sml-tk

Tcl/Tk are John Ousterhout's Tool Control Language and Tool Kit.

Tk provides convenient, high-level interface/library for building Graphical User Interfaces (GUIs) under the X windows system.

Tcl is a scripting language for driving arbitrary unix tools, including Tk. It is normally implemented as an interpreter called `wish`.

sml-tk is an SML library that allows Tk to be accessed from SML. It uses Tk concepts and conventions for describing GUI's, but the SML language.

Interface is via an inter-process **pipe**. SML process generates Tk commands (actually strings) and sends them down pipe to a standard `wish` process, and also gets data back.

Tcl is good for writing short programs, but doesn't scale very well. Sml-tk lets you use ML to write underlying applications and to describe GUI structure.

- `sml-tk` documentation is on-line.

- Tcl/Tk documentation is online as man pages in `/pkgs/tcltk4/man`.

- You'll need access to Ousterhout's book "Tcl and the Tk Toolkit."

Widget Parameters

Each type of widget has a wide variety of possible parameters, depending on its nature. In general, nearly every type of widget supports:

- Configuration options, allowing specification of geometric characteristics, text fonts, colors, etc. The text contents of text widgets and certain call-back functions are also specified here.

- Packing options, indicating how the widget is to be placed in relation to its siblings in the enclosing window or widget.

- Binding options, specifying call-back functions to be invoked when interesting **events** occur in the widget's screen area.

Unfortunately, not all options are supported by all kinds of widgets. Some options are ignored, others may cause runtime errors, when applied to inappropriate widgets.

Tk Concepts

A GUI is described hierarchically. At the top level are one or more **windows** through which the user communicates with the underlying application.

Each window contains one or more graphical objects called `widgets`.

Types of widgets include:

- Text messages, labels, form entry fields, editable text displays,...

- Buttons, including radio-style, check buttons, menu buttons, etc.

- Menus, including menu buttons, popup menus.

- Canvases: screen regions on which graphical objects can be placed.

- Frames: collections of sub-widgets.

Each window and widget has a **unique identifer** string by which it can be referenced.

Interaction Model

In sml-tk, the programmer builds a description of the **initial** state and appearance of the GUI by constructing a `datatype` value (details in a moment). This description can built using pure functional programming.

The GUI is then started up by making a function call, passing this descriptive value as an argument. This start-up call doesn't return immediately. Instead, control passes to an **event loop** inside Tk.

- By default, the event processing code just sits doing nothing.

- When the user does something like press a keyboard key or mouse button, or move the mouse onto a graphic object, the event loop controller invokes an appropriate **call-back** function as specified in the GUI description.

Call-backs

A call-back function is just an ordinary SML function that is invoked by Tk when a particular event occurs.

• It usually takes as its argument an **event** value that indicates the details of the event that triggered it, such as the coordinates of the mouse at the time when a button was pressed.

• The function does any application processing required by the event. It can also make Tk library calls to **change** the appearance of the GUI, e.g., by adding, moving or deleting graphic objects in a canvas or widgets in a window.

Control returns to the main SML loop when a call-back function executes an appropriate function call to delete the top-level window (or on an error).

SML datatypes (`tk_types.sml`)

```
type Window = WinId * Title * Widget list * SimpleAction

datatype Widget        =
        Frame   of WidId * Widget list *
               Pack list * Configure list * Binding list
      | Label   of WidId *
               Pack list * Configure list * Binding list
      | Button  of WidId *
               Pack list * Configure list * Binding list
      | Canvas  of WidId * ScrollType * CItem list *
               Pack list * Configure list * Binding list
      ...

datatype Configure =
        Width of int
      | Height of int
      | Foreground of Color
      | Background of Color
      | Text of string
      | Font of string
      | Command of SimpleAction
      | ...

datatype Color =
        NoColor | Black | White | Grey |
        Blue | Green | Red | Brown | Yellow
```

Text and Graphics

The built-in Tk widgets offer fairly sophisticated features for text and graphics.

For example, text in text widgets can be edited using the mouse to move around and to mark regions. The `wish` process maintains current knowledge of the contents of each text field. When the user performs an appropriate termination action (such as hitting RETURN), the associated call-back function must query Tk for the current text contents.

Graphics is performed in **canvas** widgets. Canvases are collections of **canvas items**, which can include

• Rectangles, Ovals, Polylines, etc.

• Icons (bitmaps or other images)

• Nested widgets

Each canvas item can have its own associated set of call-back functions to be executed when interesting events occur to that item, e.g., the cursor moves on or off the screen region associated with the item, or a button or key is pressed when the cursor is in that screen region.

The `wish` process maintains an internal concept of the location of each canvas item, which can also be queried by the SML callback functions.

Types for callbacks

```
type SimpleAction = unit -> unit

datatype Binding = Bind of eventName * Action

type eventName = string (* "<Enter>", "<ButtonPress-1>",
                           etc. *)

type Action = (TkEvent -> unit)

datatype TkEvent =
  TkEvent of int        (* %b  button number     *)
          *  string     (* %s  state field       *)
          *  int        (* %x  x field           *)
          *  int        (* %y  y field           *)
          *  int        (* %X  x_root field      *)
          *  int        (* %Y  y_root field      *)
```

SML Functions (`export.sml`)

```
val mainWinId                 : unit -> TkTypes.WinId
val newWinId             : unit -> TkTypes.WinId
val newWidgetId          : unit -> TkTypes.WidId

val startTclExn : TkTypes.Window list -> string
val exitTcl : unit -> unit
val resetTcl : unit -> unit

val addWidget : TkTypes.WinId -> TkTypes.WidId ->
                                TkTypes.Widget -> unit
val delWidget : TkTypes.WidId -> unit
val addBind : TkTypes.WidId ->
                        TkTypes.Binding list ->  unit
val addConf : TkTypes.WidId ->
                        TkTypes.Configure list -> unit

val insertText    : TkTypes.WidId -> string ->
                        TkTypes.Mark -> unit
val clearText     : TkTypes.WidId -> unit

val addCItem : TkTypes.WidId -> TkTypes.CItem   -> unit
val delCItem : TkTypes.WidId -> TkTypes.CItemId -> unit

val createAndPopUpMenu : TkTypes.Widget ->
        int TkTypes.option -> TkTypes.Coord -> unit

...
```

Hints: CTRL/C is handy.

Ignore “Warning: missed Binding” messages.

Trivial Example Program

```
(* Get names for named objects: *)

val mainID  = mainWinId()
val entID   = newWidgetId()

(* Define the widgets: *)
val label = Label(newWidgetId(),
                  [Side Left], [Text "name:"], [])
val input = let fun endInput _ =
                  changeTitle mainID (readTextAll entID)
            in Entry(entID, [], [Width 20],
                     [Bind("<Return>", endInput)])
            end
val quit  = let fun stop _ = closeWindow mainID
            in Button(newWidgetId(),
                      [Side Bottom],
                      [Text "Quit", Command stop], [])
            end

(* Group together the text entry and the label *)
val topblock = Frame(newWidgetId(),
                     [label, input], [Side Top], [], [])

(* Define the main window *)
val enterwin = (mainID, "Please enter name",
                [topblock, quit], fn _ => ());

(* ... and go! *)
startTcl [enterwin]
```