NAME

xgterm – terminal emulator for X with graphics and imaging capability

SYNOPSIS

xgterm [–*toolkitoption* ...] [–*option* ...]

DESCRIPTION

The *xgterm* program is a terminal emulator for the X Window System based largely on *xterm* but with a completely new graphics and imaging widget. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. *XGterm* also serves as a prototype for the *Widget Server* being developed by the IRAF Project at NOAO. The *Object Manager Library* it uses implements a window system toolkit as an interpreted window-object language, allowing application GUIs to be defined and executed at runtime without compiling any code, and with minimal dependence upon the underlying window system toolkit library. We will concentrate here, however, on it's use as a terminal emulator and a description of the new *Gterm* widget.

The Gterm graphics window operates almost identically to the *xterm* Tek window, however there are extensions for implementing full-screen cursors, imaging, area fills, colors, graphics erasure, a "status line" and so on. Any graphics application capable of running under an *xterm* Tek window should also be able to use *xgterm* as well. Client programs wishing to make use of the extended features, or those wishing to implement a GUI, are advised to use the OBM (*Object Manager*) library supplied with the XGterm source as part of the X11IRAF package. This provides a much better programmatic interface to all of the features available; however, as of this writing it is not yet fully documented. Users are referred to the *XImtool* task as an example of a more complex application using the *OBM Library* and *Gterm* widget, as well as demo tasks in the *guidemo* directory of the X11IRAF sources. Questions or comments may also be sent to *iraf@noao.edu*.

The VT102 text window is unchanged from the original *xterm* application. All of it's resources, command-line options and operation are identical to that used by *xterm*. The *termcap(5)* entry for *xterm* may be used for *xgterm* as well. See the *xterm(1)* man page for details.

OPTIONS

All *xterm(1)* and X Toolkit command line options are supported, there are no additional options.

RESOURCES

The program understands all of the core X Toolkit resource names and classes, all text window resources known to *xterm(1)*, as well as the *Gterm* (graphics and imaging widget) resources. The proper Class name for all resources described here is *Gterm*. A table of available *Gterm* resources and their defaults may be found below, some of the more interesting resources are described here in detail:

basePixel

Base cell of the custom colormap. This essentially allows you to reserve *basePixel* colors in the global colormap for other applications. The default is 38, if changed you'll need to also enable the *cmapInitialize* resource to force the Gterm widget to update it's global colormap resource in the X server.

cmapInitialize

Initialize the ximtool colormap at startup. When resetting the *basePixel* resource or colormap this is required in order to force the Gterm widget to update it's global colormap resource in the X server. The default is False.

cmapInterpolate

Interpolate the colormap to the number of display colors. The default is True.

cmapName

Name used for private colormap. The default for all IRAF imaging applications is *image*. Gterm widget based imaging applications which have the same value of cmapName will share the same colormap, minimizing colormap flashing and allowing multiple applications to be run at the same time.

color0
The widget background color. The default is black.

color1
The widget foreground color. The default is white.

color2 thru *color9*
Optional drawing colors. The line color used for graphics is set using an escape sequence to select the current color index. See *Gterm I/O Escape Sequences* below for more details.

crosshairCursorColor
Color of the full screen crosshair cursor.

defaultMarker
Default marker type. Options include *text*, *line*, *polyline*, *rectangle*, *box*, *circle*, *ellipse*, and *polygon*. The default is *rectangle*.

deiconifyWindow
De-iconify the Gterm graphics window when activated. The default is False.

dialogBgColor
Dialog box (i.e. status line) background color. Dialog text is text which is drawn into the dialog area at the bottom of the gterm window, it is transient and is not a permanent part of the graphics being drawn. Dialog text is normally used to interact with the user or to display messages during program operation, without affecting the graphics being drawn.

dialogFgColor
Dialog box (i.e. status line) foreground color.

ginmodeBlinkInterval
Graphics cursor blink interval, time is specified in milliseconds. The default is 0.

ginmodeCursor
Graphics mode cursor type. The default is a full screen cursor custom to the widget.

height
Height of the Gterm window. The default is 480.

idleCursor
Cursor to use when not in graphics mode.

markerHighlightColor
Highlight color for the active marker. When the pointer moves into a marker is it marked "active", the highlight color and width change to which marker is active. The default is green.

markerHighlightWidth
Highlight width for the active marker. The default is 2.

maxColors
The maximum number of colors to use in the private global colormap, the default is 216. Out of this number 10 colors (the *color0* theu *color9* values) are reserved by the widget as static colors, the remainder may be allocated for images.

raiseWindow
Raise the window when active. The default is False.

warpCursor
Warp the cursor to the window when active. The default is False.

width
Width of the Gterm window. The default is 640.

GTERM WIDGET RESOURCES

Class Hierarchy

Core -> Gterm

Resources

When crating a Gterm widget instance, the following resources are retrieved from the arguments list or from the resource database:

Name	Class Type	Default	Description
alphaFont1	XFontStruct	nil2	Graphics fonts
alphaFont2	XFontStruct	5x8	"
alphaFont3	XFontStruct	6x10	"
alphaFont4	XFontStruct	7x13	"
alphaFont5	XFontStruct	8x13	"
alphaFont6	XFontStruct	9x15	"
alphaFont7	XFontStruct	9x15	"
alphaFont8	XFontStruct	9x15	"
basePixel	Int	38	Base of private global colormap
busyCursor	String	watch	Cursor to use when application is busy
busyCursorBgColor	Foreground	white	Busy cursor background color
busyCursorFgColor	Foreground	black	Busy cursor foreground color
cacheRasters	String	whenNeeded	Save rasters as server pixmaps for faster access
cmapInitialize	Boolean	False	Initialize colormap at startup
cmapInterpolate	Boolean	True	Interpolate colormap
cmapName	String	default	Custom colormap name
cmapShadow	Int	10	Colormap shadow interval
cmapUpdate	Int	60	Colormap update interval
color0	Background	black	Default graphics background color
color1	Foreground	white	Default graphics foreground color
color2	Foreground	red	Optional drawing color
color3	Foreground	green	"
color4	Foreground	blue	"
color5	Foreground	cyan	"
color6	Foreground	yellow	"
color7	Foreground	magenta	"
color8	Foreground	purple	"
color9	Foreground	darkslategray	"
copyOnResize	Boolean	True	Copy raster when resized
crosshairCursorColor	Foreground	red	Full-screen cursor color
defaultMarker	String	rectangle	Default marker type
deiconifyWindow	Boolean	False	Deiconify window when active
dialogBgColor	Foreground	yellow	Status line background color
dialogFgColor	Foreground	black	Status line foreground color
dialogFont1	XFontStruct	nil2	Status line fonts
dialogFont2	XFontStruct	5x8	"
dialogFont3	XFontStruct	6x10	"
dialogFont4	XFontStruct	7x13	"
dialogFont5	XFontStruct	8x13	"
dialogFont6	XFontStruct	9x15	"
dialogFont7	XFontStruct	9x15	"
dialogFont8	XFontStruct	9x15	"
ginmodeBlinkInterval	Int	0	Graphics cursor blink interval
ginmodeCursor	String	full_crosshair	Graphics cursor type
ginmodeCursorBgColor	Foreground	black	Graphics cursor background color

ginmodeCursorFgColor	Foreground	white	Graphics cursor foreground color
height	Dimension	480	Height of graphics window
idleCursor	String	Plus	Idle cursor type
idleCursorBgColor	Foreground	white	Idle cursor background color
idleCursorFgColor	Foreground	black	Idle cursor foreground color
markerBoxKnotColor	Foreground	blue	Vertex knot color
markerBoxKnotSize	Int	0	Vertex knot size
markerBoxLineColor	Foreground	green	Marker border color
markerCircleKnotColor	Foreground	blue	Vertex knot color
markerCircleKnotSize	Int	0	Vertex knot size
markerCircleLineColor	Foreground	green	Marker border color
markerCursorBgColor	Foreground	black	Cursor background when in marker
markerCursorFgColor	Foreground	yellow	Cursor foreground when in marker
markerEllipseKnotColor	Foreground	blue	Vertex knot color
markerEllipseKnotSize	Int	0	Vertex knot size
markerEllipseLineColor	Foreground	green	Marker border color
markerFill	Boolean	False	Flood fill marker area with *markerFillColor*
markerFillBgColor	Foreground	black	Fill area background color
markerFillColor	Foreground	slategray	Flood fill color
markerFillStyle	Int	FillSolid	Fill area style
markerHighlightColor	Foreground	green	Marker highlight color
markerHighlightWidth	Int	2	Marker highlight line width
markerLineKnotColor	Foreground	blue	Vertex knot color
markerLineKnotSize	Int	5	Vertex knot size
markerLineLineColor	Foreground	green	Line marker color
markerLineStyle	Int	LineSolid	Line marker line style
markerLineWidth	Int	1	Line marker width
markerPgonKnotColor	Foreground	blue	Vertex knot color
markerPgonKnotSize	Int	5	Vertex knot size
markerPgonLineColor	Foreground	green	Marker border color
markerRectKnotColor	Foreground	blue	Vertex knot color
markerRectKnotSize	Int	0	Vertex knot size
markerRectLineColor	Foreground	green	Marker border color
markerTextBgColor	Foreground	slategray	Text marker background color
markerTextBorder	Int	2	Text marker border width
markerTextColor	Foreground	yellow	Text marker text color
markerTextFont	XFontStruct	6x13	Text marker font
markerTextLineColor	Foreground	green	Text marker line color
markerTextString	String	NULL	Text string
markerTranslations	String	default	Marker event-to-actions translations
maxColors	Int	216	Max colors in custom colormap
maxMappings	Int	32	Max image mappings
maxRasters	Int	512	Max image rasters
nearEdge	Int	1	Distance, in pixels, between pointer and marker edge required for translation actions for be in effect.
nearVertex	Int	4	Distance, in pixels between pointer and marker vertex (i.e. *knot*) required for translation actions for be in effect.
raiseWindow	Boolean	False	Raise window when active
translations	String	default	Event-to-actions translations
useTimers	Boolean	True	Ok to use timers
warpCursor	Boolean	False	Enable warp cursor when active

width	Dimension	640	Height of graphics window
xorFill	Boolean	False	Fill with GXxor
xorFillBgColor	Int	255	Xor-fill background color
xorFillColor	Int	2	Xor-fill color

GTERM WIDGET TRANSLATIONS AND ACTIONS

The default translations for a Gterm window are:

<Btn1Down>:	m_create()
<Btn2Down>:	crosshair(on)
<Btn2Motion>:	crosshair(on)
<Btn2Up>:	crosshair(off)
<EnterWindow>:	enter-window()
<LeaveWindow>:	leave-window()
<KeyPress>:	graphics-input()
<Motion>:	track-cursor()

The available action procedures for a Gterm window are:

ignore() Ignore an event.

graphics-input() Handle a graphics input request.

crosshair(on|off) Display a crosshair cursor.

track-cursor() Track crosshair cursor position.

enter-window() Handle an EnterWindow event.

leave-window() Handle an LeaveWindow event.

reset() Do a soft reset of the Gterm widget.

m_create() Create a new marker. Valid types include

text	*line*	*polyline*	*rectangle*
box	*circle*	*ellipse*	*polygon*

The default is *rectangle*, if no type is given the default type specified by the *markerType* resource will be used.

GTERM MARKER TRANSLATIONS AND ACTIONS

The default translations for a marker are:

!Shift <Btn1Motion>:	m_rotateResize()
<Btn1Motion>:	m_moveResize()
!Shift <Btn1Down>:	m_raise() m_markpos()
<Btn1Down>:	m_raise() m_markposAdd()
<Btn1Up>:	m_redraw() m_destroyNull()
<Btn2Down>:	m_lower()
<Key>BackSpace:	m_deleteDestroy()
<Key>Delete:	m_deleteDestroy()
<KeyPress>:	m_input()
<Motion>:	track-cursor()

Translations affect only the currently active marker, the cursor must be within *nearEdge* pixels of a marker edge, or *nearVertex* pixels of a marker vertex to take effect.

The available action procedures for a marker are

m_create(type) Create a new marker. Valid types include

text	*line*	*polyline*	*rectangle*

box *circle* *ellipse* *polygon*

The default is *rectangle*, if no type is given the default type specified by the *markerType* resource will be used.

m_destroy() Destroy the active marker.

m_destroyNull() Destroy the active marker if it is null sized.

m_set(attribute, value, ...)
Set a marker attribute. Valid attributes include

activated	*autoRedraw*	*fill*	*fillBgColor*
fillColor	*fillPattern*	*fillStyle*	*font*
height	*highlightColor*	*imageText*	*knotColor*
knotSize	*lineColor*	*lineStyle*	*lineWidth*
rotangle	*sensitive*	*textBgColor*	*textBorder*
textColor	*translations*	*type*	*visible*
width	*x*	*y*	

m_raise() Raise the active marker to the top of the display list.

m_lower() Lower the active marker to the bottom of the display list.

m_notify(event, event, ...)
Notify any clients that have registered callbacks for the specified type of events. Recognized events include

notify	*moveResize*	*modify*
redraw	*destroy*	*input*
focusIn	*focusOut*	*constraint*

m_input() Notify any clients that have registered a input callback that a input event has occurred.

m_markpos() Mark the current position of the marker, e.g., so that it can later be erased.

m_markposAdd() Execute either the markpos or add action, depending upon the pointer location. If the pointer is over an active marker at a location where the add action can be executed this is done, otherwise the markpos action is executed.

m_redraw() Redraw the active marker.

m_addPt() Add a point (i.e. vertex knot). *Polyline* and *polygon* markers only.

m_deletePt() Delete a point (i.e. vertex knot).

m_movePt() Move a point (i.e. vertex knot). *Polyline* and *polygon* markers only.

m_deleteDestroy() Delete a point or destroy a marker, depending upon the pointer position.

m_move() Move a marker.

m_resize() Resize a marker.

m_moveResize() Move a point or marker, or resize a marker, depending upon the pointer position.

m_rotate() Rotate a marker.

m_rotateResize() Rotate or resize a marker. A marker is rotated if near a vertex know, or resized if near an edge.

GTERM I/O ESCAPE SEQUENCES

XGterm uses escape sequences to provide graphics emulation. This protocol is an extension of the Tektronix 4012 graphics protocol. The basic extensions are patterned after the Retrographics VT640 graphics terminal, using GS (octal \035, aka Ctrl-]) and CAN (octal \030, aka Ctrl-x) to switch between vt100 and graphics modes. Additional extensions are defined to support advanced features such as color, area fills, graphics erasure, setting the cursor location under program control, interactive dialog via the "status line", and so on.

While these escape sequences can be used directly, the best programmatic interface is to use the OBM (*Object Manager*) library supplied with the XGterm source as part of the X11IRAF package. Any Tektronix-compatible graphics library will suffice for producing vector graphics, the added escape sequences used by the Gterm widget are required to make use of imaging, area fills, the status line, etc.

All escape sequences begin with an **ESC** character (octal \033), followed by up to three characters defining the action to be taken. All strings in capital letters refer to the ASCII code (e.g. *LF* is the ASCII linefeed code), a three digit number preceeded by a '´ refers to an octal code (e.g. " 12" is octal 12) , all others are characters in the escape code (e.g. "/bc" are the three characters '/', 'b', and 'c').

ESCAPE SEQUENCES

US
CR Switch to alpha mode. Characters are drawn in the graphics window at the "current" position (normally set beforehand with a GS/US vector move), using the alpha mode font. Receipt of any control code causes alpha mode to be exited.

GS Switch to vector polyline mode.
FS Switch to vector polypoint mode.
RS Switch to vector mode, vertices are joined as a polygon.

With all three codes, vertices and points are accumulated in a buffer and displayed when the buffer fills or when vector mode is terminated by receipt of any control code. A workstation open will be done if it hasn't already been opened, no-op sequences GS-CAN are filtered out, since they would only cause a pointless switch to the graphics frame and back without drawing. The open workstation sequence is GS,US, or by the *xterm* graphics start escape sequence "[?38h".

EM Enter message mode. In message mode input text is accumulated in a buffer and eventually passed to the object manager, which delivers the message to the referenced object. Messages are used to download the user interface to be executed by the object manager, and during execution messages are used to set the values of user interface parameters to allow the UI to track the state of the client application.

CAN Close workstation and enter command mode.

BEL Ring the screen bell.

ENQ Return terminal status. Returned values include the terminal mode, and alpha cursor x and y position.

SUB Initiate a cursor read, values are returned in window coordinates.

/SUB Return window cursor position in raster coordinates.

FF Clear the screen.

/f Set current cursor position.

0	Set character size 0. (Currently ignored).
1	Set character size 1. (Currently ignored).
2	Set character size 2. (Currently ignored).
3	Set character size 3. (Currently ignored).
/0d	Set color index.
/1d	Clear graphics screen.
/2d	Invert graphics screen
‘	Select line style 0. (Solid)
a	Select line style 1. (Dashed)
b	Select line style 2. (Dotted)
c	Select line style 3. (DashDot)
d	Select line style 4. (Dash3Dot)
/0w	Select line width 0.
/1w	Select line width 1.
/2w	Select line width 2.
/nw	Select line width 3.
/0c	Select line color 0.
/1c	Select line color 1.
/2c	Select line color 2.
/3c	Select line color 3.
/4c	Select line color 4.
/5c	Select line color 5.
/6c	Select line color 6.
/7c	Select line color 7.
/8c	Select line color 8.
/9c	Select line color 9.

IMAGING ESCAPE SEQUENCES

These are encoded as follows:

ESC <code> [*P ; P* ; ...] <*data*>

where *code* is a character sequence and *P* is an ASCII encoded parameter described below.

/nc	Select line color. Parameter is the color number in the range 0-9.
sre	*Reset*. Parameters are "reset-str".
ssz	*Resize*. Parameters are "resize-str".
rir	*Initialize raster*.
rcr	*Create a raster*. Parameters are raster number, type, width, height, and depth. Type is 1 for a normal (client) raster, 2 for cached in server memory, or 0 if you don’t care. Depth may be 1, 8, 16, or 32.
rde	*Destroy a raster*. Parameter is raster number.
rqr	*Query a raster*. Parameter is raster number. Output parameters are status, type, width, height, and depth encoded in the string ""\033[5;%d;%d;%d;%d;%d]".
rsr	*Select a raster*. Parameter is raster number.
rwr	*Write pixels to a rectangular region of a raster*. Parameters are raster number, encoding type (not used), x1, y1, nx, ny, and depth followed by (nx*ny) data pixels.
rrd	*Read from a rectangular region of a raster*. Parameters are raster number, encoding type (not used), x1, y1, nx, ny, and depth followed by (nx*ny) data pixels.

rrp *Refresh raster pixels*. Parameters are raster number, coordinate type (0 for pixel, 1 for NDC), x1, y1, nx, ny.

rsp *Set all the raster pixels in a region to a single color*. Parameters are raster number, coordinate type (0 for pixel, 1 for NDC), x1, y1, nx, ny, color, and raster operand. If nx=ny=0 the entire raster will be written. Raster operands include transient (octal 020), refresh_all (octal 040), or refresh_none (octal 100).

rco *Copy a region of the source raster to a region of the destination raster*. Parameters are raster operand, source raster number, source type, source x coord, source y coord, source width, source height, destination raster number, destination type, destination x coord, destination y coord, destination width, destination height, If the input and output regions are not the same size the subimage is automatically scaled to fit the destination region. If the destination extent DNX or DNY is negative, the image is flipped in that axis. The type of spatial scaling performed is determined by the scale factors (zoom, dezoom, or no scaling). The rasterop argument is used to exercise fine control over how the mapping is performed, e.g. to force a refresh, implement a transient mapping, or in the case of a dezoom (many-to-one) mapping, select the antialiasing technique to be used.

rwc *Write a colormap*. Parameters are colormap number, first color and the number of colors followed by NC colors triples in the data.

rrc *Return the color assignments for a region of the named colormap*. Parameters are colormap number, first color and the number of colors followed by NC colors triples in the data.

rlc *Load a colormap into the display*, optionally scaling the colormap via a linear transformation in the process. Parameters are the colormap number, the offset value, and the cursor x and Y coordinates in NDC units. The colormap is unaffected if offset=0.5, scale=1.0. A negative scale inverts the image. If map=0 the linear transformation is applied directly to the display colormap.

rfc *Free a colormap*. Parameter is the colormap number.

rwo *Write the IOmap*. Parameters are the first color and the number of colors, followed by NC color triples in the data. An iomap is an optional lookup table used to isolate the client application from the color model used within the Gterm widget. To simplify color allocation the Gterm widget defines a logical color space where color 0 is the background, 1 the foreground, 2-N are statically allocated standard colors, and colors N+1 and above are dynamically allocated by the graphics application. Less-demanding applications use only the statically allocated, shared colors. The widget internally maps these logical colors to whatever the window system requires, but providing a well-defined logical color space isolates the client from the details of color allocation in the underlying window system.

An iomap can be used to define a mapping between the color model of the client application and the Gterm color model (when we say color model here we mean color allocation schemes for 8 bit pseudocolor). By default the iomap is one-to-one. The use of an iomap frees the client from having to worry about color index translations, and allows color tables to be combined in the widget for greater efficiency when color tables are serially applied. The iomap applies to all color indices or pixel values passed in i/o operations between the client and the Gterm widget.

rro *Read the IOmap*. Return values are the first color and the number of colors, followed by NC color triples in the data.

rim *Delete all mappings* and initialize the mapping subsystem.

rsm *Define a new mapping function*, or modify an old one. If a new mapping is defined it is merely enabled, and no refreshing of the screen takes place until either some mapped data is written to or the mapping is explicitly refreshed. If an existing mapping is modified the old and new mappings are examined and only those portions of the destination rect for which the mapping changed are updated. This permits minor changes to a mapping (e.g. moving an edge) without having to redraw the entire region. Regions of the destination drawable which were previously covered by the mapping but which were exposed by modifying the mapping are redrawn.

rgm *Return the external parameters of a mapping*. Parameter is the mapping number, values returned (in the string "\033[6;%d;%d %d;%d;%d;%d;%d;%d %d;%d;%d;%d;%d;%d]") are the mapping number, rasterop, source mapping, type, x, y, width, height, and destination mapping, type, x, y, width and height.

rem *Enable a mapping*. Parameters are the mapping number and an integer flag indicating whether to refresh the mapping.

rdm *Disable a mapping*. Disabling a mapping does not affect the mapping definition, hence a disabled mapping may later be reenabled. Parameters are the mapping number and an integer flag indicating whether to erase the mapping.

rrm *Refresh a mapping*. Parameter is the mapping number.

rfm *Free a mapping*. Parameter is the mapping number.

MORE ON IMAGING

The imaging model of the Gterm widget defines the following key object or data types: *rasters*, *mappings*, and *colors*.

raster A raster is a MxN array of pixels. At present pixels are 8 bits deep but hooks are built into the widget to expand this in the future. Pixel values are indices into the Gterm virtual colormap, with values starting at zero. A raster may be any size. A raster is merely a two-dimensional array in the graphics server; it is not displayed unless mapped. An exception is raster zero, which is the graphics window. Rasters are referred to by number, starting with zero. Initially only raster zero exists; new rasters are created with the create raster escape code **rcr**. Space for rasters may be allocated either in the graphics server, or in the X server. This has implications on performance but is otherwise transparent to the client. By default rasters are allocated in the graphics server, i.e., in the X client.

mapping A mapping defines a projection of a rectangle of the source raster onto a rectangle of the destination raster. Mappings may be either enabled (active) or disabled. When a mapping is enabled, any change to a pixel in the source rect will cause the corresponding pixels in the destination rect to be updated. Mappings are referred to by number starting with one. Initially no mappings are defined. If the size of the input and output rect is not the same the input rect will be scaled by pixel replication or subsampling to fill the output rect. If the argument DW (destination width) or DH (destination height) of the destination rect is negative, the image will be flipped around the corresponding axis when copied to the destination; the region of the destination drawn into is the same in either case. Multiple mappings may reference the same source or destination raster. Mappings are refreshed in order by the mapping number. Modifying a mapping causes the changed regions of the destination rect to be refreshed.

color The *Gterm* widget provides a fixed number of preassigned colors corresponding to pixel values 0 through 9. Zero is the background color, one is the foreground color, and 2-9 (8 colors) are arbitrary colors defined by Gterm widget resources. These static colors are normally used to draw the background, frame, axes, titles, etc. of a plot, or to draw color graphics within the drawing area. The advantage of static colors is that they are shared with other X clients, and the values of these colors may be assigned by the user to personalize the way plots look.

The *Gterm* widget also allows any number (up to about 200 or so) additional colors to be defined at runtime by the client application. These color values start at pixel value 10 and go up to the maximum pixel value assigned by the client. The client application allocates colors with the write colormap escape code **rwc**. Attempts to overwrite the values of the static colors are ignored. The values of already allocated colors may be changed dynamically at runtime using write colormap code to write the desired range of color values.

Applications should not assume that there are 10 static colors and 200 or so allocatable colors. The graphcap entry for the logical device in use defines these parameters for the device. Alternatively, the read colormap code may be used to dynamically determine how many colors the server has preallocated when the application starts up.

An image may use either static and dynamic pixel values or both types of values, but in most cases imaging applications involve smoothly shaded surfaces hence will require dynamically assigned private colors.

If for some reason the client application cannot use the *Gterm* widget color model, the IOMAP feature can be used to make the widget appear to have some externally defined (i.e., client defined) color model.

The maximum number of rasters and maximum number of mappings is defined by the Gterm widget resources *maxRaster* and *maxMappings* (or in the GUI file) when the graphics application starts up. The maximum values should be much larger than most applications require. Applications should allocate raster or mapping numbers sequentially starting at 1 (more or less) to avoid running out of raster or mapping descriptors.

The {read|write}pixels escape codes operate directly on raster pixels. The mapping escape codes support two alternative coordinate systems, raster pixels and NDC (normalized device coordinates), as indicated by the ST or DT argument (source or destination coordinate type). Note that the origin of the pixel coordinate system is the upper left corner of the display window (consistent with most graphics systems), whereas the origin of the NDC coordinate system is the lower left corner (consistent with IRAF).

Pixel coordinates allow precise control of imaging but require the application to know the window size, and may result in complications e.g. if the window is resized. NDC coordinates pretty much guarantee that a mapping will involve sampling, hence are not the most efficient, but the graphics will be drawn correctly no matter how the window is resized and for most applications the performance difference is negligible. Most applications should use NDC coordinates for raster 0 (the display window), and pixel coordinates for rasters 1-N.

Although the size of rasters 1 and higher are defined by the client application, the size of raster zero, the actual gterm display window, is subject to the constraints of the window system. The client can attempt to reset the size of the gterm window using create raster escape with raster=0, however the Gterm widget, UI containing the *Gterm* widget, and the window manager are all free to deny such a request. The query raster escape should be called to determine the actual size of the window one will be drawing into.

AN EXAMPLE IMAGING APPLICATION

An example of a simple imaging application might be one that downloads an image and displays it in the gterm window, filling the window. This could be done as follows (following a graphics open and other escape codes to prepare the drawing surface).

create raster Create raster 1 the size of the pixel array to be displayed. This need not be the same as the size of the gterm display window.

set mapping Define a mapping between raster 1 and raster 0, the display window, using NDC coordinates to define the region of the display window to be filled. The mapping number is arbitrary but mappings should normally be allocated starting with 1. The mapping is automatically enabled when first defined.

write colormap (Optional). Define the pixel value to RGB color assignments for the image pixels.

write pixels This escape is called one or more times to write pixels into raster 1. At most 32K pixels can be written in each call. As each write is made the affected region of the display window will be updated.

Alternatively, one could write the pixels and then define the mapping to cause the entire image to be displayed at once.

Note that the imaging escape can be combined with normal graphics to draw text and graphics around or on top of an image region. The order in which drawing operations occur is important, e.g., to draw graphics or text on top of an image the image should be drawn first.

MARKERS

Markers are a general feature of the *Gterm* widget and are used more extensively in other programs (e.g. the prototype IRAF science GUI applications), but they have no real use in *xgterm* when used as simply a graphics terminal. All markers share some of the same characteristics, so it is worthwhile learning basic marker manipulation keystrokes (as defined using the default marker translations), especially how to delete an accidentally created marker:

- *Delete* or *Backspace* in a marker deletes it.
- MB1 anywhere inside a marker may be used to drag the marker.
- MB1 near a marker corner or edge, depending on the type of marker, resizes the marker.
- Shift-MB1 on the corner of most markers will rotate the marker.
- Markers stack, if you have several markers and you put one on top of the other. The active marker is highlighted to tell you which of the stacked markers is active. If the markers overlap, this will be marker "on top" in the stacking order.
- MB2 in the body of a marker "lowers" the marker, i.e. moves it to the bottom of the stacking order.

ENVIRONMENT

XGterm sets the environment variables ‘‘TERM’’ and ‘‘TERMCAP’’ properly for the size window you have created. It also uses and sets the environment variable ‘‘DISPLAY’’ to specify which bit map display terminal to use. The environment variable ‘‘WINDOWID’’ is set to the X window id number of the *xgterm* window.

SEE ALSO

xterm(1), resize(1), X(1), pty(4), tty(4)
Xterm Control Sequences (in the *xterm* source directory)

BUGS

Many of the same bugs affecting *xterm* also apply here.

Xgterm is not normally installed with setuid permissions. On some Linux systems where the /dev/tty and /dev/pty devices have root ownership and permission 600 this can cause problems. Workarounds are to either install *XGterm* with setuid permissions or modify the /dev/tty and /dev/pty devices to have permission 666.

COPYRIGHT