ones. The memory used for these images will show up in the X server process. If you need more Visibility Memory (below) for a really large ms, it is usually better to forgo caching a large number of views.

  – Advanced - Max. Visibility Memory

    This option specifies how many megabytes of memory may be used to store visibility data from the measurement set internally. Even if you do not adjust this entry, it is useful to look at it to see how many megabytes are required to store your entire (selected) MS in memory. If the slider setting is above this, the whole selected MS will fit into the memory buffer. Otherwise, some data planes will be 'greyed out', and the selected data will have to be viewed one buffer at a time, which is somewhat less convenient. The largest process called 'glishtk' contains this buffer memory; it actually contains the entire display library, but the memory buffer can dominate its size.

1.3.6 viewerdisplaypanel - Tool

Package display
Module viewer

Tool for providing drawing windows (canvases)
include "viewerdisplaypanel.g"

Constructors

viewerdisplaypanel	Create a Viewerdisplaypanel `tool`

Functions

addgui	Add various GUI components to this Viewerdisplaypanel `tool`
adjust	Display adjustment panel for Viewerdisplaydata `tools`
animator	Return the Vieweranimator `tool` used by this `tool`
annotationdd	Return the agent of the annotation layer
annotator	Return the viewerannotations created by this Viewerdisplaypanel `tool` on construction
autoregister	Set or query the auto-register state of this Viewerdisplaypanel `tool`
canvasmanager	Return the Viewercanvasmanager `tool` used by this `tool`
canvasprintmanager	Return the Viewercanvasprintmanager `tool` used by this Viewerdisplaypanel `tool`
disablecontrols	Temporarily disable the controls on this Viewerdisplaypanel `tool`
done	Close down (destroy) this Viewerdisplaypanel `tool`
dying	Is this Viewerdisplaypanel `tool` in the process of destroying itself?
enablecontrols	Re-enable the controls on this Viewerdisplaypanel `tool`
getoptions	Return a record describing some modifiable state of this `tool`
hold	Prevent refreshes occuring on this Viewerdisplaypanel `tool`
holdsdata	Returns whether this Viewerdisplaypanel `tool` can hold Viewerdisplaydata `tools`
isregistered	Query whether a Viewerdisplaydata `tool` is registered on this Viewerdisplaypanel `tool`
leftframe	Return the left frame in the Viewerdisplaypanel `tool` window
lowerframe	Return the lower frame in the Viewerdisplaypanel `tool` window
newcanvasmanagergui	Create and (probably) show a new graphical user interface for this `tool`'s Viewercanvasmanager `tool`
newcanvasprintmanagergui	Create and (probably) show a new graphical user interface for this `tool`'s Viewercanvasprintmanager `tool`
newcmap	Query whether this Viewerdisplaypanel `tool` is using a new colormap
register	Register a Viewerdisplaydata `tool` for drawing

registercolormap	Explicitly register a colormap on the Viewerdisplaypanel `tool`
registerednames	Return a vector of the names of registered Viewerdisplaydata `tools`
registrationflags	Return a record describing which Viewerdisplaydata `tools` are registered on this `tool`
release	Allow refreshes to occur on this Viewerdisplaypanel `tool`
rightframe	Return the right frame in the Viewerdisplaypanel `tool` window
setoptions	Set the modifiable state of this `tool`
setzoom	Set a specific zoom
status	Return a record describing various attributes of the Viewerdisplaypanel `tool`
testpattern	Turn a test pattern on or off on the Viewerdisplaypanel `tool`
type	Return the type of this `tool`
unregister	Unregister a Viewerdisplaydata `tool` from this Viewerdisplaypanel `tool`
unregisterall	Unregister all Viewerdisplaydata `tools` from this Viewerdisplaypanel `tool`
unregistercolormap	Explicitly unregister a colormap on the Viewerdisplaypanel `tool`
unzoom	Unzoom the current view
upperframe	Return the upper frame in the Viewerdisplaypanel `tool` window
viewer	Return the Viewer `tool` which manages this `tool`
writexpm	Write an X11 Pixmap of the Viewerdisplaypanel `tool` to disk
zoom	Zoom by a given factor

General description

Description

This `tool` is a fundamental component of the Viewer `tool`. It provides a drawing area (or "canvas") on which Viewerdisplaydata `tools` can draw themselves. Multiple Viewerdisplaypanel `tools` can be placed on the screen, and each can have multiple Viewerdisplaydata `tools` registered to draw themselves. In this way, for example, multiple views of the same data can be simultaneously displayed, and contour Viewerdisplaydata `tools` can be drawn "on top" of raster Viewerdisplaydata `tools`, for example. Exact copies of Viewerdisplaypanel `tools` can be written to X11 Pixmap files, and high-quality PostScript representations of Viewerdisplaypanel `tools` can be saved and printed.
Viewerdisplaypanel `tools` can be adorned with various graphical user interface components, including a "controlbox" which allows the user to select controls like zooming and colormap fiddling, and a "tapedeck" which enables the user to manually or automatically move through data having more than two (non-redundant) axes.
Three points related to the registration of Viewerdisplaydata `tools` on Viewerdisplaypanel `tools` are worth making here:

1. *The order of registration of Viewerdisplaydata* `tools` *on a specific Viewerdisplaypanel* `tool` *is important.* The maximum area of sky, or position-velocity space, or of whatever coordinate system is being used, is determined solely by the first registered Viewerdisplaydata `tool`. All

other Viewerdisplaydata `tools` registered for display on a particular Viewerdisplaypanel `tool` simply draw those parts of their data which ought to be visible on the Viewerdisplaypanel `tool`. A consequence of this is that the zoom setting of the Viewerdisplaypanel `tool` must be reset to the full view whenever the first registered Viewerdisplaydata `tool` is removed, replaced, or re-read. Independent of the order in which Viewerdisplaydata `tools` are registered on a given Viewerdisplaypanel `tool`, the raster-type Viewerdisplaydata `tools` are always drawn first, followed by contours, and lastly axis labels.

2. *Axis labels are drawn by Viewerdisplaydata* `tools`*, and only by the first registered Viewerdisplaydata* `tool`*.* In the Viewer `tool`, axis labelling information belongs to the Viewerdisplaydata `tools` rather than to the Viewerdisplaypanel `tools`, which is commonly seen in other applications. This is because the only entity which knows *exactly* how to draw the axis labels for a particular piece of data is the Viewerdisplaydata `tool` which draws the data itself. Control over the appearance of axis labels is therefore available via the individual Viewerdisplaydata `tools`.

3. *Registering a specific Viewerdisplaydata* `tool` *on more than one Viewerdisplaypanel* `tool` *can reduce display efficiency.* The drawings made of data by a Viewerdisplaydata `tool` are cached, so that they only need to be recalculated under various circumstances, such as the settings for the Viewerdisplaydata `tool` changing, or the size of the Viewerdisplaypanel `tool` on which they are drawing changing. The caching will in general fail to operate efficiently if a Viewerdisplaydata `tool` is registered on more than Viewerdisplaypanel `tool`. This is best avoided by making another Viewerdisplaydata `tool` from the same dataset, and registering this second `tool` on the second Viewerdisplaypanel `tool`.

Additional to the `tool functions` described for the Viewerdisplaypanel `tool`, this `tool` also possesses the `tool functions` described in the Viewerstandardgui interface documentation. Note that some functions may not be documented and therefore shouldn't be used unless you really know what you are doing.

Graphical User Interface

Description

The standard graphical user interface provided by the Viewerdisplaypanel `tool` is shown in Figure 1.5. Surrounding the actual display area itself are five GUI components: the menu bar at the top, the button bar at the bottom, the tracking bar[s] just above the buttons, the controlbox on the left, and the (currently disabled) "tapedeck" control on the right. Applications

programmers are able to control which of these components appear on the Viewerdisplaypanel `tools` that they create, and hence throughout AIPS++, Viewerdisplaypanel `tools` can appear slightly different from the example given here.
Note: All AIPS++ GUIs have a so-called popup help. Just rest your mouse cursor on top of a GUI element (e.g. a button) and a little frame will appear outlining the funtionality of this element. If you then press and hold down the right mouse button you will get a detailed description. The information given there should be sufficient to let you operate that GUI element.

The menu bar

In Figure 1.5, a typical Viewerdisplaypanel `tool` is shown in all its glory. The menu bar along the top has just three menus:

- File, offering selections to create and display new Viewerdisplaypanel `tools`, access to various managers associated with the Viewer `tool`, and selections to finish the Viewerdisplaypanel `tool` itself, and to finish use of the entire Viewer `tool`.

  The File menu is where additional Viewerdisplaypanel `tools` can be created for use. Multiple Viewerdisplaypanels can be on-screen, and to the best of their ability they will share the available color resources. Three classes of Viewerdisplaypanel can be made: *Index* panels, on which normal raster maps can be displayed, with pixel intensities mapped to colors from a colormap; and *RGB* (red-green-blue) and *HSV* (hue-saturation-value) panels, on which multi-component raster maps can be put together. Viewerdisplaydatas which are set up to use a colormap (eg. 'Rainbow', 'Greyscale', etc.) will only draw when registered on an *Index* panel. Similarly, Viewerdisplaydatas which are set up to draw to the red, green or blue color channel will only draw themselves on RGB panels. And likewise, Viewerdisplaydatas which are set up to draw to the hue, saturation or value color channel will only draw when registered on HSV panels.

  For the *RGB* and *HSV* panel types, up to three raster maps can be displayed concurrently, overlaid on each other, with each mapped to a different color component. For example, an optical image could be displayed using the green component, and an X-ray image "overlaid" using the red component. In locations of strong X-ray emission and weak optical emission, the map will be predominantly red; weak X-ray emission coincident with strong optical emission will be predominantly green, and in between will be various levels of yellow.

  Two sub-classes of Viewerdisplaypanels can be made: *non-flashing* and *most colors* panels. The latter provides the most colors available, but with the cost of ugly "color flashing" behaviour as the mouse cursor is moved in to and out of the Viewerdisplaypanel window. The former provides generally less colors, but avoids this color flashing, and is generally more appropriate when analysis involving multiple windows is underway.

The menu also gives access to Viewerslicepanel `tools` in all color modes. Viewerdisplaydata `tool` with at least three axes can be displayed on these in slice mode. See the documentation for Viewerslicepanel `tool` for more.

- DisplayData, which is where various Viewerdisplaydata `tool` can be registered (or unregistered) for display on the Viewerdisplaypanel `tool`. Viewerdisplaydata `tool` are registered for display by ensuring that their checkbox is lit in this menu. When more than one Viewerdisplaydata `tool` is available for registration, additional convenience selections are placed at the bottom of the menu, enabling the user to register or unregister all the Viewerdisplaydata `tool` in the list.

- Tools contains a submenu ImageAnalysis ImageAnalysis tools all listen to regions and postions emitted by the region/positioning tool. The Tools menu also contains a Annotations... menu item, which opens the the Annotations GUI.

  You can interactively make a region (boxes and polygons) from the display with the cursor. First you select the appropriate button (or icon; box or polygon) from the viewer control buttons arranged vertically along the left-hand side of the display. Whatever mouse button you used to select the icon is now active for creating the region. Either drag out a box, or select vertices of a polygon.

  You create the region by double clicking inside of it. When you do this, the Viewerdisplaypanel `tool` will emit an event called `region`. See below for a description of that event.

  - ImageAnalysis This menu brings up a window with roll-ups for several image analysis operations. Each roll-up has a tab widget for each of the registered Viewerdisplaydatas. You can select the one you want to analyze by pressing the button showing the image name.

    * Statistics This roll-up shows image statistics on the selected image. It listens to event created by the region tools (of the control box. Whenever you emit such a region the fields in this GUI will be updated.
      When you generate an interactive region the statistics are computed only for the current displayed plane of the image. You can extend this spatial region over a range of animation-axis pixels via the Extend check box and by entering a pixel range in the adjacent entry box. For example, entering 10 20 would apply the spatial region to planes 10 through 20 along the animation axis.

Statistics can also be computed for the full data Full or the current plane Plane as selected with the animator tapedeck. Check the Robust button to also retrieve robust statistics. You can also bring up a pgplotter tool to show a statistic histogram by clicking the plot button or alternatively Auto-plot to have that plot updated automatically. The next row of buttons allow you to copy the values shown to the clip board (retrieve with `x := dcb.paste()`). Auto-copy automatically updates the clip board.

* Positions The Position rollup consists of a plot button (only if the image has more than 2 dimensions), a copy button, an autocopy switch and position fields to identify the current position of the cursor.
  The position rollup listen's to position events generated by it's parent Viewerdisplaypanel `tool` (via the crosshair tool) as well as motion events generated by the Viewerdisplaypanel `tool` from the movement of the mouse over the display area (when position tracking is turned on). When one of these events are received by the position rollup, the current position details are updated.
  You can plot the profile (at the cursor position and along the movie axis) with the button labelled `Plot`.
  When you submit a `region event` (see below), the plot will be updated with a new profile. This profile respresents the statistics in the submitted region along the movie axis of the image. The default statistic is `mean` and can be changed in via the `Adjust` button near the bottom of the profile window.
  You can copy the position details to the default Clipboard `tool` (**dcb**) with the `Copy` button. This copies a record to the clipboard with the format below. If you prefer, you can autocopy the positions to the Clipboard by checking the `Autocopy` button. Then, every time you generate a position, a record is automatically copied the Clipboard.
  Format of copied record:
  rec.ddname
  rec.position .pixel .world .numeric .string .intensity .value .mask
  rec.region .pixelblc .pixeltrc .stat
  rec.profile .cs .cs .x .pixel .abs .world .abs .y .data .mask .unit
  The profile can be customized via the `Adjust` button near the bottom of the profile window. Features such as autoscaling, rest frequency labelling and region profiles are available here.
  You can zoom in the profile window by using the left mouse button using the same method that is used to zoom in the viewer panel. Similarly, an `Unzoom` button is also available.

* Regions

Every time you generate a region, it is also inserted into the region entry widget (a check box flashes to indicate something went in) at the top of the regions rollup (roll it down to see the entry widget !). This enables you to copy it to the default Clipboard `tool` (**dcb**) and also paste into it from the default Clipboard `tool`. When a region is captured, a little indicator light next to the entry widget flashes to indicate something went in. There is also a `Show` button which prints the bounding box of the region to the logger and shows the regions already generated as a white outline. This button turns into `Hide` once it is pressed to disable the region drawing.
It is also possible to accumulate regions, and then make a compound region from all of them. This is done via the `Regions` rollup; roll it down and look at the row of buttons at the bottom. In this mode, if you have pressed the `Show` button, the regions are shown individually as they are generated.
The second button from the left controls the accumulation. If it is labelled `Start`, when you press it (and its label will then change to `Finish`), regions are internally accumulated (no `region` event is emitted and the region is not inserted in the region entry widget). When you have accumulated all the regions you want, you may press `Finish`. When accumulation mode is active, a check box to the right is lit and the number of regions accumulated displayed. Now a compound region is made from all the regions you accumulated according to the select you make from the left-most menu; you can make a union or intersection. When this region is created, the `region` event is emitted.

 * Slices Every time you use the polyline tool, a `polyline` event is generated containing the points of the polyline. These points are used here to
    · Generate an interpolated 1-D slice along the polyline and optionally plot it (use the `Plot` and `Autoplot` buttons).
    · For DirectionCoordinates, the distance and position angle between the polyline vertices is computed. If the polyline contains more than two points the sum over all distances and the sum over all angles is shown.

   Use the `Copy` and `Autocopy` buttons to transfer the slice, distance and position angle information to the clip board.
 * Summaries This part just shows the same Image.summary would show, wrapped up in a nice gui text area.

- Help, which provides selections with which assistance can be obtained on the Viewer `tool` itself, or more widely within AIPS++.

The button bar Along the bottom of the Viewerdisplaypanel `tool` are a few buttons, which are available for commonly executed tasks:

- The Adjust... button is used to make adjustments to the way the various Viewerdisplaydata `tool` draw themselves. Pressing Adjust just once will create adjustment windows for all Viewerdisplaydata `tool` that are presently registered on the Viewerdisplaypanel `tool`. Here is a detailed description of the elements of the Adjust GUI.
- The Unzoom button is a short-cut to fully unzooming the display—when pressed, the Viewerdisplaypanel `tool` will re-draw itself displaying as much of the *first-registered* Viewerdisplaydata `tool` as is possible.
- The Clear button is a short-cut to unregistering all Viewerdisplaydata `tool` which are presently registered on this Viewerdisplaypanel `tool`. When pressed, the Viewerdisplaypanel `tool` will be cleared.
- The Print... button is a short-cut to selecting Print... from the File menu, and will bring up the print window for this Viewerdisplaypanel `tool`, which naturally provides printing operations. For more go to Viewercanvasprintmanager `tool`.
- Finally, the Done button will remove this Viewerdisplaypanel `tool` from the viewer.

The tracking bar[s]

Separate tracking bars are generated for each Viewerdisplaydata `tool` registered on the Viewerdisplaypanel `tool`. They continually report the value and position of the point of the display area under the mouse cursor.
Pressing `space bar` in the display area will "freeze" the tracking displays or restore them again to continual tracking. Pressing the `L` key will send the displayed values to the logger. (An even more detailed record may also be sent to the clipboard, by using the Positions rollup under the Tools/ImageAnalysis menu item).
Note that if you have "click-to-focus" set in your windowing system, you may need to click on the displaypanel's window before it will recognize the key presses above. This can be confusing, since the tracking bar[s] will respond to mouse movement in any case.

The control box

The control box (on the left hand side of most Viewerdisplaypanel `tools`) provides a more convenient way to access the various controls available on Viewerdisplaypanel `tools`. Clicking on a button will assign that control to the mouse button which was used to click, over-riding any previous assignment to that mouse button.
For most tools the following actions apply:

- "Esc" key cancels the tool.

- "Double click" of the button assigned to the tool executes it (eg. with regions it sends out an event).
- "Handles" (little boxes on the edges) can be used to resize the tool.

From the top button downwards, the controls are:

- **Zooming:** The top-most button, depicting a magnifying glass with a plus sign inside, is the zoom control button. Zooming is accomplished by pressing down the selected mouse button at the start point, dragging the mouse away from that point, and releasing the selected mouse button when the zoom box encloses the desired zoom area. Once the button is released, the zoom rectangle can be moved by clicking inside it with the selected mouse button and dragging it around, and resized by selecting one of the handles at the corners and dragging.

  To zoom in, simply double click with the selected button inside the rectangle, and when the Viewerdisplaypanel `tool` is re-drawn, the area enclosed by the rectangle will be expanded to fill the Viewerdisplaypanel `tool`, in as much as other constraints allow this. Double clicking outside the rectangle, will result in a zoom out, whereby the previous full view will be scaled down to fit within the rectangle, in as much as other constraints allow this.

- **Panning:** The next button down, whose icon is the outline of a hand, is the panning control button. Panning is accomplished by pressing down the selected mouse button at the point you wish to move, dragging the mouse to the position where you want the first point moved to, and releasing the selected mouse button.

  Note: The arrow keys, Page Up, Page Down, Home and End keys, and scroll wheel (if any) can also be used to scroll through your data once you have zoomed in. For these to work, the mouse must be over the display panel drawing area, but no mouse tool need be active. All but the left and right arrows apply only to the vertical direction. However, you can use the Crtl key in conjunction with the arrows to move one page at a time in any direction.

- **Stretch-shift colormap fiddling:**
- **Brightness-contrast colormap fiddling:**
- **Positioning:** The fifth control is a crosshair icon, which enables the user to place a crosshair marker on the Viewerdisplaypanel to indicate a position. Depending on the context, the emitted positions may be used, for example, to display image spectral profiles or flag measurement set data. After selecting a mouse button for the crosshair, click on the Viewerdisplaypanel to place it, and drag to move it around. Double-click is not needed with this control; the position is emitted whenever the crosshair's assigned mouse button is pressed or the crosshair is moved. **Note: position selection is quite different from *position tracking*.**

- **Rectangle and Polygon region drawing:** Rectangular and polygonal regions can easily be constructed. The rectangle region is generated exactly the same way as the zoom rectangle, and is emitted by double clicking within the rectangle. Polygon regions can be constructed by progressively clicking the selected mouse button at the desired location of each vertex, and clicking in the same location twice to complete the polygon. Once constructed, it can be moved by dragging inside the polygon, and reshaped by dragging the various handles at the vertices. Double click within the polygon to emit the polygonal region. Compound image regions can be built up using these controls. The rectangular control can also be used to specify areas to be flagged in measurement sets.
- **Polyline drawing:** A polyline can be constructed with this tool. This tool is almost identical to the polygon region tool. Create the points by clicking at the positions wanted and the double-click to finish the line. this generates a polyline event.

The animation tapedeck

In its 'Normal' mode, the tapedeck-style controls down the right-hand side of most Display Panels provide basic movement between image planes, along a selected third dimension of the displaydatas. This set of buttons will only be enabled when the first-registered displaydata reports that it has more than one plane along the 'Z axis' specified in its 'Adjust' panel. In the most common case the animator controls the frequency channel being viewed. From the top down, the tape deck controls allow the user to:

- rewind to the start of the sequence (i.e. the first plane)
- step backwards by one plane
- play backwards, or repetitively step backwards
- stop any current play
- play forwards, or repetitively step forwards
- step forwards by one plane
- fast forward to the end of the sequence

Below the tapedack buttons is an editable text box indicating the current frame number and a sunken label showing the total number of frames. One can type a channel number into the current frame box to jump to that channel, for example, when 'Frequency' is on the 'Z axis'.
Below these is a right-arrow button; pressing it will display a popup frame containing sliders for setting the animation speed and the number of frames to skip when playing animations.
Below all these controls is a pair of radio buttons selecting animator mode: either 'Normal' (the default mode described above) or 'Blink'. In 'Blink' mode, when

more than one raster image is registered on the Display Panel, the tapedeck will control which one is being displayed at the moment. The images registered should cover the same portion of the sky, using the same coordinate projection. Several images can also be displayed side-by-side in this mode, with the same zoom applied to all. For details on operating the Display Panel in this mode, see the Display section of the *AIPS++ Cookbook, vol. 1*.

Events

The Viewerdisplaypanel `tool` *emits several events. A few of them can be of interest for the programmer.*

- pseudoregion *The pseudoregion event is emitted after double-clicking a region tool. The record this event holds contains unfromatted information about the region. Use* region *instead.*
- pseudoposition *The pseudoposition event is emitted after double-clicking a positioning tool. The record this event holds contains unfromatted information about the positon. Use* position *instead.*
- motion *The motion event returns a record containing (screen)pixel and world coordinate (and units) of the position the cursor is moved. It is for internal use only.*
- region *Note: This event only occurs when Viewerdisplaydata* `tools` *are registered.*

  An event called `region` *is emitted when you interactively create a region with the cursor. This event is always emitted, unless region accumulation mode is active (see above). When you terminate accumulation mode, the* `region` *event is then emitted.*

  The region event $value record is shown diagramatically below.

  ```
  \$value
    region                # The region tool
    type                  # The region type (box, polygon etc)
  ```

- position *Note : This event only occurs when Viewerdisplaydata* `tools` *are registered.*

  An event called `position` *is emitted when you interactively created a position with the cursor (see above).*

  The $value of the event is a record with the fields `position` *and* `profile`.

  The position event record is shown diagramatically below.

  ```
  \$value
    position              # Location of cursor
  ```

```
    pixel              #   Absolute integer pixel coordinate
    world              #   Absolute world coordinate
      numeric          #     as a numeric vector
      quantity         #     as vector of quantities
      string           #     as a formatted string
      measure          #     as measures (further subfields not shown)
    intensity          #   Intensity of image
      mask             #     mask value
      value            #     value of image (quanity)
      pixel            #     absolute integer pixel location
  profile              # The z-profile (if > 2D image)
    x                  #   abcissa
      pixel            #     pixel coordinate
        abs            #       absolute
        rel            #       relative
      nativeworld      #     world coordinate in native units
        abs            #       absolute
        rel            #       relative
      name             #     name of abcissa axis
      nativeunit       #     native abcissa unit
      currentworld     #     world coordinate as currently plotted (if plotted
)
      currentunit      #     abcissa unit as currently plotted
      currentveltype   #     velocity type as currently plotted
      currentabs       #     absolute or relative as currently plotted
      isSpectral       #     is the abcissa a spectral coordinate
    y                  #   ordinate
      unit             #     unit of ordinate
      data             #     ordinate values
```

- polyline *This event is generated by the* `polyline` *tool, when the user double-clicks, or drags and drops an existing polyline or its points.*
- statistics *Note: This event only occurs when Viewerdisplaydata* `tools` *are registered.*

  If the statistics rollup is rolled down, then statistics are written to the listboxes (`Plane`, `Full` *button press or region event emitted) then a* `statistics` *event is generated.*

  The statistics event record is shown diagramatically below. The `stats` *subfield is a record with the same format as that generated by the return record of function Image.statistics.*

```
\$value
  region          # The region from which the statistics were evaluated
  stats           # The statistics record
```

```
npts
sum
sumsq
min
max
mean
sigma
rms
blc
blcf
trc
trcf
minpos
minposf
maxpos
maxposf
```

viewerdisplaypanel - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Create a Viewerdisplaypanel `tool`

Synopsis

viewerdisplaypanel(parent, viewer, width, height, nx, nx, maptype, newcmap, mincolors, maxcolors, autoregister, holdsdata, show, widgetset)

Arguments

parent	*parent frame for the panel* *Allowed: agent or F* *Default: F*
viewer	*the Viewer* `tool` *which will manager this* `tool` *Allowed: Viewer* `tool`
width	*width in pixels of the drawing area* *Allowed: integer* *Default: 250*
height	*height in pixels of the drawing area* *Allowed: integer* *Default: 250*
nx	*number of panels in x* *Allowed: integer* *Default: 1*
nx	*number of panels in y* *Allowed: integer* *Default: 1*
maptype	*how colors will be addressed on this Viewerdisplaypanel* `tool` *Allowed: 'index' 'rgb' 'hsv'* *Default: 'index'*
newcmap	*install a private colormap? (ignored if a parent frame is provided)* *Allowed: boolean* *Default: F*
mincolors	*minimum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
maxcolors	*maximum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
autoregister	*accept auto-register data on this Viewerdisplaypanel* `tool`*?* *Allowed: boolean* *Default: F*
holdsdata	*is this Viewerdisplaypanel* `tool` *going to hold data?* *Allowed: boolean* *Default: T*
show	*show the Viewerdisplaypanel* `tool`*?* *Allowed: boolean* *Default: T*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns

Viewerdisplaypanel `tool` *or fail*

Description

This is the only constructor for Viewerdisplaypanel `tools`*. It constructs (creates) a Viewerdisplaypanel* `tool` *which can subsequently be used for the display of Viewerdisplaydata* `tools`*. If* `widgetset` *is left* `unset`*, then the Viewerdisplaypanel* `tool` *will use the Widgetserver* `tool` *used by its parent Viewer* `tool`*.*
This constructor should not be called by users: *it is called by the newdisplaypanel function of the Viewer* `tool`*, which returns a Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.addgui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Add various GUI components to this Viewerdisplaypanel `tool`

Synopsis

addgui(guihasmenubar, guihascontrolbox, guihasanimator, guihasbuttons, hasdismiss, hasdone, isolationmode)

Arguments

guihasmenubar	*add a menu bar (top)?* *Allowed: boolean* *Default: T*
guihascontrolbox	*add a control box (left)?* *Allowed: boolean* *Default: T*
guihasanimator	*add an animator/tapedeck (right)?* *Allowed: boolean* *Default: T*
guihasbuttons	*add buttons (bottom)?* *Allowed: boolean* *Default: T*
hasdismiss	*add a "Dismiss" button to the GUI?* *Allowed: unset or boolean* *Default: unset*
hasdone	*add a "Done" button to the GUI?* *Allowed: unset or boolean* *Default: unset*
isolationmode	*prevent user accessing other components of this Viewer* `tool`*?* *Allowed: boolean* *Default: F*

Returns
T

Description

This function can be used to add specified graphical user interface components to an existing Viewerdisplaypanel `tool`. *This function can only be called once per Viewerdisplaypanel* `tool`, *so get it right the first time!*
If both `hasdismiss` *and* `hasdone` *are given as* `T`, *then a warning will be emitted, and* `hasdismiss` *will be forcibly set to* `F`. *If both* `hasdismiss` *and* `hasdone` *are left* `unset`, *then* `hasdone` *will be forcibly set to* `T` *and* `hasdismiss` *to* `F`. *If only one of these two arguments is specified, and the other left* `unset`, *then the* `unset` *argument will default to the opposite of the given argument.*
If `isolationmode` *is* `T`, *then the GUI added to the Viewerdisplaypanel* `tool`*will not provide the user with the buttons or menu items to create new Viewerdisplaypanel* `tools` *or Viewerdatamanager* `tools`. *Typically this option should be used whenever a non-global Viewer* `tool` *is constructed.*
See the GUI description on hpw to operate the GUI.

---

viewerdisplaypanel.adjust - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Display adjustment panel for Viewerdisplaydata `tools`

Synopsis
adjust(displaydata)

Arguments

displaydata	*name of displaydata* *Allowed: string* *Default: F*

Returns
T

Description

In the default mode this function creates and shows a graphical user interface for each Viewerdisplaydata `tool` *which is registered for display on this Viewerdisplaypanel* `tool`*. If the name of a Viewerdisplaydata* `tool` *is given only it's graphical user interface will be displayed. The nice thing about creating the adjustment panels in this way is that they can be instantly remapped to the screen even if the user has hit the "Dismiss" button. Generally, other methods of obtaining the Viewerdisplaydata* `tool` *GUIs result in GUIs which have "Done" buttons.*

---

viewerdisplaypanel.animator - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the Vieweranimator `tool` *used by this* `tool`

Synopsis
animator()

Returns
Vieweranimator `tool`

Description

This function simply returns the Vieweranimator `tool` *which is used by this Viewerdisplaypanel* `tool`*. The return value can be used to manually and automatically step through the frames available to the Viewerdisplaypanel* `tool`*.*

---

viewerdisplaypanel.annotationdd - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the agent of the annotation layer

Synopsis
annotationdd()

Returns
drawingdisplaydata agent

Description
This function retruns an DDD agent which can be used to put annotations onto the canvas.

---

viewerdisplaypanel.autoregister - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Set or query the auto-register state of this Viewerdisplaypanel `tool`

Synopsis
autoregister(set)

Arguments

set	*Auto-register on or off* *Allowed: boolean or unset* *Default: unset*

Returns
boolean

Description

This function, if called with a boolean value for `set`*, will set the auto-register state of this Viewerdisplaypanel* `tool` *to the given value, and return the same value. Otherwise, it simply returns a boolean indicating whether auto-register is on (T) or off (F).*
Any Viewerdisplaydata `tools` *subsequently created via the loaddata function of the Viewer* `tool`*, with* `autoregister=T`*, will automatically be registered on this Viewerdisplaypanel* `tool` *if auto-register is true for this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.canvasmanager - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the Viewercanvasmanager `tool` *used by this* `tool`

Synopsis
canvasmanager()

Returns
Viewercanvasmanager `tool`

Description

This function simply returns the Viewercanvasmanager `tool` *which was constructed and is used by this Viewerdisplaypanel* `tool`*. The return value provides functions to modify various settings of the Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.canvasprintmanager - Function

Package **display**
Module **viewer**
Tool **viewerdisplaypanel**

Return the Viewercanvasprintmanager `tool` *used by this Viewerdisplaypanel* `tool`

Synopsis
canvasprintmanager()

Returns
Viewercanvasprintmanager `tool`

Description

This function returns the Viewercanvasprintmanager `tool` *that was constructed and is used by this Viewerdisplaypanel* `tool` *to provide printing services. The return value provides functions which can be used to control printing options.*

---

viewerdisplaypanel.viewer - Function

Package **display**
Module **viewer**
Tool **viewerdisplaypanel**

Return the Viewer `tool` *which manages this* `tool`

Synopsis
viewer()

Returns
Viewer `tool`

Description

This function simply returns the Viewer `tool` *which manages this Viewerdisplaypanel* `tool`*, and probably constructed it.*

---

viewerdisplaypanel.dying - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Is this Viewerdisplaypanel `tool` *in the process of destroying itself?*

Synopsis
dying()

Returns
boolean

Description

This function is provided so that tools created by (and belonging to) the Viewerdisplaypanel `tool` *can establish whether a request to destroy them came from the Viewerdisplaypanel* `tool`*itself. There are some tools, for example the Viewercanvasmanager* `tool`*, which will only allow their owning Viewerdisplaypanel* `tool` *to call their* `done` *function.*

---

viewerdisplaypanel.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Close down (destroy) this Viewerdisplaypanel `tool`

Synopsis
done()

Returns
T

Description

If you no longer need to use this Viewerdisplaypanel `tool`*, calling this function will free up its resources. Any Viewerdisplaydata* `tools` *which are registered on this Viewerdisplaypanel* `tool` *will be unregistered. Any graphical user interfacesassociated with this* `tool` *will be removed from the screen. An event with name* `'done'` *will be emitted when the Viewerdisplaypanel* `tool` *is destroyed.*

---

viewerdisplaypanel.upperframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the upper frame in the Viewerdisplaypanel `tool` *window*

Synopsis
upperframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *menubar should be placed.*

---

viewerdisplaypanel.lowerframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the lower frame in the Viewerdisplaypanel `tool` *window*

Synopsis
lowerframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *buttonbar should be placed.*

---

viewerdisplaypanel.leftframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the left frame in the Viewerdisplaypanel `tool` *window*

Synopsis
leftframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *controlbox should be placed.*

---

viewerdisplaypanel.rightframe - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the right frame in the Viewerdisplaypanel `tool` *window*

Synopsis
rightframe()

Returns
frame

Description

This function is generally used internally to obtain the frame into which the Viewerdisplaypanel `tool` *Vieweranimator* `tool` *tapedeck should be placed.*

---

viewerdisplaypanel.writexpm - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Write an X11 Pixmap of the Viewerdisplaypanel `tool` *to disk*

Synopsis
writexpm(filename)

Arguments

filename	*name of X11 Pixmap file* *Allowed: string*

Returns
T

Description

This function writes an exact replica of the display area part of the Viewerdisplaypanel `tool` *to disk in X11 Pixmap format. This format can be converted to "portable pixmap format" via the utility* **xpmtoppm**, *and then onwards to many other pixmap formats, for example TIFF, BMP, etc.*

---

viewerdisplaypanel.annotator - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the viewerannotations created by this Viewerdisplaypanel `tool` *on construction*

Synopsis
annotator()

Returns
This Viewerdisplaypanel `tools` *Viewerannotations* `tool`*tool*

Description

This function returns a reference to this Viewerdisplaypanel `tools` *Viewerannotations* `tool`*tool.This can be used to draw shapes onto the viewerdisplaypanel, open the annotations GUI, delete shapes etc. For more information see viewerannotations*

---

viewerdisplaypanel.newcmap - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Query whether this Viewerdisplaypanel `tool` *is using a new colormap*

Synopsis
newcmap()

Returns
boolean or unset

Description

This function returns `T` *if the Viewerdisplaypanel* `tool` *was constructed using a new (private) X Windows colormap,* `F` *if the Viewerdisplaypanel* `tool` *is using the system X Windows colormap, or* `unset` *if this information is unknown. It will be unknown in cases where the Viewerdisplaypanel* `tool` *was inserted into an existing frame, in which case the Viewerdisplaypanel* `tool` *will use the same X Windows colormap as the parent frame. (Recall that the constructor argument* `newcmap` *is ignored if a parent frame is provided.)*

---

viewerdisplaypanel.holdsdata - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Returns whether this Viewerdisplaypanel `tool` *can hold Viewerdisplaydata* `tools`

Synopsis
holdsdata()

Returns
boolean

Description

This function simply returns a boolean value indicating whether this Viewerdisplaypanel `tool` *was constructed with the* `holdsdata` *argument T or F. Viewerdisplaypanel* `tools` *for which* `holdsdata=T` *will have functions for registering and unregistering Viewerdisplaydata* `tools`*. The only use of Viewerdisplaypanel* `tools` without `holdsdata=T` *is for the Viewercolormapmanager* `tool` *graphical user interface as of this writing.*

---

viewerdisplaypanel.status - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a record describing various attributes of the Viewerdisplaypanel `tool`

Synopsis
status()

Returns
record

Description

This function returns some fairly low-level information about the Viewerdisplaypanel `tool`*. The record returned has upto two top-level fields:* `pixelcanvas`*, and* `paneldisplay` *if this Viewerdisplaypanel* `tool` *holds data. Fields which may exist in the* `pixelcanvas` *sub-record include the following (those tagged "dynamic" can change during the lifetime of a Viewerdisplaypanel* `tool`*):*

- `width`*: width of the entire display area ("pixelcanvas") in pixels (dynamic).*
- `height`*: height of the pixelcanvas in pixels (dynamic).*
- `depth`*: bit depth of the pixelcanvas, normally 8-bit.*
- `maptype`*: maptype of the pixelcanvas, normally 'index'.*
- `xdpi`*: approximate resolution in the X direction, in dots per inch (dpi).*
- `ydpi`*: approximate resolution in the Y direction.*
- `colortablesize`*: number of colors currently allocated to the pixelcanvas (dynamic).*
- `maxcolortablesize`*: maximum number of colors that could immediately be allocated to the pixelcanvas (dynamic).*
- `buildincolormaps`*: names of the colormaps that are known to the pixelcanvas.*
- `registeredcolormaps`*: a record having sub-fields* `names`*, which lists the names of the registered colormaps, and* `sizes`*, which lists the length of each of the named, registered colormaps. Normally the values in* `sizes` *should add up to* `colortablesize`*, since the colormaps will dynamically resize to fill the colortable.*

Fields which may exist in the `paneldisplay` *sub-record (they are all dynamic) include:*

- `nxpanels`*: The number of panels in x direction on this paneldisplay.*
- `nypanels`*: The number of panels in y direction on this paneldisplay.*
- `xorigin`*: The blcX of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `yorigin`*: The blcY of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `xsize`*: The extent in x of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `xsize`*: The extent in y of this paneldisplay on the pixelcanvas (0.0-1.0).*
- `linearblc`*: linear coordinates* `[x, y]` *of the bottom-left corner of each panel in the paneldisplay.*

- `lineartrc`*: linear coordinates* `[x, y]` *of the top-right corner of each panel in the paneldisplay.*
- `worldblc`*: world coordinates* `[x, y]` *of the bottom-left corner of each panel in the paneldisplay.*
- `worldtrc`*: world coordinates* `[x, y]` *of the top-right corner of each panel in the paneldisplay.*
- *axisunits: the units of the world axes*

---

viewerdisplaypanel.getoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a record describing some modifiable state of this `tool`

Synopsis
getoptions()

Returns
record

Description

This function returns a record which describes some of the internal state of this Viewerdisplaypanel `tool`*. The record differs from that returned by the* `status tool function` *in both format and content: it is suitable for construction of an Autogui* `tool`*, and contains explicitly modifiable parameters. A brief description of the fields which* may *exist in the returned record follows:*

- `colortablesize`*: the number of colors currently allocated to this Viewerdisplaypanel* `tools` *pixelcanvas.*
- `papercolors`*: a boolean specifying whether screen or paper colors (black/white or white/black background/foreground) should be used.*
- `leftmarginspacepg`*: left margin space in PgPlot character units.*
- `bottommarginspacepg`*: bottom margin space in PgPlot character units.*

- `rightmarginspacepg`: *right margin space in PgPlot character units.*
- `topmarginspacepg`: *top margin space in PgPlot character units.*
- `nxpanels`: *The number of panels in x direction on this paneldisplay.*
- `nypanels`: *The number of panels in y direction on this paneldisplay.*
- `xspacing`: *The space between adjacent panels in x direction.*
- `yspacing`: *The space between adjacent panels in y direction.*

---

viewerdisplaypanel.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Set the modifiable state of this `tool`

Synopsis
setoptions(options)

Arguments

options	*options record describing the requested state* *Allowed: record*

Returns
T
This function should be used to set the internal state of a Viewerdisplaypanel `tool`*, according to the field names of the record returned by the getoptions* `tool` `function`*. Installing new options on a Viewerdisplaypanel* `tool` *will generally cause the refresh (that is, a total re-drawing) of any Viewerdisplaydata* `tools` *registered on the Viewerdisplaypanel* `tool`*. This is because modifying the options will cause a change in the appearance of the data.*

---

viewerdisplaypanel.hold - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Prevent refreshes occuring on this Viewerdisplaypanel `tool`

Synopsis
hold()

Returns
T

Description

This function will prevent refreshing occuring on this Viewerdisplaypanel `tool` *until an equal number of calls to the release function have been made. Typically this should be used when multiple operations need to be carried out before the user is shown the result of the operations.*

---

viewerdisplaypanel.release - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Allow refreshes to occur on this Viewerdisplaypanel `tool`

Synopsis
release()

Returns
T

Description

This function will re-enable refreshing occuring on this Viewerdisplaypanel `tool` *provided an equal number of calls have been made to the hold function.*

---

viewerdisplaypanel.disablecontrols - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Temporarily disable the controls on this Viewerdisplaypanel `tool`

Synopsis
disablecontrols()

Returns
T

Description

This function will simply suspend the actions of controls on this Viewerdisplaypanel `tool`*. This means, for example, that the zoom and colormap fiddling controls will not work temporarily.*

---

viewerdisplaypanel.enablecontrols - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Re-enable the controls on this Viewerdisplaypanel `tool`

Synopsis
enablecontrols()

Returns
T

Description

This function re-installs the controls on this Viewerdisplaypanel `tool` *after they have been disabled with the disablecontrols function.*

---

viewerdisplaypanel.newcanvasmanagergui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Create and (probably) show a new graphical user interface for this `tool`*'s Viewercanvasmanager* `tool`

Synopsis

newcanvasmanagergui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	*parent frame in which to place the Viewercanvasmanager* `tool` *Allowed: agent or F* *Default: F*
show	*show the Viewercanvasmanager* `tool`*?* *Allowed: boolean* *Default: T*
hasdismiss	*add a "Dismiss" button to the GUI?* *Allowed: boolean* *Default: F*
hasdone	*add a "Done" button to the GUI?* *Allowed: boolean* *Default: F*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns

`tool` *having Viewerstandardgui interface*

Description

This function will create a new graphical user interface which can be used to control this Viewerdisplaypanel `tool`*'s Viewercanvasmanager* `tool`*. By default, the GUI is placed on the screen, although it can be constructed for later use.*

---

viewerdisplaypanel.newcanvasprintmanagergui - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Create and (probably) show a new graphical user interface for this `tool`'s Viewercanvasprintmanager `tool`

Synopsis

newcanvasprintmanagergui(parent, show, hasdismiss, hasdone, widgetset)

Arguments

parent	*parent frame in which to place the Viewercanvasprintmanager `tool`* *Allowed: agent or F* *Default: F*
show	*show the Viewercanvasprintmanager `tool`?* *Allowed: boolean* *Default: T*
hasdismiss	*add a "Dismiss" button to the GUI?* *Allowed: boolean* *Default: F*
hasdone	*add a "Done" button to the GUI?* *Allowed: boolean* *Default: F*
widgetset	*Widgetserver `tool` to use to build the GUI* *Allowed: Widgetserver `tool` or unset* *Default: unset*

Returns

`tool` *having Viewerstandardgui interface*

Description

This function will create a new graphical user interface which can be used to control this Viewerdisplaypanel `tool`'s Viewercanvasprintmanager `tool`. By default, the GUI is placed on the screen, although it can be constructed for later use.

---

viewerdisplaypanel.testpattern - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Turn a test pattern on or off on the Viewerdisplaypanel `tool`

Synopsis
testpattern(onoff)

Arguments

onoff	*turn pattern on (T) or off (F)* *Allowed: boolean*

Returns
T

Description

Calling this function with `onoff=T` *will place a "test pattern" on the Viewerdisplaypanel* `tool` *which is drawn with each refresh. The test pattern is really available for, surprise, surprise, testing! It is actually the test pattern which is used to display the registered colormaps in the graphical user interface for the Viewercolormapmanager* `tool`*.*

---

viewerdisplaypanel.type - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return the type of this `tool`

Synopsis
type()

Returns
'viewerdisplaypanel'

Description

This function simply returns the string `'viewerdisplaypanel'`*, which can be used to identify this* `tool` *as a Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.registercolormap - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Explicitly register a colormap on the Viewerdisplaypanel `tool`

Synopsis
registercolormap(map)

Arguments

map	*name of colormap to register* *Allowed: string*

Returns
T

Description

If the given colormap name is valid, then it will be explicitly registered on this Viewerdisplaypanel `tool`*. The result of this will be a redistribution of the available color cells to the registered colormaps. A change will only be noticed if previously the named colormap was* not *registered on this Viewerdisplaypanel* `tool`.
This function is generally reserved for internal use; careless use of it could cause the Viewer `tool` or AIPS++ to fail.

---

viewerdisplaypanel.unregistercolormap - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Explicitly unregister a colormap on the Viewerdisplaypanel `tool`

Synopsis
unregistercolormap(map)

Arguments

map	*name of colormap to unregister* *Allowed: string*

Returns
T

Description

This function simply explicitly un*registers the named colormap, provided it is a valid colormap and is already registered on this Viewerdisplaypanel* `tool`*.*
This function is generally reserved for internal use, and careless use of it could cause the Viewer `tool` or AIPS++ to fail.

---

viewerdisplaypanel.register - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Register a Viewerdisplaydata `tool` *for drawing*

Synopsis
register(displaydata)

Arguments

displaydata	*Viewerdisplaydata* `tool` *to register* *Allowed: Viewerdisplaydata* `tool`

Returns
T, F or fail

Description

This function registers the given Viewerdisplaydata `tool` *on this Viewerdisplaydata* `tool`*. That means that at the next and subsequent refreshes, until it is unregistered, the given Viewerdisplaydata* `tool` *will attempt to draw itself on this Viewerdisplaypanel* `tool`.
A return value of `T` *indicates that the Viewerdisplaydata* `tool` *was successfully registered on this Viewerdisplaypanel* `tool`*. A return value of* `F` *indicates that the Viewerdisplaydata* `tool` *was already registered, so no action was performed. The return value will be* `fail` *if the Viewerdisplaydata* `tool` *was unknown to, or could not be found by, this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.unregister - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Unregister a Viewerdisplaydata `tool` *from this Viewerdisplaypanel* `tool`

Synopsis
unregister(displaydata)

Arguments

displaydata	*Viewerdisplaydata* `tool` *to unregister* *Allowed: Viewerdisplaydata* `tool`

Returns
T, F or fail
This function removes the given Viewerdisplaydata `tool` *from the list of Viewerdisplaydata* `tools` *which will (attempt to) draw themselves on this Viewerdisplaypanel* `tool` *at each refresh.*
A return value of `T` *indicates that the Viewerdisplaydata* `tool` *was successfully unregistered from this Viewerdisplaypanel* `tool`*. A return value of* `F` *indicates that the Viewerdisplaydata* `tool` *was not previously registered, so no action was performed. The return value will be* `fail` *if the Viewerdisplaydata* `tool` *was unknown to, or could not be found by, this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.isregistered - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Query whether a Viewerdisplaydata `tool` *is registered on this Viewerdisplaypanel* `tool`

Synopsis
isregistered(displaydata)

Arguments

displaydata	*Viewerdisplaydata* `tool` *to query* *Allowed: Viewerdisplaydata* `tool`

Returns
T, F or fail
This function simply returns `T` *or* `F` *indicating (respectively) that the given Viewerdisplaydata* `tool` *is or is not currently registered for display on this Viewerdisplaypanel* `tool`*. The return value will be* `fail` *if the Viewerdisplaydata* `tool` *was unknown to, or could not be found by, this Viewerdisplaypanel* `tool`.

---

viewerdisplaypanel.registerednames - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a vector of the names of registered Viewerdisplaydata `tools`

Synopsis
registerednames()

Returns
Vector of strings

Description

This function returns a vector of names of all Viewerdisplaydata `tools` *registered on this Viewerdisplaypanel* `tool`. This function is generally reserved for internal use.

---

viewerdisplaypanel.registrationflags - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Return a record describing which Viewerdisplaydata `tools` *are registered on this* `tool`

Synopsis
registrationflags()

Returns
record

Description

This function returns a record whose top-level field names are those of the Viewerdisplaydata `tools` *known to the Viewer* `tool` *which owns this Viewerdisplaypanel* `tool`*. The value of each field is a boolean, which if T indicates that that Viewerdisplaydata* `tool` *is registered for display on this Viewerdisplaypanel* `tool`.
This function is generally reserved for internal use.

---

viewerdisplaypanel.unregisterall - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Unregister all Viewerdisplaydata `tools` *from this Viewerdisplaypanel* `tool`

Synopsis
unregisterall()

Returns
T

Description

This function simply calls the unregister function for every Viewerdisplaydata `tool` *that is registered on this Viewerdisplaypanel* `tool`*. It is akin to "clearing" the Viewerdisplaypanel* `tool`*. The unregistrations are wrapped in a* `hold/release` *scope, so that only one refresh is made at the end of this function.*

---

viewerdisplaypanel.unzoom - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Unzoom the current view

Synopsis
unzoom()

Returns
T

Description

This function will modify the coordinates on the Viewerdisplaypanel `tool` *such that the full view (of at least the first registered Viewerdisplaydata* `tool`*) is visible.*

---

viewerdisplaypanel.setzoom - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Set a specific zoom

Synopsis
setzoom(blc, trc)

Arguments

blc	*bottom-left linear coordinates of zoom* *Allowed: 2-vector*
trc	*top-right linear coordinates of zoom* *Allowed: 2-vector*

Returns
T

Description

This function sets the linear coordinate system of the Viewerdisplaypanel `tool` *to the specified blc and trc. Linear coordinates are generally identical to the pixel coordinates of the first registered Viewerdisplaydata* `tool`*.*

---

viewerdisplaypanel.zoom - Function

Package	**display**
Module	**viewer**
Tool	**viewerdisplaypanel**

Zoom by a given factor

Synopsis
zoom(xfac, yfac)

Arguments

xfac	*scale factor in X direction* *Allowed: float* *Default: 2.0*
yfac	*scale factor in Y direction* *Allowed: unset or float* *Default: unset*

Returns
T

Description

If `yfac=unset`*, then* `yfac` *is forced to be identical to* `xfac`*. This function zooms in by the specified amount in each direction, centred on the centre of the current viewable area. For scale factors greater than one, the view is magnified, and for less than one, the view is demagnified. There will be some interplay between the* `aspect` *option of Viewerdisplaydata* `tools`*and the zoom setting.*

1.3.7 viewerslicedp - Tool

Package display
Module viewer

Tool for providing 3D slice canvases
include "viewerslicedp.g"

Constructors

viewerslicedp	*Create a Viewerslicepanel* `tool`

Functions

preload	*Cache all views in display memory*
setoptions	*Set the modifiable state of this* `tool`

Description

This `tool` *sets up a canvas with three panels on it. These panels present the user with different views of the Viewerdisplaydata* `tools`*. Only data which has more than two axes and more than one pixel on the third axis will be displayed on this panel. The top left panel shows the "standard" axis1 vs. axis2. The top right panel displays axis3 vs. axis2 and the bottom right panel shows axis1 vs. axis 3.*
The functions of this tool are almost identical to the Viewerdisplaypanel `tool`*. This documentation only highlights functions not available to this tool and functions additional or different to the Viewerdisplaypanel* `tool`*.*
This specialized displaypanel has no printing facility yet, although X11 pixmaps can be created using the the function viewerslicedp.writexpm.

Graphical User Interface

The standard graphical user interface provided by the Viewerslicepanel `tool` *is shown in Figure 1.6. Surrounding the actual display area itself are four GUI components: the menu bar at the top, the button bar at the bottom, the tracking* *bar/s just above the buttons and the controlbox on the left. The menu and the tracking bar are identical to the ones in the Viewerdisplaypanel* `tool`*.*

The button bar *This* `tool` *provides a* `Preload...` *button. The* `Print...` *button is currently not available.*

The control box *The control box has one additional button, the* `Multi Panel Crosshair` *tool. If this tool gets dragged across the canvas on one panel it "animates" the other two panels since the coordinates on those two are dependend on ths first one.*

- **Multi Panel Cursor:** *Press the "Esc" key to cancel the crosshair drawn.*

All other GUIelements inherited from the Viewerdisplaypanel are described here.

viewerslicedp - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerslicedp**

Create a Viewerslicepanel `tool`

Synopsis

viewerslicedp(parent, viewer, width, height, maptype, newcmap, mincolors, maxcolors, autoregister, holdsdata, show, widgetset)

Arguments

parent	*parent frame for the panel* *Allowed: agent or F* *Default: F*
viewer	*the Viewer* `tool` *which will manager this* `tool` *Allowed: Viewer* `tool`
width	*width in pixels of the drawing area* *Allowed: integer* *Default: 250*
height	*height in pixels of the drawing area* *Allowed: integer* *Default: 250*
maptype	*how colors will be addressed on this Viewerslicepanel* `tool` *Allowed: 'index' 'rgb' 'hsv'* *Default: 'index'*
newcmap	*install a private colormap? (ignored if a parent frame is provided)* *Allowed: boolean* *Default: F*
mincolors	*minimum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
maxcolors	*maximum number of colors to allocate* *Allowed: Integer or unset* *Default: unset*
autoregister	*accept auto-register data on this Viewerslicepanel* `tool`*?* *Allowed: boolean* *Default: F*
holdsdata	*is this Viewerslicepanel* `tool` *going to hold data?* *Allowed: boolean* *Default: T*
show	*show the Viewerslicepanel* `tool`*?* *Allowed: boolean* *Default: T*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: unset*

Returns

Viewerslicepanel `tool` *or fail*

Description

This is the only constructor for Viewerslicepanel `tools`*. It constructs (creates) a Viewerslicepanel* `tool` *which can subsequently be used for the display of*

Viewerdisplaydata `tools`. *If* `widgetset` *is left* `unset`, *then the Viewerslicepanel* `tool` *will use the Widgetserver* `tool` *used by its parent Viewer* `tool`.
This constructor should not be called by users: *it is called by the newdisplaypanel function of the Viewer* `tool`, *which returns a Viewerslicepanel* `tool`.

---

viewerslicedp.setoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerslicedp**

Set the modifiable state of this `tool`

Synopsis
setoptions(options)

Arguments

options	*options record describing the requested state* *Allowed: record*

Returns
T
This function is distributes the options over all views of the Viewerdisplaydata `tool`. *Otherwise it is identical to viewerdisplaypanel.setoptions*

---

viewerslicedp.preload - Function

Package	**display**
Module	**viewer**
Tool	**viewerslicedp**

Cache all views in display memory

Synopsis
preload()

Returns

T
This function animates over all views of the Viewerdisplaydata `tool`*. After this loop has been completed the Multipanel Crosshair tool will instantly cause an animation on the other panels, because there is no computation involved.*
Note: If axis labels are enabled there won't be instant animation and you will notice a delay between moving your cursor and display on the other panels.
Note: Once you have decided to execute this function you can't stop it until it has finished.

1.3.8 viewerstandardgui - Tool

Package	**display**
Module	**viewer**

Interface for operating Viewer `tool` *graphical user interfaces*

Constructors

Functions

disable	*Disable this graphical user interface*
dismiss	*Hide this graphical user interface*
done	*Close down (finished with) this graphical user interface*
enable	*Enable this graphical user interface*
gui	*Map this graphical user interface*
ismapped	*Is this graphical user interface mapped to the screen or its parent frame?*
map	*Map the graphical user interface to the screen*
unmap	*Remove the graphical user interface from the screen*

Description

This is not a tool. Instead, this simply defines an interface which is shared by all graphical user interfaces which are constructed by the Viewer `tool` or by its support `tools`.

viewerstandardgui.map - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Map the graphical user interface to the screen

Synopsis
map(force)

Arguments

force	*force map even if this* `tool` *seems to be mapped already* *Allowed: boolean* *Default: F*

Returns
boolean

Description

This function should be used to map a graphical user interface to the screen. If the `tool` *made its own top-level frame, then that frame will be mapped to the screen. Otherwise, the next highest level frame will be mapped into its parent frame, which itself may or may not be presently mapped to the screen.*
A boolean value of `T` *is returned if an action was taken to map the GUI. Otherwise,* `F` *is returned, implying that the* `tool` *believed itself to already be mapped.*

---

viewerstandardgui.unmap - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Remove the graphical user interface from the screen

Synopsis
unmap(force)

Arguments

force	*force unmap even if this* `tool` *seems to be unmapped already* *Allowed: boolean* *Default: F*

Returns
boolean

Description

This function should be used to remove a graphical user interface from the screen. If the `tool` *made its own top-level frame, then that entire frame will be removed from the screen. Otherwise, the next highest level frame will be removed from its parent frame, which itself may or may not be presently mapped to the screen.*
A boolean value of `T` *is returned if an action was taken to unmap the GUI. Otherwise,* `F` *is returned, implying that the* `tool` *believed itself to already be unmapped.*

---

viewerstandardgui.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Close down (finished with) this graphical user interface

Synopsis
done()

Returns
T

Description

If you no longer need to use this graphical user interface, calling this function will remove it from the screen, disable any interaction with it, and free up its resources. This function is called when the user presses the `Dismiss` *button on the GUI* if *the GUI was constructed with* `hasdone=T` *and* `hasdismiss=F`.

---

viewerstandardgui.ismapped - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Is this graphical user interface mapped to the screen or its parent frame?

Synopsis
ismapped()

Returns
boolean

Description

This function simply returns a boolean value of `T` *which indicates that this graphical user interface is mapped to the screen (or into its parent frame), or* `F`*, meaning it is presently unmapped.*

---

viewerstandardgui.dismiss - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Hide this graphical user interface

Synopsis
dismiss()

Returns
boolean

Description

This function simply calls unmap with `force=F`*. The return value is the result of this invokation. This function is called when the user presses the* `Dismiss` *button on the GUI* if *the GUI was constructed with* `hasdismiss=T`*.*

viewerstandardgui.gui - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Map this graphical user interface

Synopsis
gui()

Returns
boolean

Description

This function simply calls map with `force=F`*. The return value is the result of this invokation.*

---

viewerstandardgui.disable - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Disable this graphical user interface

Synopsis
disable()

Returns
boolean

Description

If this graphical user interface `tool` created its own top-level frame, then this function disables ("greys") that frame, sets the cursor to a "watch" icon, and returns `T`. Otherwise, it does nothing and returns `F`.

---

viewerstandardgui.enable - Function

Package	**display**
Module	**viewer**
Tool	**viewerstandardgui**

Enable this graphical user interface

Synopsis
enable()

Returns
boolean

Description

If this graphical user interface `tool` created its own top-level frame, then this function enables ("ungreys") that frame, returns the cursor to the standard pointer shape, and returns `T`. Otherwise, it does nothing and returns `F`.

1.3.9 viewerdatamanager - Tool

Package	**display**
Module	**viewer**

Tool for importing data into the Viewer
include "viewerdatamanager.g"

Description

The Data manager is a graphical user interface only tool to provide a context sensitive import of data into the Viewer `tool`. It is a front-end to the viewer.loaddata function in the Viewer `tool`.

This GUI has three major control areas and a couple of buttons an a menu:

File Chooser *The top-left corner shows a "File Chooser", a text field to enter directories and a listbox to select your data files on disk. This listbox shows the filtered content of the specified directory, i.e. only types understood by the Viewer* `tool` *are listed. The file types supported are can be found in viewer.loaddata.*

Tool Chooser *Here all supported* Glish *tools are listed, e.g if you have opened an image tool with the variable name* `im` *it will show up here.*

Display Type selector *Once you have selected a supported file or tool, the box called "DisplayData Type" to the right, shows all the drawing types which are supported for this kind of data. The possible types are:*

- *Raster Image*
- *Contour Map*
- *Vector Map*
- *Marker Map*
- *Sky catalog overlay*

Update *Update the contents of the current directory and* Glish *tools.*

Autoregister *Your data will automatically be registered (displayed) on all Viewerdisplaypanel* `tools`*. In some cases like the* `Skycatalog` *you might want to disable this to make sure you don't register it by itself.*

Menu bar *The* File *entry of the menu bar is identical to the same menu item in viewerdisplaypanel gui.*

1.3.10 viewerannotations - Tool

Package display
Module viewer

Tool for drawing annotations onto a displaypanel.
include "viewerannotations.g"

General description

Description

This `tool` *allows the user to interactively draw shapes and text onto Viewerdisplaypanel* `tools`*. It has a GUI, allowing selection of the drawing tool to use, and also the ability to change settings for a specified drawing object. Annotations can also be constructed and modified via option records passed to functions on the command line. The* `tool` *also supports printing of the user's annotations onto postscript. On creation of a Viewerdisplaypanel* `tool`*, a viewerannotation is automatically created, the GUI of which can be accessed by selecting the "Annotations..." option from the "Tools" menu. Each viewerannotation* `tool` *must be associated with a Viewerdisplaypanel* `tool`*.*

Graphical User Interface

Description

The graphical user interface provided by the viewerannotations tool is designed to provide access to common settings and actions used when annotating a displaypanel. The GUI contains buttons, which are broken into two main categories. There are three control buttons, at the top left of the GUI . They can be used to assign a mouse button to control of annotations, to move / rotate shapes and to delete shapes. The other buttons are all tools, which allow the user to create new shapes. After clicking on a shape, the user can position the shape on the displaypanel. The button on the far right, which is the marker tool opens a pop up menu when clicked on. This menu shows a list of available markers, which can then be placed on the displaypanel.

The other main section of the GUI is a expandable options section. By default the roll-up is "rolled-up", and hence the options are not visible. By rolling down the options section, the user has access to a context specific menu which shows options for the shape they have currently selected.

The figure above shows the annotation control GUI. Everything below the Options roll-up is context specific, depending on which shape you have selected. The control buttons, and also the tool buttons can be seen above the roll-up.

The figure above shows annotations objects being edited. The selected object is shown using green "handles". The handles can be moved to change the size of the shape, and the shape can be moved by clicking and dragging inside the shape. Keyboard modifiers can be used to change the action performed by mouse events. Holding "Control" on the keyboard changes the effect of moving the handles to that of rotation, and the effect of dragging a shape to that of scaling the object (about its center). If more than one shape occupies a location on the screen, or if a larger object is covering a smaller object, the underlying objects can be accessed by repeatedly clicking in the same spot.

Note: *All AIPS++ GUIs have a so-called popup help. Just rest your mouse cursor on top of a GUI element (e.g. a button) and a little frame will appear outlining the funtionality of this element. If you then press and hold down the right mouse button you will get a detailed description. The information given there should be sufficient to let you operate that GUI element.*

Events

The Viewerannotations `tool`*tool does not emit any events. It is not really required to, since its job is to control event handling internally. All of the actual mouse events are handled by the underlying proxy. maxposf*

viewerannotations - Constructor

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Create a Viewerannotations `tool`*tool*

Synopsis
viewerannotations(displaypanel, title, widgetset)

Arguments

displaypanel	*displaypanel to attach to* *Allowed: agent*
title	*the title to use for the GUI frame* *Allowed: String* *Default: "Unknown Draw Area"*
widgetset	*Widgetserver* `tool` *to use to build the GUI* *Allowed: Widgetserver* `tool` *or unset* *Default: ddlws*

Returns
Viewerannotations `tool`*tool or fail*

Description

This is the only constructor for the Viewerannotations `tool`*tool. It constructs (creates) a Viewerannotations* `tool`*tool which attaches itself to the supplied Viewerdisplaypanel* `tool`*. This constructor will generally not need to be called by users, since each Viewerdisplaypanel* `tool`*creates its own Viewerannotations* `tool`*tool at construction time.*

---

viewerannotations.done - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Destroy this Viewerannotations `tool`*tool.*

Synopsis
done()

Returns
T

Description

This function is used when the Viewerannotations `tool`*tool is no longer required. When the Viewerdisplaypanel* `tool`*to which the annotator is attached is 'done', this function is called automatically.*

---

viewerannotations.dismiss - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Remove the Viewerannotations `tool`*tool GUI from the screen*

Synopsis
dismiss()

Arguments

Returns
T

Description

This function simply removes (if it is visible) the Viewerannotations `tool`*tool GUI from the screen. The GUI can be redisplayed, either by the user clicking the appropriate menu option, or by a call to the* `gui` *function.*

---

viewerannotations.print - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Print the annotations onto the supplied pixelcanvas

Synopsis

print(pixelcanvas)

Arguments

pixelcanvas	*PixelCanvas proxy to print the annotations to* *Allowed: Pixelcanvas proxy*

Returns

T

Description

This function is mainly designed for internal use. When the Viewerdisplaypanel `tool` *is to be printed onto a Postscript PixelCanvas, the pixelcavas is passed to this function, which draws all the current annotations onto it.*

---

viewerannotations.newshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Create a new shape based on a record

Synopsis
newshape(shapesettings)

Arguments

shapesettings	*Settings describing the shape* *Allowed: Record*

Returns

Description
This function makes a new shape based entirely on the information provided in the record passed to it. Once on screen, the shape can be manipulated by the user.

---

viewerannotations.deleteshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Delete a shape from the annotations

Synopsis
deleteshape(index)

Arguments

index	*Index of the shape* *Allowed: Integer*

Returns

Description

This function deletes the shape with the specified index. The index of the currently active shape can be determined using the `whichshape` *function.*

---

viewerannotations.whichshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Return the index of the currently active shape

Synopsis
whichshape()

Returns
Integer

Description

This function returns the index of the selected / active shape.

---

viewerannotations.createshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Create a shape, based on options and also user input

Synopsis
createshape()

Returns

Description

This function allows the user to create a shape by "dragging it out". Some of the options including shape type, and miscellaneous options must be supplied via the option record. Options regarding position of the shape can be omitted, and these *will be obtained the next time the user clicks in the draw area. This allows for the user to select a shape, and "drag it out".*

---

viewerannotations.availableshape - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Return a list of the available shapes

Synopsis
availableshape()

Returns
Array of Strings

Description

This function simply returns an array which is filled with a list of the available shapes.

---

viewerannotations.setkey - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Set the key (mouse button) which the user will use to control the annotations

Synopsis
setkey(keysym)

Arguments

keysym	*New key symbol* *Allowed: Key Symbol, some are defined as glish global variables*

Returns
T

Description

Different tools can be assigned to different mouse buttons. This function allows the user to specify which button they wish to use to control the annotations.

---

viewerannotations.getalloptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Return options for all the shapes currently displayed

Synopsis
getalloptions()

Returns
Record

Description

This will return a record, which contains the information for all of the annotation shapes. The format of the record is as follows:

- *Shape 1:*
  - *Options for shape 1...*
- *Shape 2:*
  - *Options for shape 2...*
- *Shape n:*
  - *Options for shape n...*

;

---

viewerannotations.setalloptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Sets the annotations based on information provided in the record

Synopsis
setalloptions()

Returns
Record

Description

This sets the annotations based on a record. All current shapes are deleted, and replaced with those specified in the option record. The format for the record passed as a parameter is the same as that which is returned by `getalloptions`.

---

viewerannotations.getshapeoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Returns the option record for the specified shape

Synopsis
getshapeoptions(whichshape)

Arguments

whichshape	*Return options for this shape* *Allowed: Integer*

Returns
Record

Description

This function returns the option record for the shape with the specified index. The index of the current / selected shape can be determined by using the `whichshape` *function.*

Example

```
#Return the options for the currently selected shape:
annotator := mydisplaypanel.annotator();
shapeoptions := annotator.getshapeoptions(annotator.whichshape());
```

---

viewerannotations.setshapeoptions - Function

Package	**display**
Module	**viewer**
Tool	**viewerannotations**

Set the options of a specific shape

Synopsis

setshapeoptions(whichshape, newoptions)

Arguments

whichshape	*Set the options for this shape* *Allowed: Integer*
newoptions	*Options to set* *Allowed: Record*

Returns

Description

This function sets options for a specific shape. The index of the currently selected / active shape can be determined using the `whichshape` *function.*

---

viewerannotations.addlockedtocurrent - Function

Package **display**
Module **viewer**
Tool **viewerannotations**

"Lock" a shape to the currently selected one

Synopsis
addlockedtocurrent(whichshape)

Arguments

whichshape	*Which shape do you wish to lock to the selected one* *Allowed: Integer*

Returns

Description

This function can lock a shape to the currently selected one. For example, if the currently selected object is an arrow, and the `whichshape` *variable is set to the index of text, then whenever the arrow is moved by the user, so too will the text.*

---

viewerannotations.removelockedfromcurrent - Function

Package **display**
Module **viewer**
Tool **viewerannotations**

Remove a locked item from the currently selected shape.

Synopsis
removelockedfromcurrent(lockeditemindex)

Arguments

lockeditemindex	*Index of shape to remove from lock* *Allowed: Integer*

Returns

Description

This functions performs the opposite of `addlockedtocurrent`. *It allows the user to 'unlock' a shape from the currently selected one.*

1.3.11 skycatalog - Tool

Package display
Module viewer

Tool to create and manipulate Viewer `tool` *readable skycatalog tables*
include "skycatalog.g"

Constructors

skycatalog	*Create a skycatalog* `tool`

Functions

addunit	*Add a unit to a column*
done	*Close this* `tool`
fromascii	*Create the* `tool` *from an ascii file*
fromcomponentlist	*Create the* `tool` *from a componentlist*
mergecolumns	*Merge n columns into one*
name	*Return the name of the table*
renamecolumn	*Rename a column*
table	*Return the underling table* `tool`
type	*Return the* `tool` *type*

Description

This `tool` *allows the user to create so called skycatalog tables readable by the Viewer* `tool`. *This can be done from ASCII tables or componentlists. It also provides a front-end to the table* `tool` *to manipulate the structure of the* *skycatalog.*

Example

Here is a small example on how to convert an ascii table "my.ascii" to an AIPS++ table "my.tbl":
The ASCII file looks like this:

Name	*No*	*RA*	*DEC*	*NED-ID*	*Max*	*mom0*
A	*A*	*A*	*A*	*A*	*R*	*R*
NED	*16*	*19:44:47*	*-14:46:51*	*NGC6822ee*	*24.255*	*159.8*
NED	*22*	*01:04:46*	*+02:07:04*	*IC1613e*	*13.002*	*328.1*

```
include 'skycatalog.g';
sca := skycatalog('myscat1.tbl');
sca.fromascii(asciifile='my.ascii',hasheader=T,longcol='RA',latcol='DEC');
# cosmetics
sca.mergecolumns("Name No");
sca.done()
```

Example

An example using a componentlist:

```
im := imagemaketestimage()
cl := im.findsources()
sca := skycatalog('myscat2.tbl')
sca.fromcomponentlist(cl)
sca.done()
# That's it !
```

skycatalog - Constructor

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Create a skycatalog `tool`

Synopsis

skycatalog(filename)

Arguments

filename	*Name of the skycatalog table*	
	Allowed: String	
	Default: unset	

Returns
skycatalog `tool` *or fail*

Description

Construct a skycatalog `tool`. *The specified filename will either be used to associate this* `tool` *wwith skycatalog table on disk or to create a new skycatalog table, which can be filled from a componentlist or ASCII file.*

---

skycatalog.addunit - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Add a unit to a column

Synopsis
addunit(colname, unit)

Arguments

colname	*column name* *Allowed: String*
unit	*Unit to put into column* *Allowed: String*

Returns

Description

---

skycatalog.done - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Close this `tool`

Synopsis

done(delete)

Arguments

delete	*delete the table* *Allowed: Boolean* *Default: F*

Returns

Description

---

skycatalog.fromascii - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Create the `tool` *from an ascii file*

Synopsis

fromascii(asciifile, hasheader, longcol, latcol, dirtype, unit)

Arguments

asciifile	*The source ASCII file* *Allowed: String* *Default: unset*
hasheader	*Does the ASCII file include a header definition?* *Allowed: Boolean* *Default: F*
longcol	*The column name or number containing the Longitude component* *Allowed: String or Integer (see description)* *Default: unset*
latcol	*The column name or number containing the Latitude component* *Allowed: String or Integer (see description)* *Default: unset*
dirtype	*The direction type of the coordinate* *Allowed: String* *Default: J2000*
unit	*The unit of the coordinates* *Allowed: String* *Default: deg*

Returns
T or F

Description

This functions converts an ASCII table into an AIPS++ skycatalog table. These tables can be read by the Viewer `tool` *and displayed as a skycatalog overlay. Whitespace in the ASCII tables is interpreted as a column separator and the number of columns per row must be constant. The ASCII table can start with a description of the columns and the data types in that column. If this header is not specified the* `tool` *tries to determine the data type and will name the columns Column1..ColumnN. In that case the columns containing the coordinate can be given as integer indices.*
The argument dirtype indicates what direction coordinate reference type your coordinate has. At the moment only J2000, B1950, GALACTIC and SUPERGAL are valid types.
The unit is assumed to be the same for long and lat and only has to be specified if the coordinate is given decimals.

---

skycatalog.fromcomponentlist - Function

Package display

Module viewer
Tool skycatalog

Create the `tool` *from a componentlist*

Synopsis
fromcomponentlist(code)

Arguments

code	*The componentlist* `tool` *Allowed: componentlist* `tool` *Default: unset*

Returns
T or F

Description
Invoke this function to fill the skycatalog table with components of a componentlist.

---

skycatalog.mergecolumns - Function

Package display
Module viewer
Tool skycatalog

Merge n columns into one

Synopsis
mergecolumns(columns, name)

Arguments

columns	*Names of the columns to merge* *Allowed: String*
name	*name of the output column* *Allowed: String* *Default: unset*

Returns

Description

If the name argument is unset, the name of the output column is taken from the first column in column array. The output column will always be of type string.

---

skycatalog.name - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Return the name of the table

Synopsis

name()

Returns

String

Description

This function returns the name of the table.

---

skycatalog.renamecolumn - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Rename a column

Synopsis
renamecolumn(oldname, newname)

Arguments

oldname	*column name* *Allowed: String*
newname	*new column name* *Allowed: String*

Returns

Description

---

skycatalog.table - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Return the underling table `tool`

Synopsis
table()

Returns
table `tool`

Description

---

skycatalog.type - Function

Package	**display**
Module	**viewer**
Tool	**skycatalog**

Return the `tool` *type*

Synopsis
type()

Returns
skycatalog

Description

1.3.12 is_viewer - Function

Package display
Module viewer

Is the argument a viewer tool?

Synopsis
is_viewer(tool)

Description
This global function can be used to determine if the supplied argument is a viewer tool. If so it returns True, otherwise it returns False.

Arguments

tool	*The variable that you wish to test* *Allowed: Any glish variable* *Default: no default*

Returns
T or F, indicating if the argument is a viewer tool or not.

Example

```
include 'viewer.g';
if (is_viewer(dv)) {
  myviewer := dv;
} else {
  myviewer := viewer();
}
```

This example shows how you can use this function to determine if the default viewer (dv) is a viewer tool.