+ createParent: (Widget *)p and - setParent: (Widget *)p
    When a widget is created it needs to be given a parent. The parent widget will be the widget's containing window. If no parent is given (ie, a parent of nil), then a toplevel Frame will be allocated automatically

- setWindowTitle: (const char *)s
    Sets the window manager title of the top level window

- pack, - packWith: (const char *)c
    Roughly, packing a widget makes it draw on the screen. The Tk packer allows complicated options to control widget layout: you can pass options via -packWith:. See documentation on Tk to learn more about packing details.

- getWidth, - setWidth: (int)n, - getHeight, - setHeight: (int)n, - getPositionX, - getPositionY, - setPositionX: (int)n Y: (int)n, etc.
    These methods allow you to control the position and size of the widget.

- (const char *)getWidgetName, -(const char *)getObjcName
    Get various names for the widgets. The WidgetName is a Tcl name for the window: the ObjcName is the name of the object itself. These are used if you choose to send strings off to Tcl directly for evaluation.

---

# Simple Widgets

There are a variety of simple widgets which do straightforward things.

## Frame

Frames are boxes other widgets fit in. They correspond to the Tk "frame" and "toplevel" widgets. Frames can be new windows, or subwindows in an existing window. You only need to create frames yourself if building complicated composite widgets: by default, a frame will be built automatically for widgets without parents.

## Label

Labels are text strings used for displaying short information.

- setText: (const char *)t
    set the label's text string and resize it to fit.

## Button

Buttons a user can click. Each button has a label and a command to be executed when the button is pressed.

## ButtonPanel

Several buttons bound together in one frame.

- `addButtonName: (const char *)n Command: (const char *)c`
  add a new button to the panel with the given name and command.

---

# Data Display Widgets

TkObjc has two basic classes for displaying numerical data.

## BLTGraph

A time series graph tool, based on BLT's graph widget. BLTGraph currently implements just a basic graph with multiple datasets, but should eventually support scaling and scrolling. For each BLTGraph you create one or many GraphElements, one per dataset to plot. GraphElements can be configured for appearance, and data can be added to the element to draw.

- `title: (const char *)t, - axisLabelsX: (const char *)xl Y: (const char *)yl`
  set labels and title on the graph.
- `setScaleModeX: (int)xd Y: (int)ys`
  takes two boolean values - whether to autoscale every timestep or instead to jump scale.
- `setRangesXMin: (double)xmin Max: (double)xmax YMin: (double)ymin Max: (double)ymax`
  sets the ranges for the graph, turns off autoscaling entirely
- `(GraphElement *)createElement`
  builds a new GraphElement for you to plot data with
- `destroyElement: (GraphElement *)g`
  remove the given graph element, destroy the plot.

GraphElements are where the real work are, where you give it data to plot. One BLTGraph can have multiple GraphElements.

- `setLabel: (const char *)label`
  set the label for this element
- `setColor: (const char *)color, - setWidth: (int)w, - setDashes: (int)i, - setSymbol: (const char *)s`
  change the apperance of the graph. The argument to dashes specifies the dash width. Valid symbols are `line` (default), `square`, `circle`, `diamond`, `plus`, `cross`, `splus`, `scross`.
- `addX: (double)x Y: (double)y`

- (BOOL)setColor: (Color)c ToName: (const char *)colorName
    set the specified color to the given colorname.

-(BOOL) setColor: (Color) c ToGrey: (double) g
    set the specified color to the grey value in [0, 1.0].

-(BOOL) setColor: (Color) c ToRed: (double) r Green: (double) g Blue: (double) b
    set the specified color to the red, green, blue vector in the range [0, 1.0].

## Raster

2 dimensional, colour pixel images. Raster is based on a Tk frame widget with our own code for fast display of images. You can draw coloured dots on a Raster, or generic ¡XDrawer¿s (not documented). Raster widgets are double buffered - the pixels you draw are not actually put on the screen until `- drawSelf` is called. In addition, Rasters handle mouse clicks.

- erase
    erase the window to black

- drawSelf
    render the Raster on the screen (complete the double buffer)

- setColorMap: (XColorMap *)colormap
    sets the colourmap used to translate logical colors to screen colors.

-(XColormap *) getColormap;
    returns the id of the colormap

- drawPointX: (int)x Y: (int)y Color: (Color)c;
    draw a point at the given coordinates with the given color.

- fillRectangleX0: (int)x0 Y0: (int)y0 X1: (int)x1 Y1: (int)y1 Color: (Color)c
    fill a whole rectangle with color

- setButton: (int)n Client: c Message: (SEL)s
    when button n (ButtonLeft, ButtonMiddle, or ButtonRight) is pressed, send the given message to the specified client object.

- (Display *)getDisplay
    returns the id of the display

- handleButton: (int)n X: (long)x Y: (long)y
    fields the button clicks

## ZoomRaster

ZoomRaster is a subclass of Raster that implements a zoomable image. It handles translation between logical coordinates and screen coordinates.

- increaseZoom, - decreaseZoom, - setZoomFactor: (int)z
    change the zoom size

designates the object to which this item refers

- setClickSel: (SEL)sel
  sets the message that will be effected upon a click on this item
- setMoveSel: (SEL)sel
  sets the message that will effect the motion of the item on the canvas
- setPostMoveSel: (SEL)sel
  sets the message that will dictate what happens after the item is moved
- createItem
  passthrough method to be implemented by subclasses
- createBindings
  establishes the bindings for the buttons: button 3 registers a "click", button 1 up and down sets the focus, and button 1 held down initiates motion.
- createEnd
  calls createItem and createBindings
- clicked
  sends the clickselector message to the referent
- initiateMoveX: (long)delta_x Y: (long)delta_y
  prepares for movement of the item within the canvas

## Circle : CanvasItem

- setX: (int)x Y: (int)y
  set the x,y coordinates for the center of the circle
- setRadius: (int)r
  set the radius of the circle

## Line : CanvasItem

- setTX: (int)tx TY: (int)ty LX: (int)lx LY: (int)ly
  set the endpoints of the line

## Rectangle : CanvasItem

- setTX: (int)tx TY: (int)ty LX: (int)lx LY: (int)ly
  set the two diagonal corner coordinates of the rectangle

## TextItem : CanvasItem

- setX: (int)x Y: (int)y
  sets the coordinate for the center of the text
- setText: (const char *)text