TkObjc Documentation

`Tcl/Tk`

`(ftp://ftp.smli.com/pub/tcl/)`
is a scripting language and graphical widget set. TkObjc is a library of wrapper classes around Tk and BLT widgets. It's purpose is to provide a simple graphical interface while hiding most Tk-specific code from library users. To create and use graphical widgets, the user merely needs to create and use objects. Many of the objects here are straightforward wrappings of Tk widgets, but some (ButtonPanel, for instance) are combinations of other widgets, and others (Raster) are novel code.

```
TkObjc is currently based on Tcl 7.5, Tk 4.1
(ftp://ftp.smli.com/pub/tcl/)
, BLT 2.1
(ftp://ftp.neosoft.com/pub/tcl/NEW/)
or BLT 2.3
(http://www.tcltk.com/blt/index.html)
,and tclobjc-1.2
(ftp://ftp.santafe.edu/pub/swarm/needed-software/tclobjc-1.2.tar.gz)
(available from Swarm authors).  Very little of this code is library-version dependent,
however, as most of it works by directly calling the Tk interpreter.
```

```
The basic purpose of tkobjc is to package Tk functionality.  Therefore, tkobjc's behaviour
is similar to the Tk toolkit.  For simple usage one should be able to get fairly far
just by looking at this document, the header files, and the Swarm examples:  more
complicated graphical output will require the programmer have some familiarity with
Tk.
```

---

globalTkInterp

TkObjc gets most of its work done by talking to the Tcl interpreter object `globalTkInterp`, an instance of class TkExtra. Strings handed to the interpreter's eval: method are evaluated by the Tcl interpreter and any side effects are performed. The libtclobjc library provides the functionality that glues Tk to Objective C. Users with simple needs never need to talk to the Tk interpreter directly.

Widget Base Class

All graphical widgets inherit from the Widget base class. Widget defines most of the behaviour needed: Widgets are created by the user with a particular parent, and then "pack"ed in order to draw them on the screen. All widgets have three essential things: a widget name used when running Tcl code, an Objective C name when sending messages from Tcl to those objects, and a parent.

```
+ 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.

```
- setText:  (const char *)t
     set the button's text string and resize to fit.
- setCommand:  (const char *)c
     set the button's command, what to do.  This is a Tcl string passed off to
     the interpreter.  See the ControlPanel code in simtools for an example.
```

Input Widgets

All widgets which take input from the user inherit from a base class, InputWidget. InputWidgets get their input in one of two ways: by being readable, or by being linked to a C variable.

InputWidget (base class)

```
- (const char *)getValue
     read the value as a string
- setValue:  (const char *)v
     set the value as a string
- linkVariable:  (void *)p Type:  (int)type
     Link the widget directly to a C variable - any changes from the user will
     be automatically stored in the variable.  Types are TCL_LINK_INT, TCL_LINK_BOOLEAN,
     TCL_LINK_DOUBLE, and TCL_LINK_STRING.
```

Entry

Handles data of any input type - user types in a string.

CheckButton

Boolean data only, click to set.

Compound Widgets

Some TkObjc widgets consist of several other widgets bound together into one object.

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
```

add the datapoint (x,y) to the graph

- resetData
    remove all data from this element, start over

-(BLTVector *)getXData, -(BLTVector *)getYData
    Get the data in the graph as a BLTVector (undocumented)

Histo

Histogram display tool, based on BLT's barchart. The number of bins is fixed at creation time, then the user hands the Histo an array of datapoints (double or int) to display (or optionally an array of datapoints and *locations* where the bars should be drawn (specified as doubles).

- title: (const char *)t, - axisLabelsX: (const char *)xl Y: (const char *)yl
    title the histogram, give it axis labels.

- setNumPoints: (int)n Labels: (const char * const*) l Colors: (const char * const *)c
    initialize the histogram, tell it how big its dataset is. Labels and Colors are arrays of strings (one string per point) for text labels and the colours of the bars (the last two arguments are optional).

- drawHistoWithDouble: (double *)points, - drawHistoWithInt: (int *)points
    an array of points (two possible types) to draw in the histogram.

- drawHistoWithDouble: (double *)points atLocation: (double *)locations, - drawHistoWit (int *)points atLocation: (double *)locations
    An array of points (two possible types) to draw in the histogram, and an array of specific locations in which to draw them (the bar width at every location will be of width 1.0).

Graphical Output

TkObjc has one basic class (and a subclass) for displaying graphical data. The model used here is a 2d raster display: an array of display elements. Display elements are typically simply pixels, blocks of a solid colour (specified via an XColormap). Other options are a pixmap (XPixmap) or arbitrary Xlib display code (implementation of XDrawer).

The display widgets used here do not have Tk counterparts: they are raw Xlib code written for flexibility and (relative) speed.

XColormap

Mechanism used to map numbers in the range [0, 255] to colour names. Create an XColormap, allocate colours in it, and pass it to a Raster widget for drawing.

- (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

```
- (unsigned)getZoomFactor
     returns an integer representing the zoom size
- handleConfigureWidth: (unsigned)newWidth Height: (unsigned)newHeight
     reconfigures zoom when the window is resized
```

XDrawer, XPixmap

Using XDrawer and XPixmap, you can write arbitrary Xlib code to draw in the window. XDrawer is just a protocol for the `drawOn:X:Y:GC:Caller:` message.

```
XPixmap provides packaging for drawing pixmaps onto displays.

- setFile: (const char *)filename
     designates the file from which to read the pixmap image
- createEnd
     loads the pixmap and sets the display and window
-(Pixmap) getPixmap
     returns the id of the pixmap data structure
-(BOOL) getMasked
     returns YES if a mask has been set, NO if it hasn't
-(Pixmap) getMask
     returns the id of the pixmap mask
-(unsigned) getWidth, -(unsigned) getHeight
     returns the width or height of the pixmap
```

Canvas

There are the beginnings of a packaging of the Tk canvas widget that allows arbitrary drawing of complicated shapes. This code is still under development and is not thoroughly documented. Canvas

is a subclass of Widget and so inherits those methods but overrides `createEnd`.

CanvasItem

This is the superclass of any item that might be drawn on a Canvas, e.g. the geometric shapes, textItems, etc. This is an abstract class.

```
- setCanvas: canvas
     designates the id of the Canvas in which this item resides
- setTargetId: target
```

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

sets the content of the text

CompositeItem

A CompositItem is simply a CanvasItem that consists of several pieces. (Why doesn't this inherit from CanvasItem?) The interface is identical to CanvasItem except for the additional method `moveX:Y`. Again, this is an abstract class.

```
- moveX: (long)delta_x Y: (long)delta_y
```

passthrough method to be implemented by subclass

NodeItem : CompositeItem

```
- setX: (int)x Y: (int)y
    position the item on the canvas
- (int)getX, - (int)getY;
    return the x or y coordinate for the item
- setString: (const char *)string
    designate the string that will label this node
- setColor: (const char *)aColor
    designate the fill color of the node
- setBorderColor: (const char *)aColor
    designate the border color of the node
- setBorderWidth: (int)aVal
    set the width of the border in pixels
```

LinkItem : CompositeItem

```
- setFrom: from
    designate the object (node) that will be the source of the link
- setTo: to
    designate the node that will be the destination of the link
- setColor: (const char *)aColor
    set the color of the link
- update
    redraw the link (esp due to motion of nodes)
```

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08