E U R O P E A N S O U T H E R N O B S E R V A T O R Y

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

VLT Software

Real Time Display

User Manual

Doc.No. VLT-MAN-ESO-17240-0866

Issue 2.8

Date 5/16/99

Prepared	A. Brighton	5/16/99	
	Name	Date	Signature
Approved	P. Biereichel		
	Name	Date	Signature
Released	G. Monnet		
	Name	Date	Signature

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.0	**25/07/95**	**All**	**First preparation**
2.0	**22/11/95**	**All**	**update**
2.1	**22/02/96**	**All**	**update**
2.4	**01/10/96**	**overview, reference, installation**	**update**
2.5	**28/07/97**	**All**	**update**
2.6	**15/01/98**	**All**	**update, minor changes after split into *tclutil* and *astrotcl* packages, mostly updated reference section.**
2.8	**07/08/98**	**All**	**Updated for Frame5.5, HTML reference pages, updated man pages, added cross references**

Table of Contents

1 Introduction

The Real Time Display software, or RTD, described in this document was designed to display atronomical images, either from image files or from some external source, such as a CCD camera or external process via shared memory. One of the development goals was to be able to display images coming from a CCD camera as rapidly as possible, while still having a flexible, user friendly user interface.

1.1 Purpose

The purpose of this manual is to describe the widgets and functions implemented in the Real Time Display. In addition, instructions for installing the RTD package and using RTD in applications is provided.

1.2 Scope

This document is primarily aimed at software developers using the RTD package. No real end-user documentation is provided here.

1.3 Applicable Documents

This document is based on the following documents:

[1] VLT-SPE-ESO-17240-0250, 1.0, 03/04/95 -- VLT Software - Real-Time-Display Software Spec.

[2] VLT-PRO-ESO-10000-0228, 1.0 10/03/93 -- VLT Software Programming Standards

1.4 Reference Documents

The following documents are referenced in this document:

[1] VLT-MAN-ESO-19400-1550 1.0 19/01/98 -- Tcl and C++ Utilities, Programmer's Manual

[2] VLT-MAN-ESO-19400-1551 1.0 19/01/98 -- Astronomical Tcl and C++ Utilities

1.5 Abbreviations and Acronyms

The following abbreviations and acronyms are used in this document:

RTD	Real Time Display
VLT	Very Large Telescope

1.6 Stylistic Conventions

The following styles are used:

`teletype`
: for examples extracted from the source code, directory and file names, names of programs and functions, and commands as they have to be typed (e.g. in the installation procedure).

`<name>`
in the examples, for parts that have to be substituted with the real contents before typing.

2 Overview

The RTD consists of a Tk image extension implemented in C++ called *rtdimage*, a collection of [incr Tcl] widget classes, a real-time image server and C++ library and a demo application showing how it is all used. This section should give a general overview of the system and its components.

2.1 Tk Image Extension *rtdimage*

At the heart of the RTD software is a Tk image type called *rtdimage* that was developed as a C++ class hierarchy. Tk has a documented method of adding new image types and the advantage of using it as opposed to developing a new Tk widget is that you can put a Tk image in a canvas window and use all of the canvas graphics functionality to draw over the image. This can be used to display labels and markers on the image or for more complicated *star maps* overlaying the image.

The rtdimage extension was developed as a C++ class hierarchy and library, which is described in more detail in later sections of this manual. There are classes for implementing Tcl commands, images and widgets, classes for managing the image transformations and display, the colormap and communication with the real-time image server for displaying images from a CCD camera. The functionality of all these C++ classes is available at the Tcl/Tk level as *rtdimage* subcommands and via a *remote control* interface. The following subcommands are currently implemented:

***rtdimage* Subcommand**	**Description**
alloccolors	Allocate or free colors in the colormap.
autocut	Set the cut levels automatically using median filtering or other algorithm.
bitpix	Return data type (BITPIX field) of the FITS image.
camera	Start, stop, pause or continue a CCD camera real-time display.
clear	Clear or blank out the display.
cmap	Load or manipulate a (MIDAS) colormap.
colorramp	Generate an image to use for the colormap display (color bar or ramp).
colorscale	Apply a color scaling algorithm, such as linear or logarithmic scaling to the image.
configure	Set or query configuration options.
convert	Convert X,Y between different coordinate system types.
cut	Set the cut levels directly.
dispheight	Return display height of image (after transformations).
dispwidth	Return display width of image (after transformations).
dump	Dump or save the image as a FITS file.
fits	Access FITS image header information.
flip	Flip the image in the X or Y directions or both.
frameid	return the frame Id (used for rapid frames and rtdServer communication).
get	Return X,Y coordinates and pixel value(s) for a screen position.
graphdist	Plot a graph of the pixel value distribution.
height	Return height of image (before transformations).
isclear	Return true if the image is cleared (no image loaded).
itt	Load or manipulate a (MIDAS) intensity transfer table or ITT.
max	Return maximum image pixel value.
mband	Draw a measure band on the canvas to display the world coordinate distance.
min	Return minimum image pixel value.
mmap	Access the *mmap* memory in which the FITS image data and header are stored.

***rtdimage* Subcommand**	**Description**
object	Return the astronomical object name, if known, for the viewed object.
pan	Support for a panning window.
perftest	Toggle interactive performance testing on or off.
pixtab	Support for displaying a table of image pixel values as the mouse moves.
preview	When a camera is running, stop it and make a local copy of the image.
radecbox	Returns a list of 4 values {ra0 dec0 ra1 dec1} that form an ra,dec box with the given center point and radius.
remote	Start remote control onterface.
rotate	Rotate the image by swapping X and Y coordinates.
scale	Magnify or shrink the image by integer factors.
shm	SysV shared memory access to image header and data.
spectrum	Plot a graph of pixel values along a line in the image.
statistics	Calculate statistics on the section of the image being displayed.
type	Return the type of the raw image data (short, int, float, ...)
update	Update the image.
userfreq	Set the maximum real-time update frequency.
view	Make a *view* of an image, a second image displaying the same image, possibly at a different magnification and offset.
warp	Warp (move) the mouse pointer in the image.
wcscenter	Return the center of the image in world coordinates.
wcsdist	Return the world coordinate distance between 2 screen pixel points.
wcsequinox	Return the world coordinates equinox of the image.
wcsheight	Return the height of the image in arcmin.
wcsradius	Return the radius (diagonal to center) in arcmin.
wcsset	Set basic world coordinates information.
wcsshift	Shift the world coordinates center of the image.
wcswidth	Return the widthof the image in arcmin.
width	Return width of image (before transformations).
zoom	Support for a zoom window.
zoomview	Support for an alternate implementation of the zoom window using a *view* of the raw image (now the default)

In addition, the following configuration options are supported for the rtdimage command:

***rtdimage* option**	**Description**
-displaymode	Sets display mode to 0 or 1 (default 1), see reference manual for details.
-file	Sets (FITS) image file to load and display.
-fitheight	Shrink to fit image in window of this height.
-fitwidth	Shrink to fit image in window of this width.
-newimagecmd	Sets Tcl command to be evaluated whenever a new image is displayed.
-shm_header	If true, put the image (FITS) header in shared memory for access via the remote control interface.
-shm_data	If true, put the image (FITS) data in shared memory for access via the remote control interface.
-subsample	Use subsampling when shrinking images (rather than max value).

***rtdimage* option**	**Description**
-usexsync	Flag: if true, try to use X synchronisation extension of displaying images.
-usexshm	Flag: if true, try to use X shared memory for displaying images.
-verbose	Flag: if true, print diagnostic messages at run time.
-name	Set the name of the image (makes debugging easier)
-min_colors	Set the minimum number of colors to allocate before using a private colormap.
-max_colors	Set the maximum number of colors to allocate.

See rtdimage(n) for more details.

2.2 RTD [incr Tcl] Widget Classes

The RTD software includes a library of [incr Tcl] widget classes[1]. Some of these widget classes are based on the *rtdimage* extension or are designed to be used with it and some are simply general user interface components. Here is a list of the widgets designed to work with the *rtdimage* extension directly: (see the User's Guide for screendumps of these widgets):

Widget Class	**Description**
RtdImage	An *rtdimage* in a canvas window, with optional scrollbars, support for *rapid frames*, *spectrum* lines, setting *cut levels*, etc.
RtdImageColorRamp	Makes a *color ramp* or bar, used to display the colors in the colormap and manipulate the colormap by rotating, shifting, stretching, etc.
RtdImageColors	Popup window for selecting a colormap, intensity, color scaling algorithm and for controlling the number of colors allocated.
RtdImageCtrl	Combination of other widgets: displays the image with a zoom window, panning window, control panel and color ramp. Adds support for application menu commands.
RtdImageCut	Popup window displaying the pixel value distribution and various ways of setting the image cut levels.
RtdImageFrame	Widget for displaying a *rapid frame* displaying a section of the image in the image window.
RtdImagePanel	Control panel and information display for an RtdImage widget, displays X,Y coordinates, world coordinates and pixel values, min and max pixel values, low and high cut levels. Supports scaling, rotating, flipping the image.
RtdImagePerf	Widget for displaying interactive performance testing data when the facility is toggled on by the *perftest* rtdimage subcommand.
RtdImagePick	Popup window to pick a star or object in the image and display statistics, such as FWHM, position, angle, etc.
RtdImagePopup	A top level widget for displaying a section of the image and zoom controls, also for use as a rapid frame.
RtdImagePan	A Panning window for RtdImage: displays a small *view* of the image with a panning rectangle to control the portion of the image that is displayed.
RtdImagePixTable	Popup window displaying a table of pixel values while tracking the mouse pointer.
RtdImagePixel	Displays the X and Y coordinates and pixel value at the mouse position.

1. [incr Tcl] is an object oriented extension to Tcl that makes it easy to implement widgets at the Tcl level.

Widget Class	Description
RtdImageTrans	Displays a label and a *choice* menu button for selecting and displaying the current image transformations (scale, rotate, flip).
RtdImageSpectrum	Popup window displaying a graph of image pixel values along a line drawn over the image.
RtdImageZoom	A Zoom window, displaying a magnified view of the X Image being displayed, while tracking the mouse pointer.
RtdImageZoomView	An alternative implementation of the Zoom window, using a magnified view of the raw image. This version is now the default, since it is more flexible.
RtdImageMBand	Used to display a measure band over the image while dragging the mouse pointer (Button-3), with support from built in C++ methods.
RtdPerfTester	Widget for controlling the front end of the performance benchmark tool.
RtdRecorder	Widget containing *rtdrecorder* and *rtdplayback* objects. Allows the real-time recording and playing back of CCd images.
RtdRMPEdit	Widget for selection of image sub-regions from cameras (used in multicasting applications, hence the name: see Appendix).
RtdServerTool	Popup window controlling the RtdServer communication for test and simulation purposes.

2.3 rtdServer and rtdImageEvent library

rtdServer is the process that manages the image event mechanism. Clients register to the rtdServer via the rtdInitImageEvt call. When a client attaches to a camera source an incoming image event will be forwarded to this client. Image events received from image sources where no clients are attached are simply discarded. Clients can also attach to image sources that not have registered yet as the rtdServer supports an independence between image event producer and image event consumer.

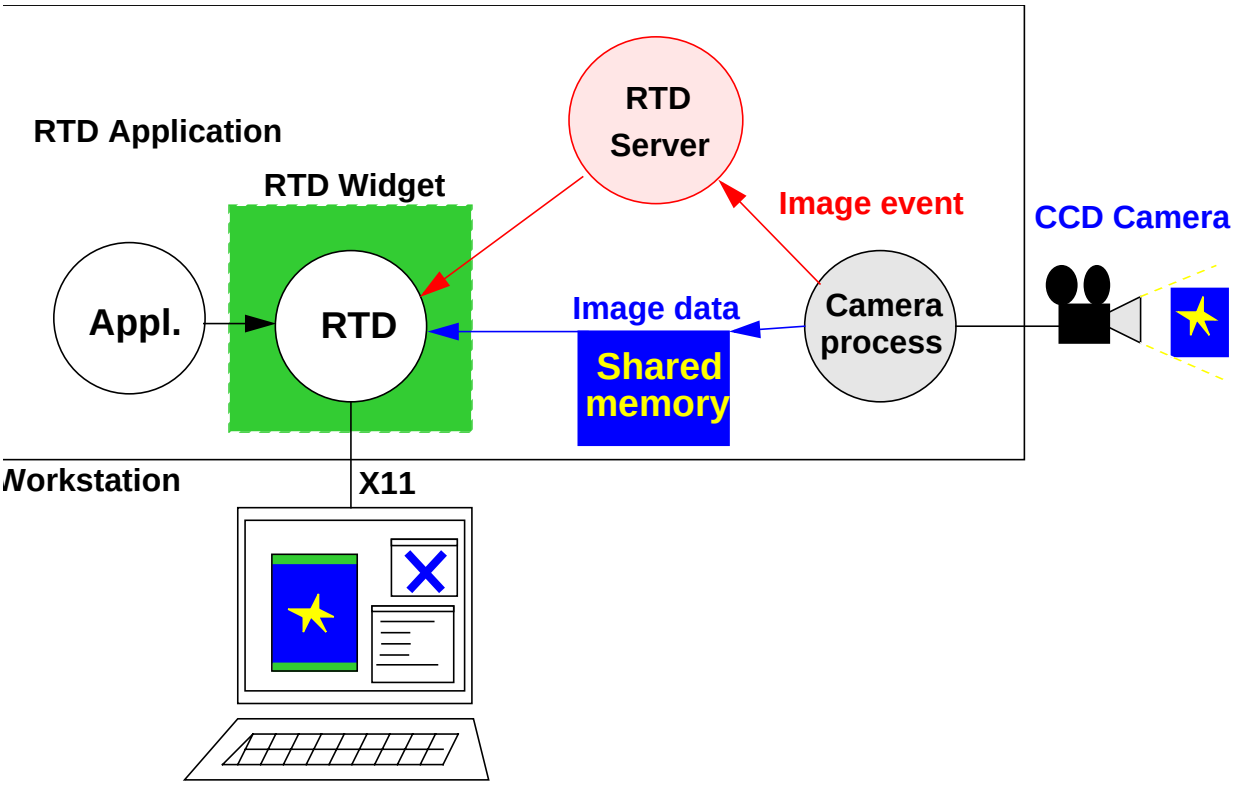
Furthermore rtdServer contains a simulator part that can be used to simulate the generation of image events. This feature is reserved for testing purposes only. Similarly, it also contains a performance test facility, in which several areas of shared memory are sent to a client Rtd in quick succession, and measurements are taken on certain performance parameters (see RtdPerformanceTool(3/n).

The rtdServer also implements semaphore locking of shared memory, to avoid the possibility of the RTD client reading the shared memory at the same time as the CCD writes (this is known as “image jitter”). The server program expects the CCD software to set a semaphore against any shared memory that has been written to (effectively to lock it). The server will then increment this semaphore by the number of RTD clients less one. If semaphores are not implemented in the incoming image event, no action is taken. The overall locking scheme is discussed in more detail in rtdSem(1). Semaphore locking is implemented in the simulator facility.

A C library, *rtdImgEvt,* is available for client applications to communicate with the rtdServer. It contains functions for connecting to the server, attaching to and detaching from a given CCD camera. It also contains functions for the CCD developer to lock areas of shared memory with semaphores.

2.4 RTI - Real-Time Image Library

The *RTI* Real-Time Image Library is a C++ class hierarchy that is used by the *rtdimage* software internally and was designed to be independent of X or Tk so that it can be used by other applications to preprocess images before sending them to be displayed. The interface is through the base class



ImageData, which hides the different raw image data types and implements such algorithms as color scaling, median filtering, scaling, rotating and flipping images.

This class also interfaces with the saoimage World Coordinates code (rtdwcs module) to give World Coordinates support for the image when the required FITS keywords are present.

The input to the ImageData class is normally the raw image and the output is an image in X image format, suitable for display without changes in an X window. An X image in this case is just an array of bytes that can be assigned to a real XImage in the display application. Since these classes don't know anything about X or Tk, they are fed from the outside with pointers to the raw and X image data, the dimensions of the raw and X images and colormap information (the number of colors available and the color values).

2.5 Remote Control Interface

The RTD supports remote access via a simple C interface (module *rtdrmt*). This interface works a bit like the Tk send command, except that it uses a socket interface and restricts the commands that may be executed. Remote clients can connect to a running RTD widget and send RTD subcommands to be executed and get the results, either immediately or via a callback. Some of the RTD subcommands are designed especially for use with this interface. For example, the *shm* command gives access to the image header and data as shared memory. Remote clients can execute any of the RTD image subcommands. In addition, applications can extend the list of available remote commands by specifying a Tcl command to be called for unknown subcommands (*not impl. yet*).

2.6 Coordinate Systems

The RTD understands and supports the following coordinate systems:

Coordinate System	Description
canvas	canvas coordinates (canvas scroll area)
screen	canvas window coords (visible area)
image	basic image pixel coords (at mag 1, no transformations)
chip	detector chip coordinates (based on ESO extended FITS keywords: DET CHIP...)
wcs *equinox*	world coordinates in H:M:S
deg *equinox*	world coordinates in degrees

Where it makes sense to do so, the RTD subcommands accept coordinates of the form:

```
$x $y $coord_type
```

Where $x and $y are the coordinate values and the coordinate type is one of the types in the above table. For world coordinates, the equinox may also be included as part of the coordinate type specification, for example:

```
$image get $ra $dec "wcs 1950"
```

2.7 Simulation Tool *rtdctrl*

A simulation tool exists to simulate the action of a CCD camera to test the real-time functionality of the RTD. It is found in the directory `lib/rtd/demos`. To start the application go to this directory and type `rtdctrl`; the application front-end consists of a panel with a button to allow the loading of a FITS image, sliders to adjust the time between image send events, start, stop, and close buttons. When the start button is invoked, images are (effectively) sent from the CCD camera named "RTD-SIMULATOR" to the server daemon. The image information will be forwarded to any client which is attached to the simulator.

`rtdctrl` allows also the locking system to be tested explicitly by first inducing image jitter in the RTD and then removing this by applying semaphore locking to the shared memory areas.

See the User's Guide and the reference manual RtdServerTool(n) for more details.

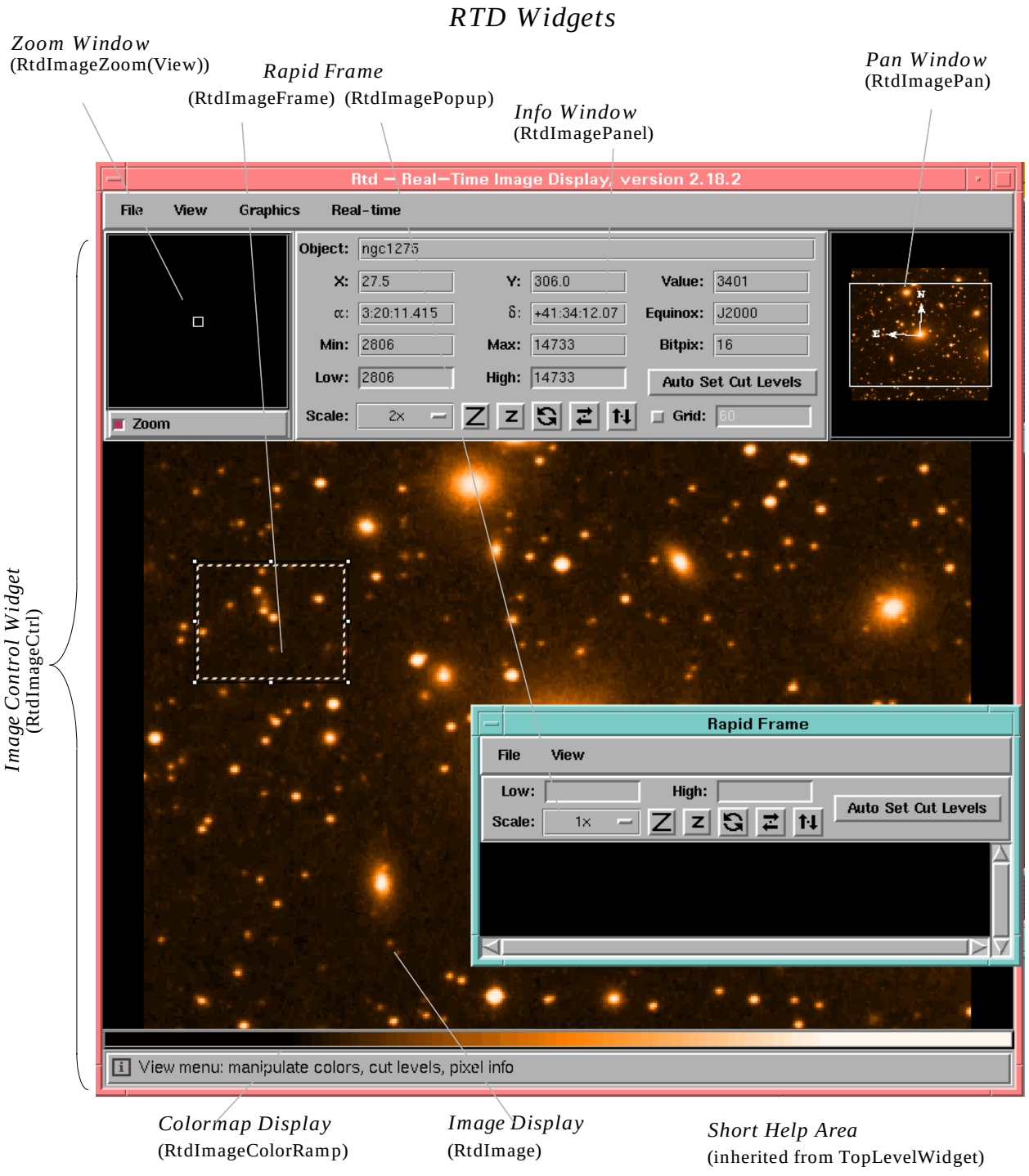
2.8 Demo Application *rtd*

To demonstrate and test all of the widgets and features in the RTD release, a demo application is included. It is found in the directory: `lib/rtd/demos`. The application, called *rtd*, displays a window with a test FITS image, a panning window, zoom window, control panel, menu bar, colormap display and short help display. To demonstrate the real-time display features, you start the *rtdServer* daemon first and then start the simulation by pressing a button in a control window. See the User's Guide and the reference manual rtd(1) for details.

3 User's Guide

This section describes, in a general way, all of the RTD widgets, how they are used in the *rtd* demo application and how they are implemented. For details, see the reference section at the end of this manual.

The best place to start, in order to get an idea what the software can do, is to look at the demo application `rtd` in the `rtd/rtdImage/demos` directory of the RTD release. The main window is shown below:



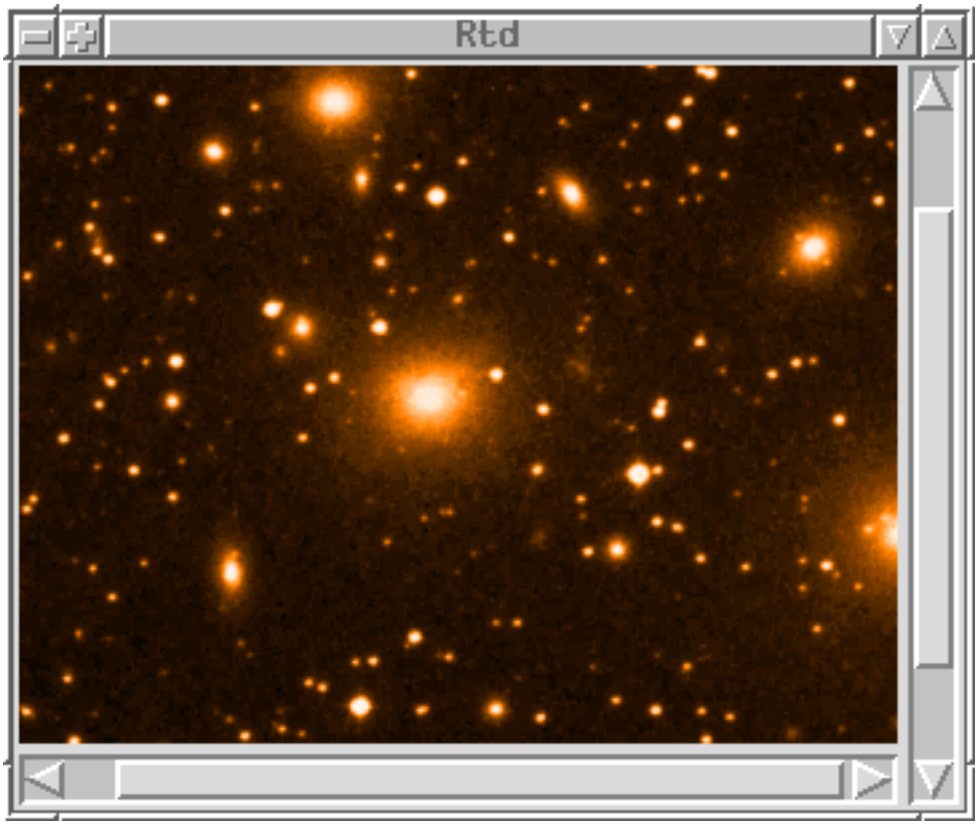
3.1 RTD Images and Widgets[1]

All of the widgets in the figure on the previous page that display an image (also the colormap display), are based on the extended Tk image type *rtdimage*. Tk4.0 introduced a new “image” command and a C interface for adding new image types. A Tk image is much like a Tk widget in that it is both an object and a Tcl command. *rtdimage* is designed for real-time image display. Images can be loaded from shared memory or FITS format files. For real-time usage, a background daemon process *rtdServer*(1) communicates with the rtdimage software over a socket interface to display and update images rapidly from shared memory. An *rtdimage* is created with the `image create` Tk command. After this, you can use the image in a Tk canvas by specifying it with the `-image` option. For example:

```
set image [image create rtdimage ...]
$canvas create image 0 0 -image $image ...
```

3.1.1 Image Window

A number of [incr Tcl] widgets have been developed around the *rtdimage* type. The `RtdImage` widget creates a canvas window and puts the image in it, so that it can be treated just like any other Tk widget and inserted in an application with the Tk *pack* command. In addition, the `RtdImage` widget implements methods and creates other widgets for dealing with line graphics, rapid frames, image color management, cut levels, pixel information and statistics. Besides the main image window, the zoom window, the pan window, rapid frame and colormap display window are all based on (i.e.: use an instance of) the `RtdImage` widget.



1. See the Reference Manual at the end of this document for details on the widget options and methods.

Below is an example that shows how to create a simple `RtdImage` widget and perform some simple operations, such as loading a fits file, setting the cut levels, scaling or displaying the graphics popup window.

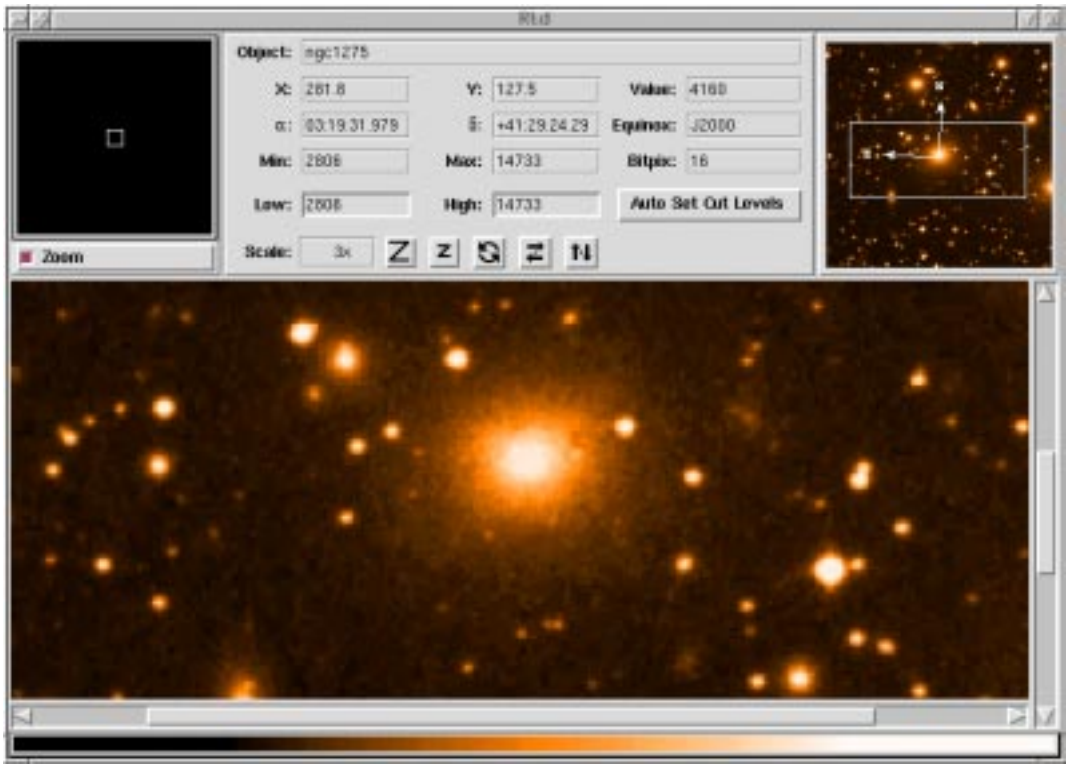
```
# create and pack RtdImage
pack [RtdImage .image -scrollbars 1] -fill both -expand 1

# load a test image, set the cut levels, scale 3x
.image config -file test.fits
.image cut 0 1000
.image scale 3 3
```

See *RtdImage(n)* for more details.

To make building applications easier, a compound widget `RtdImageCtrl` can be used. It includes a main image, zoom window, pan window, colormap display and image info display. This covers most of what an application will need for the main window, other than the menu bar and mini-help area.

If you replace `RtdImage` in the example above with `RtdImageCtrl`, you would get a window like the one below:



See RtdImageCtrl(n) for more details.

3.1.2 Zoom Window

The zoom window displays a magnified area of the image by tracking mouse pointer motion events. The zoom factor and window size are configurable and the user can turn zooming on and off via a toggle button.

Here is an example, taken from the `RtdImageCtrl` code, that creates and packs a zoom window:

```
pack [RtdImageZoomView $this.zoom \
```

```
    -target_image $this \
    -width $zoom_width -height $zoom_height \
    -factor $zoom_factor \
    -usexshm $usexshm \
    -bd 3 -relief groove] \
    -side left -fill y -in $panel
```

See the man page *RtdImageZoomView(n)* for an explanation of the options used.

3.1.3 Pan Window

The pan window, implemented by the `RtdImagePan` widget, displays a *view* of the main image at a smaller magnification and a rectangle indicating the size and position of the visible portion of the image. The pan window is given a maximum size at start-up and always tries to fit the image in the window by shrinking it by integer factors. The rectangle can be dragged around in the pan window with the left mouse button to *pan* or move the image. The panning rectangle is implemented using the same class as the main image uses for line graphics: class CanvasDraw.

Here is an example, taken from the `RtdImageCtrl` class, demonstrating how to create a pan window:

```
pack [RtdImagePan $this.pan \
    -target_image $this \
    -width $pan_width \
    -height $pan_height \
    -usexshm $usexshm \
    -verbose $verbose \
    -bd 3 \
    -relief groove] \
    -side right -in $panel
```

See *RtdImagePan(n)* for more details.

3.1.4 Colormap Display

The colormap display widget, `RtdImageColorRamp`, at the bottom of the image also uses an instance of the `RtdImage` widget with generated data to display, from left to right, equally spaced, all the colors in the colormap. The image data is generated by the `rtdimage colorramp` subcommand. Bindings in the colorramp window are set so that dragging with mouse button 1 will shift the colormap, dragging with Shift-1 will rotate it, dragging with button 2 will stretch or squeeze the colormap and dragging button 3 will reset the colormap back to its original state.

An `RtdImageColorRamp` widget can be easily created and inserted in the window hierarchy, as the following example shows:

```
pack [RtdImageColorRamp $this.colorramp \
  -height $colorramp_height] \
  -side bottom -fill x
```

See *RtdImageColorRamp(n)* for more details.

3.1.5 Image Info Panel

The `RtdImagePanel` widget displays a combination of other widgets with relevant image information, including: the image object name, file name or camera name (`LabelEntry`), the current raw image pixel and world coordinates, the current pixel value, the low and high image cut

levels, the minimum and maximum image pixel values, the current image scale (magnification) factor and the current settings for rotate and flip x and y. The cut levels are editable and hitting return in the entry sets the image cut levels. The image scale factor can be selected from a menu.

Here is an example, taken from class `RtdImageCtrl`, of how to create and pack an `RtdImagePanel` widget:

```
pack [RtdImagePanel $this.info \
  -image $this \
  -bd 3 -relief groove] \
  -side left -fill both -expand 1 -in $panel
```

See *RtdImagePanel(n)* for more details.

3.1.6 Rapid Frame

The rapid frame is designed to display a small section of the main image very rapidly, either in the main image (class `RtdImageFrame`) or in a popup window (class `RtdImagePopup`). The region being displayed in the rapid frame is marked by a black and a white dashed rectangle, which can be moved and resized interactively by dragging with the mouse. The rapid frame creates an instance of the `RtdImage` widget (for popup frames) or a second `rtdimage` *canvas image* item (for embedded frames), to display a section of the image. A notification facility is used to notify application specific code about changes in the position and size of the rapid frame, which normally will involve communication with the *rtdServer*, a daemon process that multiplexes real-time image events from CCD cameras to the display applications.

Here is an example showing how class `RtdImage` creates a rapid frame. This code is evaluated after the user draws a rectangle over the image. A callback procedure gets the coordinates and *Id* of the frame and creates it as follows:

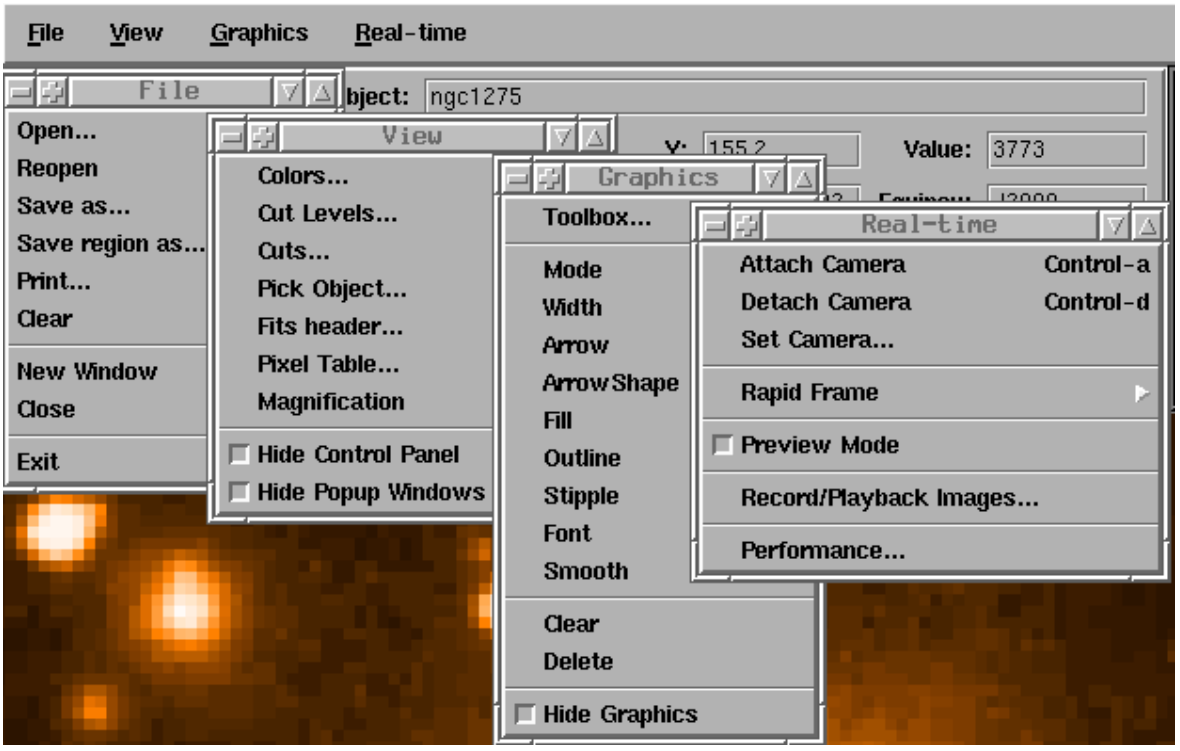
```
RtdImageFrame $this.rapid \
    -target_image $this \
    -popup $popup \
    -xoffset $xoffset \
    -yoffset $yoffset \
    -width $width \
    -height $height \
    -subsample $subsample \
    -usexshm $usexshm \
    -withdraw [expr !$popup] \
    -region_id $region_id \
    -verbose $verbose \
    -command $rapid_frame_command
```

Since this frame is displayed as a canvas image item, there is no need to use pack here. For more details, see the man pages *RtdImageFrame(n)* and *RtdImagePopup(n)* in the *Reference* section.

3.1.7 Mini-Help area

The bottom window in the example *rtd* application is the mini-help window. This window displays a short help text whenever the mouse pointer enters a widget. Methods for managing this window and its contents are inherited from one of the widget base classes `TopLevelWidget` or `FrameWidget`. See the *tclutil* package documentation for a description of these classes.

3.2 RTD Menus



The figure above shows some of the menus available in the menu bar in the *rtd* demo application (the menus are shown in the *tearoff* state). There are four main menu buttons: *File*, *View*, *Graphics* and *Real-time*, and a number of pullright (cascade) menus.

3.2.1 File Menu

The *File* menu contains general file handling utilities, for example entries for *opening* and *saving* FITS files and for *clearing* the image display.

The Open item allows a current FITS image to be displayed.

Reopen can be used to refresh the image after the file was modified via shared memory.

The Save as... item opens a file selector dialogue and allows the current image to be saved as a FITS file.

Save region as... allows you to save a selected region of the image to a FITS file.

The Print item prints out the current display.

The Clear item clears the display.

The New Window item clones the RTD.

The Close item closes the RTD.

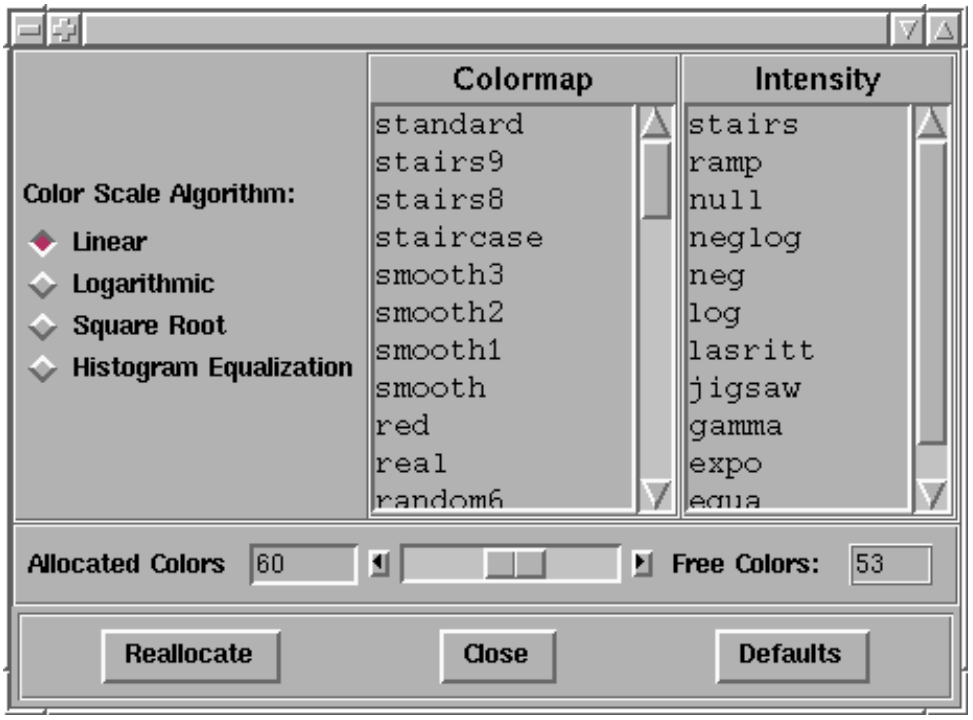
The Exit item quits the application.

3.2.2 View Menu

The *View* menu contains items that affect the look of the image, for example by manipulating the

colors or cut levels, and items that give you some view of the image data, such as a graph of the pixel distribution.

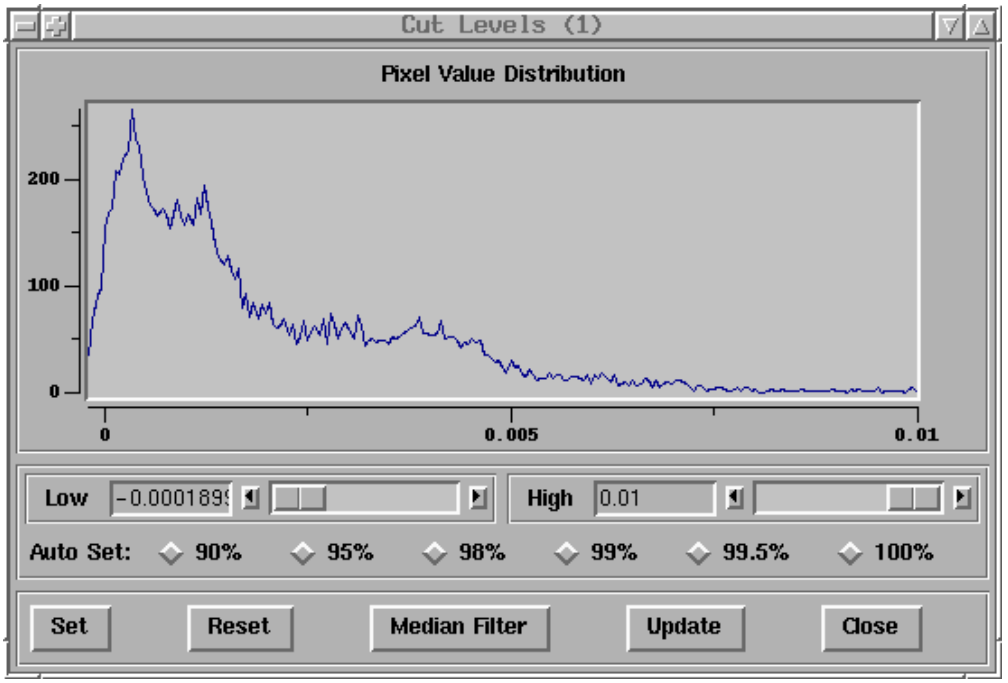
Colors... pops up a window to set the colormap, intensity, color scaling algorithm and the number of colors allocated.



Selecting a colormap from the list installs a new colormap from the relevant MIDAS colormap file. Selecting an intensity, modifies the current colormap by applying the given MIDAS intensity file. The number of colors allocated can be set with the scale widget. In this way, you can decide to use more colors for better image quality, or fewer colors, so that other applications will still be able to get enough colors. See *RtdImageColors(n)* for a description of this widget.

The *Cut Levels* item pops up a window to set the image cut levels, or the lowest and highest pixel values considered when distributing colors to pixel values.

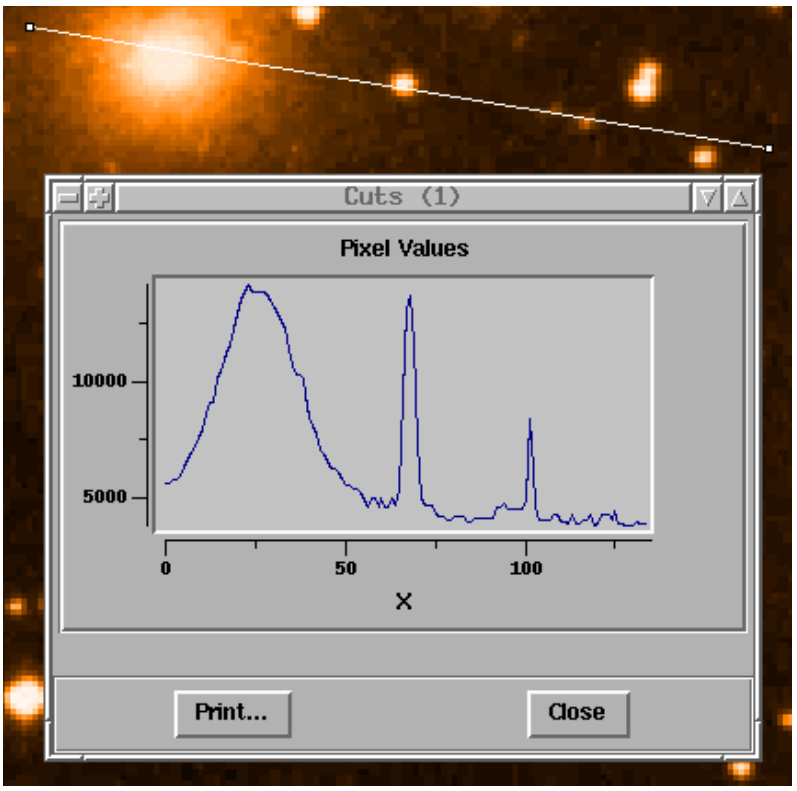
.



The graph shows a plot of the pixel value distribution in the image and the scales, arrow buttons and entry widgets allow you to manipulate the cut levels and the graph. The *Auto Set* buttons allow you to set the cut levels by *percent of pixels* within the cut levels. The *Set* button sets the cut levels to the selected values. *Reset* sets the low and high cut levels to the min and max image values resp. *Median Filter* applies a median filtering algorithm to the image to set the cut levels. *Close* removes the window again. See the man page *RtdImageCut(n)* for more details.

The *Cuts...* item sets the drawing mode in the canvas to *line* and changes the cursor to reflect this. After the user drags out a line on the image, a popup window is displayed with a graph of the pixel values along the given line

.



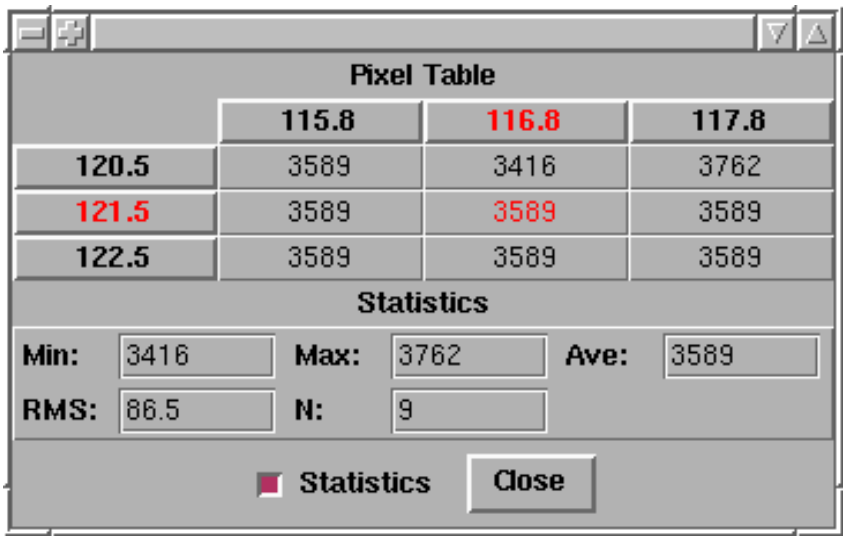
Once the spectrum graph is displayed, dragging the line or resizing it updates the graph with the image values under the line. See *RtdImageSpectrum(n)* for a description of this widget.

The *Pick Object*... item pops up the Pick Object window, allowing you to select an object in the main image. Once an object is selected, statistics are displayed about the object, including the center coordinates in image pixles and world coordinates (if supported by the image), the object size and rotation, calculated using a FWHM algorithm. The selected object is marked by a blinking cross in the image, which also shows the size and rotation of the object. You can also zoom the small image in and out to get a better view of the object.

This widget is described in *RtdImagePick(n)*.

The *FITS Header...* item displays the FITS file header information in a text window, in the cases where the image was loaded as a FITS file (For real-time images, the header is normally empty, since the necessary information is received via socket from the rtdServer).

The *Pixel Table* item pops up a window displaying a table of pixel values while tracking mouse pointer motion events.



The row and column headings are the X and Y coordinates in the raw image and the table items are the image pixel values at the given coordinates. Note that normally, the origin is at lower left in the raw image. This changes when the image is rotated or flipped. The table is constantly updated

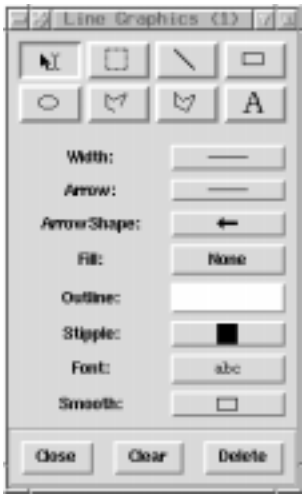
when the mouse pointer is moved over the image. See *RtdImagePixTable(n)* for more details.

The *Magnification* item brings up a new menu which allows the user to select to zoom ratio for the RTD.

The three toggle buttons at the bottom of the *View* menu are self-explanatory, allowing the user to turn certain components of the RTD on or off.

3.2.3 Graphics Menu

The *Graphics* menu contains entries for interactive line graphics editing in the canvas window. The line graphics are normally restricted to the area of the image. For convenience, the menu offers the same functions as are available in the popup window:



The top of the *Canvas Graphics* popup has an area for setting the drawing mode. The other items are used for setting options, such as line width, arrow type, fill and outline color. These all correspond to the *canvas* item options of the same name. The `CanvasDraw` widget that implements the line graphics here is a generic canvas graphics widget and not RTD specific. It is part of the *tclutil* package, on which the *rtd* package is based.

3.2.4 Real-time Menu

The *Real-time* menu contains entries that deal with real-time functionality.

Attach Camera attaches the RTD to the current CCD (see *Set Camera* below) via the rtdServer daemon process. Any images received by the server from the CCD will then be forwarded to and displayed on the RTD.

Detach Camera detaches the RTD from the currently attached CCD.

Set Camera brings up a simply entry dialogue to allow the user to enter the name of the current CCD camera. This is not attached to until *Attach Camera* is invoked.

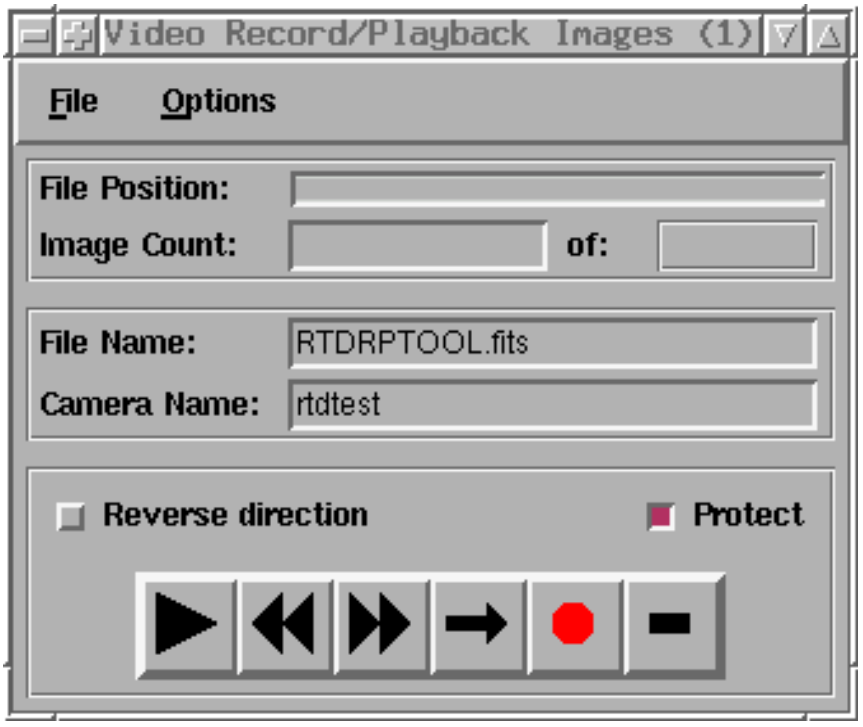
The *Rapid Frame* entry lets you create a rapid frame either in the *canvas* window or in a *separate* popup window. This sets the drawing mode in the canvas containing the main image and displays a *left button* cursor to indicate that you should drag out an area on the image with mouse button 1.

The *Preview Mode* item is useful when in real-time mode. It makes a local copy of the shared memory image from a camera, so that it will not be overwritten or destroyed. This is a toggle button, so selecting it again will return you to real-time mode. If no camera is running, this button does nothing.

The *Record/Playback Images...* item allows the recording of images from a camera to file, and the playback of images from file to the RTD. The images are recorded as a FITS cube; thus this tool can be used to replay FITS cubes produced by other applications.

In 'Record' mode, the tool allows the user to select a maximum size for the output file (the default being set to 5 Mb). In addition, the 'cycle mode' can be toggled on or off: this indicates whether the recording should automatically loop round to the beginning of the file when the file is at its maximum size. It is also possible to select a specific sub-region of the incoming images to record; this could be useful when the incoming images are very large or the area of interest is small.

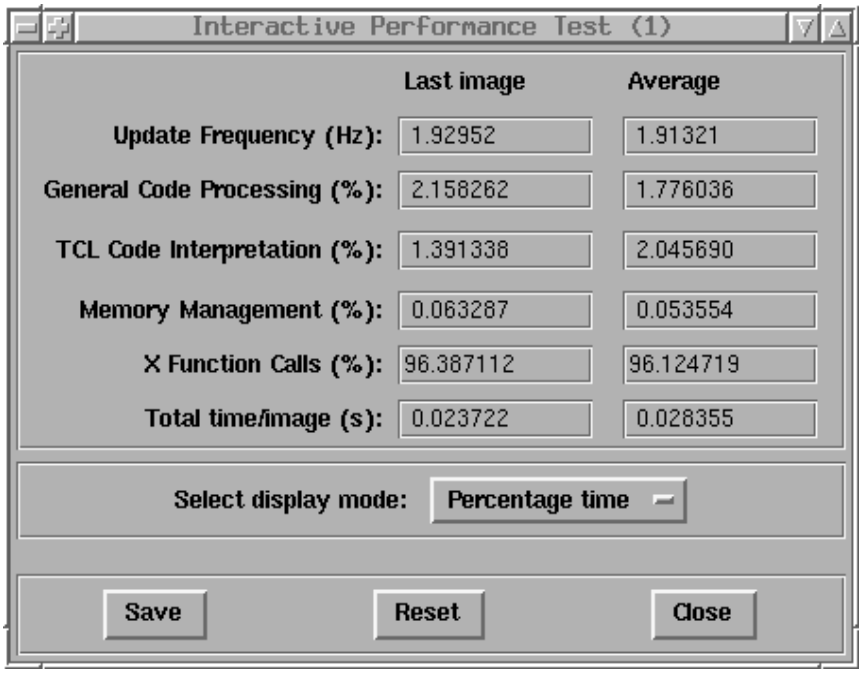
In 'Playback' mode, the cycle mode still allows the user to select whether the playback will operate in a continuous loop. The speed of the playback is also variable, with the user able to specify slow, fast, or real-time updates. The real-time update (i.e. images re-displayed at the same rate as they were received) is only available on FITS cubes produced using the recorder tool; if the option is selected on another cube file, the playback speed defaults to 'slow'.



It is also possible to playback images in reverse order and to single-step through the image file (return one image to the RTD at a time).

When the dialogue is invoked initially, the camera name specified is the current default RTD camera. When recording is commenced, the recorder tool will connect to the camera named in the 'Camera Name' field. When playback of images is started, the parent RTD automatically attaches to the camera 'RTDRPTOOL' and receives images in the usual manner. When playback finishes, the RTD detaches from this camera and reattaches to its original camera.

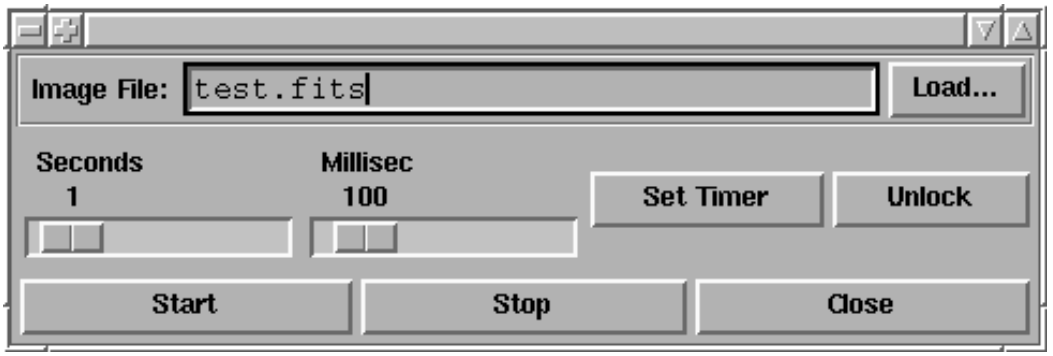
The *Performance...* menu item pops up a dialogue containing interactive performance data. These data refer to parameters that relate to the amount of processing carried out within the RTD on certain chores. These chores are: Tcl code interpretation, memory management, X function calls, and general processing.



The performance data can be displayed in three ways; as absolute time per image event, as percentage of total time per image event, or as normalised time. In the latter case, the X function calls and memory management processes are divided through by the size of the image (in kB) to allow comparison of performance across platforms for different image types and sizes. The normalised format is not applicable to general processing and Tcl code interpretation.

Support for a “benchmarking performance tester” has been added to the RTD release. See the reference section, RtdPerfTester(n) for more information.

The *Rtd Simulation Control* pops up a window to control the real-time simulation used for testing the RTD display and real-time functionality when no camera interface is available. To start the application, go to the `demos` directory and type `rtdctrl`.



To start the simulation, you load a FITS file in the dialogue and press the *start* button to notify the rtdServer to start sending images. Then select *Attach Camera* from the File menu of the RTD to tell the rtdimage to start listening for real-time image events. *Detach Camera* stops the application from accepting images from the rtdServer.

`rtdctrl` allows also the locking system to be tested explicitly by first inducing image jitter in the RTD and then removing this by applying semaphore locking to the shared memory areas. Load a

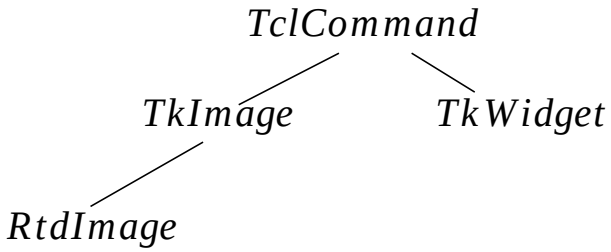
suitable FITS file (avoiding floating-point formats as these can produce floating-point exceptions when they jitter) and unlock the server by invoking the *Unlock* button. After this, send the images to the server by invoking *Send* and adjust the update rate until image jitter is obtained. If the server is currently multi-buffered, it may be necessary to change the number of buffers to 1 in the source code for rtdImgEvt and rtdimage libraries: see the entry for rtdlock(1) in the reference section for instructions on how to do this.

The RTD and server daemons should be transparent in the absence of implementation of the locking. When the *Lock* button is invoked, locking should be implemented on the shared memory area, and the image jitter should vanish (albeit at a slower update rate).

3.3 Implementation

3.3.1 Central C++ Classes

The central C++ class in the implementation of the *rtdimage* Tcl command and image type is `RtdImage`. It is a subclass of `TkImage`, which handles the generic image related tasks. `TkImage` is a subclass of `TclCommand`, which handles the Tcl command related features. Both of these classes are part of the *tclutil* package, which the *rtd* package uses. Class `RtdImage` implements most of the *rtdimage* subcommands and configuration options.



Manipulating the image data itself is done by the class `ImageData` (see *RTI* library below). `RtdImage` keeps a pointer to an `ImageData` object to manage transforming the raw image to an XImage for display.

The actual display of the image is handled by the `ImageDisplay` class. It decides whether or not it can use X shared memory to improve performance and handles the creation of the XImage and the copying of the image to the X server for display.

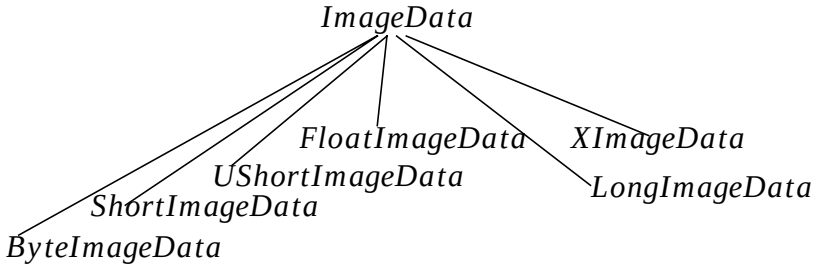
`RtdImage` uses a class derived from `RtdCamera` to manage real-time image events. `RtdCamera` hides the interface to the real-time server daemon process and takes care of managing the shared memory for the images and the socket communication with the server.

See the *Reference* section of this manual for a complete description of all classes used.

3.3.2 RTI - C++ Real-Time Interface Library for Manipulating Images

The RTD software includes a separate C++ class library for manipulating images. Through the base class *ImageData(3)*, you can create images of any raw data type and convert them to XImages for display, using any one of a number of color scaling algorithms. Scaling (resizing), rotating and flipping are supported as well as *median filtering* and cut levels. The details of the different underlying raw image data types is hidden by use of virtual functions and, in some cases, cpp macro templates

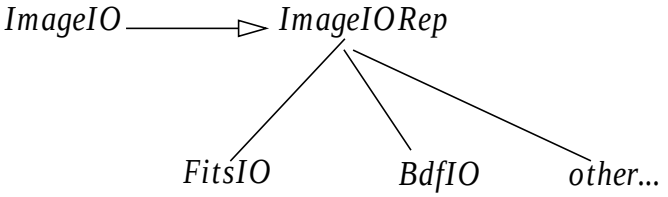
for subclass methods[1].



This class library is independent of *X* or any other libraries (other than the standard system libraries for C and C++), so it could, in principle, be used by client applications, perhaps running on different machines, to preprocess images before sending them to be displayed. The rtdimage display software will accept images in standard FITS style (with origin at lower left) or in already finished XImage style (class `XImageData` - with origin at upper right and no color scaling needed). In order to use the library, an application needs to have an image, either as a FITS file or as a pointer to image data in memory (or shared memory) and the application needs to know how many colors are available and what their values. This information can be obtained via the RTD *remote* interface (see rtdRemote(3). The `ImageData` class is described in the man page *ImageData(3)* in the *Reference* section.

3.3.3 Adding New Image Formats

Currently, the RTD only supports FITS format images. Although it is planned to keep FITS as the internal image format, we do plan to add support for other image formats. This will be done by deriving subclasses the abstract base class `ImageIORep`. Note that The public interface is through the class `ImageIO`, which uses a pointer to an `ImageIORep` subclass for reference counting.



To add support for a new image format, you can derive a class from ImageIORep that reads in the image and converts it to FITS format. (Note: Class BdfIO, for Midas BDF images, is not implemented yet). Note that these classes are now part of the *astrotcl* package, on which the rtd package is based.

3.4 Programming with Real-Time Images

One of the main features of the RTD widget is the ability to display real-time images. These images are typically coming from an aquisition system that controls a CCD camera or other kind of detectors such as the DCS (Detector Control Software). This section describes how programmers providing such real-time images can interface to the RTD and display their images.

As already described in the architecture overview, the real-time image must be provided in shared memory by the aquisition system. This means that the aquisition system is the creator and owner of the shared memory segment (see also *`shmget(2)`*, *`shmctl(2)`*, *`shmat(2)`*) .

1. We used cpp macros rather than C++ templates here because C++ templates are not very portable across compilers yet - at least not the compilation part. Besides, the cpp macros are, in some ways, more flexible. You can choose which methods to define as templates and which to define as special cases.

To interface with the RTD widget, the library *rtdImgEvt* is provided with the following functions:

Routine	Description
`rtdInitImageEvt`	Register to rtdServer, usually as a camera identified by a unique name.
`rtdSendImageInfo`	Send image event to rtdServer.
`rtdRecvImageInfo`	Receive an image event from rtdServer (normally used only by the RTD widget).
`rtdAttachImageEvt`	Subscribe for event notification from rtdServer (normally used only by the RTD widget).
`rtdDetachImageEvt`	Unsubscribe for event notification from rtdServer.
`rtdClose`	Close connection to rtdServer.

The basic idea is that an application providing real time images, such as DCS, uses *`rtdInitImageEvt`*() to identify itself to the rtdServer, for example.:

```
#include “rtd/rtdImageEvent.h”
rtdIMAGE_EVT_HNDL  eventHndl;
if (rtdInitImageEvt(“myCamera”,&eventHndl,NULL) == RTD_ERROR){
        printf(“Could not initialize image event !\n”);
        usage();
}
```

After this call, a camera source named “*myCamera*” is registered with the rtdServer. The handle returned of type rtdIMAGE_EVT_HNDL is a handle needed for subsequent calls to rtdImgEvt functions.

When the camera source has a new image ready in shared memory, the structure rtdIMAGE_INFO describing the image must be prepared. rtdIMAGE_INFO contains of following fields (Any unused fields should be set to 0, see memset(3)).:

Member	**Description**
`char version`	Protocol version (filled by rtdSendImageInfo)
`char` **`frameId`**	Rapid frame identifier. 0 means normal image. 1 and above identifies the the image is a rapid frame.
`char` **`dataType`**	Defined in the rtdIMAGE_TYPE enumeration: BYTE, SHORT, XIMAGE, SHORT, USHORT, INT, FLOAT, DOUBLE. XIMAGE means that the image provided already has been color scaled.
`char` **`bytePerPixel`**	Number of bytes used per pixel.
`int` **`shmId`**	ID for the shared memory block
`short frameX,` `short frameY`	X,Y offsets of the image frame in the canvas (used only for rapid frames and should otherwise be set to 0).
`short` **`xPixels`**	Pixels in horisontal direction (equals the width of the image).
`short` **`yPixels`**	Pixels in vertical direction (equals the height of the image)

Member	Description
`short blockLines`	If different from 0, block mode is applied. Block mode means that only a small part of the image changed and the rtd widget only has to update a fraction of the image (currently not implemented).
`short blockOffset`	Y offset on image (currently not implemented).
`int highCut`	High cut level as set by the aquisition system.
`int lowCut`	Low cut level as set by the aquisition system.
`short binningX`	Binning factor applied on image
`short binningY`	Binning factor applied on image.
`struct timeval timeStamp`	UTC time when image was aquired.

The following fields were added to support World Coordinates Set all fields to 0 if World Coordinates are not supported

Member	Description
`double ra`	Center right ascension in degrees
`double dec`	Center declination in degrees
`double secpix`	Number of arcseconds per pixel
`double xrefpix`	Reference pixel X coordinate
`double yrefpix`	Reference pixel Y coordinate
`double rotate`	Rotation angle (clockwise positive) in degrees
`int equinox`	Equinox of coordinates, 1950 and 2000 supported
`double epoch`	Epoch of coordinates, used for FK4 / FK5 conversion, no effect if 0
`char proj[8]`	Projection: one of: “-SIN”, “-TAN”, “-ARC”, “-NCP”, “-GLS”, “-MER”, “-AIT”, “-STG”, “PLATE”, “LINEAR”, “PIXEL”, ... (as supported by the wcs library: *wcssubs*, from Doug Mink)

The following fields were added to support image synchronization

Member	Description
`int semId`	ID of semaphore set
`int shmNum`	Number of semaphore in the set
`int reserved[10]`	reserved for future use

These fields were added to support detector "*chip*" coordinates for real-time images. The chip origin

Member	Description
`short startx` `short starty`	First window pixel in the X (Y) direction within the detector physical system.

is assumed by be at lower left, as for FITS.

The image event is forwarded with the function *rtdSendImageInfo* to the rtdServer. The fields marked as bold are **mandatory** and must have a value in order to display the image.

In order for rtd clients (rtd widget applications) to receive the event, they must first attach to the camera source, which in this case is "*myCamera*". As an example of an application the rtd demo application can be used, for example :

```
rtd -camera myCamera
```

and from the real time menu select "*Attach Camera*". For more information on the functions mentioned above please refer to the man pages. And remember if you program with shared memory images to delete them when they are no longer needed (such as when your aquisition system exits).

The RTD include files and libraries are normally installed in $*prefix*/include and $*prefix*/lib, which must be added to the path in the Makefile (For the VLT, $*prefix* is normally set to `/vlt/dflow`. The default is `/usr/local`). $

```
RTD_LIB=-L$(PREFIX)/lib -lrtd
RTD_INCLUDE=-I$(PREFIX)/include
```

3.4.1 Multi-buffering and Semaphore Locking of Shared Memory

The RTD and the server support the use of multiple areas of shared memory in a *multi-buffering* scheme. This means that the CCD software creates several shared memory areas and cycles the image data around these as it is created. This reduces the possibility of the CCD writing to shared memory at the same time as the RTD reads from the same segment (*image jitter*). The number of areas of shared memory that can be created is resource dependent: for very large images it may not be possible to create more than one segment. In addition, systems have a limit on the number of shared memory areas that can be created. With Solaris 2.5, this can be altered by adding the following line to the `/etc/system file`:

```
set shmsys:shminfo_shmseg=200
```

As an additional insurance against this image jitter, it is possible to use semaphores to lock areas of shared memory. The use of semaphores is not mandatory (the server and the RTD are transparent to their implementation); however, convenience routines are provided in the rtdImgEvt library to create and manage shared memory and the corresponding semaphores together.

The locking scheme implemented by the RTD and the server daemon process is as follows. When the shared memory is created, a semaphore set is created, the number in the set being the same as the number of shared memory segments. Thus each member of the set corresponds to an item of shared memory, and so shared memory can be locked by setting the appropriate semaphore to a high state. Shared memory is locked with respect to the CCD; the CCD should be prevented from writing to the buffer when the semaphore corresponding to that buffer is high.

Before the CCD dumps image data to a shared memory buffer, it should set the corresponding semaphore to one. The image event is then sent to the server daemon, which increments the same semaphore by the number of RTD clients minus one. When each RTD client has finished with the data (i.e. has displayed the image) the semaphore is decremented by one. Thus when all RTD cli-

ents have finished with the data, the semaphore will be zero and the CCD software will again be free to write to the shared memory.

After the server has sent the images to the RTDs, it is up to the RTDs to ensure that each image event is dealt with, otherwise (in the case of skipped images) semaphores may be left in high states and this could lock up the CCD completely. Thus, even if an image event can not be displayed (due to a high update rate) it is essential that the image event is at least serviced to decrement the appropriate semaphore.

In addition, each semaphore has a timeout period (which is `#defined` in rtdSem.h). If a semaphore is set for longer than this period, it is assumed to be dead, and can be reset by the CCD. This protects against the possiblity of an RTD crashing and leaving a semaphore in a high state.

The following convenience routines exist for the benefit of the CCD software developer who wishes to interface with the shared memory locking offered by the RTD:

Routine	Description
`rtdShmCreate`	creates the required number of shared memory buffers, and enough semaphores to cover these.
`rtdShmFill`	fills the required shared memory buffer with image data, and locks this buffer with the appropriate semaphore.
`rtdShmFillFirst`	cycles over all currently allocated buffers, and fills the first buffer that is not locked.
`rtdShmLocked`	determines whether a given buffer is locked or not.
`rtdShmStruct`	fills the image information structure with shared memory / semaphore information prior to being sent to the server.
`rtdShmDelete`	removes semaphores and shared memory buffers from the system.

To support this additional functionality, the following fields have been added to the image information structure. As mentioned above, these are not mandatory.

- int semId - the semaphore identifier assigned by the system,
- int shmNum - the number of the shared memory buffer in the multi-buffered cycle (e.g. if five buffers were allocated, this would be a number between 0 and 4).

These are filled in automatically if the **rtdShmStruct** routine is used (this also fills in the shared memory ID).

An example of how these routines may be used is given below:

```
#include “rtdSem.h”
#include “rtdImageEvent.h”

#define WIDTH           128
#define HEIGHT          128
#define DATASIZE        16
#define NUM_BUF         5

static void generate_data(char *);

void main() {
    rtdIMAGE_EVT_HNDL  eventHndl;
    rtdIMAGE_INFO      imageInfo;
    rtdShm             shmInfo;
```

```
        char               *data;
        unsigned int       i = 0;

        memset(&imageInfo, '\0', sizeof(rtdIMAGE_INFO));

        if (rtdInitImageEvt("My_CCD_Camera", &eventHndl, NULL) == RTD_ERROR) {
            /* ... handle error ... */
        }

        if (rtdShmCreate(NUM_BUF, &shmInfo, WIDTH, HEIGHT, DATASIZE) == -1) {
            /* ... handle error ... */
        }

        while ( [some condition] ) {
            generate_data(data);

            while (rtdShmFill(i, data, &shmInfo, 0) == -1) {
                i++;
            }

            imageInfo.dataType = DATASIZE;
            imageInfo.xPixels  = WIDTH;
            imageInfo.yPixels  = HEIGHT;
            imageInfo.frameX   = 0;
            imageInfo.frameY   = 0;
            imageInfo.frameId  = 0;

            rtdShmStruct(i, &imageInfo, &shmInfo);

            /* forward image event */
            rtdSendImageInfo(&eventHndl, &imageInfo, NULL);

            sleep(1);

            i = (i + 1) % NUM_BUF;
        }

        rtdShmDelete(&shmInfo);
    }
```

This is a very simple way of sending locked data to the server, and does not use all the functions mentioned above. The functions are designed to allow as much flexibility in the CCD software as possible.

Additional routines are specified in rtdSem.c. These are not currently useful in the CCD software as they are not consistent with the locking scheme described above. They are used in the RTD, and are described briefly below in case they become useful to the CCD at a later development stage.

Routine	Description
`rtdShmServicePacket`	Given an image event structure, this routine decrements the required semaphore. This is used by the RTD to service missed image events.
`rtdShmDecrement`	Simply decrements the required semaphore.
`rtdShmReset`	Simply sets the required semaphore to zero.

3.5 Interfaces for Remote Access

3.5.1 Remote Control C Interface Library

The RTD supports remote access via a simple socket interface. This interface works a bit like the Tk *send* command in that you send a command to execute and get the result back, however in this case, only RTD subcommands are allowed. Remote clients can connect to a running RTD widget and send RTD subcommands to be executed and get the results. Some of the RTD subcommands are designed especially for use with this interface. For example, the *shm* command gives access to the image header and data as shared memory. Remote clients can execute any of the RTD image subcommands. In addition, applications can extend the list of available remote commands by specifying a Tcl command to be called for unknown subcommands.

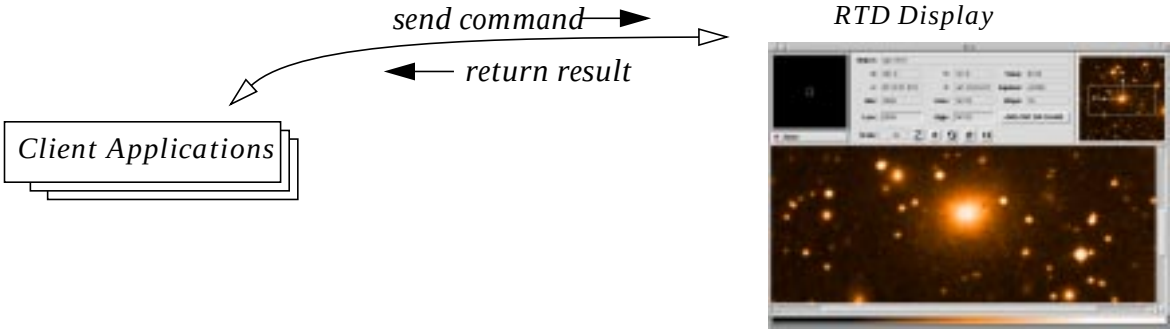
The remote control interface is implemented on the client side by the client writing to and reading from the socket or optionally through a simple C library (module *rtdrmt*). On the server side, the C++ class *RtdRemote(3)* is used to start listening for connections. By default, a port number is chosen automatically and the information is saved in the file `$HOME/.rtd-remote` as 3 values: *processID*, *hostname* and *port*. Since the server is started as an rtdimage subcommand (*remote*), you can also specify a port number to use.

Client commands are sent as *strings* over a socket, one *line* for each command sent. The strings are interpreted directly as rtdimage subcommands. The client waits for a reply on the same socket used to send the request. The reply consists of 2 lines. The first line contains 2 numbers: the *status* of the command (0 for OK) and the *length* of the command result that follows.:

Socket Protocol:

Command Format	**Result Format**
command <newline>	status length <newline> result[length]

The figure below illustrates the connection:



3.5.1.1 Example Usage

The following example demonstrates the use of the remote RTD interface, using the example C library:

```
static char* send_rtd(char* cmd)
{
    char* result = NULL;
    int status = rtdRemoteSend(cmd, &result);
    printf("%s ==> %s: %s\n", cmd, (status ? "FAILED" : "OK"), result);
    return result;
}
```

```
main()
{
    int data_id, header_id;

    /*
     * open the
     */
    if (rtdRemoteConnect(0, NULL, 0) != 0)
        exit(1);

    /* send some commands to RTD to be evaluated */
    send_rtd("wcscenter");
    send_rtd("bitpix");
    send_rtd("scale");
    send_rtd("width");
    send_rtd("height");
    send_rtd("config -file ngc1316r.fits");
    send_rtd("width");
    send_rtd("height");

    data_id = atoi(send_rtd("shm get data"));
    header_id = atoi(send_rtd("shm get header"));

    exit(0);
}
```

The following example shows how you could use the remote interface without using the C library. This example uses Tcl (with the TclX extension):

```
# open a socket to a running Rtd application and return the file
# descriptor for remote commands

proc connect_to_rtd {} {
    global env
    # get the hostname and port info from the file ~/.rtd-remote,
    # which is created by rtdwhen the remote subcommand is used
    if {[catch {set fd [open $env(HOME)/.rtd-remote]} msg]} {
        puts "can't open ~/.rtd-remote: make sure rtd is running: $msg"
        exit 1
    }

    lassign [read $fd] pid host port
    close $fd

    if {[catch {exec kill -0 $pid} msg]} {
        puts "could it be that rtd is not running? ($msg)"
        exit 1
    }

    set fd [server_connect -nobuf $host $port]
    return $fd
}

# send the command to rtd and return the results or generate an error

proc send_to_rtd {args} {
    global rtd_fd
```

```
        puts $rtd_fd $args
        lassign [gets $rtd_fd] status length
        set result {}
        if {$length > 0} {
            set result [read $rtd_fd $length]
        }
        if {$status != 0} {
            error $result
        }
        return $result
    }

    puts "testing the RTD remote interface..."

    # open the connection
    set rtd_fd [connect_to_rtd]

    # this allows us to use the remote rtdimage as if it were local
    set image send_to_rtd

    # send some commands to RTD to be evaluated
    puts "WCS image center is: [$image wcscenter]"
    puts "image type: [$image bitpix]"
    puts "current scale factor: [$image scale]"
    puts "image dimensions: [$image width] x [$image height] pixels"
    puts "loading a new file: ngc1316r.fits [$image config -file ngc1316r.fits]"
    puts "setting cut levels: [$image autocut]"
    puts "new image dimensions: [$image width] x [$image height] pixels"

    puts "shared memory Id for image data: [$image shm get data]"
    puts "shared memory Id for image header: [$image shm get header]"
```

Here is the output from the above script (rtd was already running):

```
testing the RTD remote interface...
WCS image center is: 03:19:48.243 +41:30:40.31 J2000
image type: 16
current scale factor: 1 1
image dimensions: 353 x 353 pixels
loading a new file: ngc1316r.fits
setting cut levels:
new image dimensions: 512 x 512 pixels
shared memory Id for image data: 7400
shared memory Id for image header: 601
```

3.5.2 RTD Features and Subcommands that Support Remote Interfaces

The RTD remote interface is made most useful by a number of supporting features in the *rtdimage* (Tcl level) command interface. For example:

- Access to the image header and data in *shared memory* (set or get the shared memory Id for the image data or header and specify who should delete it when done.
- Coordinate arguments may be specified in any supported *coordinate system* and there are commands to convert coordinates from one coordinate system to another.

3.5.2.1 Shared Memory Access to Image Header and Data

The RTD uses the FITS image format internally. Currently no other formats are supported, however when they are added, the other formats will be converted to FITS internally. The RTD can be told to use sysV shared memory for the FITS image data and / or header on start-up (by default image files are memory mapped using *mmap*). If an application attempts to get the sysV shared memory Id for an image, the RTD automatically starts using shared memory for it. External applications can use the shared memory created by the RTD or they can create their own and tell the RTD to use it. The RTD *shm* subcommand controls the sysV shared memory access. It has the following syntax:

```
<imageName> shm set $data_size $data_id $data_owner
                    ?$header_size $header_id $header_owner?
<imageName> shm get data
<imageName> shm get header
<imageName> shm create $size
<imageName> shm delete $Id
<imageName> shm update
```

The *shm set* subcommand allow you to set the shared memory Ids to use to access the image data and header. The data and header in the area specified should be in FITS format. If the header is not specified, the previous header is reused. For both data and header, the size of the area (in bytes) and the shared memory Id must be specified. In addition a flag indicating who *owns* the shared memory is specified (if true, then the area will be deleted when no longer needed).

The *shm get* subcommand returns the shared memory Id of the data or header. If the data or header is not currently in shared memory, it is copied to a new shared memory area and the Id for this area is returned.

The *shm create* subcommand creates a new shared memory area with the given size and returns the Id. The memory should be deleted with the *shm delete* subcommand when no longer needed.

The *shm delete* command deletes the shared memory with the given Id (which should have been returned from the *shm create* subcommand).

The *shm update* command causes the display to be updated to reflect any changes in the image memory.

3.5.2.2 Coordinate Systems

The RTD understands and supports the following coordinate systems:

Coordinate System	Description
canvas	canvas coordinates (canvas scroll area)
screen	canvas window coords (visible area)
image	basic image pixel coords (at mag 1, no transformations)
chip	detector chip coordinates.
wcs *equinox*	world coordinates in H:M:S
deg *equinox*	world coordinates in degrees

Where it makes sense to do so, the RTD subcommands accept coordinates of the form:

```
$x $y $coord_type
```

Where $x and $y are the coordinate values and the coordinate type is one of the types in the above table. For world coordinates, the equinox may also be included as part of the coordinate type specification, for example:

```
$image get $ra $dec “wcs 1950”
```

There is also a subcommand *convert* for explicitly converting coordinates from one system to another, for example:

```
$image convert coords $ra $dec “wcs 1950” x y canvas
```

converts $ra and $dec in equinox 1950 to canvas coordinates $x,$y, while:

```
$image convert dist $sw $sh screen iw ih image
```

converts a screen distance $sw,$sh to an image distance $iw,$ih.

4 Reference

Following is a list of man pages from the various widgets, programs and library calls provided in the RTD software.

4.1 COMMANDS

4.1.1 rtd(1)

NAME

rtd - real-time image display application

SYNOPSIS

rtd ?options? ?filename?

DESCRIPTION

The rtd application is used to diplay FITS images in real-time. The application is based on "rtdimage", a Tk image extension for displaying FITS images.

OPTIONS

-file name
"name" specifies a FITS format file to load and display.

-subsample bool
If bool is true, subsampling is used when shrinking the image, i.e.: if the image is shrunk by 1/3, only every third pixel is displayed. Otherwise, the maximum value is taken from the group of pixels.

-usexshm bool
If bool is true (default), attempt to use X shared memory for the image display, if available. This improves performance considerably, but is only available when working on the system console.

-usexsync bool
If true, try to use X synchronisation.

-verbose bool
If bool is true, diagnostic messages are printed out to show what is going on internally (for debugging use).

-default_cmap
This option sets the default colormap file to use when starting up. Only the root of the filename should be specified for this option, for example: "ramp" for a grey level colormap. For a list of available colormap files, see the colormaps directory in the rtd release.

-min_colors n
-max_colors n
Specify the min and max number of colors to allocate before using a private colormap.

-default_itt
This option is similar to -default_cmap, except it sets the default ITT (intensity transfer table) file to use at startup. ITT files are also stored in the colormaps directory.

-xscale xs
-yscale ys
Set the default scaling factors (default: 1).

```
-camera name
        Set the camera name for real-time image events:
        default: taken from the RTD_CAMERA environment
        variable, if set, otherwise set to RTDSIMULATOR, for
        simulation test mode.

-zoom_factor number
        Set the scale factor for the zoom window (default 5 x
        the original image).

-colorramp_height h
        Set the height of the colorramp subwindow (default: 20).

-with_zoom_window bool
        If bool is true (default), add a zoom window.

-with_pan_window bool
        If bool is true (default), add a panning window.

-dozoom bool
        If true, turn on zoom window.

-disp_image_icon bool
        If true, display a copy (view) of the image as an icon.

-drag_scroll bool
        If true, set bindings to scroll with the middle mouse button.

-scrollbars bool
        If true, display scrollbars to scroll the image.

-port port
        Default port for remote connections (0 means system chooses a
        port).

-debug bool
        Debugging flag: enables real-time simulation with testProg
        (below).

-testprog path
        For testing: name of test program used to generate real-time
        updates (tRtd).

-interval n
        For testing: interval between updates in ms.

-with_perftest bool
        If true, display performance tester utility in menu bar.

-with_warp bool
        Option to warp the mouse pointer.
```

FILES

```
$RTD_LIBRARY/../demos/rtd.tcl
```

SEE ALSO

```
rtdimage(n), RtdImageCtrl(n), rtdimage_wish(1), rtdServer(1),
RtdServerTool(n), RtdImageZoom(n), RtdImageZoomView(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.1.2 rtdCubeDisplay(1)

NAME

```
rtdCubeDisplay - simple FITS cube display program
```

SYNOPSIS

```
rtdCubeDisplay -f <file name> -c <camera name> [-t <msec>] [-l] [-v]
```

DESCRIPTION

```
rtdCubeDisplay displays FITS cube images for the real-time display.
By specifying a FITS cube file (option -f) and a camera name (option -c)
the images are extracted from the file and an image event is sent to
the rtdServer. In order to display the image a rtd widget application
e.g. 'rtd' must register to the same camera name as specified above.
For the 'rtd' application this is done by  setting the RTD_CAMERA
environment.

Options:

 -f <file name>        FITS cube images
 -c <camera name>      Camera name to identify real time source
 -v                    Enables verbose mode
 -t <msec>             Delay time between images (default 500msec)
 -l                    Loop (forever)
```

SEE ALSO

```
rtdServer(1)
```

```
- - - - - -
Last change: 07 May 99
```

4.1.3 rtdimage_wish(1)

NAME

rtdimage_wish - a Tk wish shell with the rtdimage extension

SYNOPSIS

```
rtdimage_wish ?fileName arg arg ...?
```

DESCRIPTION

rtdimage_wish is the Tk wish shell created by the RTD package to demonstrate the real-time display widget features. This version of wish(1) contains, in addition to the rtdimage extension, the BLT, [incr Tk] and TclX extensions. BLT is required for displaying graphs and tables, [incr Tk] is used for its class system and TclX is used for various utility commands and interprocess communication (rtdServer(1).

ADDING THE RTDIMAGE EXTENSION

The rtdimage extension can be included in a Tk application shell in the standard way, by adding a call to RtdImage_Init(interp) in the Tcl_AppInit() routine, which every Tk application must define:

```
/* initialize the rtdimage type */
if (RtdImage_Init(interp) == TCL_ERROR) {
    return TCL_ERROR;
}
```

Note that since the rtdimage extension is implemented in C++, it is required that main() also be compile with C++. Normally, main() is included in the same C file with Tcl_AppInit(), so you have 2 choices: you can rename the file tkAppInit.c to tkAppInit.C and compile and link it with a C++ compiler, or you can put main() in a separate file and compile and link it with a C++ compiler.

SEE ALSO

rtdimage(n), wish(1), BLT(n), incrTcl(n), TclX(n)

- - - - - -

Last change: 07 May 99

4.1.4 rtdServer(1)

NAME

rtdServer - image event dispatcher for RTD

SYNOPSIS

rtdServer [-v] [-p <port number>]

DESCRIPTION

rtdServer is the process that manages the image event mechanism. Clients register to the rtdServer via the rtdInitImageEvt call. When a client attaches to a camera source an incomming image event will be forwarded to this client. Several clients can attach to the same camera source as the multicasting of event notification is supported by the rtdServer.
Image events received from image sources where no clients are attached are simply discarded. Clients can also attach to image sources that not have registered yet as the rtdServer supports a independence between image event producer and image event consumer.

Furthermore rtdServer contains a simulator part that can be used to simulate the generation of image events. This feature is reserved for testing purposes only. Similarly, it also contains a performance test facility, in which several areas of shared memory are sent to a client Rtd in quick succession, and measurements are taken on certain performance parameters (see RtdPerformanceTool(3/n).

The rtdServer also implements semaphore locking of shared memory, to avoid the possibility of the RTD client reading the shared memory at the same time as the CCD writes (this is known as "image jitter"). The server program expects the CCD software to set a semaphore against any shared memory that has been written to (effectively to lock it). The server will then increment this semaphore by the number of RTD clients less one. If semaphores are not implemented in the incoming image event, no action is taken. The overall locking scheme is discussed in more detail in rtdSem(1). Semaphore locking is implemented in the simulator facility.

ENVIRONMENTS

The rtdServer (and RTD) uses 5555 as the default port number. If another copy of the rtdServer is needed e.g. for debugging purposes or if another port number shall be used then set the environment variable RTD_SERVER_PORT to the port number before starting the rtdServer (and RTD).

SEE ALSO

rtdCubeDisplay(1), rtdInitImageEvt(3), rtdSendImageInfo(3), rtdSem(1)

- - - - - -
Last change: 07 May 99

4.1.5

4.2 C++ CLASSES, C ROUTINES

4.2.1 ColorMapInfo(3)

NAME

ColorMapInfo - A C++ class for reading and managing color map files

SYNOPSIS

```
#include "ColorMapInfo.h"

enum {MAX_COLOR=256};            /* work with 8 bit color */

// one of these is used to hold colormap info for each colormap
// file read
class ColorMapInfo {
...
public:
    ColorMapInfo(char* name, ColorMapInfo* next = NULL);
    ~ColorMapInfo();

    // create and return ColorMap from a file description
    static ColorMapInfo* read(char* filename, ColorMapInfo* next = NULL);

    // member access
    char* name();
    ColorMapInfo* next();

    // set the red, green and blue values from the colormap data
    // and interpolate based on the count of available colors
    void interpolate(XColor* colorCells, int colorCount);

    // rotate the colormap by the given amount
    void rotate(int amount, XColor* src, XColor* dest, int colorCount);

    // shift the colormap by the given amount
    void shift(int amount, XColor* src, XColor* dest, int colorCount);
};
```

DESCRIPTION

This class is used by class ImageColor to read in and manage a single colormap file. The constructor is not normally called from outside the class. To create an object of this class, the static "read" member function reads in a colormap file (256 lines of RGB values between 0.0 and 1.0) and returns a pointer to a new ColorMapInfo instance for the file. Reading in the colormap only stores the values in memory, To apply the colormap file to the default colormap, the interpolate method is called.

Most methods take a colorCount argument, which is the number of colors allocated in the colormap. The rotate and shift methods take an integer "amount" argument, which is typically the difference in mouse movements in some widget and is used to rotate (with wrap around) or shift (without wrap around) the colormap by the given amount.

METHODS

static ColorMapInfo* read(char* filename, ColorMapInfo* next = NULL)
: Create and return a ColorMapInfo from a file description. The next argument is used to build a list of loaded colormaps.

char* name()
: Return the name (file name) of the colormap loaded.

```
ColorMapInfo* next()
        Return a pointer to the next colormap in the list.

void interpolate(XColor* colorCells, int colorCount)
        Set the red, green and blue values in the colormap (in the
        colorCells array) from the loaded colormap data and
        interpolate based on the count of available colors.

void rotate(int amount, XColor* src, XColor* dest, int colorCount)
        Rotate the colormap by the given amount. "src" is the source
        colormap, dest is the destination colormap and colorCount
        gives the number of colors in src and dest.

void shift(int amount, XColor* src, XColor* dest, int colorCount);
        Shift the source colormap by the given amount, putting the
        result in dest. colorCount is the number of colors in src and
        dest.
```

FILES

```
$RTD_LIBRARY/../colormaps/*.lasc - MIDAS colormap files
```

SEE ALSO

```
ImageColor, ITTInfo(3C++)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.2 ImageColor(3)

NAME

ImageColor - A C++ class for managing image colors

SYNOPSIS

```
#include "ImageColor.h"

/*
 * An instance of this class is used to manage colors and colormaps
 * for RtdImage and derived widgets
 */
class ImageColor {
...
public:
    // constructor
    ImageColor(Display*, Visual*, int numColors);

    // member functions
    int numFreeColors();
    int allocate(int numFreeColors);
    int reallocate(int numFreeColors);

    // load (reload) a color map from the given file
    int loadColorMap(char* filename);

    // load (reload) an ITT from the given file
    int loadITT(char* filename);

    // rotate the colormap by the given amount
    int rotateColorMap(int amount);

    // shift the colormap by the given amount
    int shiftColorMap(int amount);

    // scale the current colormap/ITT by the given amount
    int scaleITT(int amount);

    // reset colormap to original state
    int reset();

    // start using a private colormap
    int usePrivateCmap();

    // return true if we are using a private colormap
    int usingPrivateCmap() {return (colormap_ != defaultCmap_);}

    // if we are using a private colormap, set it for the given window
    int setColormap(Tk_Window);

    // member access
    int freeCount() {return freeCount_;}
    int colorCount() {return colorCount_;}
    unsigned long* pixelval() {return pixelval_;}
    unsigned long pixelval(int i) {return pixelval_[i];}
    XColor* colorCells() {return itt_ ? ittCells_ : colorCells_;}
};
```

DESCRIPTION

This class is used to manage the colormap for an image display

```
application. This is normally the default colormap, however, if not
enough colors can be allocated, a private colormap can be created and
an attempt is made to reduce color flashing by copying the colors from
the default colormap to the new one.  Normally, only a single instance
of the ImageColor class is required per application. Class ImageColor
modifies the colormap by allocating a given number of color cells and
assigning color values to cells based on MIDAS colormap and ITT
(intensity transfer table) files. Methods are also available for
rotating, shifting, stretching and squeezing the colormap.
```

MIDAS COLORMAP FILES

```
The colormap files used here were taken in ascii form from the MIDAS
distribution.  A colormap file has 256 lines of red, green, blue
floating point values between 0.0 and 1.0. The values are scaled to the
size of the colormap (the number of allocated colors) at run time.
```

MIDAS ITT FILES

```
An ITT file is like a colormap file, except that there is only one
value per line (256 lines). Each value is between 0.0 and 1.0 and is
used to modify the colormap. For example, to get a negative of an
image, a negative ITT would have values starting at 1.0 and ending at
0.0, equally spaced.
```

CONSTRUCTOR

```
The constructor takes as arguments, a pointer to the X display (for
colormap operations) and the number of colors to allocate. If there
are not that many colors available, then the actual number allocated
will be less.
```

METHODS

```
int numFreeColors()
        Return the number of free color cells available (uses a binary
        search between 0 and MAX_COLOR).

int allocate(int numFreeColors)
        Allocate at most numColors color cells. If there are not that
        many colors available, then the actual number allocated will
        be less.

int reallocate(int numFreeColors)
        Free and then re-allocate at most numColors color cells.

int loadColorMap(char* filename)
        Load a color map from the given file, where file contains 256
        lines of (r g b) values.

int loadITT(char* filename)
        Load an intensity transfer table (ITT) from the given file
        where file contains 256 ITT values, one per line.

int rotateColorMap(int amount)
        Rotate the colormap by the given amount.

int shiftColorMap(int amount)
        Shift the colormap by the given amount.

int scaleITT(int amount)
        Scale (squeeze or stretch) the current colormap/ITT by the
```

```
                given amount.

        int reset()
                Reset colormap to original state.

        int freeCount()
                Return the number of free colors available.

        int colorCount()
                Return the number of colors allocated.

        unsigned long* pixelval()
                Return the array of pixel values for the allocated colors in
                the colormap.

        unsigned long pixelval(int i)
                Return the colormap pixel value at the given index.

        XColor* colorCells()
                Return pointer to array of XColors used for colormap.

        int usePrivateCmap()
                Start using a private colormap.

        int usingPrivateCmap()
                Return true if we are using a private colormap.

        int setColormap(Tk_Window)
                If we are using a private colormap, set it for the given window.
```

SEE ALSO

ColorMapInfo, ITTInfo(3C++)

- - - - - -

Last change: 07 May 99

4.2.3 ImageData(3)

NAME

ImageData - C++ Base Class for Managing Image Data

SYNOPSIS

```
#include "ImageData.h"

class ImageData {
...
public:
    // maximum scale factor
    enum {MAX_IMAGE_SCALE = 50};

    // types of color scaling
    enum ImageColorScaleType {
        LINEAR_SCALE,           // linear scale
        LOG_SCALE,              // logarithmic or exponential scale
        SQRT_SCALE,             // square root scale
        HISTEQ_SCALE            // histogram equalization
    };

    virtual ~ImageData();

    static ImageData* makeImage(const char* name, const ImageIO&, int verbose = 0);

    int write(const char* filename);

    void doTrans(double& x, double& y, int distFlag = 0,
                 double xOffset = 0.0, double yOffset = 0.0,
                 int width = 0, int height = 0);

    void undoTrans(double& x, double& y, int distFlag = 0,
                   double xOffset = 0.0, double yOffset = 0.0,
                   int width = 0, int height = 0);

    void coordsToDist(double& x, double& y, int width = 0, int height = 0);
    void distToCoords(double& x, double& y, int width = 0, int height = 0);
    int getIndex(double x, double y, int& ix, int& iy);
    void setXImageData(byte* xImage, int width, int height);
    void setScale(int xScale, int yScale);
    void shrinkToFit(int width, int height);
    void update();
    void updateOffset(double x, double y);
    virtual void getDist(int& numValues, double* xyvalues);
    int getSpectrum(double* xyvalues, int x0, int y0, int x1, int y1);
    virtual void setCutLevels(double min, double max, int scaled);
    virtual void autoSetCutLevels(double percent = 98.0);
    virtual void medianFilter(double* pattern = NULL);
    virtual void colorScale(int ncolors, unsigned long* colors);
    virtual int dataType() = 0;
    virtual ImageData* copy() = 0;
    void saveParams(ImageDataParams&);
    void restoreParams(ImageDataParams&);
    virtual char* getValue(char* buf, double x, double y) = 0;

    virtual void getValues(double x, double y, double rx, double ry,
                           char* xStr, char* yStr, char* valueStr,
                           char* raStr, char* decStr, char* equinoxStr) = 0;

   virtual void getValues(double x, double y, double* ar, int nrows, int ncols) = 0;
    virtual void getValues(double x, double y, int w, int h, float* ar) = 0;
```

```
    virtual double getValue(double x, double y) = 0;
    virtual int getStatistics(double x, double y, int w, int h,
                              double& meanX, double& meanY,
                              double& fwhmX, double& fwhmY,
                              double& symetryAngle,
                              double& objectPeak, double& meanBackground);

    Mem& data();
    Mem& header();

    int getFitsHeader(ostream& os);

    void data(const Mem& data);
    void header(const Mem& header);

    WCS& wcs();

    byte* xImage();
    const ImageIO& image();
    void colorScaleType(ImageColorScaleType t);
    ImageColorScaleType colorScaleType();
    int ncolors();
    unsigned long* colors();
    unsigned long color0();
    unsigned long colorn();
    void setColors(int ncolors, unsigned long* colors);
    void expo(double e);
    double expo();
    int width();
    int height();
    int dispWidth();
    int dispHeight();
    int xScale();
    int yScale();
    int flipX();
    void flipX(int b);
    int flipY();
    void flipY(int b);
    int rotate();
    void rotate(int);
    double highCut();
    double lowCut();
    double minValue();
    double maxValue();
    void subsample(int b);
    void verbose(int b);
    void name(const char* name);
    char* name();
    void object(const char *object);
    char* object();
    int update_pending();
    void update_pending(int b);
    LookupTable lookup();
    int lookup(LookupTable);
    void clear();
};
```

DESCRIPTION

This class is part of the RTI or Real-Time Image library and is independent of X and Tcl/Tk, so it could, in principle, be used by a

```
separate process from the one displaying the image.  Class ImageData
is used for managing the image data and the conversion of the raw
image data to X image data with transformations (scaling, flipping and
rotating). Note that the term "X image", in this case refers to a
pointer to an array of bytes and not the XImage struct itself, which
is used to actually display the image later. The base class ImageData
is the only class visible to the outside, however, there is one
subclass for each underlying raw image data type. These subclasses
implement the type specific operations on the raw image. Here is the
class hierarchy:

ImageData
        Base class, only class visible outside.

   ShortImageData
   UShortImageData
   ByteImageData
   LongImageData
   FloatImageData
        Derived classed for images with the raw data types: short,
        ushort, byte, long and float

   XImageData
        This class is like ByteImageData, except that the raw image is
        already in XImage format (i.e.: different Y axis direction and
        no need for color scaling or cut levels).
```

CREATING AN IMAGE OBJECT

```
The ImageData class is designed to work, in principle, independently
of the actual image format, although in the end, something resembling
FITS format is expected. An ImageData object can be created with the
"makeImage()" method.

ImageData::makeImage() takes an arbitrary image name, an ImageIO
object, and a verbose flag and returns a pointer to an object that is
a subclass of ImageData specialized in the data type for that image
(ShortImageData, FloatImageData, etc.).

The caller creates the ImageIO object and can control how the image is
constructed. ImageIO is a reference counted class that can be
initialized with a pointer to a subclass of ImageIORep. The astrotcl
package currently defines only one subclass called FitsIO, however
other packages may add other subclasses to implement support for new
image types. You can create an ImageIO object by passing the
constructor a pointer to a FitsIO object, and then use this for the
ImageData::makeImage() call:

    ImageIO imio(new FitsIO(width, height, type, 0.0, 1.0, header, data));
    image = ImageData::makeImage("myimage", imio, 0);

Or you can do it all in one step and let the compiler do the conversion:

    image = ImageData::makeImage("myimage", new FitsIO(...), 0);

Note that the internal image format is always FITS. Other formats may
be converted to FITS internally (see ImageIO).
```

WORKING WITH IMAGES

```
The main task of the the ImageData class is to transform raw image
data of some type to data that can be displayed in an application
window. This includes transformations, such as scaling, rotating and
```

```
flipping and color scaling, or the mapping of pixel values to a
limited number of colormap values.  Since applications don't always
display the entire image at once, methods are available to transform
only a given section of the image. Whenever a part of the raw image is
to be copied to the X image, the current transformations and color
scaling algorithm are taken into account. The transformations are
controlled by flags. These can be accessed through the inline member
functions rotate(), flipX(), flipY(), xScale() and yScale(). The color
scaling algorithms (linear, logarithmic, square root and histogram
equalization) are used to create a lookup table that is used when
transforming the image.  Basically, the lookup tables (taken from
saoimage) convert shorts to bytes. Each subclass defines the method to
convert the raw image pixel values to shorts, which are then converted
to bytes via the lookup table, according to the current color scaling
algorithm.
```

COORDINATES

```
Arguments representing coordinates and dimensions are generally
expected to be in image coordinates. Since the caller usually has
screen coordinates, these must be converted first to image coordinates
by reversing the transformations using the method "undoTrans".
```

METHODS

```
makeImage(name, imageIo, verbose)
        Return a pointer to a derived class of ImageData specialized
        in the given type of image. See also above and class ImageIO.

write(filename)
        Save the image to a file. For FITS images, if a header was
        present, it is reused, otherwise FITS keywords are inserted
        indicating the image type, width and height along with the
        date and a number of numbered "blank cards" or FITS keyword
        fields that can be modified by other applications as needed.
        The fields have names starting with BLANK followed by 2 digits
        (from BLANK00 to BLANK28). See FitsIO for more information.

doTrans(x, y, dist_flag)
undoTrans(x, y, dist_flag)
        apply (doTrans) or reverse (undoTrans) the transformations on
        given coordinates (scale, rotate, flipX and flipY). If
        dist_flag is non-zero, x and y are treated as a distance, so
        that "flipX" and "flipY" are not done. Note that both methods
        also reverse the Y axis (when dist_flag is 0), since image
        coordinates have the origin at lower left rather than upper
        left as for canvas coordinates. These methods essentially convert
        between canvas (displayed) coordinates and image coordinates.

setXImageData(xImage, width, height)
        Set the destination XImage buffer and dimensions.  This class
        copies the rawimage to xImage, doing any necessary
        transformations along the way.

setScale(xScale, yScale)
        Set the scaling factor.  The scaling factors are positive or
        negative integers (default 1). Positive integers are used to
        zoom in on the image (2 means twice the original
        size). Negative integers are used to zoom out (-2 means 1/2
        the original size). The software imposes an arbitrary limit on
        the scaling factor of +-MAX_IMAGE_SCALE.

shrinkToFit(width, height)
```

```
        Set the scaling factor so that the image will fit within the
        given dimensions.

update()
        Update the entire image from the raw image if necessary.  If
        nothing has changed since the image was last updated, this
        call does nothing, otherwise the entire raw image is copied to
        the X image with transformations.

updateOffset(x, y)
        Update the image area starting at the given offset and
        continuing to the end of the raw image or the end of the X
        image data, which ever comes first. The raw data starting at
        the offset (x,y) is copied to the X image starting at (0,0)
        with transformations. Note that x and y are in image
        coordinates.  This method is used when the X Image is the same
        size as the visible window and displays the part of the image
        at some x,y scroll offset.

getDist(numValues, xyvalues)
        Scan the image and generate X,Y values to show the
        distribution of pixel values in the image. "numValues" is the
        max number of X,Y pairs to put in xyvalues (if there are not
        enough values, numValues is modified to reflect the actual,
        smaller number of values). This method is used to generate a
        BLT(n) graph of the pixel value distribution in the image.

getSpectrum(xyvalues, x0, y0, x1, y1)
        Scan the raw image along the line given by x0,y0 and x1,y1 and
        generate an array of (index, pixel value) information. Return
        the number of (index,value) pairs generated for the line.
        This method is used to generate a BLT(n) graph of the pixel
        values in an arbitrary line in the image. As always, the
        arguments are expected in image coordinates.

setCutLevels(min, max)
        Manually set the cut levels. This affects color scaling (see
        colorScale()).

autoSetCutLevels(percent)
        Scan the image to find the distribution of pixel values (using
        getDist()) and set the cut levels so that the given percent of
        pixels are within the low and high cut values.

medianFilter(pattern)
        Automatically set the cut levels using a median filtering
        algorithm.  The optional "pattern" parameter may be specified
        as a pointer to an array containing a fixed pattern of pixels
        (the same size as the image),
        which is used by the algorithm.

colorScale(ncolors, colors)
        Create the color scale lookup table for the image.  "ncolors"
        is the number of available colors in the colormap.  "colors"
        is an array of pixel values for the available colors.  The
        color scaling algorithm used is set with the (inline) method
        colorScaleType(), and defaults to ImageData::LINEAR_SCALE.

dataType()
        Return the type of the raw image data as an enumeration value
        (see enum ImageDataType above).

copy()
```

```
        This virtual method returns an allocated copy of the image.
        The copy will share the same raw image pointer and color
        lookup table. Normally, a new X image pointer is assigned
        after this call, so that the same raw image can be transformed
        in different ways. This is used by the rtdimage widget to
        implement "views", such as the panning window and zoom window.

copyParamsFrom(image)
        Copy the cut levels and transformation params from the given
        image.

getValue(buf, x, y)
        Print the coordinates and raw data value at the given x,y
        image coords to the given char* buffer.  A "-" is printed if
        the x,y coords are out of range.

getValue(x, y)
        Return the raw image value at the given x,y coordinates as a
        double.  The input x,y is assumed to be in image coordinates.
        If the coordinates are out of range, 0.0 is returned.

getValues(x, y, xStr, yStr, valueStr, raStr, decStr, equinoxStr)
        Print the values at the given x,y coordinates to the given
        buffers for display. X and Y are specified in image
        coordinates and the values written to the buffers are in image
        and world coordinates, resp. If the given point is out of
        range, the buffers are set to the empty string.

getValues(x, y, ar, nrows, ncols)
        Fill the given array with the pixel values surrounding the
        given image coordinate point.  nrows and ncols give the
        dimensions of the array. Any values that are outside of the
        image are set to HUGE_VAL.  Note: it is assumed that nrows and
        ncols are odd numbers and that the array is one row and column
        larger (nrows+1 x ncols+1), so that it can hold the X and Y
        index headings.  The X heading is in the first row of the 2
        dimensional array and contains the X coordinate values. The Y
        coordinate values are in the first column.

header()
data()
        Return a reference to the class "Mem" object used to represent the
        FITS image header (or data).

header(m)
data(m)
        Set a new (FITS) image header or image data object (of class Mem).

xImage()
        Get a pointer to the X image data.

colorScaleType()
colorScaleType(type)
        Get/set the current color scale algorithm (see enum
        ImageColorScaleType above).

ncolors()
colors()
        Get the number of colors in the colormap or the pointer to the
        color values (set with colorScale() or setColors()).

setColors(ncolors, colors)
        Set the number of colors in the colormap or the pointer to the
```

```
        color values.

color0()
colorn()
        These two methods return the color values for blank pixels
        (color0()) and saturated pixels (colorn()). These two color
        cells are normally reserved for this purpose and are not
        otherwise used in the image. color0 is normally black and
        colorn is normally white. See ImageColor.

expo(e)
expo()
        Set/get the exponent used for logarithmic or square
        root color scaling.

width()
height()
        Get the dimensions of the raw image.

dispWidth()
dispHeight()
        Get the dimensions of the image after transformations.

xScale()
yScale()
        Get the X and Y scale (magnification) factors (Set
        with setScale()).

flipX()
flipX(bool)
flipY()
flipY(bool)
rotate()
rotate(bool)
        Turn flipping in the X or Y direction or rotating on
        or off for the image.

highCut()
lowCut()
        Get the high or low cut value (set with setCutLevels()).

minValue()
maxValue()
        Get the minimum or maximum raw image value.

subsample(bool)
        Set the subsample flag, to true to use every nth pixel
        when shrinking an image, or false to use the maximum
        pixel.

verbose(bool)
        Sets a flag: if true, diagnostic messages are printed
        out at run time.

name(name)
name()
        Set/get the name of the image. This is some arbitrary string
        used to identify the image. The set routine makes a local copy
        of the string,

wcs()
        This method returns a reference to the WCS class object for this
        image. This object provides a number of methods for working with
```

```
               world coordinates and converting between pixel and world coordinates.

       clear()
               Temporarilly clear the X image data to make the image blank
               (until the next call to "update" or "updateOffset").
```

SEE ALSO

```
       rtdimage(n), RtdImage, ImageColor(3C++), ImageDisplay(3C++),
       WCS, BLT(n)


- - - - - -
Last change: 07 May 99
```

4.2.4 ImageDisplay(3)

NAME

ImageDisplay - A C++ class for managing the display of an XImage

SYNOPSIS

```
#include "ImageDisplay.h"

class ImageDisplay {
...
public:

    // constructor
    ImageDisplay(Display *display, Visual *visual, GC gc,
                 int depth, int useXShm, int verbose);

    // destructor
    ~ImageDisplay();

    // create or update an XImage with the given size
    int update(int width, int height);

    // copy the XImage to a Drawable in the X Server
    void put(Drawable, int src_x, int src_y, int dest_x, int dest_y, int width, int
height);

    // return a pointer to the XImage data
    unsigned char* data();

    // inline query methods
    int width();
    int height();
    int bitmapPad();
    int bytesPerLine();

    // return true if we are really using X shared memory
    int usingXShm();
};
```

DESCRIPTION

This class manages the creation, display and disposal of an XImage, optionally using X shared memory, if available.

CONSTRUCTOR

The constructor takes as arguments: the X display, visual, GC and image depth, all for later reference in X calls. In addition, 2 flags may be specified: "useXshm" is set to true if X shared memory should be tried for and "verbose" is set to true if diagnostic messages should be printed out at run time.

METHODS

There are two main methods. One to create or update an XImage and one to copy it to the X server:

int update(int width, int height)
: Create or update the XImage so that it has the given width and height, using X shared memory, if applicable.

```
       void put(Drawable d, int src_x, int src_y,
                   int dest_x, int dest_y,
                   int width, int height)
               Copy the contents of the XImage to the given drawable with the
               given arguments, using X shared memory, if applicable.

       In addition, there are inline methods defined to query the XImage
       width and height, bytes per line and padding.  Note: always use
       "bytesPerLine()" rather than width() in calculations, since padding in
       X shared memory can make the two different.
```

SEE ALSO

```
       RtdImage


- - - - - -
Last change: 07 May 99
```

4.2.5 ImageZoom(3)

NAME

ImageZoom - C++ class for RtdImage Zoom Window

SYNOPSIS

```
#include "ImageZoom.h"

/*
 * This class implements the Zoom window for the RtdImage class
 */
class ImageZoom {
protected:
    Tk_Window  tkwin_;          // zoom window
    GC gc_;                     // graphics context for copying pixels
   GC rect_gc_;             // graphics context for drawing box aroung center pixels
    int width_;                 // width of displayed image
    int height_;                // height of displayed image
    int zoomFactor_;            // zoom factor (1...n)
    int zoomStep_;              // value used to calculate zoom = width/factor
    ImageDisplay *xImage_;      // class object for zoom window's X image
    int status_;                // return value from constructor

public:
    // constructor: initialize the zoom window
    ImageZoom(Tk_Window tkwin, GC copyGC, int width, int height, int factor,
                           int usingXShm, int verbose);

    // destructor: clean up resources
    ~ImageZoom();

    // called for motion events in the image to do the zooming
    void zoom(unsigned char* data, int x, int y, int w, int h, int xs, int ys);

    // return status after constructor for error checking
    int status() {return status_;}
};
```

DESCRIPTION

This class is used to implement one version of the RtdImage zoom window, a small window displaying a magnified area of the main image while tracking mouse pointer motion events. See RtdImageZoomView(n) for the other, which is implemented as "view" of an RtdImage widget.

This simple class gets the necessary X window information from the constructor arguments. The "zoom()" method is then called for mouse pointer motion events with a pointer to the XImage data for the main image, the mouse coordinates, the width of the zoom image and the zoom factor. The zoom is done at the given factor directly from the given X Image data and a rectangle is drawn in the middle to indicate the size of a pixel in the main image.

METHODS

void zoom(unsigned char* data, int x, int y, int w, int h, int xs, int ys)

Called for motion events in image window when zooming is on.

Args:

data - pointer to data being displayed

x, y - coords in displayed image (XImage coordinates)

```
                w, h - width (bytesPerLine) and height of displayed image
                xs, ys - x and y magnification factors

       int status()
               Return status after constructor for error checking.
```

SEE ALSO

```
       RtdImage, RtdImageCtrl(n), RtdImageZoom(n), RtdImageZoomView(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.6 ITTInfo(3)

NAME

```
ITTInfo - C++ class for reading and managing MIDAS ITT (intensity
            transfer table) files
```

SYNOPSIS

```
#include "ITTInfo.h"

// one of these is used to hold ITT info for each ITT
// file read
class ITTInfo {
...
public:
    ITTInfo(char* name, ITTInfo* next = (ITTInfo*)NULL);
    ~ITTInfo();

    // create and return ITT from a file description
    static ITTInfo* read(char* filename, ITTInfo* next = (ITTInfo*)NULL);

    // member access
    char* name() {return name_;}
    ITTInfo* next() {return next_;}

    // Copy the rgb color values from colorCells to copyCells and interpolate based
    // on the ITT table and the count of available colors
    void interpolate(XColor* src, XColor* dest, int colorCount);

    // Copy the rgb color values from colorCells to copyCells as above,
    // and also scale the ITT values by the given amount
    void scale(int amount, XColor* src, XColor* dest, int colorCount);
};
```

DESCRIPTION

```
This class is used to read in and manage an ITT (intensity transfer
table) file in MIDAS format, which is 256 lines of values between 0.0
and 1.0, which are applied to the colormap color values to modify the
colormap.

Most methods take a colorCount argument, which is the number of colors
allocated in the colormap. The scale method take an integer "amount"
argument, which is typically the difference in mouse movements in some
widget and is used to stretch or squeeze the ITT (and colormap) by the
given amount.
```

METHODS

```
static ITTInfo* read(char* filename, ITTInfo* next = NULL)
        Create and return an ITTInfo from a file description.  The
        next argument is used to build a list of loaded ITTs.

char* name()
        Return the name (file name) of the ITT loaded.

ITTInfo* next()
        Return a pointer to the next ITT in the list.

void interpolate(XColor* src, XColor* dest, int colorCount)
        Copy the rgb color values from src to dest and interpolate
        based on the ITT table and the count of available colors.
```

```
       void scale(int amount, XColor* src, XColor* dest, int colorCount);
              Copy the rgb color values from src to dest as above, and also
              scale the ITT values by the given amount. Values greater than
              1.0 "stretch" the ITT/colormap, values between 0.0 and 1.0
              "sqeeze" it.
```

FILES

```
       $RTD_LIBRARY/../colormaps/*.iasc - MIDAS ITT files
```

SEE ALSO

```
       ImageColor, ColorMapInfo(3C++)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.7 RtdCamera(3)

NAME

RtdCamera - A C++ class for managing realtime image updates

SYNOPSIS

```
#include "RtdCamera.h"

class RtdCamera {
protected:
    char* name_;                // some unique name (name of Tk image...)
    int verbose_;               // flag: if true, print diagnostic messages
    Tcl_Interp* interp_;        // Tcl interp (for file events, error handling)
    rtdIMAGE_EVT_HNDL* eventHndl_; // image event handle
    int evtError_;              // error count for image events
    char* eventScript_;         // tcl script to evaluate for each event
    void* shmPtr_;              // pointer to shared memory area for image
    int shmId_;                 // shared memory ID for image event
    char* camera_;              // camera name
    int attached_;              // flag: true if we are attached to the image event
                                // server
    int width_, height_;        // image dimensions
    int type_;                  // image type

    // member called by fileEventProc for realtime image events
    int fileEvent();

    // called to display new image from shared memory
    // (defined in a derived class)
   virtual int display(int frameId, int type, int width, int height, void* data) = 0;

    // start accepting events from the camera
    int attach(const char* camera);

    // stop accepting events from the camera
    int detach();

public:

    // constructor
    RtdCamera(const char* name, Tcl_Interp*, int verbose);

    // destructor
    ~RtdCamera();

    // static file handler, called by Tk file handler for realtime image events
    static void fileEventProc(ClientData, int mask);

    // start/stop/pause or continue accepting images
    int start(const char* camera);
    int stop();
    int pause();
    int cont();

    // make a allocated copy of the shared memory image data
    void* copyImage();

    // return the current status of the camera
    int paused();
    int attached();
    int stopped();
    void* shmPtr();
```

```
};
```

DESCRIPTION

RtdCamera is the abstract base class for managing real-time images coming from a CCD camera. It is designed as a base class, so that it doesn't have to know anything about how to actually display an image. Class RtdImage derives a simple class from this base class and redefines the "display" method to display the incoming image.

An instance of this class (actually a derived class) is created for the rtdimage(n) "camera start" command. It opens a connection to the RtdServer(1) daemon process and sets up a Tk file event handler to listen for image events. When an image event is received, this class decodes it and calls the virtual "display" method to display the image.

METHODS

int fileEvent()
: This method is called when there is a message to read from the realtime event server. Read the message and call a virtual method to display the image and evaluate the tcl event script, if there is one (not currently used, see rtdimage(n) camera command).

void* copyImage()
: return a malloc'ed copy of the shared memory image data.

void fileEventProc(ClientData clientData, int mask)
: This static method is called when there is a message to read from the realtime event server: pass it on to a member function.

int attach(const char* camera)
: Start accepting events from the camera.

int detach()
: Stop accepting events from the camera.

int start(const char* camera)
: Start accepting images from the named camera. The "name" argument is some string that identifies the caller, such as the image name. "camera" is a string that identifies the camera.

int stop()
: Stop accepting images from the camera.

int pause()
: This is like stop, but keeps the camera around so that you can use "cont" to continue.

int cont()
: Continue the camera after a pause.

SEE ALSO

RtdImage, rtdImageEvt, RtdServer(1), rtdimage(n)

```
- - - - - -
Last change: 07 May 99
```

4.2.8 RtdImage(3)

NAME

RtdImage - The C++ class implementing the rtdimage Tk image type

PARENT CLASS

TkImage

SYNOPSIS

```
#include "RtdImage.h"

class RtdImageOptions : public TkImageOptions {...};

#define RTD_OPTIONS ...

class RtdImage : public TkImage {
...
public:
    RtdImage(Tcl_Interp*, const char* instname, int argc, char** argv,
             Tk_ImageMaster master, const char* imageType,
             Tk_ConfigSpec* specs = (Tk_ConfigSpec*)NULL,
             RtdImageOptions* options = (RtdImageOptions*)NULL);

    ~RtdImage();

    virtual int call(const char* name, int len, int argc, char* argv[]);

    static int CreateImage(Tcl_Interp*, char *name, int argc, char **argv,
                    Tk_ImageType*, Tk_ImageMaster, ClientData*);

    static void eventProc(ClientData clientData, XEvent *eventPtr);

    static void motionProc(ClientData clientData);

    int displayImageEvent(int frameId, int type, int width, int height,
                          int xoffset, int yoffset, const Mem& data);

    static int rtd_set_cmap(ClientData, Tcl_Interp* interp, int argc, char** argv);

    int alloccolorsCmd(int argc, char* argv[]);
    int autocutCmd(int argc, char* argv[]);
    int bitpixCmd(int argc, char* argv[]);
    int cameraCmd(int argc, char* argv[]);
    int clearCmd(int argc, char* argv[]);
    int cmapCmd(int argc, char* argv[]);
    int colorrampCmd(int argc, char* argv[]);
    int colorscaleCmd(int argc, char* argv[]);
    int convertCmd(int argc, char* argv[]);
    int cutCmd(int argc, char* argv[]);
    int dispheightCmd(int argc, char* argv[]);
    int dispwidthCmd(int argc, char* argv[]);
    int dumpCmd(int argc, char* argv[]);
    int fitsCmd(int argc, char* argv[]);
    int flipCmd(int argc, char* argv[]);
    int frameidCmd(int argc, char* argv[]);
    int freqCmd(int argc, char *argv[]);
    int getCmd(int argc, char* argv[]);
    int graphdistCmd(int argc, char* argv[]);
    int heightCmd(int argc, char* argv[]);
    int isclearCmd(int argc, char* argv[]);
```

```
int ittCmd(int argc, char* argv[]);
int maxCmd(int argc, char* argv[]);
int maxFreqCmd(int argc, char* argv[]);
int mbandCmd(int argc, char* argv[]);
int minCmd(int argc, char* argv[]);
int mmapCmd(int argc, char* argv[]);
int motioneventCmd(int argc, char* argv[]);
int objectCmd(int argc, char* argv[]);
int panCmd(int argc, char* argv[]);
int perfTestCmd(int argc, char *argv[]);
int pixtabCmd(int argc, char* argv[]);
int previewCmd(int argc, char* argv[]);
int radecboxCmd(int argc, char* argv[]);
int remoteCmd(int argc, char* argv[]);
int remoteTclCmd(int argc, char* argv[]);
int rotateCmd(int argc, char* argv[]);
int scaleCmd(int argc, char* argv[]);
int shmCmd(int argc, char* argv[]);
int spectrumCmd(int argc, char* argv[]);
int statisticsCmd(int argc, char* argv[]);
int typeCmd(int argc, char* argv[]);
int updateCmd(int argc, char* argv[]);
int viewCmd(int argc, char* argv[]);
int warpCmd(int argc, char* argv[]);
int wcssetCmd(int argc, char* argv[]);
int wcsshiftCmd(int argc, char* argv[]);
int wcscenterCmd(int argc, char* argv[]);
int wcsdistCmd(int argc, char* argv[]);
int wcsequinoxCmd(int argc, char* argv[]);
int wcsheightCmd(int argc, char* argv[]);
int wcsradiusCmd(int argc, char* argv[]);
int wcswidthCmd(int argc, char* argv[]);
int widthCmd(int argc, char* argv[]);
int zoomCmd(int argc, char* argv[]);
int zoomviewCmd(int argc, char* argv[]);

CoordinateType getCoordinateType(const char* s);

int convertCoordsStr(int dist_flag,
                     char* inx_buf, char* iny_buf,
                     char* outx_buf, char* outy_buf,
                     double& x, double& y,
                     char* in_type, char* out_type);

int convertCoords(int dist_flag, double& x, double& y,
                  char in_type, char out_type);

int canvasToScreenCoords(double& x, double& y, int dist_flag);
int canvasToImageCoords(double& x, double& y, int dist_flag);
int canvasToWorldCoords(double& x, double& y, int dist_flag);
int screenToCanvasCoords(double& x, double& y, int dist_flag);
int screenToImageCoords(double& x, double& y, int dist_flag);
int screenToWorldCoords(double& x, double& y, int dist_flag);
int imageToCanvasCoords(double& x, double& y, int dist_flag);
int imageToScreenCoords(double& x, double& y, int dist_flag);
int imageToWorldCoords(double& x, double& y, int dist_flag);
int worldToCanvasCoords(double& x, double& y, int dist_flag);
int worldToImageCoords(double& x, double& y, int dist_flag);
int worldToScreenCoords(double& x, double& y, int dist_flag);
int imageToChipCoords(double& x, double& y, int dist_flag);
int canvasToChipCoords(double& x, double& y, int dist_flag);
int screenToChipCoords(double& x, double& y, int dist_flag);
int worldToChipCoords(double& x, double& y, int dist_flag);
```

```
    int chipToImageCoords(double& x, double& y, int dist_flag);
    int chipToCanvasCoords(double& x, double& y, int dist_flag);
    int chipToScreenCoords(double& x, double& y, int dist_flag);
    int chipToWorldCoords(double& x, double& y, int dist_flag);

    static ImageColor* colors();
    int displaymode() const;
    int fitWidth() const;
    int fitHeight() const;
    int subsample() const;
    char* file() const;
    char* newImageCmd() const;
    char* name() const;
    int usexshm() const;
    int usexsync() const;
    int shm_header() const;
    int shm_data() const;
    int min_colors() const;
    int max_colors() const;
    int verbose() const;

    int dispWidth();
    int dispHeight();

    int imageType();

    int isWcs();

    char* cameraPreCmd();
    char* cameraPostCmd();
    ImageData* image();
};
```

DESCRIPTION

The RtdImage C++ class implements the rtdimage Tk image type. It is a subclass of class TkImage, which implements the more general, Tk image related interface, while RtdImage implements the more specific real-time display features (see TkImage).

Class RtdImage is not normally accessed from other classes (other than derived classes) directly, only via the Tcl command interface (see RtdImage(n)) and the remote control interface (see rtdRemote), so this description is aimed at those who want to understand the class inorder to modify it or subclass from it.

The main interface is to the Tcl interpreter and the Tk image code. The main entry point is through the initialization routine:

```
extern "C" int Rtd_Init(Tcl_Interp* interp)
```

This routine is declared extern "C", since the C routine tclAppInit() needs to call it (note: tclAppInit() is usually found in the file tkAppInit.c and is required in any Tcl/Tk application for adding extensions). To add the rtdimage extension to an application, the following lines are added to its tclAppInit() routine:

```
if (Rtd_Init(interp) == TCL_ERROR)
    return TCL_ERROR;
```

Note that, since the rtdimage extension is implemented in C++, main() also needs to be compiled and linked with a C++ compiler. Since main()

is also included in the tkAppInit.c file in the standard Tk distribution, you either have to compile tkAppInit.c with a C++ compiler (rename it to tkAppInit.C first) or you have to extract the main() routine and put it in a separate file (say, main.C) and compile it with a C++ compiler (In this case, main() is a very simple two-liner that simply calls Tk_Main() and returns).

Rtd_Init() installs the new image type "rtdimage" so that static RtdImage member functions are called (from a table of function pointers) whenever an rtdimage is created, displayed or deleted. The static member functions are passed a pointer to client data, which is actually a pointer to the RtdImage class instance (set in the image create procedure). This pointer is used to access the class member functions to implement the image display and subcommands.

IMAGE CREATION

When an image of type rtdimage is created (by the Tk "image create" command), the static method "CreateImage" is called. It creates an instance of the RtdImage class and sets the client data pointer to the class instance for later reference (in the display and delete static methods).

IMAGE DISPLAY HANDLING

Once an image has been created, a static method in the parent class (TkImage::DisplayImage) is called whenever the image needs to be displayed or redisplayed. It, in turn calls the non-static RtdImage method displayImage() with the coordinates of the area of the image that needs to be redrawn and the X "drawable" to draw to. The display method updates the XImage for the given area and displays it in the given X drawable (see class ImageDisplay).

X SHARED MEMORY

RtdImage attempts to use X shared memory, if the "-usexshm" option is on (default). This improves performance, but is only available when working on the workstation console. This is taken into consideration in the display routine, when deciding whether or not to use X pixmaps to cache image data in the X server.

DISPLAY MODES

Two different display modes are supported:

In displaymode 0, the image is always copied completely to the X server as needed. This makes scrolling smoother, since fewer trips to the X server are needed, but is not practical for large images or greatly magnified images due to memory and bandwidth constraints. This mode is however used for example, by the pan window, since it must always display the entire image (at a small size).

In displaymode 1 (default), only the part of the image that is visible in the image window is considered. This generally means more frequent trips to the X server, but with less data, so in the end the performance remains acceptable for any size image.

The "-displaymode" option in the Tcl command controls the setting of the display mode.

UPDATING THE DISPLAY

Image redisplay can be forced at any time by calling the parent class method imageChanged(). This is done, for example, when an rtdimage subcommand modifies the image in some way, for example by rotating or scaling it. The imageChanged() method tells Tk the "logical" size of the image, which is independent of the window size or the size of the XImage and/or Pixmap being used.

SCROLLING AND CANVAS WINDOW

An image of type rtdimage can only be displayed in a canvas window. The RtdImage class keeps track of the canvas window's scrolling offsets and uses them to help determine which part of the image to display. The image handling code (TkImage::GetImage) gets called for each widget in which the image is displayed, and this is where the check is made and the canvas window handle is saved for later reference.

VIEWS

Normally, a Tk image can be displayed in multiple widgets and changes in size, etc. are propagated. For the rtdimage, a different scheme was needed for sharing images, since changes in size should not be propagated to all instances of the image. For example, a panning window should display the same image, but at a smaller size and a zoom window should display the same image at a larger size, etc. The concept of a "view" of an rtdimage was implemented. This is a simple array of pointers to RtdImage objects that is updated whenever the main image is updated.

ADDING NEW OPTIONS

The return value from the "image create rtdimage" command in Tk is the name of the image and also the name of a new Tcl command. The options to the create command are the same as the options to the image "configure" subcommand. These options are kept in a table and can be fairly easily extended by adding entries to the table and to a simple class. Derived classes can also use the RTD_OPTIONS macro to avoid duplicating code.

```
#define RTD_OPTION(x) Tk_Offset(RtdImageOptions, x)
#define RTD_OPTIONS \
   {TK_CONFIG_BOOLEAN, "-usexshm",    NULL, NULL, "1", RTD_OPTION(usexshm),    0}, \
   {TK_CONFIG_BOOLEAN, "-usexsync",    NULL, NULL, "1", RTD_OPTION(usexsync),    0},
\
   {TK_CONFIG_BOOLEAN, "-verbose",    NULL, NULL, "0", RTD_OPTION(verbose),    0}, \
   {TK_CONFIG_BOOLEAN, "-shm_header",  NULL, NULL, "0", RTD_OPTION(shm_header),  0},
\
   {TK_CONFIG_BOOLEAN, "-shm_data",    NULL, NULL, "0", RTD_OPTION(shm_data),    0},
\
    {TK_CONFIG_INT,      "-displaymode", NULL, NULL, "1", RTD_OPTION(displaymode),
0}, \
   {TK_CONFIG_INT,    "-min_colors",  NULL, NULL, "1", RTD_OPTION(min_colors),  0},
\
   {TK_CONFIG_INT,    "-max_colors",  NULL, NULL, "1", RTD_OPTION(max_colors),  0},
\
   {TK_CONFIG_INT,    "-fitwidth",   NULL, NULL, "0", RTD_OPTION(fitWidth),   0}, \
   {TK_CONFIG_INT,    "-fitheight",  NULL, NULL, "0", RTD_OPTION(fitHeight),  0}, \
   {TK_CONFIG_BOOLEAN, "-subsample",   NULL, NULL, "1", RTD_OPTION(subsample),   0},
\
   {TK_CONFIG_STRING,  "-file",       NULL, NULL, "",  RTD_OPTION(file),       0}, \
    {TK_CONFIG_STRING,   "-newimagecmd", NULL, NULL, "",   RTD_OPTION(newImageCmd),
0}, \
    {TK_CONFIG_STRING,  "-name",        NULL, NULL, "",  RTD_OPTION(name), 0}
```

```
For each rtdimage configuration option, you need an entry in the above
table (from RtdImage.C) and a member in the RtdImageOptions class in
RtdImage.h:

class RtdImageOptions : public TkImageOptions {
public:
    int displaymode;            // set mode used to display image:
                               // 0 ==> XImage is size of image, update whole image
to pixmap
                                // 1 ==> XImage is size of window (default mode)

    int fitWidth;               // fit the image in a window with this width
    int fitHeight;              // and this height by shrinking the image

     int subsample;               // if true, don't count neighboring pixels when
shrinking image
    int usexshm;                // if true, use X shared memory if available.
    int usexsync;               // if true, use X synchronisation if available.
    int verbose;                // if true, print program info to stdout

    int shm_header;            // if true, keep image FITS headers in shared memory
    int shm_data;               // if true, keep image FITS data in shared memory
                                // (see RtdRemote remote access interface)

    int min_colors;             // min (max) number of colors to allocate, if this
many are
    int max_colors;             // not available, use private colormap.

    char* file;                 // name of image file, if any

    char* name;                 // name for image (for debugging)

   char* newImageCmd;          // tcl command to evaluate whenever a new (different)
                               // image is loaded (for updates, see camera command)

    int fixUpdateRate;          // flag: user has specified a fixed update rate, as
below.
    double userUpdateTime;      // the minimum time between updates, as specified
                                // by the user.

    // constructor
    RtdImageOptions()
        : displaymode(1),
          fitWidth(0), fitHeight(0),
          subsample(0),
          usexshm(1),
          usexsync(1),
          verbose(0),
          shm_header(0),
          shm_data(0),
          min_colors(30),
          max_colors(60),
          file(NULL),
          name(NULL),
          newImageCmd(NULL),
          fixUpdateRate(0),
          userUpdateTime(0.) {}
};


The values in this class are treated as read-only for the most part by
RtdImage. They are normally only set by Tk_ConfigureWidget when the
```

image is created or configured. RtdImage accesses these options through inline functions defined at the end of RtdImage.h:

```
    // read-only access to configuration options
    static ImageColor* colors() {return colors_;}
    int displaymode() const {return options_->displaymode;}
    int fitWidth() const {return options_->fitWidth;}
    int fitHeight() const {return options_->fitHeight;}
    int subsample() const {return options_->subsample;}
    char* file() const {return options_->file;}
    char* newImageCmd() const {return options_->newImageCmd;}
    char* name() const {return ((options_->name && *options_->name)
                        ? options_->name : instname_);}
    int usexshm() const {return options_->usexshm;}
    int usexsync() const {return options_->usexsync;}
    int shm_header() const {return options_->shm_header;}
    int shm_data() const {return options_->shm_data;}
    int min_colors() const {return options_->min_colors;}
    int max_colors() const {return options_->max_colors;}
    int verbose() const {return options_->verbose;}
```

Where options_ is the name of the RtdImageOptions class object.

ADDING NEW IMAGE SUBCOMMANDS

The rtdimage subcommands are also defined in a table. In the TclCommand class hierarchy, there is a scheme that allows subcommands to be inherited and extended at any level of the inheritance hierarchy. (Options can also be inherited by deriving a subclass of RtdImageOptions and using the #define RTD_OPTIONS in the option array.) The following table is used to declare rtdimage subcommands in RtdImage.C:

```
/*
 * declare a table of image subcommands and the methods that handle them.
 *
 * NOTE: keep this table sorted, so we can use a binary search on it !
 * (select lines in emacs and use M-x sort-lines)
 */
static class RtdImageSubCmds {
public:
    char* name;      // method name
    int (RtdImage::*fptr)(int argc, char* argv[]); // ptr to method
    int min_args;    // minimum number of args
    int max_args;    // maximum number of args
} subcmds_[] = {
    {"alloccolors", &RtdImage::alloccolorsCmd,  0,  1},
    {"autocut",     &RtdImage::autocutCmd,      0,  2},
    {"bitpix",      &RtdImage::bitpixCmd,       0,  0},
    {"camera",      &RtdImage::cameraCmd,       1,  4},
    {"clear",       &RtdImage::clearCmd,        0,  14},
    {"cmap",        &RtdImage::cmapCmd,         1,  2},
    {"colorramp",   &RtdImage::colorrampCmd,    0,  0},
    {"colorscale",  &RtdImage::colorscaleCmd,   0,  1},
    {"convert",     &RtdImage::convertCmd,      7,  7},
    {"cut",         &RtdImage::cutCmd,          0,  2},
    {"dispheight",  &RtdImage::dispheightCmd,   0,  0},
    {"dispwidth",   &RtdImage::dispwidthCmd,    0,  0},
    {"dump",        &RtdImage::dumpCmd,         1,  5},
    {"fits",        &RtdImage::fitsCmd,         1,  2},
    {"flip",        &RtdImage::flipCmd,         1,  2},
    {"frameid",     &RtdImage::frameidCmd,      0,  0},
    {"get",         &RtdImage::getCmd,          3,  5},
```

```
    {"graphdist",   &RtdImage::graphdistCmd,    5,  5},
    {"height",      &RtdImage::heightCmd,       0,  0},
    {"isclear",     &RtdImage::isclearCmd,      0,  0},
    {"itt",         &RtdImage::ittCmd,          1,  2},
    {"max",         &RtdImage::maxCmd,          0,  0},
    {"mband",       &RtdImage::mbandCmd,        6,  6},
    {"min",         &RtdImage::minCmd,          0,  0},
    {"mmap",        &RtdImage::mmapCmd,         0,  7},
    {"motionevent", &RtdImage::motioneventCmd,  0,  1},
    {"object",      &RtdImage::objectCmd,       0,  0},
    {"pan",         &RtdImage::panCmd,          1,  3},
    {"perftest",    &RtdImage::perfTestCmd,     1,  2},
    {"pixtab",      &RtdImage::pixtabCmd,       1,  3},
    {"preview",     &RtdImage::previewCmd,      1,  1},
    {"radecbox",    &RtdImage::radecboxCmd,     3,  3},
    {"remote",      &RtdImage::remoteCmd,       0,  1},
    {"remotetcl",   &RtdImage::remoteTclCmd,    1,  1},
    {"rotate",      &RtdImage::rotateCmd,       0,  1},
    {"scale",       &RtdImage::scaleCmd,        0,  2},
    {"shm",         &RtdImage::shmCmd,          0,  7},
    {"spectrum",    &RtdImage::spectrumCmd,     9,  9},
    {"statistics",  &RtdImage::statisticsCmd,   0,  0},
    {"type",        &RtdImage::typeCmd,         0,  0},
    {"update",      &RtdImage::updateCmd,       0,  1},
    {"userfreq",    &RtdImage::maxFreqCmd,      1,  1},
    {"view",        &RtdImage::viewCmd,         2,  11},
    {"warp",        &RtdImage::warpCmd,         2,  2},
    {"wcscenter",   &RtdImage::wcscenterCmd,    0,  2},
    {"wcsdist",     &RtdImage::wcsdistCmd,      4,  4},
    {"wcsequinox",  &RtdImage::wcsequinoxCmd,   0,  0},
    {"wcsheight",   &RtdImage::wcsheightCmd,    0,  0},
    {"wcsradius",   &RtdImage::wcsradiusCmd,    0,  0},
    {"wcsset",      &RtdImage::wcssetCmd,       0,  11},
    {"wcsshift",    &RtdImage::wcsshiftCmd,     3,  3},
    {"wcswidth",    &RtdImage::wcswidthCmd,     0,  0},
    {"width",       &RtdImage::widthCmd,        0,  0},
    {"zoom",        &RtdImage::zoomCmd,         1,  3},
    {"zoomview",    &RtdImage::zoomviewCmd,     1,  5}
};
```

The above table maps subcommand name to class method that implements the command. The additional fields indicate the minimum and maximum number of arguments the subcommand expects (for the subcommands, argc is the number of arguments and argv[0] the first argument after the subcommand name). The parent class (and any future derived classes) also declare a similar table and also the virtual method "call", that calls a method in a class, given the method name:

```
/*
 * Call the given method in this class with the given arguments
 * If the method is not defined here, pass on the search to the
 * parent class. Since this is a virtual function, the search starts
 * in the most specific class.
 */
int RtdImage::call(const char* name, int len, int argc, char* argv[])
{
 ...
}
```

DERIVING A SUBCLASS OF RTDIMAGE

If you want to derive a class from RtdImage, in order to modify or

extend its behavior in some way, this is the way you would do it:

In tkAppInit.C, replace the call to Rtd_Init with your own copy of that function: say MyRtd_Init(), so that you can install your own derived class. The new function and declarations would look something like this:

```
class MyRtdImage : public RtdImage {
     ...
}

// image structure needed for Tk images
static Tk_ImageType myRtdImageType = {
     "rtdimage",                    /* name */
     MyRtdImage::CreateImage,       /* createProc */
     TkImage::GetImage,             /* getProc */
     TkImage::DisplayImage,         /* displayProc */
     TkImage::FreeImage,            /* freeProc */
     TkImage::DeleteImage,          /* deleteProc */
     (Tk_ImageType *) NULL          /* nextPtr */
};


// called from tkAppInit
extern "C" int Rtd_Init(Tcl_Interp* /* interp */)
{
    Tk_CreateImageType(&myRtdImageType);
    return TCL_OK;
}


/*
 * This static method is called by the Tk image code to create
 * a new (MyRtdImage) image.
 */
int RtdImage::CreateImage(
    Tcl_Interp *interp, // Interpreter for application containing image.
    char *name,     // Name to use for image.
    int argc,       // Number of arguments.
    char **argv,    // Argument strings for options
                    // (not including image name or type)
    Tk_ImageType *typePtr, // Pointer to our type record (not used).
    Tk_ImageMaster master, // Token for image, to be used by us in
                           // later callbacks.
    ClientData *clientDataPtr)// Store manager's token (this ptr)
                              // for image here,
                              // it will be returned in later callbacks.
{
    MyRtdImage* im = new MyRtdImage(interp, name, argc, argv, master);
    *clientDataPtr = (ClientData) im;
    return im->status();
}
```

After this, you can define methods for new image commands (to be called from Tcl level), redefine some existing behavior or add interfaces to C/C++ code or other processes.

To add new Tcl image subcommands, declare a table like the one described above (each class in the hierarchy that defines subcommands declares one of these). For example:

```
/*
 * declare a table of image subcommands and the methods that handle them.
```

```
 */
static class MyRtdImageSubCmds {
public:
     char* name;      // method name
    int (MyRtdImage::*fptr)(int argc, char* argv[]); // ptr to method
    int min_args;    // minimum number of args
    int max_args;    // maximum number of args
} subcmds_[] = {
    {"foo",         &MyRtdImage::fooCmd,          1,  3},
    {"bar",         &MyRtdImage::barCmd,          2,  4},
    ....
};
```

Now we need to declare the virtual member function "call" to call the method to handle the Tcl subcommands (this is the same in every class, but still needs to be redefined - it probably could also be defined as a macro):

```
/*
 * Call the given method in this class with the given arguments
 * If the method is not defined here, pass on the search to the
 * parent class. Since this is a virtual function, the search starts
 * in the most specific class.
 */
int MyRtdImage::call(const char* name, int len, int argc, char* argv[])
{
     ...
}
```

The rtdimage options can also be extended, if needed, for example:

```
class MyRtdImageOptions : public RtdImageOptions {
public:
  char *grid_tag;               // canvas tag for all grid items
  char *component;              // NDF component to display

  MyRtdImageOptions() : grid_tag(NULL), component(NULL)  {}
};

#define MYRTD_OPTION(x) Tk_Offset(MyRtdImageOptions, x)
#define MYRTD_OPTIONS \
    RTD_OPTIONS, \
    {TK_CONFIG_STRING,  "-grid_tag",    NULL, NULL,
       "ast_grid_item", MYRTD_OPTION(grid_tag), 0}, \
    {TK_CONFIG_STRING,  "-component",   NULL, NULL,
       "data", MYRTD_OPTION(component), 0}
```

Each class in the hierarchy should follow these conventions and define the options as above, so that derived classes can access them and add to them.

SEE ALSO

TkImage, RtdImage(n), image(n), canvas(n)

- - - - - -

Last change: 07 May 99

4.2.9 rtdimage(3)

NAME

rtdimage - Real-Time Display Image, a Tk Image Type

SYNOPSIS

```
image create rtdimage ?option value ...?
```

DESCRIPTION

Tk4.0 introduced a new "image" command and a C interface for adding new image types. A Tk image is much like a Tk widget in that it is both an object and a Tcl command. "rtdimage" is an extended Tk image type designed for real-time image display. Images can be loaded from shared memory or FITS format files, over sockets or HTTP.
For real-time usage, a background daemon process rtdServer(1) communicates with the rtdimage software over a socket interface to display and update images rapidly from shared memory. A more general purpose remote control interface is also available (see rtdRemote).

CREATING RTDIMAGES

An "rtdimage" is created with the "image create" Tk command. After this, you can use the image in a Tk canvas by specifying it with the "-image" option. For example:

```
set image [image create rtdimage ...]
$canvas create image 0 0 -image $image ...
```

Most Tk image types may be used in any Tk widget, however, for our purposes, it was necessary to restrict the use to canvas widgets only. This was necessary in order to handle scrolling efficiently.

OPTIONS

The following options may be specified when creating or configuring an rtdimage:

-displaymode mode
: The rtdimage supports two different display modes: 0 and 1. In display mode 0, space is allocated in the X server for the entire image. This makes scrolling faster, but uses enormous amounts of memory when the image is very large or is scaled to a large size. Still, this mode is useful in cases where the entire image is always displayed, such as in a panning window. In displaymode 1 (default), space is only allocated for the visible part of the image. This makes scrolling somewhat slower, but uses much less memory.

-file name
: "name" specifies a FITS format file to load and display.

-fitwidth winwidth
-fitheight winheight
: These two options specify the size of the window into

```
        which the image must fit. The image will be scaled
        (shrunk) equally in the X and Y directions to fit as
        closely as possible inside the window.

-newimagecmd command
        The given tcl command is evaluated every time a new
        image is loaded. This command is not called for
        real-time image updates, unless the image dimensions
        or data type changed. See the "camera" subcommand
        for getting notification of real-time image updates.

-subsample bool
        If bool is true, subsampling is used when shrinking
        the image, i.e.: if the image is shrunk by 1/3, only
        every third pixel is displayed. Otherwise, the maximum
        value is taken from the group of pixels.

-usexshm bool
        If bool is true (default), attempt to use X shared
        memory for the image display, if available. This
        improves performance considerably, but is only
        available when working on the system console.

-verbose bool
        If bool is true, diagnostic messages are printed out
        to show what is going on internally (for debugging
        use).

shm_header bool
shm_data bool
        If bool is true, the image FITS header (or data) is kept in
        shared memory so that it can be accessed from a remote process
        (see rtdRemote).
```

COORDINATES

```
The rtdimage subcommands support the following types of coordinates:

    canvas     - canvas coordinates (canvas scroll area)
    screen     - canvas window coordinates (visible area)
    image      - basic image pixel coords (at mag 1, no transformations)
    wcs        - world coordinates in H:M:S D:M:S
    deg        - world coordinates in degrees

The rtdimage "convert" subcommand can be used to convert between any
two coordinate systems. In addition, most rtdimage subcommands accept
coordinates using the following syntax:

    $x $y  coord_type

For example:

    set val [$image get $x $y canvas]
    set val [$image get $ra $dec "wcs 1950"]
    set val [$image get 42.1 38.3 "deg 2000"]

For world coordinates, the equinox may be optionally specified as part
of the coordinate type. The default is 2000.
```

IMAGE FORMATS

```
An rtdimage can load and display FITS format images or images written
to shared memory via rtdServer(1). The following FITS image data types
```

```
are supported: float, long, short, ushort, byte or XImage.  Except for
XImage, The order of lines is the same as for FITS files, with the
origin at lower left. XImage is a special image type, which is taken
to be already in a format that can be displayed with no color
scaling. Support for other image types is planned, however the
internal image type will remain FITS. New image types can be added by
deriving a new subclass from the ImageIO class.
```

COLOR ALLOCATION

```
All rtdimages in an application share the same default
colormap.  On startup, the rtdimage attempts to allocate as
many color cells as possible, leaving about 10 free for other
applications.  The number of color cells allocated can be
changed with the "alloccolors" subcommand. If another
application (netscape, for example) has already grabbed all of
the colors, a private colormap will be used. An attempt is made
to keep most of the window manager colors intact, to avoid color
flashing, at least in the GUI elements.
```

MOTION EVENTS

```
Since handling pointer motion events in Tcl code is fairly slow, the
rtdimage code does some of the common work internally by setting
values in a global array called "RtdImage". These values can be best
accessed by specifying the "-textvariable" option to a Tk label or
entry widget. The global "RtdImage" array contains the following
values, which are updated on motion events:

        RtdImage(X)           X image coordinate
        RtdImage(Y)           Y image coordinate
        RtdImage(VALUE)       pixel value at X,Y
        RtdImage(RA)          world coordinate RA value
        RtdImage(DEC)         world coordinate DEC value
        RtdImage(EQUINOX)     world coordinate equinox

The world coordinate values are set to empty strings if the image
header does not support world coordinates.

The same motion handler that sets the above variables also contains
support for zoom windows (zoom and zoomview commands) and pixel tables
(pixtab command).
```

IMAGE COMMANDS

```
The return value from the "image create rtdimage" command is
the name of the image and also the name of a new Tcl command
that can be used to operate on the image. The Tcl command
has the following subcommands:

<imageName> alloccolors ?numColors?
        With no arguments, this command returns a Tcl list
        containing the number of allocated and the number of
        free colors.  With one argument, the command attempts
        to reallocate numColors colors.  The number of colors
        actually allocated depends on what other applications
        are running (see COLOR ALLOCATION).

<imageName> autocut ?-percent number?
        This command automatically sets the cut levels (the
        lowest and highest image pixel values considered in
        colormap scaling).  Two different algorithms are
        supported. The default (and fastest version) is median
```

```
        filtering.  If -percent is specified, the argument is
        a number between 0 and 100, such as 90 for 90%, where
        that percent of the image pixels should be within the
        cut values. i.e.: if you look at the graph (see
        graphdist command) of the pixel value distribution,
        you would take the top 90% of the graph and set the
        cut levels to left and right ends of the graph.
        Note: if this command is called, it is assumed that
        cut levels can be set automatically when a new image
        is loaded. See also the "cut" command.

<imageName> camera start cameraName ?tclCommand?
<imageName> camera stop
<imageName> camera pause
<imageName> camera continue
        The "camera start" command sends a message to the
        rtdServer daemon process telling it to start sending
        images from the given camera.  Actually the server
        sends only image events, short messages over a socket
        interface, while the images are written to and read
        from shared memory. Camera is the name of a camera
        that must be known to the rtdServer (see rtdServer(1)
        for more information).  The optional ?tclCommand?
        argument to "start" should be a string containing a
        Tcl command to be evaluated whenever a new image event
        is received and displayed. The "camera stop" command
        tells the rtdServer to stop sending image events. The
        "pause" and "continue" subcommands can be used to
        temporarily stop the image events and restart them,
        without having to know the name of the camera.

<imageName> clear
<imageName> clear ximage
<imageName> clear ?-reuse $reuse
                   -ra $ra -dec $dec -equinox $equinox -radius $radius
                   -width $width -height $height?
        This command is used to blank out the display by generating and
        loading a blank image. With no arguments a small blank image is
        generated with a default header. If "-ximage" is specified, the
        image is only cleared temporarily, until the next image update.

        In the last case, the optional arguments are used to generate
        a dummy image that supports world coordinates, so that you can
        plot objects on a blank background. Any missing values are set
        to a default value.

        Optional arguments:

        reuse   - flag: if true, reuse previous image, if it is the same
        ra, dec - center point for WCS coords (in decimal degrees)
        radius  - used to initialize WCS coords (CDELT1 and 2)
        equinox - equinox for WCS coords
        width   - width of generated image in pixels
        height  - height of generated image in pixels

<imageName> cmap file   <colormapFile>
<imageName> cmap rotate <amount>
<imageName> cmap shift  <amount>
<imageName> cmap pixels
<imageName> cmap reset
        This command performs operations and queries on the colormap.
        If a colormap file is specified, it should contain 256 lines
        of red, green and blue values between 0.0 and 1.0 (MIDAS
```

```
        colormaps are saved in this format).  The values will be
        distributed among the available colors and installed as a new
        colormap. For rotate and shift, the amount can be any
        integer. The colormap will be rotated (or shifted) by that
        amount. "pixels" returns a Tcl list of the colormap pixel
        values (for use by external applications using the RTI
        library, class ImageData). To get the number of colors in the
        colormap, you can use the "alloccolors" subcommand with no
        arguments or "llength" on the result of the pixels subcommand.
        "reset" resets the colormap to its original state.  The RTD
        release includes a large number of MIDAS colormap files in the
        colormap directory.

<imageName> colorramp
        This command generates an rtdimage displaying the
        colors in the colormap as a ramp or colorbar. This
        image will have the same size as the window containing
        it. This command should be called again from Tcl if
        the window is resized.

<imageName> colorscale ?scale_type?
        This command sets or queries the algorithm to be used
        for assigning the limited number of available colors
        to image pixels. If scale_type is specified, it should
        be one of: linear, log, sqrt or histeq, indicating the
        color scaling algorithm: linear scaling, logarithmic,
        square root or histogram equalization, resp.  With no
        arguments, the current color scale type is returned.

<imageName> convert coords inx iny in_coord_type outx outy out_coord_type
<imageName> convert dist inx iny in_coord_type outx outy out_coord_type
        This command is used to convert between different coordinate
        representations. inx and iny and the input coords (or
        distance) in the given input coordinate system. "convert
        coords" treats x,y as a point, while "convert dist" treats it
        as a distance. outx and outy, if not empty, are the names of
        variables that will hold the resulting coordinates.  If outx
        and outy are empty strings, the values are returned as a tcl
        list "x y".

        The available coordinate systems are:

        canvas     - canvas coordinates (canvas scroll area)
        screen     - canvas window coords (visible area)
        image      - basic image pixel coords (at mag 1, no transformations)
        wcs        - world coordinates in H:M:S
        deg        - world coordinates in degrees

        The world coordinate types: "wcs" and "deg" may also include
        the epoch: Example:

        $image convert coords $ra $dec "wcs 1950" x y canvas

        Note: the coordinate types may be abbrieviated, since only the
        first char is actually checked.

<imageName> cut ?low high?
        This command sets or queries the cut levels. If low
        and high are specified, then the cut levels are set so
        that pixels below the low value will all have the
        lowest color while those above high will all have the
        highest color value.  If no arguments are given, the
        current cut values are returned.
```

```
        Note: if the cut levels are set with this command,
        it is assumed that they should not be changed automatically
        when a new image is loaded (see autocut subcommand).

<imageName> dispwidth
<imageName> dispheight
        These commands return the logical width and height of
        the image after transformations (scaling and
        rotating).  This is the size of the displayed image,
        assuming the window is large enough. This command also
        takes the image's "requested width" into account (set by
        by "view update" subcommand).

<imageName> dump <filename>
        This command dumps the current image to the given file in FITS
        format.  If a FITS header is present, it is used, otherwise
        FITS keywords are inserted indicating the image type, width
        and height along with the date and a number of numbered "blank
        cards" or FITS keyword fields that can be modified by other
        applications as needed.  The fields have names starting with
        BLANK followed by 2 digits (from BLANK00 to BLANK28).

<imageName> flip <direction> ?bool?
        With two arguments, flip (or stop flipping) the image
        in the given direction, where direction is one of x,
        y, xy or "none" for flipping in the x, y, or x and y
        directions or neither.  The boolean value turns
        flipping on (1) or off (0) in the given direction(s).
        With one argument, the command returns the current
        value for the given argument.

<imageName> frameid
        This command returns the frame Id of this image. The
        frame Id is a unique number used to identify the image
        to the rtdServer for use with rapid frames.

<imageName> get x y coord_type ?nrows ncols?
        Returns a Tcl list of image values at the given X,Y
        coordinates.  X and Y are interpreted in the given coordinate
        system (see COORDINATES above).  The return value is a tcl
        list where each item consists of a list of {X Y Value}, where
        X and Y are the adjusted coordinates in the raw image and
        Value is the raw data value at that point or "-" if out of
        range.  If nrows and ncols are greater than 1, the command
        returns a Tcl list of nrows x ncols values, each a list of
        rows, centered at the given point.

<imageName> graphdist bltGraph bltElem numValues
        This command displays the distribution of pixel values
        in the image in the given BLT graph widget. The data
        for the given BLT graph element will be set directly
        to the graph without going through tcl (see
        blt_graph(n)). The number of points to plot is given
        by the numValues argument.

<imageName> itt file <ITTFile>
<imageName> itt scale <scaleFactor>
        This command operates on MIDAS style intensity
        transfer tables or ITTs. If an ITT file is specified,
        it should contain 256 intensity values in the range
        0.0 to 1.0, one per line. The colormap will be
        modified by applying the intensities to it.  The
        colormap can also be stretched or squeezed by applying
```

```
        an integer scale factor to the ITT.  The RTD release
        contains a number of ITTs in the colormaps directory.

<imageName> max
        Returns the highest pixel value in the image.

<imageName> mband x0 y0 x1 y1 cord_type show_angle
        Draw a measure band on the canvas to show the distance in
        world coordinates (diagonal, vertical and horizontal).

        This method was originaly implemented in Tcl/[incr Tk], but
        was redone here for better performance.

        x0 and y0 are the starting coordinates of the drag, x1 and y1
        are the coordinates from the motion events and show_angle is a
        flag: if true, show the horizontal and vertical distance,
        otherwise only the diagonal.

        The coordinates are accepted in the given coordinate system
        "coord_type", see COORDINATES above.

<imageName> min
        Returns the lowest pixel value in the image.

<imageName> pan start <tclCommand> <shrinkFactor>
            pan stop
        This command supports a panning image, which is, in
        this case, a second rtdimage image or "view" of the
        main image, scaled to a small size with a rectangle
        indicating the visible portion of the image.  If
        "start" is specified, the given tcl command will be
        evaluated whenever the image size changes, due to
        scaling or loading a new image, or whenever the image
        position has changed due to scrolling.  The tcl
        command will be called with 5 arguments: x1 y1 x2 y2,
        which are the coordinates of the visible part of the
        image, scaled by the given "shrinkFactor", and a flag
        indicating whether the image is new (1) or an update
        of the existing image (0).  This can be used to draw
        the panning rectangle on the panning image.  To stop
        the command from being called, use the "pan stop"
        subcommand.

<imageName> pixtab start <nrows> <ncols>
<imageName> pixtab stop
        This command supports displaying a table of pixel values
        around a point.  All this commmand does is set a flag causing
        Tcl array variables to be updated on motion events, which can
        cause the display to be updated via the "-textvariable" widget
        option on the table items.
        The array name is fixed as: RtdPixTab and the elements are
        indexed as $RtdPixTab(i,j), where the left and top sides of
        the table (array) are the X and Y image coordinates, resp. and
        the rest are image pixel values.

<imageName> preview <bool>
        If bool is true and real-time images are being
        displayed, the viewing mode is set to "preview mode",
        otherwise, it is set back to "real-time mode". In
        preview mode, the camera is stopped (if it was
        running) and a local copy of the shared memory image
        is made, so that it can be freed or modified without
        affecting the image.
```

```
<imageName> radecbox <ra> <dec> <radius>
        ra and dec are the world coords (h:m:s or decimal deg) and
        radius is expected in arcmin.  The return value in Tcl is a
        list of 4 values {ra0 dec0 ra1 dec1} that form a ra,dec box
        with the given center point and radius.

<imageName> remote -server ?$port?
<imageName> remote -client $host $port
        This command is used to enable remote control of the RTD image
        widget. The -server option is used by the application
        containing the rtdimage widget. The -client option is used by
        the remote client via the rtdRemote interface (internally
        only).

        If -server is specified, the widget starts listening for
        commands on the given port. If port is 0 (default), a port
        number will be chosen. A file is created in the user's home
        directory "~/.rtd-remote", which contains the pid, hostname
        and port number of the running Rtd process.

        If -client is specified, the given port will be used to send
        results to clients when they have requested this type of
        "callback" operation.  The client should be already listening
        on the given port. Note that this command is only used by the
        rtdRemote library internally.

<imageName> rotate ?bool?
        Rotate (or stop rotating) the image. Currently,
        rotation is only done by swapping the x and y axis. If
        bool is specified, rotation is turned on(1) or off(0).
        Otherwise, the current setting is returned.

<imageName> scale ?sx sy?
        With 2 arguments, the image is scaled (magnified) by
        the given X and Y amount.  With no arguments, the
        current scaling factors are returned (as a tcl list of
        2 integers). The scaling factors are positive or
        negative integers (default 1). Positive integers are
        used to zoom in on the image (2 means twice the
        original size). Negative integers are used to zoom out
        (-2 means 1/2 the original size). The software imposes
        an arbitrary limit on the minimum and maximum scaling
        factor allowed.

<imageName> shm set $data_size $data_id $data_owner
                    ?$header_size $header_id $header_owner?
<imageName> shm get data
<imageName> shm get header
<imageName> shm create $size
<imageName> shm delete $Id
<imageName> shm update
        This subcommand provides access to the shared memory in which
        the FITS raw image data and header are stored. The raw image
        is normally stored in mmap shared memory, but SysV shared
        memory will be used if either the -shm_data 1 or the
        -shm_header 1 option was specified when creating the image.

        The "set" command allow you to set the shared memory Ids to
        use to access the image data and header. The data and header
        in the area specified should be in FITS format. If the header
        is not specified, the previous header is reused. For both data
        and header, the size of the area (in bytes) and the shared
```

```
        memory Id must be specified. In addition a flag indicating who
        "owns" the shared memory is specified (if true, then the area
        will be deleted when no longer needed).

        The "get" command returns the shared memory Id of the data or
        header as well as the offset in the shared memory area where
        the header or data begins, the length of the header or data
        and the total size of the shared memory. The result of the
        "get header" or "get data" command is a Tcl list of the 4
        numbers {shmId offset length size}, where length is the length
        of the header or data and size is the total size of the shared
        memory area. If the data or header is not currently in shared
        memory, an error is returned. (RTD must be started with option
        -shm_data 1 and/or -shm_header 1 for this command to work).

        The "create" command creates a new shared memory area with the
        given size and returns the Id. The memory should be deleted
        with the "delete" subcommand when no longer needed.

        The "delete" command deletes the shared memory with the given
        Id (which should have been returned from the "create"
        subcommand).

        The "update" command causes the display to be updated to
        reflect any changes in the image memory.

<imageName> spectrum <bltGraph> <bltElem> x0 y0 x1 y1 coord_type
        This command is used to display a graph of a "cut" of the
        image along a given line. x0, y0, x1 and y1 are the end points
        of a line in the image (in the given coordinate system, see
        COORDINATES above).  <bltGraph> is the path name of a BLT
        graph widget to display the plot of the pixel intensities
        along the line.  <bltElem> is the name of the element in the
        graph that should receive the data.  The data is sent directly
        to the graph for display.  The return value in Tcl is the
        number of points to plot.

<imageName> statistics
        statistics subcommand: calculate statistics on the section of
        the image being displayed.  The return value in Tcl is a list
        of the following values:

          {x y ra dec equinox fwhmX fwhmY angle objectPeak meanBackground}

        where:

        x              = adjusted X image coordinate
        y              = adjusted Y image coordinate
        ra             = RA position (calculated from mean X pos within array)
        dec           = DEC position (calculated from mean Y position within array)
        equinox        = equinox of RA and DEC
        fwhmX          = FWHM in X
        fwhmY          = FWHM in Y
        angle          = angle of major axis, degrees, along X = 0
        objectPeak     = peak value of object above background
        meanBackground = mean background level

<imageName> type
        Returns the data type of the raw image as a string:
        one of: float, long, short, ushort, byte or
        XImage. The last type, XImage is a special pseudo
        type, the same as a byte image, except that the Y axis
        is reversed and it is assumed to not need color
```

```
        scaling.

<imageName> update
        This command makes sure that the image is up to date with the
        raw data (which may have changed via shared memory, mmap, etc).

<imageName> view add <path> ?propagateScale?
<imageName> view remove <path>
<imageName> view update <path> x y width height viewx viewy coord_type
<imageName> view enter  <path>
<imageName> view leave  <path>
        The view command is used to specify a viewing image to view
        the same image, possibly at a different size.  The new view
        will share data with the original and be updated when the
        original is updated.  This can be used, for example, to build
        a panning window or a rapid frame.

        <path> must be the name of a second rtdimage image. The two
        images will communicate internally to always display the same
        image, possibly scaled to different sizes. The subcommands are:

        add
                Adds a new view to this image.

        remove
                Removes the view.

        update
                Updates the view from this image with the given image
                x,y offset, width and height and the position of the
                image view origin in the given coordinate type. This
                command can be used to implement a zoom window or
                rapid frame, since it controls which portion of the
                image is displayed.

        enter
                If 2 images are in the same canvas, make <path> the
                current one (receives motion events, ...).

        leave
                Undo the enter command.

        If the optional "add" argument "propagateScale" is true,
        changes in the scale factors in the master image will
        propagate to the view (this is the default behavior).

<imageName> warp <x> <y>
        Warp (move) the mouse pointer by the given x and y amounts
        (pixels).


<imageName> wcscenter ?-format <format>?
        This command returns the world coordinates of the center of
        the image.  The optional format option determines the format
        of the result:

        -format 0 ==> H:M:S [+-]D:M:S (default)
        -format 1 ==> RA DEC (in degrees)

        The return value is a tcl list, formatted according to the
        format option, or an empty string if the coordinates are out
        of range or WCS is not supported.
```

```
<imageName> wcsdist x0 y0 x1 y1
        This command returns the world coordinate distance between 2
        points after transformations.  The arguments are expected in
        canvas coords (canvasx, canvasy, doubles).  The return value
        in Tcl is the WCS distance between the given points after
        transformations.

<imageName> wcsheight
        This command returns the height of the image in arcmin or the
        empty string if WCS is not supported.

<imageName> wcswidth
        This command returns the width of the image in arcmin or the
        empty string if WCS is not supported.

<imageName> wcsradius
        This command returns the radius (distance from center to
        corner) of the image in arcmin or the empty string if WCS is
        not supported.

<imageName> wcsset <ra> <dec> <secpix> <xrefpix> <yrefpix> <nxpix> <nypix>
                   <rotate> <equinox> <epoch> <projection>
<imageName> wcsset
        If arguments are specified, this subcommand sets up the WCS
        structure from the given information about the image:
        Arguments:
          ra      = Center right ascension in H:M:S
          dec     = Center declination in D:M:S
          secpix  = Number of arcseconds per pixel
          xrefpix = Reference pixel X coordinate
          yrefpix = Reference pixel Y coordinate
          nxpix   = Number of pixels along x-axis
          nypix   = Number of pixels along y-axis
          rotate  = Rotation angle (clockwise positive) in degrees
          equinox = Equinox of coordinates, 1950 and 2000 supported
          epoch   = Epoch of coordinates, used for FK4/FK5 conversion no
                    effect if 0
          proj    = Projection

        With no arguments, the command returns a list of the basic WCS
        parameter values: {ra dec secpix nxpix nypix rotate equinox epoch}.

<imageName> wcsshift <ra> <dec> <coorsys>
        This command resets the center of the WCS structure.
        Arguments:
          ra        = New center right ascension in degrees
          dec       = New center declination in degrees
          equinox   = must be 2000 or 1950

<imageName> width
<imageName> height
        These commands return the width and height of the raw
        image in pixels.

<imageName> zoom start <frame> <zoomFactor>
<imageName> zoom stop
        (Note: This command is no longer supported: please use
        zoomview (below) instead.)

        This command is used to implement a zoom window, a window
        displaying a magnified section of the image at the location of
        the mouse pointer.  There are currently two versions of this
        command (see the zoomview subcommand below). In this version,
```

```
            a Tk frame is specified to hold the zoomed image, which is
            copied directly from the XImage whenever the mouse pointer
            moves over the image. This version is faster, but when the
            main image is shrunk, the zoom will not be very accurate.  If
            "start" is specified, zooming begins in the given window, and
            can be stopped with the "zoom stop" subcommand.

      <imageName> zoomview start <view> <zoomFactor>
      <imageName> zoomview stop
            This command can be used as an alternative to the zoom
            command above. It uses a "view" of the main rtdimage, so the
            zoom image is always accurate, even when main image is shrunk.
            The "view" argument to "zoomview start" should be the name of
            a second rtdimage, which is a "view" of the main image, added
            with the rtdimage "view" subcommand.  The zoomFactor is the
            magnification relative to the main image. For example, if the
            zoomFactor is 5 and the main image is scaled to 1/2, the zoom
            window scale factor would be 4.  Once started, the main image
            will automatically track mouse movements and update the zoom
            window's x and y offsets as needed to display the relevant
            magnified section of the image.
```

ENVIRONMENT VARIABLES

```
      RTD_LIBRARY - If set, this should point to the directory
                    containing the rtdimage Tcl library files.
```

FILES

```
      $RTD_LIBRARY/                  - Tcl/Itcl library files
      $RTD_LIBRARY/colormaps         - MIDAS colormap/ITT files
      $RTD_LIBRARY/images            - sample FITS images
      $RTD_LIBRARY/bitmaps           - X bitmaps used at runtime
      $RTD_LIBRARY/demos             - rtdimage demo application
```

SEE ALSO

```
      RtdImage(n), rtdServer(1), rtdImageEvt, RTI(3), BLT(n),
      canvas(n)



- - - - - -
Last change: 07 May 99
```

4.2.10 rtdImageEvent(3)

NAME

```
rtdImageEvent     - Real-Time image event client interface.

rtdInitImageEvt   - initialize and register to rtdServer.

rtdSendImageInfo  - send image event information to rtdServer.

rtdAttachImageEvt - attach to image event notification.

rtdDetachImageEvt - detach notification of image events.

rtdRecvImageInfo  - receive image event information from rtdServer.

rtdClose          - close event handel.
```

SYNOPSIS

```
#include "rtdImageEvent.h"
int rtdInitImageEvt(char              *requestor,
                    rtdIMAGE_EVT_HNDL *eventHndl,
                    char              *error)

int rtdSendImageInfo(rtdIMAGE_EVT_HNDL  *eventHndl,
                     rtdIMAGE_INFO      *imageInfo,
                     char               *error)

int rtdAttachImageEvt(rtdIMAGE_EVT_HNDL *eventHndl,
                      char              *camera,
                      char              *error)

int rtdDetachImageEvt(rtdIMAGE_EVT_HNDL *eventHndl,
                      char              *camera,
                      char              *error)

int rtdRecvImageInfo(rtdIMAGE_EVT_HNDL *eventHndl,
                     rtdIMAGE_INFO     *imageInfo,
                     int                verbose,
                     char              *error)

int rtdClose(rtdIMAGE_EVT_HNDL *eventHndl,
             char              *error)
```

DESCRIPTION

```
rtdInitImageEvt() registers the current process e.g. image
aquisition process or rtdWidget to the rtdServer running on
the local workstation. The requestor is a string passed to identify
the process. The function returns with a valid event handle
which is used for subsequent calls to the rtdServer.

rtdSendImageInfo() is used to send image event information
when an image is ready to be displayed in shared memory.
The eventHndl is the one passed from rtdInitImageEvt,
the imageInfo is information about the image.

rtdAttachImageEvt() attaches a process to event notification of
an image source. eventHndl is the handle returned by rtdRecvImageInfo.
camera is the name of the system providing images e.g. aquisition
system. After an attach the received image events can be retrieved
by a call to rtdRecvImageInfo().
```

rtdDetachImageEvt() stops the notification of image events. eventHndl is the handle returned by rtdRecvImageInfo. camera is the name of the system providing images.

rtdRecvImageInfo() is used to receive the image event information from the rtdServer. Image events are received when the process is attached to event notification. eventHndl is the handle returned by rtdInitImageEvt, imageInfo a pointer to a rtdIMAGE_INFO structure. If verbose is non-zero diagnostic messages are printed.

rtdClose()
Closes connection to rtdServer. Use when finished with real-time display or repeated errors occuring on rtdSendImageInfo.

RETURN VALUES

RTD_OK upon success or
RTD_ERROR upon failure.

NOTE

The error field in all functions is reserved for future use.

ENVIRONMENT

The port number of rtdServer is normally specified in /etc/services. If the user want to use a different port number the this can be set in the environment RTD_SERVER_PORT.

EXAMPLE

```
// sample application which send a SHORT image to real-time display
#include <sys/ipc.h>
#include <sys/shm.h>
#include "rtdImageEvent.h"

rtdIMAGE_EVT_HNDL  eventHndl;
rtdIMAGE_INFO      imageInfo;
char               *errMsg;
int                shmId;
char               *shmPtr;

if (rtdInitImageEvt("My_CCD_Camera",&eventHndl,errMsg) == RTD_ERROR)
    {
    fprintf(stderr,"rtdInitImageEvt error:%s",errMsg);
    ... handle error ...
    }

shmId    = shmget(IPC_PRIVATE,512*512*sizeof(short),0666);

 shmPtr   = (char *)shmat(shmId,NULL,0);
if (shmPtr == -1)
    { .. handle error ... }

... generate the image ...

memset(&imageInfo, '\0', sizeof(rtdIMAGE_INFO));
imageInfo.dataType = SHORT;
imageInfo.shmId    = shmId;
imageInfo.xPixels  = 512;
imageInfo.yPixels  = 512;
```

```
/ send image event
f (rtdSendImageInfo(&eventHndl,&imageInfo,errMsg) == RTD_ERROR)
    {
    fprintf(stderr,"rtdSendImageInfo error:%s",errMsg);
    ... handle error ...
    }

/ if finishing close connection and delete shared memory
tdClose((&eventHndl,errMsg);

f (shmId) shmctl(shmId,IPC_RMID,NULL);
```

SEE ALSO

rtdServer(1)

- - - - - -

Last change: 07 May 99

4.2.11 RtdRemote(3)

NAME

RtdRemote - C++ class supporting remote access to an rtdimage

SYNOPSIS

```
#include "RtdRemote.h"

class RtdRemote {
...
public:
    RtdRemote(Tcl_Interp*, int port, int verbose);
    virtual ~RtdRemote();

    static void fileEventProc(ClientData, int mask);
    static void clientEventProc(ClientData, int mask);

    int status() {return status_;}};
```

DESCRIPTION

This class is used internally (through subclassing) by the RtdImage C++ class to support remote access via a socket interface. See rtdRemote for a description of that interface.

When a remote process wants access a running rtdimage application, rtdimage commands are sent via the socket interface. A subclass of this class defines the "call" virtual method to determine the correct method to call for each message. In this case, RtdImage defines a local class that is a subclass of RtdRemote and passes the "call" method on to its own "call" method that it uses for image subcommands.

This class keeps a table of client connections (there could be multiple connections at once, although this is probably not the norm). For each client, there is a socket connection used to send commands and receive results and an additional socket connection used for callbacks.

A socket message contains the length of the command (as a binary int in network byte order), a one byte flag indicating whether the answer should be immediate or via the callback socket and finally the contents of the command.

EXTENDING THE COMMAND SET

There are a number of ways to extend the available commands for the remote interface. One is through subclassing of class RtdImage at the C++ level (and extending the "call" method). Another way is by adding a command to the rtdimage Tcl interface to allow for a Tcl command to be evaluated for any "unknown" remote commands.

SEE ALSO

RtdImage, rtdRemote

- - - - - -

Last change: 07 May 99

4.2.12 rtdRemote(3)

NAME

rtdRemote - C interface for remote access to rtdimage based widgets

SYNOPSIS

```
#include "rtdRemote.h"

typedef void (*RtdRemoteErrorHandler)(char* message);

int rtdRemoteConnect(int pid, char* host, int port);
void rtdRemoteDisconnect();

int rtdRemoteSend(char* cmd, char** result);
int rtdRemoteGetResult(int socket, char** result);

RtdRemoteErrorHandler rtdRemoteSetErrorHandler(RtdRemoteErrorHandler);
char* rtdRemoteGetError();
```

DESCRIPTION

This man page describes a simple remote interface to rtdimage based applications. With this interface, a client application can connect to a running application displaying an rtdimage, send commands and get results.

REMOTE COMMANDS

The commands are sent as ASCII strings via socket and have the same syntax as the rtdimage Tcl commands, except that no instance name is required. The command strings are not "evaluated" by Tcl, but are interpreted by the rtdimage code. Any commands that are not handled directly by the rtdimage C++ code may be passed on to a registered Tcl handler proc or [incr Tk] method. In this way, the list of available remote commands can be extended in the Tcl/Tk application.

INTERFACE

rtdRemoteConnect(pid, hostname, port)

Connect to a remote rtdimage application. If pid, hostname and port are zero (null), they are read from the file $HOME/.rtd-remote, if it exists. This file is created by by an rtdimage widget when it starts to listen for a remote connection (see rtdimage, "remote" subcommand). Otherwise, if you know the pid, hostname and port, you can specify them here. This routine initializes an internal static structure with information about the connection.

rtdRemoteDisconnect()

Disconnect from remote rtdimage.

rtdRemoteSend(cmd, result)

The routine sends the given command to the remote rtdimage for evaluation and returns the status of the command. The result argument is set to point to the command results. The result pointer points to an internal buffer that is only valid until the next call to this routine.

The command syntax is the same as for the "rtdimage" widget (image type), except that the instance name is missing. Example:

```
            char* result;
             int status = rtdRemoteCmd("wcscenter", &result);
             if (status == 0) {
                 ...
             }

        If the command could not be sent, result is set to a NULL
        pointer and an error status (1) is returned. The error message
        can be retrieved with rtdRemoteGetError().

rtdRemoteSetErrorHandler(errorHandler)
        Set an error handler to be called when errors occur, format:
            void errorhandler(char* msg).

rtdRemoteGetError();
        Return the text of the last error message.
```

EXAMPLE

```
/*
 * The following example demonstrates the use of the remote rtd interface:
 */


/*
 * this routine is used for convenience in testing below
 * Send the command to the rtdimage, then print and return the result.
 */
static char* send_rtd(char* cmd)
{
    char* result = NULL;
    int status = rtdRemoteSend(cmd, &result);
    printf("%s ==> %s: %s\n", cmd, (status ? "FAILED" : "OK"), result);
    return result;
}

main()
{
    int data_id, header_id;

    /*
     * connect to running rtd.
     * uses default args taken from ~/.rtd-remote file
     */
    if (rtdRemoteConnect(0, NULL, 0) != 0)
        exit(1);

    /* send some commands to RTD to be evaluated */
    send_rtd("wcscenter");
    send_rtd("bitpix");
    send_rtd("scale");
    send_rtd("width");
    send_rtd("height");
    send_rtd("config -file ngc1316r.fits");
    send_rtd("width");
    send_rtd("height");

    data_id = atoi(send_rtd("shm get data"));
    header_id = atoi(send_rtd("shm get header"));

    exit(0);
}
```

SEE ALSO

rtdimage, RtdRemote

- - - - - -

Last change: 07 May 99

4.2.13 ITCL CLASSES, TCL WIDGETS

4.2.14 Rtd(n)

NAME

Rtd - real-time image display application class

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

Rtd <path> ?options?

DESCRIPTION

This class defines the top level window for the rtd (real-time image display) application. The window contains a menubar with rtd related items, an RtdImageCtrl widget for displaying the image, and related info and a short help window.

The easiest way to use this class is via the "start" method (inherited from TopLevelWidget). This creates an instance of this class and passes any command line options as public variables to the class and waits for window to be exited. Alternatively, you can create the instance in the usual way for itcl classes and withdraw the main window ".", if it is not being used.

One global variable is assumed to have been defined:

rtd_library - dir containing Rtd Tcl sources.

ITK COMPONENTS

icon
: Optional RtdImage image icon.

image
: RtdImageCtrl(n) widget containing image and control panel.

WIDGET OPTIONS

-camera
: Camera name: default: $env(RTD_CAMERA), if set, otherwise RTDSIMULATOR.

-color_scale
: Set the default color scale algorithm to one of: {linear log sqrt histeq}.

-colorramp_height
: Height of the colorramp subwindow.

-debug
: Debugging flag: enables real-time simulation with $testProg (below).

-default_cmap

```
       Default (midas) colormap.

-default_itt
       Default (midas) intensity transfer table.

-disp_image_icon
       Flag: if true, display a copy (view) of the image as an icon.

-dozoom
       Flag: if true, turn on zoom window.

-drag_scroll
       Flag: if true, set bindings to scroll with the middle mouse
       button.

-file
       Image file to display.

-float_panel
        Float the control panel (better real estate control on small
        displays).

-interval
       For testing: interval between updates in ms.

-max_colors
       Specify the max number of colors to allocate before using a
       private colormap (not impl.).

-max_scale
       Maximum allowed scale value.

-min_colors
       Specify the min number of colors to allocate before using a
       private colormap (not impl.).

-min_scale
       Minimum allowed scale value.

-noop
       Dummy option, used when cloning the main window, in place of
       "-file".

-pan_height
       Height of panning window.

-pan_width
       Width of panning window.

-panel_layout
       Panel layout order: set to one of {saoimage reverse default} to
       change the layout ordering of the panel windows.  "saoimage" puts
       the info first, followed by pan and zoom, "reverse" reverses the
       default order, which is {zoom info pan}.

-pickobjectorient
       -orient option for Pick Object window.

-port
       Default port for remote connections (0 means system chooses a
       port).

-scrollbars
```

```
       Flag: if true, display scrollbars to scroll the image.

-shm_data
       This flag controls whether the FITS image data is kept in sysV
       shared memory (see the rtdRemote interface for use of this).

-shm_header
       This flag controls whether the FITS image header is kept in sysV
       shared memory (see the rtdRemote interface for use of this).

-subsample
       Flag: if true, use faster subsampling algorithm when shrinking
       images, otherwise use max pixel algorithm.

-testprog
       For testing: name of test program used to generate real-time
       updates.

-use_zoom_view
       Flag: if true, use a "view" of the main image for the zoom window
       otherwise zoom directly from the X display.  The advantage of the
       first approach (-use_zoom_view 1) is that the zoom is accurate
       even when the main image is shrunken.  The second (-use_zoom_view
       0) is faster and allows more accurate positioning.

-usexshm
       Flag: if true, try to use X shared memory for images.

-usexsync
       Lag: if true, try to use X synchronisation.

-verbose
       Flag: if true, print diagnostic messages.

-with_colorramp
       Flag: if true (default), show the color ramp window.

-with_grid
       Option to include grid button (default to off, since it doesn't
       work well yet on some images).

-with_pan_window
       Flag: if true (default) make a panning window.

-with_perftest
       With performance tester utility in menu bar.

-with_warp
       Option to warp the mouse pointer.

-with_zoom_window
       Flag: if true (default) make a zoom window .

-xscale
       Default scaling factor (just for backwards compatibility with
       tcscam; don't use!).

-zoom_factor
       Zooming factor.

-zoom_height
       Height of zoom window.
```

```
-zoom_view_propagate
       Flag: if true, changes in main image scale will propagate to the
       zoom window, otherwise controls are displayed so the user can
       manually change it (ZoomView only).

-zoom_width
       Width of zoom window.
```

PUBLIC METHODS

```
attach_camera {}
       Attach the current camera.

clear {}
       Called for "Clear" menu item. Clear the image and delete all
       graphics.

clone {}
       Make a new main window.

detach_camera {}
       Detach the current camera.

feedback {msg}
       This method can be redefined in a subclass to get feedback during
       startup.

quit {}
       Quit the application.

record {}
       Methods for the playing and recording of images.

setXdefaults {}
       Set default X resources for colors and fonts, and set some default
       key bindings. This is done in a method so that it can be
       overridden by a subclass.  These are built-in defaults that the
       user can also override in the ~/.Xdefaults file.

set_camera {}
       Popup a window to query for new camera.
```

PROTECTED METHODS

```
add_file_menu {}
       Add the File menubutton and menu.

add_graphics_menu {}
       Add the Graphics menubutton and menu.

add_menubar {}
       Add the menubar at the top of the window.

add_realtime_menu {}
       Add the Real-time menubutton and menu.

add_view_menu {}
       Add the VIew menubutton and menu.

init {}
       This method is called after the options have been evaluated.

make_rtdimage {}
```

```
            Create the rtd image widget.

        make_short_help {}
            Add the short help window and add some help texts for the menu
            buttons.

        rapid_frame_command {frameId name op x y w h}
            This method is called when the user creates, moves, resizes or
            deletes a rapid frame.

            The args are:

             frameId = unique rapid frame id for use with rtdServer

             name = unique name for the frame

             op  = {move,resize or delete},

             x, y = coords of upper left corner of frame in image

             w, h = dimensions of frame.
```

PROTECTED VARIABLES

```
        image_
            Name of main image (class RtdImageCtrl or a derived class).

        rapid_pid_
            Pid of test prog used to generate rapid frames (debug).
```

SEE ALSO

```
        TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.15 RtdImage(n)

NAME

RtdImage - itcl widget wrapper for the rtdimage type extension

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImage <path> ?options?

DESCRIPTION

The RtdImage widget is an [incr Tk] interface to the rtdimage extended Tk image type. The widget creates a canvas window with optional scrollbars and a canvas image item to hold the image. An optional canvas line graphics editor is also created by default, to manage drawing on the image. The RtdImage widget can be treated pretty much like any standard Tk widget and can be inserted in a Tk frame with the pack(n) command. Applications using the RtdImage widget, can access the underlying image object and the canvas window to overlay graphics on the image.

In addition to the methods below, this class also forwards methods implemented in the C++ rtdimage code. It is, however, usually more efficient to use the "get_image" method to get a handle for the internal rtdimage object and use it directly.

ITK COMPONENTS

canvas
: Tk canvas containing the image.

draw
: CanvasDraw(n) object, used to manage the canvas graphics.

hscroll
: Optional horizontal scrollbar.

hscrollf
: Horizontal scrollbar frame.

imagef
: Frame to hold image and scrollbars.

vscrollf
: Vertical scrollbar frame.

STANDARD OPTIONS

-borderwidth -canvasbackground -canvasborderwidth -canvasheight -canvasrelief -canvaswidth -relief

WIDGET OPTIONS

```
-cmap_dir
       Colormap initialization and directory for colormap and ITT files.

-cmap_suffix
       Suffix for colormap files.

-color_scale
       Set the default color scale algorithm to one of: {linear log sqrt
       histeq}.

-cursor
       Default cursor.

-debug
       Debugging flag.

-default_cmap
       Default cmap file.

-default_itt
       Default ITT file.

-displaymode
       Set displaymode flag 0 to optimize for smooth scrolling, 1 for
       faster updates and less memory (works best for main image).

-drag_scroll
       Flag: if true, set bindings to scroll with the middle mouse
       button.

-file
       Fits image file to display.

-fitheight
       If non-zero, shrink image to fit height.

-fits
       For compatibility with saoimage.

-fitwidth
       If non-zero, shrink image to fit width.

-graphics
       Flag: if true, create a CanvasDraw object to manage the canvas
       graphics.

-itt_suffix
       Suffix for ITT files.

-max_colors
       Specify the max number of colors to allocate before using a
       private colormap. Note: this option is currently ignored.

-max_scale
       Maximum allowed scale value.

-min_colors
       Specify the min number of colors to allocate before using a
       private colormap. Note: this option is currently ignored.

-min_scale
       Minimum allowed scale value.
```

```
-name
       -name option.

-newimagecmd
       Command to eval when a new image is loaded.

-pickobjectorient
       -orient option for Pick Object window.

-rapid_frame_command
       Optional tcl command to be evaluated when a rapid frame is
       created, moved, resized or deleted: 6 args will be appended:

        name = unique name for the frame op  = {move,resize or delete},
        x, y = coords of upper left corner of frame in image width,
        height = dimensions of frame.

-regioncommand
       Tcl command to evaluate whenever a "region" of the image is
       selected via the graphic toolbox "region" selection item. Can be
       used to select graphic items or a section of the image for an
       operation.

-scrollbars
       Flag: if true, display horizontal and vertical scrollbars.

-shelp
       Short help text.

-shm_data
       This flag controls whether the FITS image data is kept in SysV
       shared memory (see the rtdRemote interface for use of this).

-shm_header
       This flag controls whether the FITS image header is kept in SysV
       shared memory (see the rtdRemote interface for use of this).

-shorthelpwin
       Optionally specify TopLevelWidget to display short help messages.

-show_object_menu
       Flag: if true, display menus over graphic objects when selected
       with <3>.

-subsample
       Flag: if true, use quick and dirty algorithm to shrink images.

-usexshm
       X shared memory option.

-usexsync
       X synchronisation option.

-verbose
       Flag: if true, print diagnostic messages.

-with_warp
       Option to warp the mouse pointer.

-withtoolbox
       If true (default) create the GUI interface (toolbox), otherwise
       don't.
```

```
-zoomwin
       Name of zoom window to update when mouse enters this window.
```

PUBLIC METHODS

```
attach_camera {camera}
       Attach the named camera. .

center {}
       Center the image in the canvas window.

clear {}
       Clear the current image display and remove an windows that access
       it.

delete_rapid_frame {}
       Delete the rapid frame.

detach_camera {}
       Stop the camera.  note: race conditions might cause display to lag
       behind the socket data.  force an update here.

flip {xy bool}
       Flip or unflip the image and canvas items about the x or y axis,
       as given by $xy.

get_canvas {}
       Return the name of the underlying canvas widget.

get_image {}
       Return the name of the underlying rtdimage object.

get_imageId {}
       Return the canvas Id for the image.

hide_graphics {variable}
       Toggle the visibility of the line graphics (The trace variable
       name is passed here, if 1, hide the graphics...).

maybe_center {}
       If the image is smaller than the canvas window, center it .

perftest {}
       Set the performance test mode on or off.

pick_dialog {{command ""}}
       Display a dialog for selecting objects in the image and displaying
       information about the selected area of the image.

pixel_table {nrows ncols}
       Popup a window to display a table of nrows x ncols pixel values
       from the image.

preview {var}
       Set preview mode on or off in the image. In this case, the arg is
       the "name" of a global variable controlling the preview mode. It
       will be kept up to date by this class.

print {}
       Make a hard copy of the image display.

rapid_frame {popup}
       Arrange to interactively create a rapid frame to display a section
```

```
        of the image.  If popup is 1, the frame is displayed in a popup
        window, otherwise at the selected position in the canvas.

    record {camera}
        Methods for the playing and recording of images.

    reopen {}
        Reload the image file, if there is one.

    rotate {bool}
        Toggle rotation of the image and canvas items.

    save_as {{dir "."} {pattern "*"} {x0 ""} {y0 ""} {x1 ""} {y1 ""}}
        Save the current image or a section of the current image to a file
        in FITS format chosen from a file name dialog. If dir and pattern
        are specified, they are used as defaults for the file selection
        dialog.  If x0, y0, x1 and y1 are specified (canvas coordinates),
        then a section of the image is saved.

        The return value is the name of the new file, if any, or an empty
        string.

    save_region_as {}
        Save a section of the current image to a file in FITS format
        chosen from a file name dialog.

    scale {x y}
        Resize the image and the canvas graphics by the given integer
        factors (1 is no scale, -2 = 50%, 2 = 200% etc...) - deselect
        canvas graphics (so handles don't get scaled).

    set_rtd_wcs_info {frameid}
        Set up world coordinate info for an image received from the
        rtdServer.

    show_toolbox {}
        Display the toolbox window.

    spectrum {}
        Arrange to interactively create a spectrum line to display a graph
        of the image values along a given line.
```

PROTECTED METHODS

```
    camera_post_command {frameid}
        This method is called whenever a new image has been received from
        the camera and displayed.  Update the widgets that need to display
        new values The frameid will be 0 for the main image and non-zero
        for a rapid frame.

    camera_pre_command {frameid}
        This method is called when a new image has been received from the
        camera and before it is displayed.  The frameid will be 0 for the
        main image and non-zero for a rapid frame.

    focus_ {way}
        Control the focussing of the canvas. Only take focus if the
        top-level window associated with this canvas has the focus (i.e.
        it's not in another toplevel somewhere). If this isn't done then
        mysterious raises of the main image window can occur with some
        window managers (mainly CDE, with click-to-focus).

        allan: 19.6.98: disabled the above behavior, since it causes
```

```
        problems with mouse warping and confuses people. Can't verify the
        CDE behavior...

    imageconfig_ {option}
        Utility to update an option in the image Note: this works
        automatically with "widgets", but itk doesn't work with
        "images"...

    init {}
        This method is called from the base class (TopLevelWidget) after
        all the options have been evaluated.

    load_fits_ {}
        Load a FITS file (internal version: use -file option/public
        variable).

    make_rapid_frame {popup region_id x0 y0 x1 y1}
        Create a rapid frame to display a section of the image.  If popup
        is 1, the frame is displayed in a popup window, otherwise at the
        selected position in the canvas "region_id" is the canvas id of
        the object used to position and resize the image.

    make_spectrum {line_id x0 y0 x1 y1}
        Create a graph to display the image data values along the line
        just created.  "line_id" is the canvas id of the line.

    make_toolbox {}
        Make the graphics toolbox and menu.

    new_image_cmd {}
        This method is called by the image code whenever a new image is
        loaded (for updates, see camera command).

    picked_wcs_object {x y ra dec {equinox J2000} {fwhmX ""} {fwhmY ""}
            \ {angle ""} {object ""} {background ""}}
        This method can be used in bindings to cause a selection in the
        image (to pick an object/star) to return the given position rather
        than the calculated center pos. If the optional args are not
        specified, they are calculated.

    restore_scroll_pos_ {}
        Restore the relative scrolling positions.

    save_region {canvas_id x0 y0 x1 y1}
        Save the given section of the current image to a file in FITS
        format chosen from a file name dialog. The canvas_id is the id if
        the canvas object used to select the region . The canvas
        coordinates of the region are also passed as arguments.

    save_scroll_pos_ {}
        Save the current scrolling positions.

    set_drawing_area {}
        Update the allowed interactive drawing area in the canvas.

    set_scrollregion {x0 y0 x1 y1}
        Set the canvas scrollregion .
```

PROTECTED VARIABLES

```
    canvas_
        Canvas widget.
```

```
imageId_
       Canvas Id for image.

image_
       Internal rtd image .

perftest_var_
       Name of a global variable controlling performance test mode.

preview_var_
       Name of a global variable controlling preview mode.

xScroll0_
       Saved x0 relative scrolling position.

xScroll1_
       Saved x1 relative scrolling position.

yScroll0_
       Saved y0 relative scrolling position.

yScroll1_
       Saved y1 relative scrolling position.
```

COMMON CLASS VARIABLES

```
colormap_initialized_
       Flag: true if the colormap has been initialized.
```

SEE ALSO

```
FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.16 rtdimage(n)

NAME

rtdimage - Real-Time Display Image, a Tk Image Type

SYNOPSIS

```
image create rtdimage ?option value ...?
```

DESCRIPTION

Tk4.0 introduced a new "image" command and a C interface for adding new image types. A Tk image is much like a Tk widget in that it is both an object and a Tcl command. "rtdimage" is an extended Tk image type designed for real-time image display. Images can be loaded from shared memory or FITS format files, over sockets or HTTP. For real-time usage, a background daemon process rtdServer(1) communicates with the rtdimage software over a socket interface to display and update images rapidly from shared memory. A more general purpose remote control interface is also available (see rtdRemote(3)).

CREATING RTDIMAGES

An "rtdimage" is created with the "image create" Tk command. After this, you can use the image in a Tk canvas by specifying it with the "-image" option. For example:

```
set image [image create rtdimage ...]
$canvas create image 0 0 -image $image ...
```

Most Tk image types may be used in any Tk widget, however, for our purposes, it was necessary to restrict the use to canvas widgets only. This was necessary in order to handle scrolling efficiently.

OPTIONS

The following options may be specified when creating or configuring an rtdimage:

-displaymode mode
: The rtdimage supports two different display modes: 0 and 1. In display mode 0, space is allocated in the X server for the entire image. This makes scrolling faster, but uses enormous amounts of memory when the image is very large or is scaled to a large size. Still, this mode is useful in cases where the entire image is always displayed, such as in a panning window. In displaymode 1 (default), space is only allocated for the visible part of the image. This makes scrolling somewhat slower, but uses much less memory.

-file name
: "name" specifies a FITS format file to load and display.

-fitwidth winwidth
-fitheight winheight
: These two options specify the size of the window into which the image must fit. The image will be scaled (shrunk) equally in the X and Y directions to fit as closely as possible inside the window.

```
-newimagecmd command
        The given tcl command is evaluated every time a new
        image is loaded. This command is not called for
        real-time image updates, unless the image dimensions
        or data type changed. See the "camera" subcommand
        for getting notification of real-time image updates.

-subsample bool
        If bool is true, subsampling is used when shrinking
        the image, i.e.: if the image is shrunk by 1/3, only
        every third pixel is displayed. Otherwise, the maximum
        value is taken from the group of pixels.

-usexshm bool
        If bool is true (default), attempt to use X shared
        memory for the image display, if available. This
        improves performance considerably, but is only
        available when working on the system console.

-verbose bool
        If bool is true, diagnostic messages are printed out
        to show what is going on internally (for debugging
        use).

shm_header bool
shm_data bool
        If bool is true, the image FITS header (or data) is kept in
        shared memory so that it can be accessed from a remote process
        (see rtdRemote(3)).
```

COORDINATES

```
The rtdimage subcommands support the following types of coordinates:

    canvas     - canvas coordinates (canvas scroll area)
    screen     - canvas window coordinates (visible area)
    image      - basic image pixel coords (at mag 1, no transformations)
    chip       - detector chip/CCD coordinates
    wcs        - world coordinates in H:M:S D:M:S
    deg        - world coordinates in degrees

For image coordinates, the origin of the image is at (1,1) (or .5,.5
if the image is zoomed).

Detector chip coordinates may be the same as image coordinates, but
can also have an additional offset and/or binning factor. The FITS
keywords "HIERARCH ESO DET WIN STRX" and "...STRY" are used for
the offset, if present.

The rtdimage "convert" subcommand can be used to convert between any
two coordinate systems. In addition, most rtdimage subcommands accept
coordinates using the following syntax:

    $x $y  coord_type

For example:

    set val [$image get $x $y canvas]
    set val [$image get $ra $dec "wcs 1950"]
    set val [$image get 42.1 38.3 "deg 2000"]

For world coordinates, the equinox may be optionally specified as part
```

of the coordinate type. The default is 2000.

IMAGE FORMATS

An rtdimage can load and display FITS format images or images written to shared memory via rtdServer(1). The following FITS image data types are supported: float, long, short, ushort, byte or XImage. Except for XImage, The order of lines is the same as for FITS files, with the origin at lower left. XImage is a special image type, which is taken to be already in a format that can be displayed with no color scaling. Support for other image types is planned, however the internal image type will remain FITS. New image types can be added by deriving a new subclass from the ImageIO(3) class.

COLOR ALLOCATION

All rtdimages in an application share the same default colormap. On startup, the rtdimage attempts to allocate as many color cells as possible, leaving about 10 free for other applications. The number of color cells allocated can be changed with the "alloccolors" subcommand. If another application (netscape, for example) has already grabbed all of the colors, a private colormap will be used. An attempt is made to keep most of the window manager colors intact, to avoid color flashing, at least in the GUI elements.

MOTION EVENTS

Since handling pointer motion events in Tcl code is fairly slow, the rtdimage code does some of the common work internally by setting values in a global array called "RtdImage". These values can be best accessed by specifying the "-textvariable" option to a Tk label or entry widget. The global "RtdImage" array contains the following values, which are updated on motion events:

RtdImage(X)	X image coordinate
RtdImage(Y)	Y image coordinate
RtdImage(VALUE)	pixel value at X,Y
RtdImage(RA)	world coordinate RA value
RtdImage(DEC)	world coordinate DEC value
RtdImage(EQUINOX)	world coordinate equinox

The world coordinate values are set to empty strings if the image header does not support world coordinates.

The same motion handler that sets the above variables also contains support for zoom windows (zoom and zoomview commands) and pixel tables (pixtab command).

IMAGE COMMANDS

The return value from the "image create rtdimage" command is the name of the image and also the name of a new Tcl command that can be used to operate on the image. The Tcl command has the following subcommands:

<imageName> alloccolors ?numColors?

With no arguments, this command returns a Tcl list containing the number of allocated and the number of free colors. With one argument, the command attempts to reallocate numColors colors. The number of colors actually allocated depends on what other applications are running (see COLOR ALLOCATION).

```
<imageName> autocut ?-percent number?
        This command automatically sets the cut levels (the
        lowest and highest image pixel values considered in
        colormap scaling).  Two different algorithms are
        supported. The default (and fastest version) is median
        filtering.  If -percent is specified, the argument is
        a number between 0 and 100, such as 90 for 90%, where
        that percent of the image pixels should be within the
        cut values. i.e.: if you look at the graph (see
        graphdist command) of the pixel value distribution,
        you would take the top 90% of the graph and set the
        cut levels to left and right ends of the graph.
        Note: if this command is called, it is assumed that
        cut levels can be set automatically when a new image
        is loaded. See also the "cut" command.

<imageName> camera start cameraName ?tclCommand?
<imageName> camera stop
<imageName> camera pause
<imageName> camera continue
        The "camera start" command sends a message to the
        rtdServer daemon process telling it to start sending
        images from the given camera.  Actually the server
        sends only image events, short messages over a socket
        interface, while the images are written to and read
        from shared memory. Camera is the name of a camera
        that must be known to the rtdServer (see rtdServer(1)
        for more information).  The optional ?tclCommand?
        argument to "start" should be a string containing a
        Tcl command to be evaluated whenever a new image event
        is received and displayed. The "camera stop" command
        tells the rtdServer to stop sending image events. The
        "pause" and "continue" subcommands can be used to
        temporarily stop the image events and restart them,
        without having to know the name of the camera.

<imageName> clear
<imageName> clear ximage
<imageName> clear ?-reuse $reuse
                   -ra $ra -dec $dec -equinox $equinox -radius $radius
                   -width $width -height $height?
        This command is used to blank out the display by generating and
        loading a blank image. With no arguments a small blank image is
        generated with a default header. If "-ximage" is specified, the
        image is only cleared temporarily, until the next image update.

        In the last case, the optional arguments are used to generate
        a dummy image that supports world coordinates, so that you can
        plot objects on a blank background. Any missing values are set
        to a default value.

        Optional arguments:

        reuse   - flag: if true, reuse previous image, if it is the same
        ra, dec - center point for WCS coords (in decimal degrees)
        radius  - used to initialize WCS coords (CDELT1 and 2)
        equinox - equinox for WCS coords
        width   - width of generated image in pixels
        height  - height of generated image in pixels

<imageName> cmap file   ?<colormapFile>?
<imageName> cmap rotate <amount>
```

```
<imageName> cmap shift  <amount>
<imageName> cmap pixels
<imageName> cmap reset
<imageName> cmap list
<imageName> cmap private
<imageName> cmap isprivate
        This command performs operations and queries on the colormap.
        If a colormap file is specified, it should contain 256 lines
        of red, green and blue values between 0.0 and 1.0 (MIDAS
        colormaps are saved in this format).  The values will be
        distributed among the available colors and installed as a new
        colormap.

        For rotate and shift, the amount can be any integer. The
        colormap will be rotated (or shifted) by that amount.

        "pixels" returns a Tcl list of the colormap pixel values (for
        use by external applications using the RTI library, class
        ImageData). To get the number of colors in the colormap, you
        can use the "alloccolors" subcommand with no arguments or
        "llength" on the result of the pixels subcommand.

        "reset" resets the colormap to its original state.  The RTD
        release includes a large number of MIDAS colormap files in the
        colormap directory.

        For "cmap file", if the filename is not specified, the current
        colormap file name is returned.

        "cmap list" returns a list of all of the colormap files
        currently loaded.

        "cmap private" says to start using a private colormap.

        "cmap isprivate" returns true if the colormap is private.

<imageName> colorramp
        This command generates an rtdimage displaying the
        colors in the colormap as a ramp or colorbar. This
        image will have the same size as the window containing
        it. This command should be called again from Tcl if
        the window is resized.

<imageName> colorscale ?scale_type?
        This command sets or queries the algorithm to be used
        for assigning the limited number of available colors
        to image pixels. If scale_type is specified, it should
        be one of: linear, log, sqrt or histeq, indicating the
        color scaling algorithm: linear scaling, logarithmic,
        square root or histogram equalization, resp.  With no
        arguments, the current color scale type is returned.

<imageName> convert coords inx iny in_coord_type outx outy out_coord_type
<imageName> convert dist inx iny in_coord_type outx outy out_coord_type
        This command is used to convert between different coordinate
        representations. inx and iny and the input coords (or
        distance) in the given input coordinate system. "convert
        coords" treats x,y as a point, while "convert dist" treats it
        as a distance. outx and outy, if not empty, are the names of
        variables that will hold the resulting coordinates.  If outx
        and outy are empty strings, the values are returned as a tcl
        list "x y".
```

```
        The available coordinate systems are:

        canvas     - canvas coordinates (canvas scroll area)
        screen     - canvas window coords (visible area)
        image      - basic image pixel coords (at mag 1, no transformations)
        wcs        - world coordinates in H:M:S
        deg        - world coordinates in degrees

        The world coordinate types: "wcs" and "deg" may also include
        the epoch: Example:

        $image convert coords $ra $dec "wcs 1950" x y canvas

        Note: the coordinate types may be abbrieviated, since only the
        first char is actually checked.

<imageName> cut
<imageName> cut low high
<imageName> cut low high fromUser
        This command sets or queries the cut levels. If low and high
        are specified, then the cut levels are set so that pixels
        below the low value will all have the lowest color while those
        above high will all have the highest color value.

        The optional fromUser argument indicates whether or not this
        is a result of a user action and defaults to 1 (true). Once a
        user has set the cut levels, automatic cut level setting is
        disabled.  If the fromUser argument is 1, it is assumed that
        they should not be changed automatically when a new image is
        loaded. Calling the autocut subcommand resets this again (see
        the autocut subcommand).

        If no arguments are given, the current cut values are returned
        in a Tcl list {min max}.

<imageName> dispwidth
<imageName> dispheight
        These commands return the logical width and height of
        the image after transformations (scaling and
        rotating).  This is the size of the displayed image,
        assuming the window is large enough. This command also
        takes the image's "requested width" into account (set by
        by "view update" subcommand).

<imageName> dump <filename>
        This command dumps the current image to the given file in FITS
        format.  If a FITS header is present, it is used, otherwise
        FITS keywords are inserted indicating the image type, width
        and height along with the date and a number of numbered "blank
        cards" or FITS keyword fields that can be modified by other
        applications as needed.  The fields have names starting with
        BLANK followed by 2 digits (from BLANK00 to BLANK28).

<imageName> flip <direction> ?bool?
        With two arguments, flip (or stop flipping) the image
        in the given direction, where direction is one of x,
        y, xy or "none" for flipping in the x, y, or x and y
        directions or neither.  The boolean value turns
        flipping on (1) or off (0) in the given direction(s).
        With one argument, the command returns the current
        value for the given argument.

<imageName> frameid
```

```
        This command returns the frame Id of this image. The
        frame Id is a unique number used to identify the image
        to the rtdServer for use with rapid frames.

<imageName> get x y coord_type ?nrows ncols?
        Returns a Tcl list of image values at the given X,Y
        coordinates.  X and Y are interpreted in the given coordinate
        system (see COORDINATES above).  The return value is a tcl
        list where each item consists of a list of {X Y Value}, where
        X and Y are the adjusted coordinates in the raw image and
        Value is the raw data value at that point or "-" if out of
        range.  If nrows and ncols are greater than 1, the command
        returns a Tcl list of nrows x ncols values, each a list of
        rows, centered at the given point.

<imageName> graphdist bltGraph bltElem numValues
        This command displays the distribution of pixel values
        in the image in the given BLT graph widget. The data
        for the given BLT graph element will be set directly
        to the graph without going through tcl (see
        blt_graph(n)). The number of points to plot is given
        by the numValues argument.

<imageName> itt file <ITTFile>
<imageName> itt scale <scaleFactor>
        This command operates on MIDAS style intensity
        transfer tables or ITTs. If an ITT file is specified,
        it should contain 256 intensity values in the range
        0.0 to 1.0, one per line. The colormap will be
        modified by applying the intensities to it.  The
        colormap can also be stretched or squeezed by applying
        an integer scale factor to the ITT.  The RTD release
        contains a number of ITTs in the colormaps directory.

<imageName> max
        Returns the highest pixel value in the image.

<imageName> mband x0 y0 x1 y1 cord_type show_angle
        Draw a measure band on the canvas to show the distance in
        world coordinates (diagonal, vertical and horizontal).

        This method was originaly implemented in Tcl/[incr Tk], but
        was redone here for better performance.

        x0 and y0 are the starting coordinates of the drag, x1 and y1
        are the coordinates from the motion events and show_angle is a
        flag: if true, show the horizontal and vertical distance,
        otherwise only the diagonal.

        The coordinates are accepted in the given coordinate system
        "coord_type", see COORDINATES above.

<imageName> min
        Returns the lowest pixel value in the image.




<imageName> mmap set $data_filename $data_offset $data_owner
                     ?$header_filename $header_offset $header_owner?
<imageName> mmap get data
<imageName> mmap get header
```

```
<imageName> mmap create $filename $size
<imageName> mmap delete $filename
<imageName> mmap update
        This subcommand provides access to the mmap shared memory in
        which the FITS image data and header are stored. Image files
        are always mapped with mmap by default (since it is faster
        than reading the file). Applications can take advantage of
        this to modify the image data and then notify the application
        to update the image. This command makes it posible to put the
        image data and header in separate files, so that they can be
        more easily updated by other applications. If you want to put
        both header and data in the same file in the normal way, just
        use "<imageName> config -file". Otherwise you can use this
        command to quickly update the image data in a separate file.

        The "set" command allow you to set the files to use to for the
        image data and header. The data and header in the specified
        files should be in FITS format (i.e.:, a FITS file split in 2
        parts). If the header is not specified, the previous header is
        reused, if there was one.  The offset arguments indicate an
        offset in the file where the header or data start. If the file
        contains only the data or only the header, the offset argument
        should be set to 0.  A flag indicating who "owns" the file may
        be specified (if true, then the file will be deleted when no
        longer needed).

        Example: <imageName> mmap set datafile1 0 0 headerfile1 0 0
                 <imageName> mmap set datafile2 0 0
                ...

        The "get" command returns mmap information about the data or
        header.  If the data or header is not currently mapped, an
        error is returned.  The return value is a list of the form
        {filename offset owner}, the same as the arguments to the
        "<imageName> mmap set" command.

        The "create" command creates a new mmapped file with the given
        name and the given size. The mmaped file/memory should be
        released with the "delete" subcommand when no longer needed.

        The "delete" command unmaps the given file and deletes it, if
        it was created with the "mmap create" subcommand.

        The "update" command causes the display to be updated to
        reflect any changes in the image memory.

<imageName> pan start <tclCommand> <shrinkFactor>
            pan stop
        This command supports a panning image, which is, in
        this case, a second rtdimage image or "view" of the
        main image, scaled to a small size with a rectangle
        indicating the visible portion of the image.  If
        "start" is specified, the given tcl command will be
        evaluated whenever the image size changes, due to
        scaling or loading a new image, or whenever the image
        position has changed due to scrolling.  The tcl
        command will be called with 5 arguments: x1 y1 x2 y2,
        which are the coordinates of the visible part of the
        image, scaled by the given "shrinkFactor", and a flag
        indicating whether the image is new (1) or an update
        of the existing image (0).  This can be used to draw
        the panning rectangle on the panning image.  To stop
        the command from being called, use the "pan stop"
```

```
        subcommand.

<imageName> pixtab start <nrows> <ncols>
<imageName> pixtab stop
        This command supports displaying a table of pixel values
        around a point.  All this commmand does is set a flag causing
        Tcl array variables to be updated on motion events, which can
        cause the display to be updated via the "-textvariable" widget
        option on the table items.
        The array name is fixed as: RtdPixTab and the elements are
        indexed as $RtdPixTab(i,j), where the left and top sides of
        the table (array) are the X and Y image coordinates, resp. and
        the rest are image pixel values.

<imageName> preview <bool>
        If bool is true and real-time images are being
        displayed, the viewing mode is set to "preview mode",
        otherwise, it is set back to "real-time mode". In
        preview mode, the camera is stopped (if it was
        running) and a local copy of the shared memory image
        is made, so that it can be freed or modified without
        affecting the image.

<imageName> radecbox <ra> <dec> <radius>
        ra and dec are the world coords (h:m:s or decimal deg) and
        radius is expected in arcmin.  The return value in Tcl is a
        list of 4 values {ra0 dec0 ra1 dec1} that form a ra,dec box
        with the given center point and radius.

<imageName> remote ?$port?
        This command implements a remote control of the RTD image.  If
        a port number argument is specified The widget will start
        listening for commands on the given port. If port is 0, a port
        number will be chosen.

        If no port number is specified, the current port number is
        returned, or "" if there is none. This is a way to determine
        the port number at the Tcl level.

<imageName> remotetcl ?$command?
        Evaluate a Tcl command in the RTD Tcl interpreter.

<imageName> rotate ?bool?
        Rotate (or stop rotating) the image. Currently,
        rotation is only done by swapping the x and y axis. If
        bool is specified, rotation is turned on(1) or off(0).
        Otherwise, the current setting is returned.

<imageName> scale ?sx sy?
        With 2 arguments, the image is scaled (magnified) by
        the given X and Y amount.  With no arguments, the
        current scaling factors are returned (as a tcl list of
        2 integers). The scaling factors are positive or
        negative integers (default 1). Positive integers are
        used to zoom in on the image (2 means twice the
        original size). Negative integers are used to zoom out
        (-2 means 1/2 the original size). The software imposes
        an arbitrary limit on the minimum and maximum scaling
        factor allowed.

<imageName> shm set $data_size $data_id $data_owner
                    ?$header_size $header_id $header_owner?
<imageName> shm get data
```

```
<imageName> shm get header
<imageName> shm create $size
<imageName> shm delete $Id
<imageName> shm update
        This subcommand provides access to the shared memory in which
        the FITS raw image data and header are stored. The raw image
        is stored in shared memory if the -shm_data option was
        specified when creating the image and the header is stored in
        shared memory if the -shm_header option was specified.

        The "set" command allow you to set the shared memory Ids to
        use to access the image data and header. The data and header
        in the area specified should be in FITS format. If the header
        is not specified, the previous header is reused. For both data
        and header, the size of the area (in bytes) and the shared
        memory Id must be specified. In addition a flag indicating who
        "owns" the shared memory is specified (if true, then the area
        will be deleted when no longer needed).

        The "get" command returns the shared memory Id of the data or
        header.  If the data or header is not currently in shared
        memory, it is copied to a new shared memory area and the Id
        for this area is returned.

        The "create" command creates a new shared memory area with the
        given size and returns the Id. The memory should be deleted
        with the "delete" subcommand when no longer needed.

        The "delete" command deletes the shared memory with the given
        Id (which should have been returned from the "create"
        subcommand).

        The "update" command causes the display to be updated to
        reflect any changes in the image memory.

<imageName> spectrum <bltGraph> <bltElem> x0 y0 x1 y1 coord_type
        This command is used to display a graph of a "cut" of the
        image along a given line. x0, y0, x1 and y1 are the end points
        of a line in the image (in the given coordinate system, see
        COORDINATES above).  <bltGraph> is the path name of a BLT
        graph widget to display the plot of the pixel intensities
        along the line.  <bltElem> is the name of the element in the
        graph that should receive the data.  The data is sent directly
        to the graph for display.  The return value in Tcl is the
        number of points to plot.

<imageName> statistics
        statistics subcommand: calculate statistics on the section of
        the image being displayed.  The return value in Tcl is a list
        of the following values:

          {x y ra dec equinox fwhmX fwhmY angle objectPeak meanBackground}

        where:

        x              = adjusted X image coordinate
        y              = adjusted Y image coordinate
        ra             = RA position (calculated from mean X pos within array)
        dec           = DEC position (calculated from mean Y position within array)
        equinox        = equinox of RA and DEC
        fwhmX          = FWHM in X
        fwhmY          = FWHM in Y
        angle          = angle of major axis, degrees, along X = 0
```

```
        objectPeak     = peak value of object above background
        meanBackground = mean background level

<imageName> type
        Returns the data type of the raw image as a string:
        one of: float, long, short, ushort, byte or
        XImage. The last type, XImage is a special pseudo
        type, the same as a byte image, except that the Y axis
        is reversed and it is assumed to not need color
        scaling.

<imageName> update
        This command makes sure that the image is up to date with the
        raw data (which may have changed via shared memory, mmap, etc).

<imageName> view add <path> ?propagateScale?
<imageName> view remove <path>
<imageName> view update <path> x y width height viewx viewy coord_type
<imageName> view enter  <path>
<imageName> view leave  <path>
        The view command is used to specify a viewing image to view
        the same image, possibly at a different size.  The new view
        will share data with the original and be updated when the
        original is updated.  This can be used, for example, to build
        a panning window or a rapid frame.

        <path> must be the name of a second rtdimage image. The two
        images will communicate internally to always display the same
        image, possibly scaled to different sizes. The subcommands are:

        add
                Adds a new view to this image.

        remove
                Removes the view.

        update
                Updates the view from this image with the given image
                x,y offset, width and height and the position of the
                image view origin in the given coordinate type. This
                command can be used to implement a zoom window or
                rapid frame, since it controls which portion of the
                image is displayed.

        enter
                If 2 images are in the same canvas, make <path> the
                current one (receives motion events, ...).

        leave
                Undo the enter command.

        If the optional "add" argument "propagateScale" is true,
        changes in the scale factors in the master image will
        propagate to the view (this is the default behavior).

<imageName> warp <x> <y>
        Warp (move) the mouse pointer by the given x and y amounts
        (pixels).


<imageName> wcscenter ?-format <format>?
        This command returns the world coordinates of the center of
        the image.  The optional format option determines the format
```

```
        of the result:

        -format 0 ==> H:M:S [+-]D:M:S (default)
        -format 1 ==> RA DEC (in degrees)

        The return value is a tcl list, formatted according to the
        format option, or an empty string if the coordinates are out
        of range or WCS is not supported.

<imageName> wcsdist x0 y0 x1 y1
        This command returns the world coordinate distance between 2
        points after transformations.  The arguments are expected in
        canvas coords (canvasx, canvasy, doubles).  The return value
        in Tcl is the WCS distance between the given points after
        transformations.

<imageName> wcsheight
        This command returns the height of the image in arcmin or the
        empty string if WCS is not supported.

<imageName> wcswidth
        This command returns the width of the image in arcmin or the
        empty string if WCS is not supported.

<imageName> wcsradius
        This command returns the radius (distance from center to
        corner) of the image in arcmin or the empty string if WCS is
        not supported.

<imageName> wcsset <ra> <dec> <secpix> <xrefpix> <yrefpix> <nxpix> <nypix>
                   <rotate> <equinox> <epoch> <projection>
<imageName> wcsset
        If arguments are specified, this subcommand sets up the WCS
        structure from the given information about the image:
        Arguments:
          ra      = Center right ascension in H:M:S
          dec     = Center declination in D:M:S
          secpix  = Number of arcseconds per pixel
          xrefpix = Reference pixel X coordinate
          yrefpix = Reference pixel Y coordinate
          nxpix   = Number of pixels along x-axis
          nypix   = Number of pixels along y-axis
          rotate  = Rotation angle (clockwise positive) in degrees
          equinox = Equinox of coordinates, 1950 and 2000 supported
          epoch   = Epoch of coordinates, used for FK4/FK5 conversion no
                    effect if 0
          proj    = Projection

        With no arguments, the command returns a list of the basic WCS
        parameter values: {ra dec secpix nxpix nypix rotate equinox epoch}.

<imageName> wcsshift <ra> <dec> <coorsys>
        This command resets the center of the WCS structure.
        Arguments:
          ra        = New center right ascension in degrees
          dec       = New center declination in degrees
          equinox   = must be 2000 or 1950

<imageName> width
<imageName> height
        These commands return the width and height of the raw
        image in pixels.
```

```
<imageName> zoom start <frame> <zoomFactor>
<imageName> zoom stop
        (Note: This command is no longer supported: please use
        zoomview (below) instead.)

        This command is used to implement a zoom window, a window
        displaying a magnified section of the image at the location of
        the mouse pointer.  There are currently two versions of this
        command (see the zoomview subcommand below). In this version,
        a Tk frame is specified to hold the zoomed image, which is
        copied directly from the XImage whenever the mouse pointer
        moves over the image. This version is faster, but when the
        main image is shrunk, the zoom will not be very accurate.  If
        "start" is specified, zooming begins in the given window, and
        can be stopped with the "zoom stop" subcommand.

<imageName> zoomview start <view> <zoomFactor>
<imageName> zoomview stop
        This command can be used as an alternative to the zoom
        command above. It uses a "view" of the main rtdimage, so the
        zoom image is always accurate, even when main image is shrunk.
        The "view" argument to "zoomview start" should be the name of
        a second rtdimage, which is a "view" of the main image, added
        with the rtdimage "view" subcommand.  The zoomFactor is the
        magnification relative to the main image. For example, if the
        zoomFactor is 5 and the main image is scaled to 1/2, the zoom
        window scale factor would be 4.  Once started, the main image
        will automatically track mouse movements and update the zoom
        window's x and y offsets as needed to display the relevant
        magnified section of the image.
```

ENVIRONMENT VARIABLES

```
RTD_LIBRARY - If set, this should point to the directory
              containing the rtdimage Tcl library files.
```

FILES

```
$RTD_LIBRARY/                   - Tcl/Itcl library files
$RTD_LIBRARY/colormaps          - MIDAS colormap/ITT files
$RTD_LIBRARY/images             - sample FITS images
$RTD_LIBRARY/bitmaps            - X bitmaps used at runtime
$RTD_LIBRARY/demos              - rtdimage demo application
```

SEE ALSO

```
RtdImage(n), rtdServer(1), rtdImageEvt(3), BLT(n), canvas(n)



- - - - - -
Last change: 07 May 99
```

4.2.17 RtdImageColorRamp(n)

NAME

RtdImageColorRamp - itcl widget used to display contents of the

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageColorRamp <path> ?options?

DESCRIPTION

This [incr Tk] widget class displays the colors in the colormap from left to right in a generated rtdimage. In addition, bindings are added to the colorramp to rotate, shift, stretch and squeeze the colormap by dragging the mouse pointer with a button pressed.

ITK COMPONENTS

image

RtdImage item used to display colors in colormap.

WIDGET OPTIONS

-cursor

Cursor for window.

-height

Height of colorramp (width is same as window).

-shelp

Help text displayed when mouse enters widget.

-usexshm

X shared memory option.

-viewmaster

"viewmaster" image. This is also updated when colorramp changes. This allows changes to be propagated, even if using a read-only visual. .

PUBLIC METHODS

reset_colors {}

Reset the colormap.

update_colors {}

Update the colorramp after the window has been resized or the number of colors has changed (need to delay to always get the correct size).

PROTECTED METHODS

mark {pos}

```
              Mark the given position for later reference.

       mark_for_shift {pos}
              Mark the given position for later reference and set things up for
              a shift operation.  (The dummy rotate op causes an internal copy
              between cmap and itt
               that initializes the shift from the current itt.).

       rotate_colors {pos}
              Rotate the colormap by the difference between the given position
              and the position set with mark.

       save_cmap {}
              Called after a shift or scale operation is done (button up) to
              save the colormap state. We just do a null rotate here, since it
              does what we want. This prevents the colormap from reverting to
              the original state before each shift or scale operation. The
              reason it would revert is that otherwise colors shifted off to the
              left or right, for example, would be lost.

       scale_itt {pos}
              Scale the current ITT based on the difference between the given
              position and the position set with mark.

       shift_colors {pos}
              Shift the colormap by the difference between the given position
              and the position set with mark.
```

PROTECTED VARIABLES

```
       canvas_
              Canvas window containing ramp image.

       image_
              Internal rtdimage widget for colorramp.

       mark_
              Used to save a position for rotating the colormap.
```

SEE ALSO

```
       FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.18 RtdImageColors(n)

NAME

RtdImageColors - itcl widget for managing colormap for an rtdimage

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImageColors <path> ?options?

DESCRIPTION

This [incr Tk] widget presents a user interface for manipulating colors and colormaps for an RtdImage widget. The widget creates a new toplevel window containing items for "color scaling" the image, loading a MIDAS style colormap or ITT (intensity transfer table) and for setting the number of color cells allocated in the colormap.

ITK COMPONENTS

alloc
: Frame for displaying allocated/free colors.

allocated
: LabelEntryScale widget displaying the number of allocated colors.

apply
: Apply button.

buttons
: Frame for buttons.

close
: Close button.

colormaps
: Chooser(n) widget listing available colormaps.

defaults
: Defaults button.

free
: LabelValue(n) widget displaying the number of free colors.

itts
: Chooser(n) widget listing available intensity tables.

scale
: LabelChoice(n) widget for choosing a color scale algorithm.

top
: Top frame.

WIDGET OPTIONS

```
-cmap_dir
       Directory for colormap and ITT files.

-cmap_suffix
       Suffix for colormap files.

-default_cmap
       Default (midas) colormap.

-default_itt
       Default (midas) intensity transfer table.

-image
       Name of RtdImage itcl widget, set by caller.

-itt_suffix
       Suffix for ITT files.

-max_colors
       Max number of colors to allocate.

-min_free
       Min number of free colors to leave.
```

PUBLIC METHODS

```
reallocate {}
       Called when the scale value is changed to reallocate the colors.

set_cmap {cmap}
       This method is called to set the colormap for the image.

set_color_scale {alg}
       This method is called to set the color scaling algorithm.

set_defaults {}
       Set the default colormap and itt.

set_itt {itt}
       This method is called to set the itt for the image.

update_allocated {}
       Update the display to show the number of free and allocated
       colors.

update_values {im}
       Update the display with the values set in the given rtdimage.

use_private_colormap {}
       If flag is true, use a private colormap, otherwise the default.
```

PROTECTED METHODS

```
make_short_help {}
       Add a short help window.

set_allocated {num_colors}
       This method is called to set the number of allocated colors.
```

PROTECTED VARIABLES

allocated_
Number of colors allocated.

free_
Number of free colors.

COMMON CLASS VARIABLES

image_
Name of current internal rtdimage object.

images_
Array (itk RtdImage) of C++ RtdImage objects, for updating clone colors.

SEE ALSO

TopLevelWidget(n)

- - - - - -
Last change: 07 May 99

4.2.19 RtdImageCtrl(n)

NAME

RtdImageCtrl - Widget combining an RtdImage with a control panel

NAMESPACE

rtd

PARENT CLASS

rtd::RtdImage

SYNOPSIS

RtdImageCtrl <path> ?options?

DESCRIPTION

RtdImageCtrl is an itcl widget combining the RtdImage itcl widget with a control panel, zoom and panning windows.

RtdImageCtrl inherits all of the features described in RtdImage(n) and also adds the following user interface components: Zoom window (see RtdImageZoomView(n)), Panning window (see RtdImagePan(n)), Colormap display widget (see RtdImageColorRamp(n)), Image Control panel (see RtdImagePanel(n)).

Each of the added user interface components is defined in a separate [incr Tk] widget class. In addition, some methods from RtdImage are redefined in order to update the user interface display.

ITK COMPONENTS

colorramp
: Color ramp widget (RtdImageColorRamp(n)).

gridcheck
: Check button for optional WCS grid.

gridf
: Frame at lower right for the grid checkbutton and size entry.

gridsize
: LabelEntry for grid size.

info
: Info panel, RtdImagePanel(n) object used to display image controls.

pan
: Pan window (RtdImagePan(n) widget).

panel
: The RTD control panel, may be put in a frame or optionally in a popup window.

zoom
: Zoom window (RtdImageZoomView(n) widget).

zoom

```
        Zoom window: this version is not really supported any more...
```

WIDGET OPTIONS

```
-colorramp_height
       Height of the colorramp subwindow.

-default_cmap
       Default cmap file.

-default_itt
       Default ITT file.

-dozoom
       Flag: if true, turn on zoom window.

-feedback
       Command used to display feedback during startup.

-float_panel
        Floating panel option (for small displays).

-pan_height
       Height of panning window.

-pan_width
       Width of panning window.

-panel_layout
       Panel layout order: set to one of {saoimage reverse default} to
       change the layout ordering of the panel windows.  "saoimage" puts
       the info first, followed by pan and zoom, "reverse" reverses the
       default order, which is {zoom info pan}.

-port
       Default port for remote connections (0 means system chooses a
       port).

-use_zoom_view
       Flag: if true, make zoom window a view of the main image otherwise
       do a faster, but less accurate (by shrunken images) zoom from the
       xImage.

-with_colorramp
       Flag: if true (default), show the color ramp window.

-with_grid
       Option to include grid button (default to off, since it doesn't
       work well yet on some images).

-with_pan_window
       Flag: if true (default) make a panning window.

-with_warp
       Option to warp the mouse pointer.

-with_zoom_window
       Flag: if true (default) make a zoom window .

-xscale
       Default scaling factor (just for backwards compatibility with
       tcscam; don't use!).
```

```
-zoom_factor
       Zooming factor.

-zoom_height
       Height of zoom window.

-zoom_view_propagate
       Flag: if true, changes in main image scale will propagate to the
       zoom window, otherwise controls are displayed so the user can
       manually change it (ZoomView only).

-zoom_width
       Width of zoom window.
```

PUBLIC METHODS

```
clear {}
       Clear the current image display and remove an windows that access
       it (extend parent class version).

feedback {msg}
       This method is called at startup to give feedback while building
       the interface.

hide_control_panel {variable}
       Toggle the visibility of the control panel (argument is the name
       of the checkbutton variable to use).

inc_zoom {inc}
       Add the given increment to the current zoom factor and re-scale
       the image.

open {{dir "."} {pattern "*.*fit*"}}
       Open and load a new FITS image file via file name dialog.

scale {x y}
       This method is redefined here to update the scale display.  resize
       the image and the canvas graphics by the given integer factors (1
       is no scale, -2 = 50%, 2 = 200% etc...).

set_bias {}
       Pop up a window to control bias subtraction.

set_colors {}
       Pop up a window to edit the image colors.

set_grid_size {size}
       Set the size of the grid (space between lines) in arc seconds of
       dec degrees.

set_wcs_info {list}
       This command is called with a list of values from wcs_info_dialog
       above to set new world coordinates information for the current
       image.

show_grid {variable}
       Toggle the visibility of the image ra,dec grid (argument is the
       name of the checkbutton variable to use).

update_color_window {}
       Update the settings in the color popup to reflect those of the
       image.
```

```
update_colors {}
       This method is called when the colormap has been changed to update
       the display.

view_fits_header {}
       View the FITS header in a text window.

wcs_info_dialog {}
       Pop up a dialog to display/edit the basic world coordinate
       parameters.
```

PROTECTED METHODS

```
init {}
       This method is called from the base class (TopLevelWidget) after
       all the options have been evaluated.

make_colorramp {}
       Add a generated image to display the colors in the colormap.

make_control_panel {}
       Make the control panel for operating on the image.

make_grid_item {}
       Create an item in the panel to control the ra,dec grid size.

make_pan_window {panel}
       Make the pan window.

make_panel_info {panel}
       Make the panel info subwindow.

make_panel_subwindows {panel}
       Add the panel subwindows.

make_zoom_window {panel}
       Make the zoom window in the panel.

new_image_cmd {}
       This method is called by the image code whenever a new image is
       loaded.  (for real-time updates, see camera command).
```

PROTECTED VARIABLES

```
cut_
       Cut values frame.

grid_var_
       Name of trace var for grid.

zoom_state_
       Saved zoom button state.
```

SEE ALSO

```
RtdImage(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.20 RtdImageCut(n)

NAME

RtdImageCut - itcl widget for setting cut levels for an RtdImage widget

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImageCut <path> ?options?

DESCRIPTION

This widget displays a toplevel window containing a plot of the pixel value distribution in the target image and buttons and scales for manipulating the image cut levels. The cut levels are two values: the lowest and highest pixel values considered when color scaling the image, i.e.: mapping image pixel values to color values, and are used to filter out noise and other extreme values in the image.

The plot displayed uses the rtdimage "getdist" subcommand to get the pixel value distribution. This is an array of values that specify, for example, how many pixel values are between 0 and 100, between 100 and 200, and so on. This information can also be used to set the cut levels, by specifying a percent of the total number of pixels that should be within the cut levels.

A second, faster algorithm is supported by the "Median Filter" button. This is a standard algorithm that works very fast to determine reasonable cut levels.

After setting the cut levels, either manually or by one of the buttons, the new pixel value distribution is displayed. The plotting is done directly from the rtdimage C++ code to the BLT graph over its C interface.

ITK COMPONENTS

buttons
: Tk frame for buttons.

graph
: BLT graph widget for displaying pixel distribution.

percent
: LabelChoice(n) widget displaying percent values for setting cut levels.

scales
: Frame containing scale widgets to adjust cut levels.

$compo
: Button components: set, reset, median, update, or close.

${el}cut

```
        Lowcut and Highcut components (LabelEntryScale(n) widgets) for
        displaying and adjusting the cut levels.

    ${el}scale
        Lowscale and Highscale component frames.
```

WIDGET OPTIONS

```
    -command
        Tcl command to evaluate when cut levels are changed.

    -image
        Target RtdImage itcl class object.

    -num_points
        Number of points to plot.
```

PUBLIC METHODS

```
    add_button {compo text command}
        Add a button to the buttons frame.

    get_cuts {}
        Return the current low/high cut values.

    init {}
        Called after constructors have run.

    reset_cutlevels {}
        Reset the cut levels to the original min/max values.

    set_by_median {}
        Automatically set the cut values by median filtering.

    set_by_percent {percent}
        Automatically set the cut values by percent of distribution that
        should be inside the cut levels.

    update_graph {{modimg 1}}
        Update the graph after a new image has been loaded or the image
        has been modified.
```

PROTECTED METHODS

```
    entry_value {compo value}
        Write value into entry field.

    make_buttons {}
        Make the button frame at the bottom of the window.

    make_controls {}
        Make the control panel.

    make_graph {}
        Make the graph subwindow.

    make_short_help {}
        Add a short help window.

    notify_blt_zoom {pathname tickvalue}
        This method is called when blt changes the tick labels after a
        zoom.
```

```
set_cutlevels {}
       Set the cut levels in the image.

setb_cut {flg low high}
       Set low and high cut.

setb_highcut {sethigh value}
       Set entry values of the highcut scale widget and update scale
       widgets.

setb_lowcut {setlow value}
       Set entry values of the lowcut scale widget and update scale
       widgets.

update_cut {low high}
       Update min, max values of the lowcut and highcut scale widgets.

update_highcut {low high}
       Update min, max values of the highcut scale widget.

update_increment {}
       Update the increment for the slider buttons depending on the range
       XXX not currently used - problems with looping in event handlers
       (allan).

update_lowcut {low high}
       Update min, max values of the lowcut scale widget.

update_xaxis {{value 0}}
       Update xaxis after rescaling.
```

PROTECTED VARIABLES

```
graph_
       Name of graph widget.

high_
       Highest value to display (image(max) or set by user).

image_
       Internal rtdimage object.

initialized_
       Set to 1 after widget has been initizlized.

low_
       Lowest value to display (image(min) or set by user).

xVector_
       X vector for graph.

yVector_
       Y vector for graph.
```

SEE ALSO

```
TopLevelWidget(n)
```

- - - - - -

```
Last change: 07 May 99
```

4.2.21 RtdImageFrame(n)

NAME

RtdImageFrame - itcl widget for displaying a section of an rtdimage

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageFrame <path> ?options?

DESCRIPTION

This widget is used to display rapid frames for an RtdImage widget. A rapid frame is an instance of a RtdImage widget that displays a small section of the main image and can be updated faster with real-time images because it is smaller than the main image.

The area in the main image being used for the rapid frame is marked with one black and one white dashed rectangle. The rapid frame can be moved or resized in in the same way as any other graphic objects by dragging with the left mouse button over it or on one of the 8 resize handles displayed around it when it is selected. One of the dashed rectangles shows the current position of the rapid frame while the other one shows the new position and size being set.

Two types of rapid frames are supported. The first type is an RtdImage "view" of the main image embedded into its canvas window at a given x,y offset, so that it appears that main image is being updated more frequently inside the dashed box. In this case, the rapid frame is a separate rtdimage canvas image item in the same canvas with the main image, but at a different offset. The second type of rapid frame is displayed in a popup window and is implemented by the class RtdImagePopup(n).

Creating and manipulating a rapid frame usually involves communication with the rtdServer and camera, to tell the camera to start sending images at the given rate from the given area. Since this is very application specific, you can arrange to have your own Tcl command evaluated whenever a rapid frame is created, moved, resized or deleted. See the RtdImage(n) -rapid_frame_command option for how to do this.

Note that currently, only one rapid frame is allowed at a time. Creating a second one automatically deletes the first. This may be changed in a future release.

WIDGET OPTIONS

-command

Tcl command to be evaluated whenever the frame is created moved, resized or deleted: 7 args will be appended:

frameId = unique rapid frame id for use with rtdServer

name = unique name for the frame

```
         op  = {move,resize or delete}

         x, y = coords of upper left corner of frame in image

         width, height = dimensions of frame.

    -height
           Height of image frame.

    -region_id
           Canvas id of the (region) object used to position and move the
           image in the canvas.

    -subsample
           Flag: if true, pan image is "subsampled" when shrinking, otherwise
           the pixels are averaged.

    -target_image
           Target rtdimage.

    -usexshm
           X shared memory option.

    -usexsync
           X synchronisation option.

    -verbose
           Flag: if true, print diagnostic messages.

    -width
           Width of image frame.

    -xoffset
           X offset of image frame.

    -yoffset
           Y offset of image frame.
```

PUBLIC METHODS

```
    get_image {}
           Return the name of the underlying rtdimage object.

    notify_cmd {op args}
           This method is called (from the main image's CanvasDraw(n) widget)
           whenever an embedded rapid frame is moved, resized or deleted.  If
           the "-command" option was given to this class, then that tcl
           command is evaluated with the frameId, operation name (move,
           resize, delete) the x, y coords and the width and height of the
           frame.
```

PROTECTED METHODS

```
    init {}
           This method is called from the base class (FrameWidget) after all
           the options have been evaluated.
```

PROTECTED VARIABLES

```
    canvas_
           Canvas window containing rapid frame image .
```

```
draw_
       CanvasDraw object, for setting up move, resize operations on
       embedded image.

frameId_
       Rapid frame Id, needed to communicate with rtdServer.

imageId_
       Canvas image id.

image_
       Internal rtdimage for rapid frame.

rectId_
       Canvas id of rectangle used to get events for moving/resizing
       image.

target_image_
       Target internal rtdimage.
```

SEE ALSO

```
FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.22 RtdImageGrid(n)

NAME

RtdImageGrid - itcl class to display an ra,dec grid over an image.

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageGrid <path> ?options?

DESCRIPTION

RtdImageGrid is an itcl class to display an ra,dec grid over the image.

WIDGET OPTIONS

-color
: Grid line color.

-image
: Main RtdImage widget (set by caller).

-size
: Grid spacing, size of a grid box in arcsecs if not specified, a default is chosen based on the image.

PUBLIC METHODS

hide {}
: Hide (stop showing) the grid.

reset {}
: Reset the grid (redraw it, if needed, probably for a new image).

show {}
: Show the grid with the current settings.

size {}
: Return the actual size of the grid (space between lines) in arcsecs of degrees.

PROTECTED METHODS

draw {}
: Draw the grid based on the current settings.

draw_line {points}
: Draw a line with the given points.

inc_dec {dec inc}
: Return dec + inc in deg.

inc_ra {ra inc}

```
            Return ra + inc in deg.
```

PROTECTED VARIABLES

```
       canvas_
              Canvas window for image.

       image_
              Internal rtdimage widget for main image.

       pi_
              Const PI.

       size_
              Current size of grid in arc secs of deg (same as -size if
              specified, otherwise it is calculated based on the size of the
              image).
```

SEE ALSO

```
       FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.23 RtdImageIcon(n)

NAME

RtdImageIcon - itcl widget to display current image in icon window

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImageIcon <path> ?options?

DESCRIPTION

RtdImageIcon is an itcl widget used to display the current image in the icon window.

ITK COMPONENTS

image

RtdImage widget.

WIDGET OPTIONS

-image

Target RtdImage (itcl widget).

-subsample

Flag: if true, pan image is "subsampled" when shrinking, otherwise the pixels are averaged.

-usexshm

X shared memory option.

-usexsync

X synchronisation option.

-verbose

Flag: if true, print diagnostic messages.

-width

Dimensions of pan frame.

SEE ALSO

TopLevelWidget(n)

- - - - - -

Last change: 07 May 99

4.2.24 RtdImageMBand(n)

NAME

RtdImageMBand - itcl class to display a "measure band"

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageMBand <path> ?options?

DESCRIPTION

RtdImageMBand is an itcl widget class used to display a "measure band" showing the distance between two points in world coordinates.

WIDGET OPTIONS

-arrow_shape
Line arrow shape option for measure band.

-arrow_type
Line arrow type option for measure band.

-cursor
Default cursor when drawing.

-defaultcursor
Default cursor when not drawing.

-fill_color
Fill color for label rectangle.

-image
Main RtdImage widget (set by caller).

-line_color
Line color option for measure band.

-line_width
Line width option for measure band.

-outline_color
Outline color for label rectangle.

-text_color
Text color for label.

-text_font
Font to use for labels.

PUBLIC METHODS

check_stop {x y}
Stop displaying the mband if the user has moved the mouse since it

```
               was created.

       mband {x y show_angle}
               Update the display of the measure band to show the distance
               between the endpoints in WCS.

       start {x y}
               Start displaying the measure band.

       stop {}
               Stop displaying the measure band and restore cursor and bindings.
```

PROTECTED VARIABLES

```
       canvas_
               Canvas window for image.

       image_
               Internal rtdimage widget for main image.

       saved_bindtags_
               Saved canvas bindings, restored later.

       x_
               X starting point of line.

       y_
               Y starting point of line.
```

SEE ALSO

```
       FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.25 RtdImagePan(n)

NAME

RtdImagePan - itcl widget managing the RtdImage panning window

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImagePan <path> ?options?

DESCRIPTION

This widget displays a "view" of another RtdImage widget at a smaller magnification, so that the entire image is visible in a small window. A rectangle displayed over the image can be used to pan or move the image when the target image is to large to be viewed at all at once in its window. The rectangle is always notified of changes in the target image or its window, so it always displays the relative size of the visible image to the entire image. The pan window is based on the rtdimage "pan" subcommand and uses canvas graphics to display the rectangle.

Since it is not known ahead of time how large or small an image will be, the pan window is given a maximum size when created. When an image is loaded, it shrinks the image by equal integer factors until it fits in the window. Then it fits the window around the image, so as not to leave a blank (black) space around it. Rotated and flipped images are also displayed as rotated and flipped in the pan window. Only the scale factor remains fixed.

ITK COMPONENTS

image

RtdImage(n) widget for displaying a copy of the image.

STANDARD OPTIONS

-cursor -height -subsample -verbose -width -zoomwin

WIDGET OPTIONS

-cursor

Default cursor.

-height

Height of pan frame.

-shelp

Help text displayed when mouse enters widget.

-subsample

Flag: if true, pan image is "subsampled" when shrinking, otherwise the pixels are averaged.

-target_image

```
        Target RtdImage (itcl widget).

    -usexshm
        X shared memory option.

    -usexsync
        X synchronisation option.

    -verbose
        Flag: if true, print diagnostic messages.

    -width
        Width of pan frame.

    -zoomwin
        Zoom window to update.
```

PUBLIC METHODS

```
    init_panning {}
        Initialize the pan window after a new image has been loaded and
        arrange to be notified of changes in the position and size of the
        visible image.

    notify_cmd {op}
        This method is called when the user moves or resizes the panning
        rect.  op is set to "resize" or "move" (resize not currently
        supported).

    stop_panning {}
        Stop the panning callback.
```

PROTECTED METHODS

```
    draw_compass {}
        Draw an ra,dec compass indicating N and E by following lines along
        ra and dec near the center of the image.

    pan {x1 y1 x2 y2 changed}
        Update the panner rectangle to display the current position and
        size of the target image x1 y1 x2 y2 give the visible portion of
        the image if "changed" is 1, there is a new image with pos.
        different dimensions.
```

PROTECTED VARIABLES

```
    canvas_
        Canvas for panning image.

    compassfont_
        Compass label fonts.

    draw_
        Name of CanvasDraw widget for image.

    image_
        Panning image.

    marker_
        Canvas id of a second rectangle used to mark old position.

    panFactor_
        Amount panning image was shrunk (=2 = 1/2, -4 = 1/4, ...).
```

```
panImageHeight_
       Height of the panning image, after shrinking.

panImageWidth_
       Width of the panning image, after shrinking.

panner_
       Canvas id of the panning rectangle.

pi_
       Const PI.

target_canvas_
       Target image's canvas window.

target_image_
       Internal target image being panned (rtdimage) .
```

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.26 RtdImagePanel(n)

NAME

RtdImagePanel - widget for displaying relevant image information

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImagePanel <path> ?options?

DESCRIPTION

RtdImagePanel is a display and control panel for the RtdImageCtrl(n) widget. It displays the following information:

the object name, the name of the file or camera being viewed

the x,y pixel and world coordinates and pixel value under the mouse pointer

the minimum and maximum image pixel values

the image data type

the low and high cut levels

the scale (magnification), flip(X,Y) and rotate settings

In addition to displaying the current image information, the cut levels can be set by editing them and hitting return, the scale factor can be selected from a menu and the rotation and flip X/Y settings can be toggled with checkbuttons.

ITK COMPONENTS

autocut
: "Auto Set Cut Levels" button.

bitpix
: LabelValue(n) widget for the FITS bitpix value.

dec
: LabelValue(n) widget for DEC coordinate.

high
: LabelEntry(n) widget for the high cut level.

low
: LabelEntry(n) widget for the low cut level.

lrframe
: Frame at the lower right of the panel that optionally holds the "Auto Set Cut Levels" button, ...

```
max
       LabelValue(n) widget for the max pixel value.

min
       LabelValue(n) widget for the min pixel value.

object
       LabelValue(n) object displaying object name or file name.

ra
       LabelValue(n) widget for RA coordinate.

trans
       RtdImageTrans(n) widget displaying the rotate, flip and zoom
       controls.

value
       LabelValue(n) widget for pixel value.

x
       LabelValue(n) widget for X image coordinate.

y
       LabelValue(n) widget for Y image coordinate.
```

STANDARD OPTIONS

```
       -state
```

WIDGET OPTIONS

```
       -image
              Main RtdImage widget (set by caller).

       -labelfont
              Font to use for labels.

       -labelwidth
              Set the width for displaying labels.

       -max_scale
              Maximum allowed scale value.

       -min_scale
              Minimum allowed scale value.

       -shorthelpwin
              Optionally specify TopLevelWidget to display short help messages.

       -showcut
              Flag: if true, display the image cut levels.

       -showminmax
              Flag: if true, display the min and max pixel values.

       -showobject
              Flag: if true, display the object name.

       -showtrans
              Flag: if true, display the transformation widgets (zoom factor,
              etc).

       -showwcs
```

```
            Flag: if true, display ra,dec coordinates.

        -showxy
            Flag: if true, display x,y coordinates.

        -state
            Set the state to normal/disabled to enable/disable editing.

        -valuefont
            Font to use for values.

        -valuewidth
            Set the width for displaying values.

        -wcsfont
            Font to use for RA,DEC (a, b) labels (symbol).
```

PUBLIC METHODS

```
        auto_set_cut_levels {}
            Set the cut levels automatically using median filtering...

        cut_level_dialog {}
            Set the cut levels.

        updateValues {}
            Update the display with the current image values.

        update_cut_window {}
            Update the cut level display window, if needed.
```

PROTECTED METHODS

```
        make_layout {}
            Do the widget layout, aligning the items in rows and colums.

        set_cut_levels {args}
            Set the cut levels when the user types them in and hits return.
```

PROTECTED VARIABLES

```
        image_
            Internal rtdimage widget for main image.

        x_
            Saved x coordinate for update after image event.

        y_
            Saved y coordinate for update after image event.
```

SEE ALSO

```
        FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.27 RtdImagePerf(n)

NAME

RtdImagePerf - itcl widget to show current performance statistics.

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImagePerf <path> ?options?

DESCRIPTION

RtdImagePerf is an itcl widget to show current performance statistics.

ITK COMPONENTS

buttons
: Dialog buttons frame.

close
: Close button.

freq
: LabelValue(n) widget "Update Frequency (Hz)".

freq_ave
: LabelValue(n) widget for average frequency.

gen
: LabelValue(n) widget: "General Code Processing (s)".

gen_ave
: LabelValue(n) widget for average general image processing.

labl
: LabelValue(n) widget "Last image".

labl_ave
: LabelValue(n) widget "Average".

maxfreq
: LabelEntry(n) widget "Max frq:".

mem
: LabelValue(n), "Memory Management (s)".

mem_ave
: LabelValue(n) widget for average memory management.

reset
: Reset button.

save
: Save button.

```
set_units
       LabelMenu widget: "Select display mode:".

tcl
       LabelValue(n) widget: "TCL Code Interpretation".

tcl_ave
       LabelValue(n) widget: average TCL code interpretation.

total
       LabelValue(n) widget: "Total time/image".

total_ave
       LabelValue(n) widget: average total time spent per image event.

x
       LabelValue(n) widget: "X Function Calls".

x_ave
       LabelValue(n) widget: average X function calls.
```

WIDGET OPTIONS

```
-labelfont
       Font used for labels.

-labelwidth
       Set the width for  displaying labels.

-target_image
       Target (main) RtdImage itcl widget.

-valuefont
       Font used for values.

-valuewidth
       Set the width for  displaying values.
```

PUBLIC METHODS

```
cancel {}
       Close this window.

convert_units {new_units}
       This method takes the performance parameters from the dialogue and
       converts them to the new display mode, before placing them back in
       the dialogue.

save {}
       Save the current performance parameters and image information to
       file.

set_units {unit_type}
       Set the required units that the performance parameters are to be
       displayed in.
```

PROTECTED METHODS

```
add_buttons {}
       Add a row of dialog buttons at the bottom of the window.

make_labels {w}
```

```
            Make the window to display the statistics in the given frame.

        make_layout {}
            Do the window layout.
```

PROTECTED VARIABLES

```
        canvas_
            Internal canvas widget.

        current_display_mode
            Current unit type.

        image_
            Internal rtd image.

        target_canvas_
            Internal target canvas.

        target_image_
            Internal target image.
```

SEE ALSO

```
        TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.28 RtdImagePick(n)

NAME

RtdImagePick - widget to select an object in an image using a centroid alg.

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImagePick <path> ?options?

DESCRIPTION

RtdImagePick is an itcl widget to select an object (a star, for example) in an image and get statistics for it. It is based on the rtdimage(3) "statistics" subcommand, which uses a centroid algorithm to locate the center of the object.

ITK COMPONENTS

angle
: LabelValue(n) widget for "Angle of X axis".

background
: LabelValue(n) widget for "Background level".

buttons
: Dialog buttons frame.

cancel
: Cancel button.

close
: Close button.

dec
: LabelValue(n) widget for DEC (delta).

equinox
: LabelValue(n) widget for equinox.

fwhm
: LabelValue(n) for "FWHM X:Y".

nsize
: LabelValue(n) widget for number of "Pixels in x,y".

object
: LabelValue(n) widget for "Peak object level above bg".

pick
: "Pick Object" button.

ra
: LabelValue(n) widget for RA (alpha).

```
x
       LabelValue(n) widget: "Image X" .

y
       LabelValue(n) widget: "Image Y".

zoomView
       This component displays the section of the image that will be used
       for the centroid algorithm and statistics.
```

WIDGET OPTIONS

```
-command
       Command to evaluate when a selection is made or canceled.

-debug
       Debugging flag.

-factor
       Default zoom magnification factor.

-labelfont
       Font to use for labels.

-labelwidth
       Set the width for  displaying labels and values.

-maxsize
       Set the max size of the image sample area (screen dist).

-orient
       Specify the orientation of image and panel, one of {vertical
       horizontal}.

-target_image
       Target (main) RtdImage itcl widget.

-usexshm
       X shared memory option.

-usexsync
       X synchronisation option.

-valuefont
       Font to use for values.

-valuewidth
       Set the width for  displaying labels and values.

-verbose
       Flag: if true, print diagnostic messages.

-wcsfont
       Font to use for ra,dec labels (alpha, delta).
```

PUBLIC METHODS

```
cancel {{with_pick 1}}
       Cancel the current pick operation.

cancel_pick {}
       Cancel the wait for the pick_object method and reset the cursor.
```

```
close {}
       Close this window.

pick_object {{parms ""} {wait 0}}
       Let the user select a point in the image and get the statistics on
       the area. The optional argument "parms" may be set to {x0, y0},
       the size of the image box. If given, pick_object returns the
       result without user interaction. .

picked_object {}
       This method is called when the user clicks in the image to select
       an object or star for the "pick_object" method. .

picked_special_object {x y ra dec equinox}
       This method is called when the user clicks in the image to select
       an object or star for the "pick_object" method. In this case, the
       x,y and ra,dec position are fixed and only the other info should
       be calculated (used).

picked_wcs_object {retval}
       This method can be called when the user has selected an object or
       star for the "pick_object" method.  The argument should be the
       value returned from the rtdimage "statistics" subcommand.

update_scale {{fx 1} {fy 1}}
       Dummy (called by RtdImage.tcl which assumes that the scaling is
       propagated).
```

PROTECTED METHODS

```
add_buttons {}
       Add a row of dialog buttons at the bottom of the window.

blink_mark {canvas tags {color 0}}
       This method is used to make the marker given by "tags" blink on
       and off.

format_val {val}
       Format a floating point value (which may also be empty).

init {}
       This method is called after the options have been evaluated.

make_labels {w {side left}}
       Create the window for displaying the statistics in the given
       frame.  The optional "side" arg is used for packing.

make_layout {}
       This method is responsible for the window layout.

make_rect {w {side left}}
       Create the window used to display the part of the image being
       examined.  $w is the parent frame.  The optional "side" arg is
       used for packing.

mark_spot {imagex imagey image canvas angle width height {blink 0}}
       Mark the given x,y image coordinate point in the given
       image/canvas with a cross with the given width, height (image
       pixels) and angle (deg).

pick_object_in_image {}
       This method is called to allow the user to pick an object in the
```

```
        main image.  The return value is a list of:  "ra dec equinox fwhmX
        fwhmY angle objectPeak meanBackground" as returned from the
        rtdimage "statistics" subcommand, or an error.

    scale_changed {}
        Callback from isrtdZoomView when the scaling was changed.

    set_result_value {}
        Set the set_result_ variable to 1.

    set_values {list {with_rmt 1}}
        Set the values of the labels from the list (results of
        "pick_object" call).  If list is empty the labels are cleared.  If
        the list is not empty, mark the ra,dec spot in the image.

    update_now {}
        Returns statistics after image event.
```

PROTECTED VARIABLES

```
    afterId_
        Id for blink after job.

    canvas_
        Internal canvas widget.

    imageX_
        X coord of last picked object.

    imageY_
        Y coord of last picked object.

    image_
        Internal zoomView rtd image.

    initialized_
        Set to 1 after init {} was called.

    list_
        Output of last statistics command after scaling and pick object
        command.

    target_canvas_
        Internal target canvas.

    target_image_
        Internal target image.

    waiting_
        Waiting for image event before returning result.
```

SEE ALSO

```
    TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.29 RtdImagePixTable(n)

NAME

RtdImagePixTable - itcl widget for displaying a table of pixel values

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImagePixTable <path> ?options?

DESCRIPTION

This widget displays a variable sized table of raw image pixel values from an RtdImage widget with the given pixel at the center. This is meant to be bound to mouse motion events to display pixel values as the mouse moves across the image.

ITK COMPONENTS

buttons
: Button frame.

label
: Tk label "Pixel Table".

slabel
: Tk label "Statistics".

stat
: Statistics Tk frame.

tab
: BLT table frame.

pixtab_$lel
: LabelValue(n) widgets: pixtab_Min, pixtab_Max, pixtab_Ave, pixtab_RMS, pixtab_N.

WIDGET OPTIONS

-image
: Caller's RtdImage itcl object.

-labelfont
: Fonts used.

-labelwidth
: Set the width for displaying labels and values.

-nrows
: Number of rows/columns of pixels to display.

PUBLIC METHODS

```
        blank_values {}
               Method to blank out all pixel values.

        statistics {}
               Method to switch statistics window on/off.
```

PROTECTED METHODS

```
        make_buttons {}
               Make the button frame at the bottom of the window.

        make_statistics {}
               Statistics on pixels.

        make_table {}
               Make the table of pixel values with the X,Y coords at the top and
               left resp.
```

PROTECTED VARIABLES

```
        canvas_
               Canvas for image.

        image_
               Internal rtdimage object.

        making_stat_
               Flag for "making statistics widget".
```

SEE ALSO

```
        TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.30 RtdImagePopup(n)

NAME

RtdImagePopup - A toplevel widget for displaying rapid frames for RtdImage

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImagePopup <path> ?options?

DESCRIPTION

This widget is used to display rapid frames in a popup window for an RtdImage widget. A rapid frame is an instance of a RtdImage widget that displays a small section of the main image and can be updated faster with real-time images because it is smaller than the main image.

The area in the main image being used for the rapid frame is marked with one black and one white dashed rectangle. The rapid frame can be moved or resized in in the same way as any other graphic objects by dragging with the left mouse button over it or on one of the 8 resize handles displayed around it when it is selected. One of the dashed rectangles shows the current position of the rapid frame while the other one shows the new position and size being set.

Creating and manipulating a rapid frame usually involves communication with the rtdServer and camera, to tell the camera to start sending images at the given rate from the given area. Since this is very application specific, you can arrange to have your own Tcl command evaluated whenever a rapid frame is created, moved, resized or deleted. See the RtdImage(n) -rapid_frame_command option for how to do this.

Note that currently, only one rapid frame is allowed at a time. Creating a second one automatically deletes the first. This may be changed in a future release.

ITK COMPONENTS

image

RtdImage(n) widget to display in a popup window.

info

RtdImagePanel(n) widget, control panel.

mband

RtdImageMBand(n) widget: The "measure band" is displayed while the right mouse button is pressed to show the distance between points.

WIDGET OPTIONS

-command

This tcl command is evaluated whenever the frame is created moved, resized or deleted: 7 arguments will be appended to the command

```
        before it is evaluated:

        frameId: Unique rapid frame id for use with rtdServer.

        name: Unique name for the frame.

        op: Operation: one of: move,resize or delete.

        x, y: Coordinates of upper left corner of frame in main image.

        width, height: Dimensions of rapid frame.

    -height
        Height of image frame.

    -max_scale
        Maximum allowed scale value.

    -min_scale
        Minimum allowed scale value.

    -region_id
        Canvas id of the (region) object used to position and move the
        image in the canvas.

    -shelp
        Text of short help message to be displayed whenever the mouse
        enters the image window (see Toplevel.tcl).

    -shorthelpwin
        Optionally specify TopLevelWidget to display short help messages.

    -subsample
        Flag: if true, pan image is "subsampled" when shrinking, otherwise
        the pixels are averaged.

    -target_image
        Target rtdimage.

    -usexshm
        X shared memory option.

    -usexsync
        X synchronisation option.

    -verbose
        Flag: if true, print diagnostic messages.

    -width
        Width of image frame.

    -xoffset
        X offset of image frame.

    -yoffset
        Y offset of image frame.

    -zoomwin
        Zoom window to update.
```

PUBLIC METHODS

```
    get_image {}
```

```
        Return the name of the underlying rtdimage object.

    notify_cmd {op args}
        This method is called (from the main image's CanvasDraw(n) widget)
        whenever an embedded rapid frame is moved, resized or deleted.  If
        the "-command" option was given to this class, then that tcl
        command is evaluated with the frameId, operation name (move,
        resize, delete) the x, y coords and the width and height of the
        frame.

    set_cut_levels {}
        Set the cut levels.
```

PROTECTED METHODS

```
    add_menubar {}
        Add the menubar at the top of the window.

    init {}
        Add bindings and callbacks after the constructor was called.

    make_panel {}
        Make the upper panel .

    new_image_cmd {}
        This method is called by the image code whenever a new image is
        loaded (for updates, see camera command).
```

PROTECTED VARIABLES

```
    canvas_
        Canvas window containing rapid frame image, different than
        target_canvas for popup frames.

    draw_
        CanvasDraw object, for setting up move, resize operations on
        embedded image.

    frameId_
        Rapid frame Id, needed to communicate with rtdServer.

    image_
        Internal rtdimage for rapid frame.

    rectId_
        Canvas id of rectangle used to get events for moving/resizing
        image.

    target_canvas_
        Canvas widget for main image and region object marking frame.

    target_image_
        Target internal rtdimage.
```

SEE ALSO

```
    TopLevelWidget(n)
```

- - - - - -

```
Last change: 07 May 99
```

4.2.31 RtdImagePrint(n)

NAME

RtdImagePrint - popup dialog for printing an RTD image

NAMESPACE

rtd

PARENT CLASS

util::PrintDialog

SYNOPSIS

RtdImagePrint <path> ?options?

DESCRIPTION

RtdImagePrint defines a popup dialog for printing an RTD image. This class extends the PrintDialog class.

ITK COMPONENTS

options
: Print options frame.

title
: Title for option frame.

WIDGET OPTIONS

-bot_left
: Footer text to appear at bottom left.

-bot_right
: Footer text to appear at bottom right.

-fit_to_page
: Flag, if true, scale output to fit on page.

-footer_font
: Footer fonts.

-image
: Name of Itcl RtdImage or derived widget, set by caller.

-pageheight
: Page height, used when fit_to_page is 1.

-pagewidth
: Page width, used when fit_to_page is 1.

-show_footer
: Flag, if true (1), insert footers before printing.

-show_headers
: Alias for -show_footer, for backward compatibility.

-top_left
: Footer text to appear at top left.

```
-top_right
       Footer text to appear at top right.

-whole_canvas
       Flag, it true whole canvas is captured, this includes any graphics
       that extends outside the image.

-x0
       Upper left X coordinate of area of canvas to print (default bbox
       all).

-x1
       Bottom right X coordinate .

-y0
       Upper left Y coordinate .

-y1
       Bottom right Y coordinate .
```

PROTECTED METHODS

```
add_footer {}
       Add footer labels below draw area by temporarily inserting the
       text.

add_short_help {}
       Add short help texts.

init {}
       This method is called after all options have been evaluated.

print {fd}
       Print the contents of the canvas to the open filedescriptor.

remove_background {}
       Remove the background of the canvas.

rm_footer {}
       Remove the footer, if any and restore the original state.

set_background {}
       Set/remove the background of the canvas.

set_show_footer {}
       Modify show_footer.

set_whole_canvas {}
       Modify capture all canvas items.

toggle_fit_pagesize {}
       Called when the "Fit on page" button is pressed.
```

PROTECTED VARIABLES

```
canvas_
       Canvas widget.

image_
       Internal rtdimage object.

x0
```

```
              X0 of area to print.

       x1
              X1 of area to print.

       y0
              Y0 of area to print.

       y1
              Y1 of area to print.
```

SEE ALSO

```
       PrintDialog(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.32 RtdImageSpectrum(n)

NAME

RtdImageSpectrum - itcl widget for displaying graph of image data values

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdImageSpectrum <path> ?options?

DESCRIPTION

This [incr Tk] widget is used to display a BLT graph in a popup window plotting the raw image pixel values along a given line drawn interactively on the image. Once created, the graph can be continuously updated as the line is moved or resized. This widget only sets up the layout. The real work is done in the rtdimage spectrum subcommand (see rtdimage(3)), that communicates directly with the BLT graph using its C interface.

ITK COMPONENTS

fpos
: Tk frame for X,Y positions.

graph
: BLT graph of pixel values.

xpos
: Tk label for X position.

yval
: Tk label for Y position.

WIDGET OPTIONS

-image
: Name of RtdImage itcl widget, set by caller.

-line_id
: Canvas id of the spectrum line .

-x0
: X0 canvas coordinate of the spectrum line.

-x1
: X1 canvas coordinate of the spectrum line.

-y0
: Y0 canvas coordinate of the spectrum line.

-y1
: Y1 canvas coordinate of the spectrum line.

PUBLIC METHODS

notify_cmd {{op update}}
This method is called whenever the spectrum line is moved, resized or deleted or when the image changed and the graph should be updated. It updates the graph to show the image values along the line.

print {}
Make a hard copy of the graph display.

quit {}
Quit the window.

PROTECTED METHODS

dispXY {x y}
Display x, y values at cursor position.

make_buttons {}
Make the button frame at the bottom of the window.

make_graph {}
Make the graph subwindow.

PROTECTED VARIABLES

canvas_
Name of image's canvas widget.

draw_
Name of RtdImage's CanvasDraw object.

graph_
Name of graph widget.

image_
Name of internal rtdimage object.

numValues_
Number of values displayed.

xVector_
X vector for graph.

yVector_
Y vector for graph.

SEE ALSO

TopLevelWidget(n)

- - - - - -
Last change: 07 May 99

4.2.33 RtdImageTrans(n)

NAME

RtdImageTrans - itcl widget for scaling, rotating and flipping an RtdImage widget

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageTrans <path> ?options?

DESCRIPTION

RtdImageTrans is an [incr Tk] widget class for setting and displaying image transformation states, such as rotation, flipX, flipY and scale (magnification). The widget displays a menubutton with a selection of image scale factors (from 1/5x to 9x magnification), 2 optional buttons for incrementing and decrementing the scale factor, and buttons for setting rotation, flipX and flipY.

ITK COMPONENTS

choose
: LabelMenu(n) widget to choose scale factor.

flipx
: Tk checkbutton to flip the X axis.

flipy
: Tk checkbutton to flip the Y axis.

larger
: Tk button to zoom in.

rotate
: Tk checkbutton to rotate (swap X/Y axis).

smaller
: Tk button to zoom out.

STANDARD OPTIONS

-background -foreground -state

WIDGET OPTIONS

-image
: Target RtdImage (itcl widget).

-labelfont
: Font for label and value.

-labelwidth
: Set the width for displaying the label.

```
-max_scale
       Maximum allowed scale value.

-min_scale
       Minimum allowed scale value.

-show_Zz_buttons
       Flag: if true, display buttons for zooming the image in and out.

-show_trans
       Flag: if true, display the rotate, flipxy items.

-state
       Set the state to normal/disabled to enable/disable editing.

-valuewidth
       Set the width for displaying the value.
```

PUBLIC METHODS

```
fill_mag_menu {m}
       Fill the given menu with radiobuttons for changing the
       magnification of the image and keep them updated with the other
       controls.

flip {xy}
       Flip or unflip the image about the x or y axis, as given by $xy.

inc_zoom {inc}
       Add the given increment to the current zoom factor and re-scale
       the target image.

rotate {}
       Toggle rotation of the image.

update_trans {}
       Update the display based on the image scale factors (note that the
       menu values are referenced to here by their labels).
```

PROTECTED VARIABLES

```
image_
       Internal rtdimage.
```

SEE ALSO

```
FrameWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

4.2.34 RtdImageZoom(n)

NAME

RtdImageZoom - itcl widget managing the RtdImage zoom window

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageZoom <path> ?options?

DESCRIPTION

Note: It is better to use the RtdImageZoomView class, since this class is outdated and being phased out.

This [incr Tk] widget class can be used to display a magnified portion of the image while tracking mouse motion events in the image window. There are two versions of this widget, see RtdImageZoomView(n) for the other one. This version takes a caller supplied Tk frame and zooms directly from the main image's XImage to the frame, but does not display an accurate image when the main image is "subsampled" (shrunk).

The main part of this widget is implemented in C++ by the rtdimage subcommand "zoom".

This widget is a subclass of FrameWidget, so it inherits its methods and options. In addition the options and methods below are defined.

ITK COMPONENTS

check
: Checkbutton to turn zoom on/off.

frame
: Zoom frame.

WIDGET OPTIONS

-factor
: Zoom factor (window size should be a multiple of this).

-height
: Height of zoom frame.

-shelp
: Help text displayed when mouse enters widget.

-target_image
: Target RtdImage itcl widget.

-usexshm
: X shared memory option.

```
-usexsync
       X synchronisation option.

-width
       Width of zoom frame.
```

PUBLIC METHODS

```
enter_image {image}
       This method is called when the mouse ptr enters an RtdImage.  Set
       the target scale factor from the given rtdimage.

leave_image {image}
       This method is called when the mouse ptr leaves an RtdImage.
       clear out the zoom image.

scale {}
       Called when the main image is scaled. .

zoom {}
       Called when the zoom checkbutton is pressed.
```

PROTECTED VARIABLES

```
zoom_
       Internal zoom frame.
```

SEE ALSO

FrameWidget(n)

- - - - - -

Last change: 07 May 99

4.2.35 RtdImageZoomView(n)

NAME

RtdImageZoomView - itcl widget managing the RtdImage zoom window

NAMESPACE

rtd

PARENT CLASS

util::FrameWidget

SYNOPSIS

RtdImageZoomView <path> ?options?

DESCRIPTION

This [incr Tk] widget class can be used to display a magnified portion of the image while tracking mouse motion events in the image window. There are two versions of this widget, see RtdImageZoom(n) for the other one. This version uses an rtdimage "view" of the main image and changes the x and y offsets as needed. This has the advantage that it always displays the correct pixels, even when the main image is "subsampled" and there are no restrictions on the size or shape of the zoom window. The main part of this widget is implemented in C++ by the rtdimage subcommand "zoomview". This widget is a subclass of FrameWidget, so it inherits its methods and options. In addition the options and methods below are defined.

ITK COMPONENTS

check
: Checkbutton to turn zooming on/off.

f
: Frame with on/off button and scale menu.

image
: RtdImage(n) widget for zoom.

label
: Optional label for zoom factor.

larger
: Optional button to increase zoom factor.

smaller
: Optional button to decrease zoom factor.

STANDARD OPTIONS

-background -height -labelfont -width

WIDGET OPTIONS

-command
: Optional command to evaluate when the zoom factor changes.

-factor

```
       Zoom magnification factor.

-height
       Height of zoom frame.

-labelfont
       Fonts used.

-propagate
       Flag: if true, make scale of zoom window relative to target
       window.

-shelp
       Help text displayed when mouse enters widget.

-target_image
       Target (main) RtdImage itcl widget.

-usexshm
       X shared memory option.

-usexsync
       X synchronisation option.

-verbose
       Flag: if true, print diagnostic messages.

-width
       Width of zoom frame.
```

PUBLIC METHODS

```
enter_image {image}
       This method is called when the mouse ptr enters an RtdImage.  Set
       the target scale factor from the given rtdimage.

inc_zoom {inc}
       Increment or decrement the zoom factor.

leave_image {image}
       This method is called when the mouse ptr leaves an RtdImage.
       clear out the zoom image.

scale {}
       Called when the main image is scaled to draw a box around the
       center pixel.

zoom {{clear 0}}
       Called when the zoom checkbutton is pressed.
```

PROTECTED VARIABLES

```
canvas_
       Internal canvas widget.

image_
       Internal rtd image.

target_image_
       Internal target image.

target_scale_
       Scale of the target (or current target) image.
```

SEE ALSO

FrameWidget(n)

- - - - - -
Last change: 07 May 99

4.2.36 RtdRemoteTcl(n)

NAME

RtdRemoteTcl - itcl widget testing the remote Tcl interface

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdRemoteTcl <path> ?options?

DESCRIPTION

RtdRemoteTcl is an itcl widget for testing the remote Tcl interface and some Rtd functions.

ITK COMPONENTS

action
: Action frame.

cancel
: Cancel button.

close
: Close button.

testall
: "Test all above" button.

$el
: $el is one of: fliprotate zoom colors cut pixtab fitsh rapid record perft cuts pick.

STANDARD OPTIONS

-state

WIDGET OPTIONS

-file
: File in images directory to load after startup.

-rtd
: Widget name of Rtd.

PUBLIC METHODS

cancel {{val 1}}
: Cancel test procedure.

colors {}
: Color test.

cut {}

```
        Cut level window.

    cuts {}
        Spectrum window.

    etcl {cmd}
        Execute tcl command.

    fitsh {}
         FITS header.

    fliprotate {}
        Flip and rotate.

    perft {}
        Performance test.

    pick {}
        Pick window.

    pixtab {}
        Pixel table.

    rapid {}
        Rapid frame.

    record {}
        Recorder test.

    testall {}
        Complete test.

    testcancel {}
        Check if test was cancelled.

    zoom {}
        Zoom test.
```

PROTECTED METHODS

```
    init {}
        This method is called after the constructors have completed.

    make_buttons {}
        Create actions buttons.
```

PROTECTED VARIABLES

```
    cancel_
        Action cancelled flag.

    cmds_
        List of available commands.

    image
        Widget name of Rtd main Image.

    rtd_fd
        File descriptor returned by connect_to_rtd.

    verbose_
        Verbose mode.
```

SEE ALSO

TopLevelWidget(n)

- - - - - -

Last change: 07 May 99

4.2.37 RtdServerTool(n)

NAME

RtdServerTool - class for controlling the realtime image server

NAMESPACE

rtd

PARENT CLASS

util::TopLevelWidget

SYNOPSIS

RtdServerTool <path> ?options?

DESCRIPTION

The RtdServerTool [incr Tk] widget class, a subclass of TopLevelWidget, is used to manage a simulation of real-time image updates by communicating with the rtdServer and offering a Tcl level interface to it. A toplevel window is created with widgets for loading a FITS file for the simulation, setting a timer for image updates and starting and stopping the simulation.

Note: the simulation done with this class does not work for rapid frames.

WIDGET OPTIONS

-fits_file
: FITS file to use for the simulation.

-rtd
: Widget name of Rtd.

PUBLIC METHODS

close {}
: Stop all actions and delete the window.

etcl {cmd}
: Send the given command to rtd to be evaluated and return the result.

loadImage {{errdiag 1}}
: Specify an image to use for the simulation.

lock {}
: This method is called when the "Lock" button is pressed.

rapidFrame {id x y w h}
: Initialize or modify a rapid frame. The args are: the frame id (a number), the x,y coords of the frame and the width and height.

removeRapidFrame {id}
: Remove the rapid frame given by the id.

simStart {}
: Start the simulation.

```
        simStop {}
               Stop the simulation.

        timerSet {}
               Set the simulation timer.
```

PROTECTED METHODS

```
        chooseFile {}
               Display the a file browser and get the name of a FITS file to
               load.

        init {}
               This method is called after the constructors have completed.

        readInput {}
               This method is called whenever there is data to read on the
               rtdServer socket.  This is used to determine when the rtdServer
               has died.
```

PROTECTED VARIABLES

```
        readFd
               Read file descriptor for rtdServer.

        rtd_fd
               File descriptor returned by connect_to_rtd.

        writeFd
               Write file descriptor for rtdServer.
```

SEE ALSO

```
        TopLevelWidget(n)
```

```
- - - - - -
Last change: 07 May 99
```

5 Installation[1]

5.1 Before you build the RTD software

Make sure you have a proper Itcl-2.2 distribution (Tcl, Tk, BLT, TclX and ITCL extensions) with the necessary patches applied.

The Rtd requires the following software to be already installed (not included):

- itcl-2.2 - [Incr Tcl] (includes tcl7.6, tk4.2)
- BLT-2.1 - BLT Toolkit
- tclX-7.6.0 - Extended Tcl

These packages are available from the TCL archives.

See: `http://www.tcltk.com/` for Itcl and `http://www.NeoSoft.com/tcl/` for TclX and other contributed Tcl software

You can also get a copy of the whole Tcl/Tk source tree, with the patches already applied from `ftp://ftp.archive.eso.org/pub/skycat/`.

In addition, rtd requires the following packages, which are available from the URL above:

- tclutil - Tcl and C++ Utilities Package
- astrotcl - Astronomical Tcl and C++ Utilities

Note: The dependency on these two packages is hidden from applications that use the rtd library. In the installation, the rtd library contains all of the tclutil and astrotcl object files and the rtd include directory contains all of the necessary include files (for backward compatibility). The global auto_path Tcl variable is also updated automatically to include the necessary Tcl source directories.

5.2 Build the RTD Software

To make the RTD software, configure and make as follows:

```
configure
make all
make install
```

The default install dir is /usr/local. You can specify the -prefix argument to configure to change this:

- `configure -prefix $INSTALLDIR`

As an alternative, if the environment variable TCLTK_ROOT is set, it is used as the default top level directory fo Tcl/Tk.

> **Note:** The rtd configure script read files produced by the *tclutil* and *astrotcl* package configure scripts (tclutilConfig.sh and astrotclConfig.sh) to determine most compiler and shared library options and Tcl package path names. If you want to use a different compiler or shared library option, you need to reconfigure the *tclutil* and *astrotcl* packages first.

The default compilers used are g++ and gcc. If you wish to use another compiler, such as CC and cc, you can do something like this:

- `setenv CC cc`
- `setenv CXX CC`
- `configure -prefix $INSTALLDIR --with-cc`

If you prefer using shared libraries and loadable Tcl modules (see 5.5) configure with:

- `configure --enable-shared`

1. see file **`rtd/INSTALL`** for the latest updates.

Note that if you are using g++, you must also have libg++ compiled as a shared library for this to work (libg++-2.7.2 also has the "--enable-shared" option) (This requires gcc-2.7.2.1 or newer on HP-UX).

5.3 VLT Make Procedure

As an alternative to running configure and make, you can also do this:

- `cd src`
- `make all`
- `make install`

The Makefile in the $RTD/src directory runs configure and then make as described above. You can also specify options to that Makefile, for example:

- `cd src`
- `make PREFIX=$INSTALLDIR CONFIGURE_FLAGS=--with-gcc`

The PREFIX variable defaults to /usr/local and is the prefix of the directory in which to install the software. As an alternative, if the environment variable TCLTK_ROOT is set, it is used as the default top level directory fo Tcl/Tk.

5.4 Start the demo application

To run the demo:

- `cd INSTALLDIR/lib/rtd/demos`
- `./rtd`

or

- `./rtd -file FITSFILE`

Where FITSFILE is any fits format image file (look in ../images for some examples).

5.5 If you are using shared libraries

The Rtd shared library librtd.sl (in HPs) or librtd.so (on Suns) is built with the same options used to build the Tcl shared library. The options are read from the file tclConfig.sh, which is searched for in the following places:

- $prefix/lib/itcl # $prefix is the value of the -prefix
- $prefix/lib # configure option, default: /usr/local
- $TCLTK_ROOT/lib/itcl
- $TCLTK_ROOT/lib
- /usr/local/lib/itcl
- /usr/local/lib

This assumes that you have built and installed Tcl with the same "--enable-shared" and compiler option used for Rtd.

You may need to modify the SHLIB_PATH (HP) or LD_LIBRARY_PATH (Sun) environment variable so that the necessary shared libraries are found at run time. Both variables have the same format: a colon ":" separated list of directories to search for shared libraries.

From a tcl script you can load the RTD library dynamically with the command "load <path>/librtd.sl" or "load <path>/librtd.so" or it can be loaded automatically as a package. See the Tcl man pages for more information.

You can check with the program below if the shared library is loadable. If librtd.sl on HP-UX is not properly built you get the misleading error message "Not enough memory".

```
#include <dl.h>
#include <errno.h>
```

```
#include <stdio.h>
main(int argc, char *argv[])
{
  shl_t handle;
  handle = shl_load(argv[1], BIND_IMMEDIATE | BIND_VERBOSE, 0L);
  if (handle == 0)
    {
    printf(“shl_load failed %s\n”, argv[1]);
    perror(“”);
    }
  else
    printf(“shl_load ok\n”);
}
```

Appendix A: Multicasting of Images to Remote Sites

Note that the RMP package is currently not yet available!

In this appendix, we discuss the multicasting of images to remote clients. This is an extension to the "local" RTD functionality: the applications and widgets detailed below are not an integral part of the RTD and are not required by the RTD in normal operation.

In short, the multicasting functionality allows users to receive images from designated remote image servers. A multicast server process can be started on any machine on which a CCD is creating image data. Client processes can browse for any available servers and connect to the required remote camera, after which images will be received at time intervals governed by the server process operator. Several options and formats exist for the image transmission; these are discussed in more detail below.

The multicasting applications are contained in the RTD release but they do not have to be installed. See the installation notes for more information.

A.1 Overview of the Multicast Scheme

We describe here the overall scheme that was used to implement the eavesdrop functionality. In this discussion, the multicast server will be known as the "eavesdrop server", to distinguish this from the local server daemon (rtdServer).

In order to retain decoupling between processes which send and receive image data, the following scheme for the multicasting ("remote eavesdropping") has been adopted. The eavesdrop server process exists as an entirely separate process to any RTD which happens to be running, but acts as an effective RTD by connecting to the server daemon and receiving images from any active CCD in the usual manner. In the absence of any eavesdrop clients, the server will then simply discard all images that it receives (taking care to service any semaphore included in the image information structure so that the CCD software does not lock up).

When an eavesdrop client is started, the user may connect to the eavesdrop server through a server browse scheme. On connection, the eavesdrop server will take the next image event from the local server and send this to the client (having first processed the data according to the options set on the eavesdrop server panel). The client will read the data into its own local buffer before sending the image information to its own local server daemon. Thus the client acts as an effective CCD for all RTDs that are local to the client machine. This situation is illustrated in the figure below. Note that many clients may connect to a single eavesdrop server.

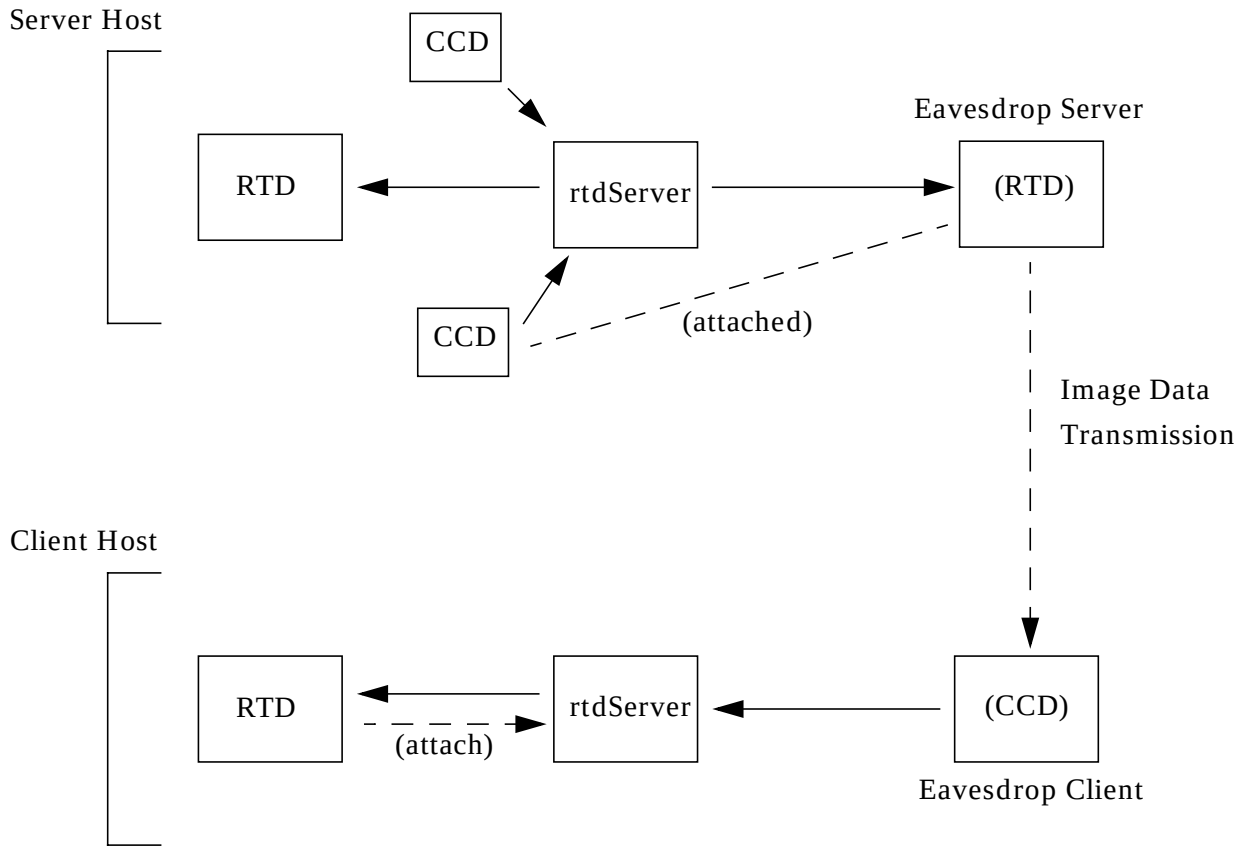


Fig. 1: The Eavesdropping Scheme

A.2 Initial Configuration

Before the multicast facility can be used effectively, a couple of minor configuration chores must be carried out.

The reliable multicast protocol operates through the creation of “RMP groups”. These are basically class D IP addresses which are used by RMP as host references to collect hosts together into a token ring. For example, the IP address “rtd.eavesdrop.domain” may be used to refer to a particular set of remote RMP hosts which may communicate as a token ring.

In the remote eavesdropping scheme, RMP group names have been mapped to unique CCD names, such that each camera has a corresponding group name. In order to be meaningful, this mapping has to be adopted consistently and across the entire system; i.e. each eavesdrop server and eavesdrop client must be aware of the correspondence between RMP group names and CCDs in order that the correct camera can be connected to by any client. This information is held as a resource file, `.rmp_config`, which must be stored in the home directory of the server operator and any client users. An example of such a file is shown below:

```
test.domain1->CAMERA1

rtd.eavesdrop->RTDSIMULATOR
```

The syntax shown here must be adhered to, as this file is read by a Tcl script. The name on the left of the arrow (->) symbol is the RMP group name for the CCD name on the right. The simulator will always be assigned the RMP group “rtd.eavesdrop”, even if there is no entry in the configuration

file, and so this name should otherwise be avoided.

A sample configuration file is contained in the RMP distribution.

In addition, the server operator is responsible for which clients are allowed to connect to the eavesdrop server. A file called `.rmp_security` should be held in the home directory of the server operator containing a list of allowed client IP addresses (not RMP addresses) or client names. An example of such a file is shown below:

```
# This is a comment
# Here are some missing lines.

rlshp2
130.246.32.3
130.246.32.2
130.246.32.36
rlspc2
```

Comments are preceded by a "#" character. Blank lines in the security file are allowed. The numerical IP addresses listed in this file are the addresses of clients that have permission to connect to and receive data from the server. In the absence of a security file, there will be no restrictions on client connections.

The security file can be edited while a server process is active, and can be accessed from the server application front end. See below for more information.

A.3 The Eavesdrop Server Application

The eavesdrop server handles receipt of images from the local rtdServer, image processing prior to the transmission of image data (this may include compression or conversion to X colour data), transmission of data to active clients using RMP, as well as interaction with the eavesdrop server operator and acknowledgement of clients.

To run the multicast server, go to the INSTALL_DIR / lib / rtd / demos directory and type rtdmcserv.

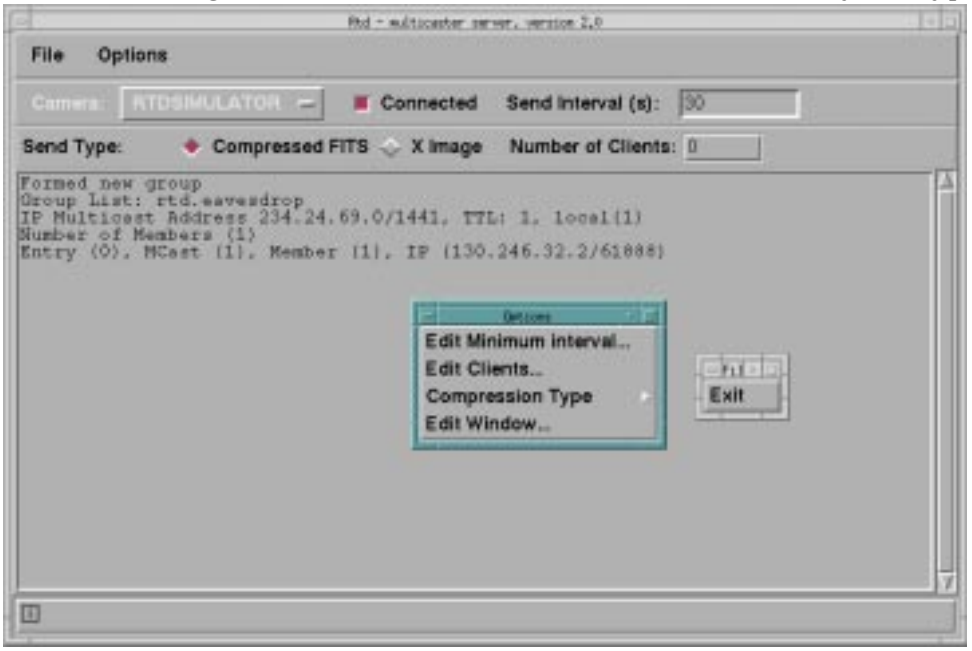


Fig. 2: The Eavesdrop Server (RtdRMPServer widget)

In order for the eavesdrop server to become "visible" to remote clients, it must first be connected to

a local CCD. To do this, select the required camera in the LabelMenu (this list is formed from the configuration file discussed in the previous section) and invoke the “Connect” toggle button to the side of this. After a short time, a message will appear on the server transcript window to the effect that a new RMP group has been created and the camera LabelMenu will become inactive. Disconnection from a camera is possible by reinvoking the “Connect” button, after which the label menu will become active again.

Images will be received from the connected camera when remote clients connect to the eavesdrop server. Diagnostic messages will appear on the transcript window as clients connect and disconnect to the server and images are transmitted.

A.3.1 Eavesdrop Server Options

The following options for the server operation are available from the dialogue itself:

- *Send type* - allows the operator to determine whether the images should be transmitted as compressed FITS images or colour scaled X data. The use of the latter case, in which 8-bit data is used instead of the data’s previous 16- or 32-bit format may be useful when large images are transmitted, although some versatility is lost from the point of view of the client as the number of colours in the image is predetermined by the server.
- *Send interval* - allows the operator to enter the amount of time between image transmissions above a minimum which is also variable (see below). The actual time between send events is determined by the availability of images at the required transmission time, and also by the fact that the server can not transmit an image while another is being sent.

The following options are available from the eavesdrop server menu:

- The *File* menu has only one menu item, *Exit*. This closes the application down.
- The *Options* menu has four menu items:
  - *Edit Minimum Time...* allows the operator to change the minimum allowed time between image transmissions. The user will not be allowed to change the send interval to be smaller than this figure. The menu item brings up a simple entry dialogue to allow the operator to enter the new minimum time.
  - *Edit Clients...* allows the editing of the security file `~/.rmp_security`. It simply runs a session of EMACS on the file. If EMACS is not present on the local system, an error message is produced. Concurrent (i.e. while the server is running) editing of the security file is still possible using a different editor.
  - *Compression Type* brings up another cascaded menu containing a list of possible compression types: UNIX compression, GNU compression, or NO compression. These options have an effect only if the send type is set to “Compressed FITS” on the front of the dialogue. It has been observed that the GNU compression gives slightly better compression than the UNIX version, although it may take significantly longer to execute. The NO compression option is useful over fast networks with small-medium sized images, where compression becomes less important.
  - *Edit Window* is considered separately below.

A.3.2 Transmission of Image Sub-Regions

It is possible to send only a portion of an image to the eavesdrop clients. This feature can be accessed from the *Options* | *Edit Window* menu item. It is useful in situations where very large images are produced which have relatively small fields of interest.

When the menu option is invoked, the following dialogue is produced.

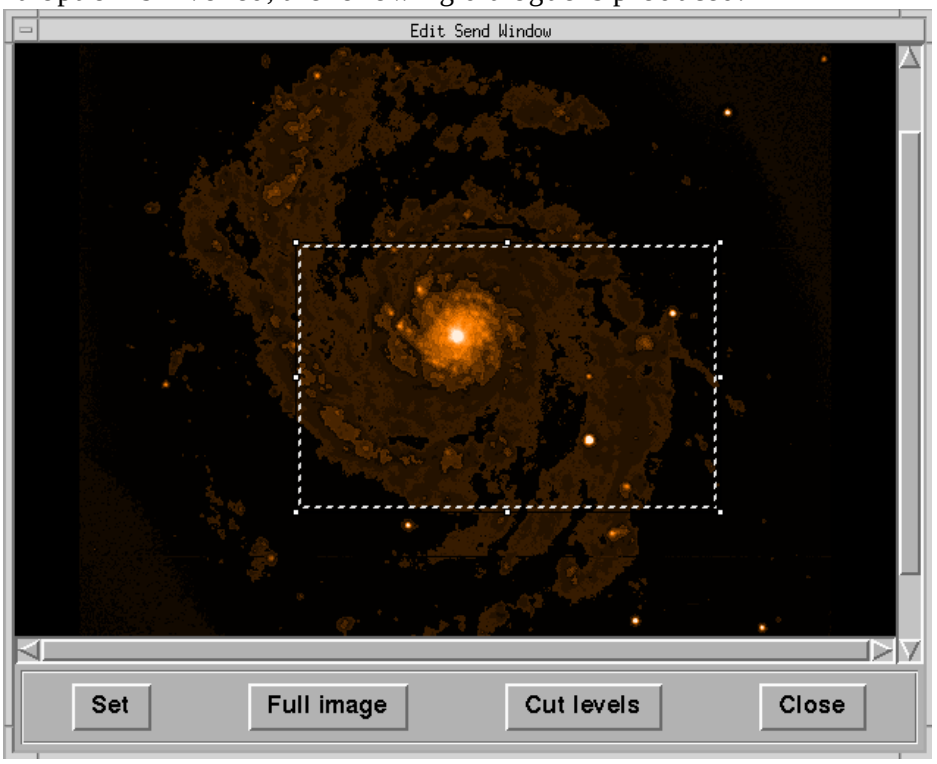


Fig. 3: The Edit Window dialogue (RtdRMPEdit widget)

The dialogue consists of an rtdimage widget (with scrollbars) and four buttons: *Set*, *Full Image*, *Cut Levels*, and *Close*. When it is invoked for the first time, the canvas is empty and the *Set* button is not active. On receipt of an image, the *Set* button becomes active and the cursor inside the canvas turns into a mouse representation, inviting the user to select a rectangle in the canvas for sending (as shown). The rectangle, once produced, can be moved or resized (as with rapid frames), and only when the *Set* button is invoked are the coordinates of the sub-region stored. The chosen coordinates are then displayed on the server transcript window.

To remove the sub-region (and continue sending the full image) invoke the *Full Image* button. A message will be written to the server to show that this is being done.

The *Cut Levels* button brings up the usual cut levels dialogue (RtdImageCut). This allows the user to change the cut levels in the edit window. This has no effect on the image send parameters.

The *Close* button closes the edit window. Any sub-region produced prior to this will be retained.

A.4 The Eavesdrop Client Application

As discussed above, the eavesdrop client acts as an effective CCD for any RTDs running on that machine, receiving images from the eavesdrop server and forwarding these to the local server daemon. The name of the client ‘CCD’ is RTDEAVESDROP; any RTD that wishes to receive images from the eavesdrop client must first connect to this camera name.

As with the server, the configuration file `~/.rmp_config` must be installed. The camera and RMP group names listed here must be consistent with those used across the entire system.

To start the eavesdrop client application, go to the `INSTALL_DIR/lib/rtd/demos` directory and

type `rtdmcclnt`. The following dialogue is produced.

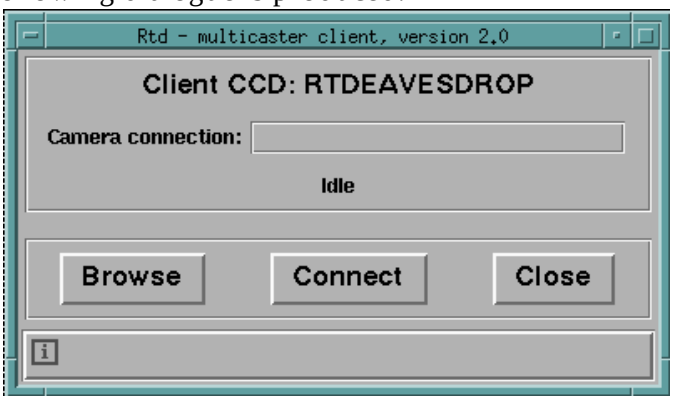


Fig. 4: The Eavesdrop Client (RtdRMPClient widget)

From the top down: the title of the dialogue is a reminder to local RTD users of the name of the eavesdrop CCD. The camera connection below this contains the name of the remote camera to which the client is connected (when it is connected). The line below that is information on the current status of the application.

The *Browse* button activates the client browse scheme. The application considers each camera listed in the `~/.rmp_config` file, and attempts to locate a remote server which is connected to this camera. This process may take some time. On completion, a list of active server connections is displayed, from which the user should select one (if there are no active servers, a message is produced to say this). The name of the camera selected is written to the “Camera connection” field.

The *Connect* button forms a link to the camera displayed in the Camera connection field. The status display updates to inform the user that the connection has been made.

The *Close* button disconnects from the remote camera and shuts the application down.

Disconnection from a remote camera is achieved either by exiting the application, or connecting to a new camera following a Browse. If RTDs wish to stop receiving images, they should disconnect themselves from the eavesdrop CCD.

A.5 Implementation Notes

This is a very brief summary of the implementation of the eavesdropping system. More information can be found in the reference section. In particular, see RtdRMPServer(n/3), RtdRMPClient(n/3), rtdmcclnt(1), rtdmcserv(1), rtdrmpclient(n), rtdrmpserver(n), and RtdRMPEdit(n).

The implementation is based around the C++ classes RtdRMP, RtdRMPServer, and RtdRMPClient, which in turn implement the rtdrmpserver and rtdrmpclient Tk widget classes. The class hierarchy for the C++ classes is shown below:

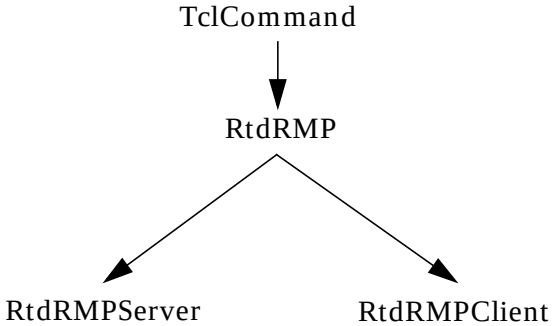


Fig. 5: The Class Hierarchy for the RMP Objects

A.5.1 The RtdRMP Base Class

The base class is abstract. It contains basic RMP functionality that must be inherited by any class that uses RMP (e.g. the ability to create/join RMP groups and transmit data). It also defines virtual functions for general IO that must be overloaded by the sub-classes. The most important methods are as follows:

- `join()` - joins the RMP group specified by the `groupName_` property,
- `group_send(char *, int)` - send the buffer (of given length) to the current group members,
- `detect(RMP *)` (virtual) - detection loop to get RMP events and process them as required,
- `handle_event(RMPEvent *)` (virtual) - event handler called by detection loop. Contains code for every possible RMP event, and acts according to the event type.
- `informUser(char *)(virtual)` - output a message to the user.

A.5.2 The RtdRMPServer Class

The RtdRMPServer class overrides the above virtual methods, as well as providing many methods for dealing with server-specific processing. The server spawns a child process in order to carry out image processing prior to sending the image. This is so that RMP events may continue to be processed while the potentially rather long image compression takes place.

The child and parent processes are synchronised as follows: the server forks, with the parent process returning immediately to the RMP detect loop. The child process performs compression tasks, and on completion places the result into a separate buffer of shared memory before raising a semaphore. The child process then exits. The parent continually polls the semaphore, and when it detects that the child has completed sends the shared memory buffer to the clients. The semaphore is then decremented.

The RtdRMPServer class also implements the methods of the rtdrmpserver Tk widget, and as such overrides the TclCommand `call()` method. See TclCommand(3) and rtdrmpserver(n) for more information.

The following are the most important RtdRMPServer methods:

- `get_image()` - called when the socket connection from the rtdServer becomes readable, this passes control to the send routines if it is time for an image to be sent,
- `send_image(rtdIMAGE_INFO *)` - drives the main image processing and forking of the parent,
- `checkShm()` - called by the parent to check the semaphore state, and send the data if it exists,
- `attach_server()` - attach the eavesdrop server to the local server daemon to receive images,
- `processImageData(rtdIMAGE_INFO *)` - return a Mem object for the buffer that is to be compressed. This routine also deals with extraction of image sub-regions, if this is required,
- `authenticateIPAddress(char *)` - authenticate whether a given IP address is an allowed client,
- `killClient(char *)` - send a message to a client address to shut the client down,
- `check_members(RMP_BOOL)` - check the current group membership list against the allowed membership list,
- .`processServerMessage(RMPEvent *)` - process a message sent from a client to the server.

A.5.3 The RtdRMPClient Class

The RtdRMPClient object overrides some of the RMP handling routines of RtdRMP as well as adding specific methods relating to the receipt and forwarding of image data. It also implements the methods of the rtdrmpclient Tk widget. See rtdrmpclient(n) for more information on these.

Note that the unpacking of the data may also be a time-consuming process, and so the client forks to allow the parent process to continue processing RMP events while the image data is uncompressed. The mechanism used is slightly different to the server: the child process processes the data and puts it into shared memory before sending the image information to the rtdServer. It then raises a semaphore and exits. The parent process detects that the data has been forwarded and so flags that the client is ready to receive another image from the eavesdrop server.

The following are the most important of the RtdRMPClient methods:

- `processData(RMPEvent *)` - called from the event handler when a message event is received, this routine calls a processing routine based on the contents of the message,
- `writeRawData(RMPEvent *)` - called from processData() in the case that the message is made up of image data, this routine puts the incoming data into a local buffer,
- `writeShmData(char *)` - unpacks and copies the image into shared memory ready for forwarding,
- `createRtdPacket()` - creates the image information and forwards this to the rtdServer,
- `checkStatus()` - called by the parent to detect whether the child process has completed,