The LablTk library release 8.06.0
and
The OCamlBrowser library explorer

Jacques Garrigue, Jun Furuse
September 18, 2014

Chapter 1

The LablTk library: Tcl/Tk GUI interface

The `lablтk` library provides access to the Tcl/Tk GUI from OCaml programs. This interface is generated in an automated way, and you should refer to Tcl/Tk books and man pages for detailed information on the behavior of the numerous functions. We also suggest to use `ocamlbrowser` to see the types of the various functions, that are the best documentation for the library itself.

Programs that use the `lablтk` library must be linked as follows:

`ocamlc` *other options* `-I +labltk labltk.cma` *other files*
`ocamlopt` *other options* `-I +labltk labltk.cmxa` *other files*

Unix:

The `lablтk` library is available for any system with Tcl/Tk installed, starting from Tcl/Tk 8.0 up to Tcl/Tk 8.6. Beware that some beta versions may have compatibility problems.

If the library was not compiled correctly, try to run again the `configure` script with the option `-tkdefs` *switches*, where *switches* is a list of C-style inclusion paths leading to the right `tcl.h` and `tk.h`, for instance `"-I/usr/local/include/tcl8.4 -I/usr/local/include/tk8.4"`.

A script is installed, to make easier the use of the `lablтk` library as toplevel.

`labltk`
: This is a toplevel including the `lablтk` library, and the path is already set as to allow the use of the various modules. It also includes code for the Unix and Str libraries. You can use it in place of `ocaml`.

Windows:

The `lablтk` library has been precompiled for use with Tcl/Tk 8.5. You must first have it installed on your system. It can be downloaded from `http://www.activestate.com/products/ActiveTcl/`. After installing it, you must put the dynamically loaded libraries `tcl85.dll` and `tk85.dll` (from the `bin` directory of the Tcl installation) in a directory included in you path.

No toplevel is available, but you can load the library from the standard toplevel with the following commands.

```
# #directory "+labltk";;
# #load "labltk.cma";;
```

You can also load it directly from the command line.

```
C:\ocaml\bin> ocaml -I +labltk labltk.cma
```

The `labltk` library is composed of a large number of modules.

Bell	Imagebitmap	Place
Button	Imagephoto	Radiobutton
Canvas	Label	Scale
Checkbutton	Listbox	Scrollbar
Clipboard	Menu	Selection
Dialog	Menubutton	Text
Entry	Message	Tk
Focus	Option	Tkwait
Frame	Optionmenu	Toplevel
Grab	Pack	Winfo
Grid	Palette	Wm

Giving a detailed account of each of these module would be impractical here. We will just present some of the basic functions in the module `Tk`. Note that for most other modules information can be found in the Tcl `man` page of their name.

1.1 Module `Tk` : Basic functions and types for LablTk

Initialization and termination

```
val openTk :
  ?display:string -> ?clas:string -> unit -> Widget.toplevel Widget.widget
```

Initialize LablTk and open a toplevel window. `display` is described according to the X11 conventions. `clas` is used for the X11 resource mechanism.

```
val mainLoop : unit -> unit
```

Start the main event loop

```
val closeTk : unit -> unit
```

Quit the main loop and close all open windows.

```
val destroy : 'a Widget.widget -> unit
```

Destroy an individual widget.

Application wide commands

```
val update : unit -> unit
```

Synchronize display with internal state.

```
val appname_get : unit -> string
val appname_set : string -> unit
```

Get or set the application name.

Dimensions

```
type units = [ `Cm of float | `In of float | `Mm of float | `Pix of int | `Pt of float ]
val pixels : units -> int
```

Converts various on-screen units to pixels, respective to the default display. Available units are pixels, centimeters, inches, millimeters and points

Widget layout commands

```
type anchor = [ `Center | `E | `N | `Ne | `Nw | `S | `Se | `Sw | `W ]
type fillMode = [ `Both | `None | `X | `Y ]
type side = [ `Bottom | `Left | `Right | `Top ]
val pack :
  ?after:'a Widget.widget ->
  ?anchor:anchor ->
  ?before:'b Widget.widget ->
  ?expand:bool ->
  ?fill:fillMode ->
  ?inside:'c Widget.widget ->
  ?ipadx:int ->
  ?ipady:int ->
  ?padx:int -> ?pady:int -> ?side:side -> 'd Widget.widget list -> unit
```

Pack a widget inside its parent, using the standard layout engine.

```
val grid :
  ?column:int ->
  ?columnspan:int ->
  ?inside:'a Widget.widget ->
  ?ipadx:int ->
  ?ipady:int ->
  ?padx:int ->
  ?pady:int ->
  ?row:int -> ?rowspan:int -> ?sticky:string -> 'b Widget.widget list -> unit
```

Pack a widget inside its parent, using the grid layout engine.

```
type borderMode = [ `Ignore | `Inside | `Outside ]
val place :
  ?anchor:anchor ->
  ?bordermode:borderMode ->
  ?height:int ->
  ?inside:'a Widget.widget ->
  ?relheight:float ->
  ?relwidth:float ->
  ?relx:float ->
  ?rely:float -> ?width:int -> ?x:int -> ?y:int -> 'b Widget.widget -> unit
```

Pack a widget inside its parent, at absolute coordinates.

```
val raise_window : ?above:'a Widget.widget -> 'b Widget.widget -> unit
val lower_window : ?below:'a Widget.widget -> 'b Widget.widget -> unit
```

Raise or lower the window associated to a widget.

Event handling

```
type modifier = [ `Alt
  | `Button1
  | `Button2
  | `Button3
  | `Button4
  | `Button5
  | `Control
  | `Double
  | `Lock
  | `Meta
  | `Mod1
  | `Mod2
  | `Mod3
  | `Mod4
  | `Mod5
  | `Shift
  | `Triple ]
type event = [ `ButtonPress
  | `ButtonPressDetail of int
  | `ButtonRelease
  | `ButtonReleaseDetail of int
  | `Circulate
  | `ColorMap
  | `Configure
  | `Destroy
  | `Enter
```

```
  | `Expose
  | `FocusIn
  | `FocusOut
  | `Gravity
  | `KeyPress
  | `KeyPressDetail of string
  | `KeyRelease
  | `KeyReleaseDetail of string
  | `Leave
  | `Map
  | `Modified of modifier list * event
  | `Motion
  | `Property
  | `Reparent
  | `Unmap
  | `Visibility ]
```

An event can be either a basic X event, or modified by a key or mouse modifier.

```
type eventInfo = {
  mutable ev_Above : int ;
  mutable ev_ButtonNumber : int ;
  mutable ev_Count : int ;
  mutable ev_Detail : string ;
  mutable ev_Focus : bool ;
  mutable ev_Height : int ;
  mutable ev_KeyCode : int ;
  mutable ev_Mode : string ;
  mutable ev_OverrideRedirect : bool ;
  mutable ev_Place : string ;
  mutable ev_State : string ;
  mutable ev_Time : int ;
  mutable ev_Width : int ;
  mutable ev_MouseX : int ;
  mutable ev_MouseY : int ;
  mutable ev_Char : string ;
  mutable ev_BorderWidth : int ;
  mutable ev_SendEvent : bool ;
  mutable ev_KeySymString : string ;
  mutable ev_KeySymInt : int ;
  mutable ev_RootWindow : int ;
  mutable ev_SubWindow : int ;
  mutable ev_Type : int ;
  mutable ev_Widget : Widget.any Widget.widget ;
  mutable ev_RootX : int ;
  mutable ev_RootY : int ;
}
```

Event related information accessible in callbacks.

```
type eventField = [ `Above
  | `BorderWidth
  | `ButtonNumber
  | `Char
  | `Count
  | `Detail
  | `Focus
  | `Height
  | `KeyCode
  | `KeySymInt
  | `KeySymString
  | `Mode
  | `MouseX
  | `MouseY
  | `OverrideRedirect
  | `Place
  | `RootWindow
  | `RootX
  | `RootY
  | `SendEvent
  | `State
  | `SubWindow
  | `Time
  | `Type
  | `Widget
  | `Width ]
```

In order to access the above event information, one has to pass a list of required event fields to the `bind` function.

```
val bind :
  events:event list ->
  ?extend:bool ->
  ?breakable:bool ->
  ?fields:eventField list ->
  ?action:(eventInfo -> unit) -> 'a Widget.widget -> unit
```

Bind a succession of `events` on a widget to an `action`. If `extend` is true then then binding is added after existing ones, otherwise it replaces them. `breakable` should be true when `break` is to be called inside the action. `action` is called with the `fields` required set in an `eventInfo` structure. Other fields should not be accessed. If `action` is omitted then existing bindings are removed.

```
val bind_class :
  events:event list ->
  ?extend:bool ->
  ?breakable:bool ->
```

```
  ?fields:eventField list ->
  ?action:(eventInfo -> unit) -> ?on:'a Widget.widget -> string -> unit
```

Same thing for all widgets of a given class. If a widget is given with label `~on:`, the binding will be removed as soon as it is destroyed.

```
val bind_tag :
  events:event list ->
  ?extend:bool ->
  ?breakable:bool ->
  ?fields:eventField list ->
  ?action:(eventInfo -> unit) -> ?on:'a Widget.widget -> string -> unit
```

Same thing for all widgets having a given tag

```
val break : unit -> unit
```

Used inside a bound action, do not call other actions after this one. This is only possible if this action was bound with `~breakable:true`.

Chapter 2

The browser/editor (ocamlbrowser)

This chapter describes OCamlBrowser, a source and compiled interface browser, written using LablTk. This is a useful companion to the programmer.

Its functions are:

- navigation through OCaml's modules (using compiled interfaces).
- source editing, type-checking, and browsing.
- integrated OCaml shell, running as a subprocess.

2.1 Invocation

The browser is started by the command `ocamlbrowser`, as follows:

`ocamlbrowser` *options*

The following command-line options are recognized by `ocamlbrowser`.

`-I` *directory*
: Add the given directory to the list of directories searched for source and compiled files. By default, only the standard library directory is searched. The standard library can also be changed by setting the `OCAMLLIB` environment variable.

`-nolabels`
: Ignore non-optional labels in types. Labels cannot be used in applications, and parameter order becomes strict.

`-oldui`
: Old multi-window interface. The default is now more like Smalltalk's class browser.

`-rectypes`
: Allow arbitrary recursive types during type-checking. By default, only recursive types where the recursion goes through an object type are supported.

`-version`
: Print version string and exit.

`-vnum`
Print short version number and exit.

`-w` *warning-list*
Enable or disable warnings according to the argument *warning-list*.

Most options can also be modified inside the application by the **Modules - Path editor** and **Compiler - Preferences** commands. They are inherited when you start a toplevel shell.

2.2 Viewer

This is the first window you get when you start OCamlBrowser. It displays a search window, and the list of modules in the load path. At the top a row of menus.

- **File - Open** and **File - Editor** give access to the editor.
- **File - Shell** creates an OCaml subprocess in a shell.
- **View - Show all defs** displays the signature of the currently selected module.
- **View - Search entry** shows/hides the search entry just below the menu bar.
- **Modules - Path editor** changes the load path. **Modules - Reset cache** rescans the load path and resets the module cache. Do it if you recompile some interface, or get confused about what is in the cache.
- **Modules - Search symbol** allows searching a symbol either by its name, like the bottom line of the viewer, or more interestingly, by its type. **Exact type** searches for a type with exactly the same information as the pattern (variables match only variables). **Included type** allows giving only partial information: the actual type may take more arguments and return more results, and variables in the pattern match anything. In both cases, argument and tuple order is irrelevant[1], and unlabeled arguments in the pattern match any label.
- The **Search entry** just below the menu bar allows one to search for an identifier in all modules (wildcards "?" and "*" allowed). If you choose the `type` option, the search is done by type inclusion (*cf.* Search Symbol - Included type).
- The **Close all** button is there to dismiss the windows created by the Detach button. By double-clicking on it you will quit the browser.

2.3 Module browsing

You select a module in the leftmost box by either clicking on it or pressing return when it is selected. Fast access is available in all boxes pressing the first few letter of the desired name. Double-clicking / double-return displays the whole signature for the module.

[1]To avoid combinatorial explosion of the search space, optional arguments in the actual type are ignored in the actual if (1) there are too many of them, and (2) they do not appear explicitly in the pattern.

Defined identifiers inside the module are displayed in a box to the right of the previous one. If you click on one, this will either display its contents in another box (if this is a sub-module) or display the signature for this identifier below.

Signatures are clickable. Double clicking with the left mouse button on an identifier in a signature brings you to its signature. A single click on the right button pops up a menu displaying the type declaration for the selected identifier. Its title, when selectable, also brings you to its signature.

At the bottom, a series of buttons, depending on the context.

- **Detach** copies the currently displayed signature in a new window, to keep it.
- **Impl** and **Intf** bring you to the implementation or interface of the currently displayed signature, if it is available.

Control-S lets you search a string in the signature.

2.4 File editor

You can edit files with it, if you're not yet used to emacs. Otherwise you can use it as a browser, making occasional corrections.

The **Edit** menu contains commands for jump (C-g), search (C-s), and sending the current phrase (or selection if some text is selected) to a sub-shell (M-x). For this last option, you may choose the shell via a dialog.

Essential functions are in the **Compiler** menu.

- **Preferences** opens a dialog to set internals of the editor and type-checker.
- **Lex** adds colors according to lexical categories.
- **Typecheck** verifies typing, and memorizes to let one see an expression's type by double-clicking on it. This is also valid for interfaces. If an error occurs, the part of the interface preceding the error is computed.

  After typechecking, pressing the right button pops up a menu that gives the type of the pointed expression and, where applicable, provides some links that can be followed.
- **Clear errors** dismisses type-checker error messages and warnings.
- **Signature** shows the signature of the current file (after type checking).

2.5 Shell

When you create a shell, a dialog is presented to you, letting you choose which command you want to run, and the title of the shell (to choose it in the Editor).

The executed subshell is given the current load path.

- **File** use a source file or load a bytecode file. You may also import the browser's path into the subprocess.

- **History** M-p and M-n browse up and down.
- **Signal** C-c interrupts, and you can also kill the subprocess.