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.

- `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.

```
      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.

```
- (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
```

```
        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