The CamlTk interface
Release `for Tk4.0`

François Pessaux and François Rouaix
Projet Cristal, INRIA Rocquencourt
Francois.Pessaux@inria.fr, Francois.Rouaix@inria.fr

May 1996

Contents

Introduction

Tk is a GUI (graphical user-interface) library for the Tcl language. Normally, Tk is controlled from Tcl. The purpose of the CamlTk interface is to provide a Caml Light library to control Tk from Caml Light programs. Thus, Tk can be used to program user-interfaces in Caml Light without knowledge of the Tcl language.

Installation

- This release runs only on Unix systems.
- You need Caml Light 0.7.
- You need to have Tcl and Tk installed (that is, at least the include filess and libraries). This release has been tested with Tcl 7.4 and Tk 4.0, and is not compatible with previous releases of TclTk.

This documentation

The present documentation is sketchy, thus some basic background about toolkits, event-driven programming, and Tk is assumed. Documentation for Tk is currently available in [1], [2], and in the `man` pages of the Tk distribution. What you will find here is:

The user's manual : a short tutorial describing how to write Caml Light programs using the Tk interface and how to compile them.

The implementor's manual : describing how the interface library is generated from a high-level description, and some technical details on the interface. This part is of interest mainly for expert Tk users who wish to extend the interface to support more functions, more widgets, or simply to fix bugs in test versions.

The Tentative reference manual : the collection of interface files from the library.

The CamlTk distribution also includes a first shot at a Caml Light browser, an *hypertext* tool that may help in finding the types of functions or types definitions available in Caml Light libraries.

Bug reports

Please report problems and bugs directly to `Francois.Rouaix@inria.fr` instead of Caml Light mailing lists. Don't hesitate to make comments and suggestions, or ask questions.

Acknowledgments

Thanks to Loic Prylli, Christophe Raffalli, Hendrik Tews, Leszek Holenderski for bug reports. Thanks to Pierre Weis for many comments and suggestions during the development and testing phases.

Chapter 1

User's Manual

This manual is a simple tutorial for CamlTk. When writing your own programs, you should consult the reference manual (chapter 3), as well as the Tk man pages. Whenever the reference manual is unsufficient, you can check the `Widgets.src` file (which syntax is found in chapter 2) since this file gives the mappings between Tk and Caml Light for functions and data.

1.1 Modules

The CamlTk interface is provided as a Caml Light library, composed of the following modules:

- `tk`, containing the initialisation functions, some frequently used functions, and all types manipulated by the library.
- a module for each widget class (e.g. `button`). The creation function and the commands specific to this class are contained in the module. The module bears the same name as the widget class, except for `toplevel` widgets which are defined in module `toplevelw`.
- various other modules grouping functions of a same family (e.g. `focus`, `winfo`).
- the `textvariable` module, for the `-textvariable` option support.
- `protocol` and `support`, which contain the internals of the interface. For example, the abstract type of widgets is defined in `support`, as well as some widget naming utilities.

All modules required by the CamlTk interface are in the `tklib.zo` library archive.

The organization of the interface is designed so that, in general, you should `#open` only the `tk` module, and for the rest of the functions use the `modulename__symbol` syntax (e.g. `text__insert`). This convention is preferred because some functions (such as `create` and `configure`) exist in many different modules. In this context, referring to functions with exact qualified names makes programs clearer and less error-prone.

Figure 1.1: The `helloworld` example

1.2 Basics

As usual with toolkits, the simplest way to learn how to write programs is by copy-paste-ing from available source code. In general, a Caml Light program using the Tk toolkit will have the following structure :

```
#open "tk";;             (* Make interface functions available *)
let top = openTk ();;    (* Initialisation of the interface *)
(* top is now the toplevel widget *)

(* Widget initialisation *)
let b = button__create top
          [Text "foobar";
           Command (function () ->
                      print_string "foobar";
                      print_newline();
                      flush stdout)]
;;
(* b exists but is not yet visible *)
pack [b][] ;;            (* Make b visible *)
mainLoop() ;;            (* User interaction*)
(* You can quit this program by deleting its main window *)
```

The first step is to initialize the interface with `OpenTk`, and this returns the *toplevel* widget. Then, the program creates some widgets, as children of the *toplevel* widget. These widgets form the user interface of your application. In this simple case, we have only one button, whose behaviour is to print a simple message on the standard output. Finally, the program enters the `MainLoop`, and user-interaction takes place. Figure 1.1 shows the interface of this program.

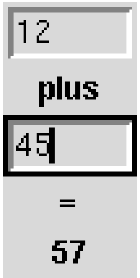
1.3 Example

Here is a complete working example with a few widgets and callbacks. This program may be found in the `test` directory of the distribution, under the name `addition.ml`.

```
#open "tk";;

let main () =
  let top = openTk ()  in
```

Figure 1.2: The `addition` example




```
  (* The widgets. They all have "top" as parent widget. *)
  let en1 = entry__create top [TextWidth 6; Relief Sunken] in
  let lab1 = label__create top [Text "plus"] in
  let en2 = entry__create top [TextWidth 6 ; Relief Sunken] in
  let lab2 = label__create top [Text "="] in
  let result_display = label__create top [] in
  (* References holding values of entry widgets *)
  let n1 = ref 0
  and n2 = ref 0  in
  (* Refresh result *)
  let refresh () =
    label__configure result_display [Text (string_of_int (!n1 + !n2))]  in
  (* Electric *)
  let get_and_refresh (w,r) =
    fun _ _ ->
      try
       r := int_of_string (entry__get w);
       refresh ()
      with
        Failure "int_of_string" ->
          label__configure result_display [Text "error"]
  in
  (* Set the callbacks *)
  entry__configure en1 [XScrollCommand (get_and_refresh (en1,n1)) ];
  entry__configure en2 [XScrollCommand (get_and_refresh (en2,n2)) ];
  (* Map the widgets *)
  pack [en1;lab1;en2;lab2;result_display] [];
  (* Make the window resizable *)
  wm__minsize_set top 1 1;
  (* Start interaction (event-driven program) *)
  mainLoop ()
;;
```

```
printexc__f main () ;;
```

The XScrollCommand callback has the property of being called each time the insertion cursor blinks in an entry widget. This property is responsible for the "electric" behavior of this program. Another approach would have been to associate an action to the event "key return has been pressed". Figure 1.2 shows the interface of this program.

1.4 Initialisation

A program using the CamlTk interface must call one of the three following functions :

```
value openTk : unit -> widget
and openTkClass : string -> widget
and openTkDisplayClass : string -> string -> widget
```

Each of these functions returns the initial toplevel widget ("." in TclTk). openTkClass lets you specify the resource class of the toplevel widget, and openTkDisplayClass lets you specify the display and the resource class.

During the initialisation, the file $HOME/.camltkrc, if it exists, will be loaded by TclTk. This might be the place where to write some special customisations in Tcl, such as changing auto-load path, ...

1.5 Widgets

The description of widget commands and options is given in the reference manual (chapter 3). The semantics of commands is however not described there, so the reader should also refer to the man pages of the Tk distribution. As of this release, all widgets of Tk 4.0 are supported, although some commands might not be implemented. Remember also that only few of them have been tested.

1.6 Bindings

Tk offers, in addition to widget-specific callbacks, a general facility for binding functions to events. (bind(n)). Bindings involves the following types, defined in the tk module:

```
type xEvent =
    ButtonPress (* also Button, but we omit it *)
  | ButtonPressDetail of int
  | ButtonRelease
  | ButtonReleaseDetail of int
  | Circulate
  | ColorMap
  | Configure
  | Destroy
  | Enter
```

```
  | Expose
  | FocusIn
  | FocusOut
  | Gravity
  | KeyPress (* also Key, but we omit it *)
  | KeyPressDetail of string      (* /usr/include/X11/keysymdef.h *)
  | KeyRelease
  | KeyReleaseDetail of string
  | Leave
  | Map
  | Motion
  | Property
  | Reparent
  | Unmap
  | Visibility
;;
type modifier =
    Control
  | Shift
  | Lock
  | Button1
  | Button2
  | Button3
  | Button4
  | Button5
  | Double
  | Triple
  | Mod1
  | Mod2
  | Mod3
  | Mod4
  | Mod5
  | Meta
  | Alt
;;
type eventInfo =
  {
  mutable ev_Above : int;               (* tk: %a *)
  mutable ev_ButtonNumber : int;        (* tk: %b *)
  mutable ev_Count : int;               (* tk: %c *)
  mutable ev_Detail : string;           (* tk: %d *)
  mutable ev_Focus : bool;              (* tk: %f *)
  mutable ev_Height : int;              (* tk: %h *)
  mutable ev_KeyCode : int;             (* tk: %k *)
  mutable ev_Mode : string;             (* tk: %m *)
```

```
  mutable ev_OverrideRedirect : bool;   (* tk: %o *)
  mutable ev_Place : string;            (* tk: %p *)
  mutable ev_State : string;            (* tk: %s *)
  mutable ev_Time : int;                (* tk: %t *)
  mutable ev_Width : int;               (* tk: %w *)
  mutable ev_MouseX : int;              (* tk: %x *)
  mutable ev_MouseY : int;              (* tk: %y *)
  mutable ev_Char : string;             (* tk: %A *)
  mutable ev_BorderWidth : int;         (* tk: %B *)
  mutable ev_SendEvent : bool;          (* tk: %E *)
  mutable ev_KeySymString : string;     (* tk: %K *)
  mutable ev_KeySymInt : int;           (* tk: %N *)
  mutable ev_RootWindow : int;          (* tk: %R *)
  mutable ev_SubWindow : int;           (* tk: %S *)
  mutable ev_Type : int;                (* tk: %T *)
  mutable ev_Widget : widget;           (* tk: %W *)
  mutable ev_RootX : int;               (* tk: %X *)
  mutable ev_RootY : int                (* tk: %Y *)
  }
;;
type eventField =
    Ev_Above
  | Ev_ButtonNumber
  | Ev_Count
  | Ev_Detail
  | Ev_Focus
  | Ev_Height
  | Ev_KeyCode
  | Ev_Mode
  | Ev_OverrideRedirect
  | Ev_Place
  | Ev_State
  | Ev_Time
  | Ev_Width
  | Ev_MouseX
  | Ev_MouseY
  | Ev_Char
  | Ev_BorderWidth
  | Ev_SendEvent
  | Ev_KeySymString
  | Ev_KeySymInt
  | Ev_RootWindow
  | Ev_SubWindow
  | Ev_Type
  | Ev_Widget
```

```
  | Ev_RootX
  | Ev_RootY
;;
type bindAction =
   BindSet of eventField list *  (eventInfo -> unit)
 | BindSetBreakable of eventField list *  (eventInfo -> unit)
 | BindRemove
 | BindExtend of eventField list *  (eventInfo -> unit)
;;
```

Binding are manipulated with the function

```
bind: widget -> (modifier list * xEvent) list -> bindAction -> unit
```

The first argument of type `widget` is naturally the widget for which this binding is defined. The second argument of type `(modifier list * xEvent) list` is the succession of events that will trigger the callback. The last argument is the action : `BindRemove` will remove a binding, `BindSet` will set a binding, and `BindExtend` will extend a binding. `BindSetBreakable` is equivalent to `BindSet`, except that the callback may call the `tk__break` function to break the chain of callbacks (see `break` in the `bind(n)` Tk man page).

In many cases, a callback needs information about the event that triggered it, such as the precise key that was pressed, the position of the mouse, etc. Therefore, a callback is of type `EventInfo -> unit`, and the callback will be invoked with argument the record containing event information. Since event information can be quite large, the user has to specify, as an `EventField list`, which fields in this record are relevant for this particular callback. The other fields will **not** contain accurate information. Check also the `Tk` documentation for finding out which fields are valid for a given event.

Example

```
bind myCanvas [[],Motion] (BindSet([Ev_MouseX; Ev_MouseY], mouse_moved)) ;;
```

will trigger the callback `mouse_moved` when the event `Motion` occurs in the `myCanvas` widget. The function `mouse_moved` may assume that the information in the fields `Ev_MouseX, Ev_MouseY` of its argument contain accurate information.

Text and Canvas bindings

Bindings on objets in `text` and `canvas` widgets obey the same rules, and the functions are

```
text__tag_bind : widget -> textTag ->
      (modifier list * xEvent) list -> bindAction -> unit
canvas__bind : widget -> tagOrId ->
      (modifier list * xEvent) list -> bindAction -> unit
```

Additional arguments are respectively the *tag* and the *tag or id* for which the binding is defined.

Class bindings

Binding for classes of widgets (resp. tags) are managed with

```
tk__class_bind : string ->
      (modifier list * xEvent) list -> bindAction -> unit
tk__tag_bind : string ->
      (modifier list * xEvent) list -> bindAction -> unit
```

The first argument should be a widget class name (e.g. "Button"), (resp. a tag name). Widget class names in CamlTk are the exact same names as in Tk.

The tk__bindtags function determines the order of bindings for a given widget.

1.7 Text variables

Text variables are available in CamlTk. However, they are not physically shared between Caml Light and Tk, that is, Tk cannot directly modify a Caml Light variable. Instead, setting or consulting a textvariable requires a transaction between Caml Light and Tk. However, the "sharing" properties of text variables in Tk are preserved. For example, if a text variable is used to physically share the content of an entry with the text of a label, then this sharing will effectively occur in the Tk world, whether your Caml Light program is concerned with the actual contents of the text variable or not.

Since text variables are global in the Tcl space, using a large amount of them might result in space leaks. Thus, there exists a creation function which relates the life of the text variable to the life of a widget. When the associated widget is destroyed, the text variable will be reused by the system.

1.8 File descriptor callbacks

A callback can be associated to a file descriptor using the following primitive:

```
tk__add_fileinput : file_descr -> (unit -> unit) -> unit
```

When some input is available on the file descriptor specified by the first argument, then the callback (second argument) is called.

```
tk__remove_fileinput : file_descr -> unit
```

Removes the file descriptor callback.

It is the programmer's responsability to check for end of file, and to remove the callback if the file descriptor is closed. Be aware that calling the update function will potentially destroy the sequentiality of read operations. Finally, note that during the invocation of your callback, only one call to read is guaranteed to be non-blocking.

tk__add_fileoutput and tk__remove_fileoutput work on the same principle.

Note: there can be only one callback at the same time on a given descriptor. In particular, it is not possible to have an input callback and an output callback simultaneously.

1.9 Timers

```
tk__add_timer : int -> (unit -> unit) -> timer
tk__remove_timer : timer -> unit
```

A callback can be scheduled to occur after a given delay using `tk__add_timer`. The first argument is the delay (in milliseconds). A timer callback can be removed before its execution with `tk__remove_timer`. If the callback has already been executed, `tk__remove_timer` has no effect. It is not necessary to call `tk__remove_timer` after a timer callback has been executed.

1.10 Exiting

The function

```
tk__closeTk : unit -> unit
```

will terminate the main interaction loop (`tk__MainLoop`) and destroy all widgets.

1.11 Errors

The CamlTk interface may raise the following exceptions:

TkError of string : this exception is raised when a TclTk command fails during an evaluation. Normally, static typing in Caml Light, in addition to some run-time verifications should prevent this situation from happening. However, some errors are due to the external environment (e.g. `selection__get`) and cannot be prevented. The exception carries the TclTk error message.

IllegalWidgetType of string : this exception is raised when a widget command is applied to a widget of a different or unappropriate class (e.g. `button__configure` applied to a *scrollbar* widget). The exception carries the class of the faulty widget.

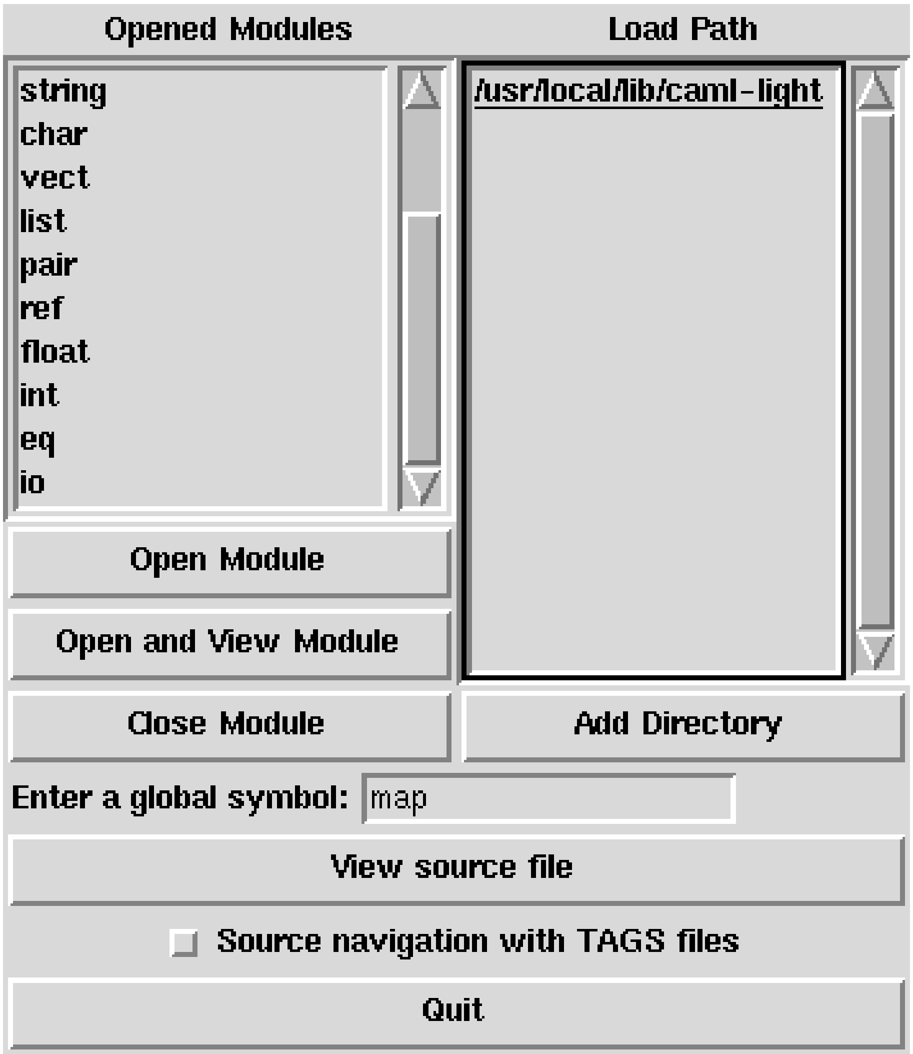
Invalid_argument of string : this exception is raised when some data cannot be exchanged between Caml Light and Tk. This situation occurs for example when an illegal option is used for creating a widget. However, this error may also be caused by a faulty or incomplete description of a widget, widget command or type.

1.12 More examples

The `test` directory of the distribution contains several test examples. The `books-examples` directory contains translations in CamlTk of several examples from [1].

All examples included in this distribution should not be considered as working applications. They are provided for documentation purposes only.

Figure 1.3: The Caml Browser



1.13 The Caml Browser

In the directory `browser` you will find a yet more complete example, featuring many different kinds of widgets. The browser mimicks the Caml Light toplevel behaviour: you can open and close modules (`#open` and `#close` directives), or add a library path (`#directory` directive). You can enter a symbol (value, type constructor, value constructor, exception), and see its definition. You can also see the complete interface of a module by double-clicking on it. Remember that the browser will show you a pretty-printed version of compiled interfaces (`.zi` files), so you will not see the comments that were in the source file.

The browser can also display source files. Hypertext navigation is also available, but does not reflect the compiler's semantics, as the `#open` directives in the source file are not taken into account.

When available, the browser also attemps to load `TAGS` file, created by the `mletags` program in the Caml Light contributions. It is therefore possible to navigate in program sources. However, note that the browser implements only a simplified version of Emacs's `tag` mechanism.

1.14 Compilation

In the following, we refer to `TCLLIBDIR` (resp. `TKLIBDIR`) as an environment variable containing the directory where `libtcl.a` (resp. `libtk.a`) is located. Consult your system administrator if you don't have this information.

Moreover, we assume a "standard installation" of Caml Light and CamlTk from the distribution. This means that all CamlTk library files are installed in the same directory as standard Caml Light library files.

The usual commands for compiling and linking are:

```
$ camlc -c addition.ml
$ camlc -custom -o addition tklib.zo addition.zo \
        -ccopt -L$TCLLIBDIR -ccopt -L$TKLIBDIR \
        -lcamltk4 -ltk -ltcl -lX11
```

Linking is a bit complex, because we need to tell the C compiler/linker where to find the `libtk.a` and `libtcl.a` libraries, and to link with all required pieces.

1.14.1 Toplevels

The distribution also installs a toplevel featuring the CamlTk interface: `camltktop`. This toplevel may be run with:

```
$ camllight camltktop
```

Note however that the usage of the toplevel is awkward (for the time being), because callbacks cannot be executed until you enter MainLoop, and you cannot easily leave MainLoop.

1.15 Translating Tk idioms

If you are a seasoned Tk programmer, you will find out that some Tk idioms have a different form in CamlTk .

1.15.1 Widgets

First of all, widget creation is more "functional". One does not specify the name of the created widget (except for `toplevel` widgets), but only its parent. The name is allocated by the library, and a handle to the widget is returned to you. Then, widgets are not "procedures". They are objects, and must be passed to widget manipulation functions. For example,

```
button .myf.bok -title "Ok" -relief raised
```

is translated by (assuming that `myf` is the parent widget `.myf`)

```
let b = button__create myf [Title "Ok"; Relief Raised] in
...
```

Then,

```
.myf.bok configure -state disabled
```

would be in Caml Light:

```
button__configure b [State Disabled]
```

This is more in the spirit of ML, but unfortunately eliminates the possibility to specify configuration of widgets by resources based on widget names. It you absolutely want to use resources then you may call the alternate `create_named` function:

```
let b = button__create_named myf "bok" [Title "Ok"; Relief Raised] in
...
```

Assuming that `myf` is the widget of path `.myf`, then `b` will have path `.myf.bok`. As in Tk, it is your responsibility to use valid identifiers for path elements.

When widgets are mutually recursive (through callbacks, for example when linking a scrollbar to a text widget), one should first create the widgets, and then set up the callbacks by calling `configure` on the widgets. This is in contrast with TclTk where one may refer to a procedure that has not yet been defined.

Partially applied widget commands (such as redisplay commands) translate quite well to Caml Light, with possibly some wrapping required due to value constructors.

1.15.2 How to find out the Caml Light name of Tk keywords and functions

Normally, a Caml Light command for a widget class will have the same name as the corresponding Tk command. However, in some cases, for example when a same command is used to read and set values, or has a variable number of type incompatible arguments, CamlTk will have several versions of the command with different names.

For the time being the best information sources are:

- the .mli files in the `lib` directory (they also appear in chapter 3). This will give you at least the functions for each widget class.
- the `Widgets.src` file of the distribution.

1.16 Debugging

Some Tk functions may have been improperly implemented in the CamlTk library. This may cause undue Tk errors (exception `TkError`, or sometimes `Invalid_Argument`). To facilitate the debugging, you can set the Unix environment variable `CAMLTKDEBUG` to any value before launching your program. This will allow you to see all data transferred between the Caml Light and the Tk processes. Since this data is essentially TclTk code, you need a basic knowledge of this language to understand what is going on. It is also possible to trigger debugging by setting the boolean reference `protocol__debug` to true (or false to remove debugging).

1.17 Changes from previous versions of CamlTk for Tk4.0

There has been really a lot of changes.

1.18 Changes from previous versions of CamlTk for Tk3.6

This list attempts to describe the incompatible changes, but is probably not exhaustive.

- Tk 4.0 is already not fully compatible with Tk 3.6, so these incompatibilities also apply here. The most visible changes concern "scrolling callbacks".
- the `xEvent` type has been modified and fix to account for some events that were not describable in previous versions
- `support__new_toplevel_widget` has disappeared. The creation of new `toplevel` widget is not exactly the same as for other widgets.
- indices for all widget classes have been merged. In particular, `menu` and `text` widgets now use the common indices.
- some widget commands have different names, for better homogeneity.
- `textvariables` are now initialized to an empty string at creation time.

Chapter 2

Implementor's Manual

Tk is still evolving, and also offers numerous user-contributed extensions. To ensure the compatibility of CamlTk with this evolution, we chose to divide the library in two parts:

- a protocol layer for communication between Caml Light and Tcl,
- implementation of Tk commands generated from a high-level description of available widgets and functions.

This chapter defines the syntax and semantics of the CamlTk description language, as well as some technical details of the protocol.

2.1 The CamlTk description language

The language allows the specification of

widgets of the standard Tk library, with options and associated commands,

functions not associated with a specific widget class

modules grouping families of functions.

A source file is simply a collection of entries, each entry describing an element of the interface. The `tkcompiler` provided in this distribution takes the source file and produces Caml Light modules forming the interface library.

2.1.1 Tk tokens

In the heart of the interface, Caml Light needs to call TclTk procedures with arguments. Essentially, arguments are lists of tokens (strings). In some cases an argument will itself be a list of strings. It is mandatory that the descriptions of types and procedure respects the "tokenization" of arguments.

The syntax for describing translations respecting tokens is called a *template*. A *template* is a list of tokens, each token being

a string, e.g. `"foobar"`, used as is by Tk.

a type, e.g. Color. The translation of a value of this type may result in several tokens

a list of tokens

2.1.2 Types

A type entry contains information to translate data from Caml Light to Tk and the converse. Each data type that needs to be exchanged by Caml Light and Tk must be described by a type entry. For each type, a conversion function from Caml Light to Tk is generated, as well as a parsing function from Tk to Caml Light if required (because values of the type are returned by some function in the library) and feasible.

A type is given by a set of value constructors, as in Caml Light. For each constructor, one must give the Caml Light name, and its template.

Example

```
type State {
   Normal ["normal"]
   Active ["active"]
   Disabled ["disabled"]
}
```

defines the Caml Light type

```
type State =
          Disabled
        | Active
        | Normal
;;
```

and the conversion function

```
let cCAMLtoTKState = function
Disabled -> TkToken"disabled"
| Active -> TkToken"active"
| Normal -> TkToken"normal"
;;
```

The parser can be generated only in restricted cases: zeroary constructors, and at most one anonymous `int` constructor and one anonymous `string` constructor (anonymous means that the template is composed only with this token). Otherwise, the compiler will produce a warning message, and you have to write the parser yourself.

2.1.3 Subtypes

As customary, types will be statically verified in Caml Light programs. However, in order to reduce the number of value constructors required by the interface (and associated naming problems), we introduced a notion of *subtypes*. A subtype is a named subset of the set of constructors of a type. In this case, type-checking (i.e. verifiying that a constructor belongs to a subtype) will occur at run-time.

If a type requires subtyping, then one should not declare the type itself, but instead define each of its subtypes. The compiler will consider the whole set of constructors for the definition of the type. Note also that the definition of a constructor may be omitted if it has already been declared.

Example

```
subtype option(standard) {
   ActiveBackground ["-activebackground"; Color]
   ActiveBorderWidth ["-activeborderwidth"; Units]
   ActiveForeground ["-activeforeground"; Color]
   Anchor ["-anchor"; Anchor]
   ...
   }
subtype option (bitmap) {
   Anchor
   Background
   ...
   }
```

defines two subtypes (`standard` and `bitmap`) of the type `option`.

Subtypes may be referred to (in argument types of functions) with the same syntax: `type(subtype)`.

Example

```
function () delete [widget(listbox); "delete"; Index(listbox); Index(listbox)]
```

2.1.4 Widgets

A widget class description is composed of its name, the set of valid options for the creation of widgets of this class, and the set of functions and commands associated to the class.

Options descriptions follow the same syntax as value constructors in type or subtype declarations, except for the presence of the `option` keyword. Actually, the options declared inside a widget description form a subtype (with the name of the class) of the `option` type. `option` is not really hardwired in the compiler, in that it obeys the same rules for subtyping as user-defined types. However the `option` type is implicitly used when producing the widget creation functions.

Function declarations are formed of

- the `function` keyword
- the result type of the function (inside parenthesis)
- the Caml Light name of the function
- the template of arguments

The Caml Light function will have as many arguments as types appearing in the template, in the same order.

Example

```
widget message {
   option Anchor
   option Background
   option Borderwidth
   option Cursor
   option Font
   option Foreground
   option PadX
   option PadY
   option Relief
   option Text
   option TextVariable
   option Width

   option Aspect ["-aspect"; int]
   option Justify ["-justify"; Justification]

   function () configure [widget(message); "configure"; option(message) list]
   function (string) configure_get [widget(message); "configure"]
   }
```

Note that the first options are given implicitly, since they belong to "standard" options, defined elsewhere. The two others are given in their full form.

Each widget class is compiled into a separate module (bearing the name of the widget class). Besides the functions described in the entry, the compiler produces two creation functions

```
value create : Widget -> option list -> Widget ;;
value create_named : Widget -> string -> option list -> Widget ;;
```

with, in fact, run-time verification of options, who must belong to the subtype `option(widgetname)`. The first argument of `create` is the parent of the created widget. The second argument of `create_named` is the new path component of the widget.

2.1.5 Modules

A module is simply a set of functions to be grouped in a separate module.

Example

```
module selection {
   function () clear ["selection"; "clear"; widget]
   function (string) get ["selection"; "get"]
   }
```

2.1.6 Builtins

Some datatypes cannot be described in the syntax of the interface description language, because they require custom converters. Thus it is possible to write them directly in Caml Light and place them in a `builtin_*.ml` file. To avoid a warning message from the compiler, one may also declare the type as external

Example

```
type Units external     # builtin_GetPixel
```

For each builtin type `foo`, one should provide a `cCAMLtoTKfoo` function, of type `foo -> protocol__TkArgs`, respecting the tokenization. If data of this type is to be returned by a function, one should also write a parser `cTKtoCAMLfoo`. For example, the file `builtin_GetPixel.ml` contains the `Units` type, used extensively in Tk for specifying distances or coordinates. It requires both a pretty-printer and a parser (which emulates the `tkGetPixel` function in the Tk library).

2.2 Compiling

The interface description source is compiled with

```
$ tkcompiler filename
```

(the file compiled defaults to `Widgets.src` if `filename` is omitted). The compiler requires the existence of a `lib` subdirectory, where it will produce `tkgen.ml`, `modules`, and various Caml Light files corresponding to entries in the interface description source.

The compiler will report the following errors

lexical errors

syntax errors see the grammar of the language below

duplicate definitions

illegal implicit use of constructors

cyclic dependancy on types

Producing the library requires the presence of several files in the `lib` directory. For more details, check the `Makefile` in `lib`.

2.3 Run-time support

Apart from the modules produced by compiling the widget description file, the CamlTk interface uses low-level support for some basic data types and communication protocol between Caml Light and Tcl.

2.3.1 Widget support

Widget naming : in the contrary of the Tcl/Tk approach, CamlTk keeps the naming of widgets internal to the library. Moreover, there is no particular data associated to widgets in the Caml Light half of the world. A widget is essentially its Tk path. The public functions are:

```
value default_toplevel_widget : Widget
       (* The default toplevel widget is ".", of type "toplevel" *)
;;
value widget_atom : Widget -> string -> Widget
;;
```

And some functions used internally are

```
value widget_name : Widget -> string
      (* Return the name (tk "path") of a widget *)
and  widget_class : Widget-> string
      (* Returns the class of a widget *)
;;
value new_widget_atom : string -> Widget -> Widget
and   new_named_widget : string -> Widget -> string -> Widget
      (* Abstract creation functions *)
;;
```

Widget typing : CamlTk attempts to check widget types before sending code to Tcl. We chose to have a unique Caml Light type for widgets, `Widget`, and to check dynamically when necessary if a widget is of the required class. Thus, internally,

```
type Widget =
  Untyped of string
| Typed of string * string
;;
```

We also maintain a table associating *widget paths* to *Widgets*. The reason for this table is that we may have to dynamically retype a widget path returned by a Tcl function.

2.3.2 Transferring control

Caml Light has been extended with primitives for evaluating Tcl procedures.

tcl_eval evaluates a Tcl source phrase. The implementation of `tcl_eval` is straightforward. During this evaluation, Tcl will perform all customary substitutions and argument parsing. Thus, one needs to be careful with Tcl special characters. This function is deprecated, and `tcl_direct_eval` is preferred.

tcl_direct_eval evaluates a Tcl procedure call. It takes as argument a value of type `TkArgs vect`, which could be thought of as an `argv` parameter. Note however that the vector may still contain token lists that have to be expanded.

Tcl has been extended with a new command `camlcb` to call Caml Light (or more precisely a given Caml Light function). `camlcb` relies on the `callback` primitive of the Caml Light runtime. Currently, the Caml Light runtime does not allow storage of Caml Light values in C global variables, because global variables cannot be roots for the garbage collector. The implementation of `camlcb` uses the same trick as signal handling.

2.3.3 Other CamlTk primitives

The other primitives, written in C, added to the Caml Light runtime deal with

initialising a Tcl interpreter : straightforward

file descriptor callbacks : the mechanism of file descriptor callbacks is available in Tk, although not as Tcl commands. We simply provide a Caml Light primitive to deal with this form of callbacks.

2.3.4 Transferring data

We provide *extensible buffers* to simplify the production of the Tcl source code to be evaluated by Tcl.

2.3.5 Callback handling

Callbacks are Caml Light functions. They have to be invoked by Tcl, using the `camlcb` command. The link between Tcl and Caml Light is made through a hash-table, associating an identifier (string) to a Caml Light function.

In order to avoid space leaks due to storing callback closures in the table, we implement a memo-mechanism associating callbacks to widgets. Except for class wide callbacks, each callback is defined for a given widget. When this widget is destroyed, the callback cannot be invoked any more, and we can remove it from the table. We thus provide a general binding for all widgets, on event `Destroy`, that will remove all callbacks associated to a widget being destroyed. If a user bindings masks the default binding for the `Destroy` event, its callback should call the function `protocol__remove_callbacks` with argument the widget being destroyed.

Chapter 3

Reference Manual

3.1 tk (builtin functions): tk initialisation functions, event bindings, extensions

```
value openTk : unit -> widget
value openTkClass : string -> widget
value openTkDisplayClass : string -> string -> widget
```

Initialisation of the Tk interface (should be called only once). All functions will return the toplevel widget. `openTkClass "class"` lets you specify the (resource) class of the toplevel widget. `openTkDisplayClass "display" "class"` lets you specify the display and the resource class.

```
value mainLoop : unit -> unit
```

Enter the main loop for user interaction

```
value closeTk : unit -> unit
```

Terminates the user-interface interaction

```
value bind : widget -> (modifier list * xEvent) list -> bindAction -> unit
```

Binding events to widgets

```
value class_bind : string ->
      (modifier list * xEvent) list -> bindAction -> unit
```

binding events for widget classes

```
value tag_bind : string ->
       (modifier list * xEvent) list -> bindAction -> unit
```

binding events for widget classes

```
value add_fileinput : file_descr -> (unit -> unit) -> unit
value remove_fileinput : file_descr -> unit
value add_fileoutput : file_descr -> (unit -> unit) -> unit
value remove_fileoutput : file_descr -> unit
```

Callbacks on Unix file descriptors

```
value  add_timer : int -> (unit -> unit) -> timer
value    remove_timer : timer -> unit
```

Schedules callbacks

3.2 `tk (other functions)`: user defined functions

```
value update_idletasks : unit ->unit
```

tk invocation: update idletasks

```
value update : unit ->unit
```

tk invocation: update

```
value appname_get : unit ->string
```

tk invocation: tk appname

```
value appname_set : string -> unit
```

tk invocation: tk appname `string`

`value raise_window_above : widget -> widget -> unit`

tk invocation: raise `widget widget`

`value raise_window : widget -> unit`

tk invocation: raise `widget`

`value place : widget -> options list -> unit`

tk invocation: place `widget options list`

`value pack : widget list -> options list -> unit`

tk invocation: pack `widget list options list`

`value lower_window_below : widget -> widget -> unit`

tk invocation: lower `widget widget`

`value lower_window : widget -> unit`

tk invocation: lower `widget`

`value destroy : widget -> unit`

tk invocation: destroy `widget`

`value bindtags_get : widget -> bindings list`

tk invocation: bindtags `widget`

`value bindtags : widget -> bindings list -> unit`

tk invocation: bindtags `widget bindings list`

`value cget : widget -> options_constrs -> string`

tk invocation: `widget` cget `options_constrs`

`value cgets : widget -> string -> string`

tk invocation: `widget` cget `string`

3.3 tk (types): types

Builtin types are:

```
type bitmap =
   BitmapFile of string                 (* path of file *)
 | Predefined of string                 (* bitmap  name *)
;;
type color =
     NamedColor of string
   | Black                      (* tk keyword: black *)
   | White                      (* tk keyword: white *)
   | Red                        (* tk keyword: red *)
   | Green                      (* tk keyword: green *)
   | Blue                       (* tk keyword: blue *)
   | Yellow                     (* tk keyword: yellow *)
;;
type cursor =
   XCursor of string
 | XCursorFg of string * color
 | XCursortFgBg of string * color * color
 | CursorFileFg of string * color
 | CursorMaskFile of string * string * color * color
;;
type units =
    Pixels of int       (* specified as floating-point, but inconvenient *)
  | Centimeters of float
  | Inches of float
  | Millimeters of float
  | PrinterPoint of float
;;
type scrollValue =
        ScrollPage of int               (* tk option: scroll <int> page *)
        | ScrollUnit of int             (* tk option: scroll <int> unit *)
        | MoveTo of float               (* tk option: moveto <float> *)
;;
type xEvent =
    ButtonPress (* also Button, but we omit it *)
  | ButtonPressDetail of int
  | ButtonRelease
  | ButtonReleaseDetail of int
  | Circulate
  | ColorMap
  | Configure
  | Destroy
  | Enter
```

```
  | Expose
  | FocusIn
  | FocusOut
  | Gravity
  | KeyPress (* also Key, but we omit it *)
  | KeyPressDetail of string      (* /usr/include/X11/keysymdef.h *)
  | KeyRelease
  | KeyReleaseDetail of string
  | Leave
  | Map
  | Motion
  | Property
  | Reparent
  | Unmap
  | Visibility
;;
type modifier =
    Control
  | Shift
  | Lock
  | Button1
  | Button2
  | Button3
  | Button4
  | Button5
  | Double
  | Triple
  | Mod1
  | Mod2
  | Mod3
  | Mod4
  | Mod5
  | Meta
  | Alt
;;
type eventInfo =
  {
  mutable ev_Above : int;               (* tk: %a *)
  mutable ev_ButtonNumber : int;        (* tk: %b *)
  mutable ev_Count : int;               (* tk: %c *)
  mutable ev_Detail : string;           (* tk: %d *)
  mutable ev_Focus : bool;              (* tk: %f *)
  mutable ev_Height : int;              (* tk: %h *)
  mutable ev_KeyCode : int;             (* tk: %k *)
  mutable ev_Mode : string;             (* tk: %m *)
```

```
  mutable ev_OverrideRedirect : bool;   (* tk: %o *)
  mutable ev_Place : string;            (* tk: %p *)
  mutable ev_State : string;            (* tk: %s *)
  mutable ev_Time : int;                (* tk: %t *)
  mutable ev_Width : int;               (* tk: %w *)
  mutable ev_MouseX : int;              (* tk: %x *)
  mutable ev_MouseY : int;              (* tk: %y *)
  mutable ev_Char : string;             (* tk: %A *)
  mutable ev_BorderWidth : int;         (* tk: %B *)
  mutable ev_SendEvent : bool;          (* tk: %E *)
  mutable ev_KeySymString : string;     (* tk: %K *)
  mutable ev_KeySymInt : int;           (* tk: %N *)
  mutable ev_RootWindow : int;          (* tk: %R *)
  mutable ev_SubWindow : int;           (* tk: %S *)
  mutable ev_Type : int;                (* tk: %T *)
  mutable ev_Widget : widget;           (* tk: %W *)
  mutable ev_RootX : int;               (* tk: %X *)
  mutable ev_RootY : int                (* tk: %Y *)
  }
;;
type eventField =
    Ev_Above
  | Ev_ButtonNumber
  | Ev_Count
  | Ev_Detail
  | Ev_Focus
  | Ev_Height
  | Ev_KeyCode
  | Ev_Mode
  | Ev_OverrideRedirect
  | Ev_Place
  | Ev_State
  | Ev_Time
  | Ev_Width
  | Ev_MouseX
  | Ev_MouseY
  | Ev_Char
  | Ev_BorderWidth
  | Ev_SendEvent
  | Ev_KeySymString
  | Ev_KeySymInt
  | Ev_RootWindow
  | Ev_SubWindow
  | Ev_Type
  | Ev_Widget
```

```
  | Ev_RootX
  | Ev_RootY
;;
type bindAction =
   BindSet of eventField list *  (eventInfo -> unit)
 | BindSetBreakable of eventField list *  (eventInfo -> unit)
 | BindRemove
 | BindExtend of eventField list *  (eventInfo -> unit)
;;
type bindings =
        TagBindings of string           (* tk option: <string> *)
        | WidgetBindings of widget             (* tk option: <widget> *)
;;
type index =
          Number of int         (* no keyword  *)
        | ActiveElement         (* tk keyword: active *)
        | End                   (* tk keyword: end *)
        | Last                  (* tk keyword: last *)
        | NoIndex               (* tk keyword: none *)
        | Insert                (* tk keyword: insert *)
        | SelFirst              (* tk keyword: sel.first *)
        | SelLast               (* tk keyword: sel.last *)
        | At of int             (* tk keyword: @n *)
        | AtXY of int * int     (* tk keyword: @x,y *)
        | AnchorPoint           (* tk keyword: anchor *)
        | Pattern of string     (* no keyword *)
        | LineChar of int * int (* tk keyword: l.c *)
        | Mark of string        (* no keyword *)
        | TagFirst of string    (* tk keyword: tag.first *)
        | TagLast of string     (* tk keyword: tag.last *)
        | Embedded of widget    (* no keyword *)
;;
type paletteType =
    GrayShades of int
  | RGBShades of int * int * int
;;
type textMark == string
;;
type textTag == string
;;
type textModifier =
    CharOffset of int           (* tk keyword: +/- Xchars *)
  | LineOffset of int           (* tk keyword: +/- Xlines *)
  | LineStart                   (* tk keyword: linestart *)
  | LineEnd                     (* tk keyword: lineend *)
```

```
  | WordStart                   (* tk keyword: wordstart *)
  | WordEnd                     (* tk keyword: wordend *)
;;
type textIndex =
   TextIndex of index * textModifier list
 | TextIndexNone
;;
type relief =
        Flat            (* tk option: flat *)
        | Groove                (* tk option: groove *)
        | Raised                (* tk option: raised *)
        | Ridge         (* tk option: ridge *)
        | Sunken                (* tk option: sunken *)
;;
type orientation =
        Horizontal              (* tk option: horizontal *)
        | Vertical              (* tk option: vertical *)
;;
type justification =
        Justify_Center          (* tk option: center *)
        | Justify_Left          (* tk option: left *)
        | Justify_Right         (* tk option: right *)
;;
type imagePhoto =
        PhotoImage of string            (* tk option: [string] *)
;;
type imageBitmap =
        BitmapImage of string           (* tk option: [string] *)
;;
type anchor =
        Center          (* tk option: center *)
        | E             (* tk option: e *)
        | N             (* tk option: n *)
        | NE            (* tk option: ne *)
        | NW            (* tk option: nw *)
        | S             (* tk option: s *)
        | SE            (* tk option: se *)
        | SW            (* tk option: sw *)
        | W             (* tk option: w *)
;;
type state =
        Active          (* tk option: active *)
        | Disabled              (* tk option: disabled *)
        | Normal                (* tk option: normal *)
;;
```

```
type colorMode =
        Color           (* tk option: color *)
        | Gray          (* tk option: gray *)
        | Mono          (* tk option: mono *)
;;
type tagOrId =
        Id of int               (* tk option: [int] *)
        | Tag of string         (* tk option: [string] *)
;;
type arcStyle =
        Arc             (* tk option: arc *)
        | Chord         (* tk option: chord *)
        | PieSlice              (* tk option: pieslice *)
;;
type joinStyle =
        Join_Bevel              (* tk option: bevel *)
        | Join_Miter            (* tk option: miter *)
        | Join_Round            (* tk option: round *)
;;
type capStyle =
        Cap_Butt                (* tk option: butt *)
        | Cap_Projecting                (* tk option: projecting *)
        | Cap_Round             (* tk option: round *)
;;
type arrowStyle =
        Arrow_Both              (* tk option: both *)
        | Arrow_First           (* tk option: first *)
        | Arrow_Last            (* tk option: last *)
        | Arrow_None            (* tk option: none *)
;;
type colormap =
        NewColormap             (* tk option: new *)
        | WidgetColormap of widget              (* tk option: [widget] *)
;;
type visual =
        Best            (* tk option: best *)
        | BestDepth of int              (* tk option: {best [int]} *)
        | ClassVisual of string * int           (* tk option: {[string] [int]} *)
        | DefaultVisual         (* tk option: default *)
        | WidgetVisual of widget                (* tk option: [widget] *)
;;
type selectModeType =
        Browse          (* tk option: browse *)
        | Extended              (* tk option: extended *)
        | Multiple              (* tk option: multiple *)
```

```
        | Single                (* tk option: single *)
;;
type side =
        Side_Bottom             (* tk option: bottom *)
        | Side_Left             (* tk option: left *)
        | Side_Right            (* tk option: right *)
        | Side_Top              (* tk option: top *)
;;
type fillMode =
        Fill_Both               (* tk option: both *)
        | Fill_None             (* tk option: none *)
        | Fill_X                (* tk option: x *)
        | Fill_Y                (* tk option: y *)
;;
type borderMode =
        Ignore          (* tk option: ignore *)
        | Inside                (* tk option: inside *)
        | Outside               (* tk option: outside *)
;;
type alignType =
        Align_Baseline          (* tk option: baseline *)
        | Align_Bottom          (* tk option: bottom *)
        | Align_Center          (* tk option: center *)
        | Align_Top             (* tk option: top *)
;;
type tabType =
        TabCenter of units              (* tk option: [units] center *)
        | TabLeft of units              (* tk option: [units] left *)
        | TabNumeric of units           (* tk option: [units] numeric *)
        | TabRight of units             (* tk option: [units] right *)
;;
type wrapMode =
        WrapChar                (* tk option: char *)
        | WrapNone              (* tk option: none *)
        | WrapWord              (* tk option: word *)
;;
type options =
        Accelerator of string           (* tk option: -accelerator [string] *)
        | ActiveBackground of color             (* tk option: -activebackground [color] *)
        | ActiveBorderWidth of units            (* tk option: -activeborderwidth [units] *)
        | ActiveForeground of color             (* tk option: -activeforeground [color] *)
        | ActiveRelief of relief                (* tk option: -activerelief [relief] *)
        | After of widget               (* tk option: -after [widget] *)
        | Align of alignType            (* tk option: -align [alignType] *)
        | Anchor of anchor              (* tk option: -anchor [anchor] *)
```

```
| ArcStyle of arcStyle          (* tk option: -style [arcStyle] *)
| ArrowShape of units * units * units           (* tk option: -arrowshape {[units] [u
| ArrowStyle of arrowStyle              (* tk option: -arrow [arrowStyle] *)
| Aspect of int         (* tk option: -aspect [int] *)
| Background of color           (* tk option: -background [color] *)
| Before of widget              (* tk option: -before [widget] *)
| BgStipple of bitmap           (* tk option: -bgstipple [bitmap] *)
| BigIncrement of float         (* tk option: -bigincrement [float] *)
| Bitmap of bitmap              (* tk option: -bitmap [bitmap] *)
| BorderMode of borderMode              (* tk option: -bordermode [borderMode] *)
| BorderWidth of units          (* tk option: -borderwidth [units] *)
| CapStyle of capStyle          (* tk option: -capstyle [capStyle] *)
| Class of string               (* tk option: -class [string] *)
| CloseEnough of float          (* tk option: -closeenough [float] *)
| Colormap of colormap          (* tk option: -colormap [colormap] *)
| Colormode of colorMode                (* tk option: -colormode [colorMode] *)
| Command of (unit -> unit)             (* tk option: -command [(unit -> unit)] *)
| Confine of bool               (* tk option: -confine [bool] *)
| Cursor of cursor              (* tk option: -cursor [cursor] *)
| Data of string                (* tk option: -data [string] *)
| Digits of int         (* tk option: -digits [int] *)
| DisabledForeground of color           (* tk option: -disabledforeground [color] *)
| ElementBorderWidth of units           (* tk option: -elementborderwidth [units] *)
| Expand of bool                (* tk option: -expand [bool] *)
| ExportSelection of bool               (* tk option: -exportselection [bool] *)
| Extent of float               (* tk option: -extent [float] *)
| FgStipple of bitmap           (* tk option: -fgstipple [bitmap] *)
| File of string                (* tk option: -file [string] *)
| Fill of fillMode              (* tk option: -fill [fillMode] *)
| FillColor of color            (* tk option: -fill [color] *)
| Font of string                (* tk option: -font [string] *)
| Foreground of color           (* tk option: -foreground [color] *)
| Format of string              (* tk option: -format [string] *)
| From of float         (* tk option: -from [float] *)
| Gamma of float                (* tk option: -gamma [float] *)
| Geometry of string            (* tk option: -geometry [string] *)
| Height of units               (* tk option: -height [units] *)
| HighlightBackground of color          (* tk option: -highlightbackground [color] *
| HighlightColor of color               (* tk option: -highlightcolor [color] *)
| HighlightThickness of units           (* tk option: -highlightthickness [units] *)
| IPadX of units                (* tk option: -ipadx [units] *)
| IPadY of units                (* tk option: -ipady [units] *)
| ImageBitmap of imageBitmap            (* tk option: -image [imageBitmap] *)
| ImagePhoto of imagePhoto              (* tk option: -image [imagePhoto] *)
| In of widget          (* tk option: -in [widget] *)
```

```
| IndicatorOn of bool           (* tk option: -indicatoron [bool] *)
| InsertBackground of color             (* tk option: -insertbackground [color] *)
| InsertBorderWidth of units            (* tk option: -insertborderwidth [units] *)
| InsertOffTime of int          (* tk option: -insertofftime [int] *)
| InsertOnTime of int           (* tk option: -insertontime [int] *)
| InsertWidth of units          (* tk option: -insertwidth [units] *)
| JoinStyle of joinStyle                (* tk option: -joinstyle [joinStyle] *)
| Jump of bool          (* tk option: -jump [bool] *)
| Justify of justification              (* tk option: -justify [justification] *)
| LMargin1 of units             (* tk option: -lmargin1 [units] *)
| LMargin2 of units             (* tk option: -lmargin2 [units] *)
| Label of string               (* tk option: -label [string] *)
| Length of units               (* tk option: -length [units] *)
| Maskdata of string            (* tk option: -maskdata [string] *)
| Maskfile of string            (* tk option: -maskfile [string] *)
| Menu of widget                (* tk option: -menu [widget] *)
| OffValue of string            (* tk option: -offvalue [string] *)
| Offset of units               (* tk option: -offset [units] *)
| OnValue of string             (* tk option: -onvalue [string] *)
| Orient of orientation         (* tk option: -orient [orientation] *)
| Outline of color              (* tk option: -outline [color] *)
| OutlineStipple of bitmap              (* tk option: -outlinestipple [bitmap] *)
| OverStrike of bool            (* tk option: -overstrike [bool] *)
| PadX of units         (* tk option: -padx [units] *)
| PadY of units         (* tk option: -pady [units] *)
| PageAnchor of anchor          (* tk option: -pageanchor [anchor] *)
| PageHeight of units           (* tk option: -pageheight [units] *)
| PageWidth of units            (* tk option: -pagewidth [units] *)
| PageX of units                (* tk option: -pagex [units] *)
| PageY of units                (* tk option: -pagey [units] *)
| Palette of paletteType                (* tk option: -palette [paletteType] *)
| PostCommand of (unit -> unit)         (* tk option: -postcommand [(unit -> unit)]
| RMargin of units              (* tk option: -rmargin [units] *)
| RelHeight of float            (* tk option: -relheight [float] *)
| RelWidth of float             (* tk option: -relwidth [float] *)
| RelX of float         (* tk option: -relx [float] *)
| RelY of float         (* tk option: -rely [float] *)
| Relief of relief              (* tk option: -relief [relief] *)
| RepeatDelay of int            (* tk option: -repeatdelay [int] *)
| RepeatInterval of int         (* tk option: -repeatinterval [int] *)
| Resolution of float           (* tk option: -resolution [float] *)
| Rotate of bool                (* tk option: -rotate [bool] *)
| ScaleCommand of (float -> unit)               (* tk option: -command [(float -> un
| Screen of string              (* tk option: -screen [string] *)
| ScrollCommand of (scrollValue -> unit)                (* tk option: -command [(scr
```

```
| ScrollRegion of units * units * units * units        (* tk option: -scrollregion
| SelectBackground of color            (* tk option: -selectbackground [color] *)
| SelectBorderWidth of units           (* tk option: -selectborderwidth [units] *)
| SelectColor of color         (* tk option: -selectcolor [color] *)
| SelectForeground of color            (* tk option: -selectforeground [color] *)
| SelectImageBitmap of imageBitmap             (* tk option: -selectimage [imageBit
| SelectImagePhoto of imagePhoto               (* tk option: -selectimage [imagePho
| SelectMode of selectModeType         (* tk option: -selectmode [selectModeType] *
| SetGrid of bool              (* tk option: -setgrid [bool] *)
| Show of char         (* tk option: -show [char] *)
| ShowValue of bool            (* tk option: -showvalue [bool] *)
| Side of side         (* tk option: -side [side] *)
| SliderLength of units        (* tk option: -sliderlength [units] *)
| Smooth of bool               (* tk option: -smooth [bool] *)
| Spacing1 of units            (* tk option: -spacing1 [units] *)
| Spacing2 of units            (* tk option: -spacing2 [units] *)
| Spacing3 of units            (* tk option: -spacing3 [units] *)
| SplineSteps of int           (* tk option: -splinesteps [int] *)
| Start of float               (* tk option: -start [float] *)
| State of state               (* tk option: -state [state] *)
| Stipple of bitmap            (* tk option: -stipple [bitmap] *)
| Stretch of bool              (* tk option: -stretch [bool] *)
| Tabs of tabType list         (* tk option: -tabs {[tabType list]} *)
| Tags of tagOrId list         (* tk option: -tags {[tagOrId list]} *)
| TakeFocus of bool            (* tk option: -takefocus [bool] *)
| TearOff of bool              (* tk option: -tearoff [bool] *)
| Text of string               (* tk option: -text [string] *)
| TextHeight of int            (* tk option: -height [int] *)
| TextVariable of textVariable         (* tk option: -textvariable [textVariable] *
| TextWidth of int             (* tk option: -width [int] *)
| ThroughColor of color        (* tk option: -troughcolor [color] *)
| TickInterval of float        (* tk option: -tickinterval [float] *)
| To of float          (* tk option: -to [float] *)
| Underline of bool            (* tk option: -underline [bool] *)
| UnderlinedChar of int        (* tk option: -underline [int] *)
| Value of string              (* tk option: -value [string] *)
| Variable of textVariable             (* tk option: -variable [textVariable] *)
| Visual of visual             (* tk option: -visual [visual] *)
| Width of units               (* tk option: -width [units] *)
| Window of widget             (* tk option: -window [widget] *)
| Wrap of wrapMode             (* tk option: -wrap [wrapMode] *)
| WrapLength of units          (* tk option: -wraplength [units] *)
| X of units           (* tk option: -x [units] *)
| XScrollCommand of (float -> float -> unit)           (* tk option: -xscrollcomman
| XScrollIncrement of units            (* tk option: -xscrollincrement [units] *)
```

```
        | Y of units            (* tk option: -y [units] *)
        | YScrollCommand of (float -> float -> unit)            (* tk option: -yscrollcomman
        | YScrollIncrement of units             (* tk option: -yscrollincrement [units] *)
;;
type options_constrs =
        CAccelerator
        | CActiveBackground
        | CActiveBorderWidth
        | CActiveForeground
        | CActiveRelief
        | CAfter
        | CAlign
        | CAnchor
        | CArcStyle
        | CArrowShape
        | CArrowStyle
        | CAspect
        | CBackground
        | CBefore
        | CBgStipple
        | CBigIncrement
        | CBitmap
        | CBorderMode
        | CBorderWidth
        | CCapStyle
        | CClass
        | CCloseEnough
        | CColormap
        | CColormode
        | CCommand
        | CConfine
        | CCursor
        | CData
        | CDigits
        | CDisabledForeground
        | CElementBorderWidth
        | CExpand
        | CExportSelection
        | CExtent
        | CFgStipple
        | CFile
        | CFill
        | CFillColor
        | CFont
        | CForeground
```

```
| CFormat
| CFrom
| CGamma
| CGeometry
| CHeight
| CHighlightBackground
| CHighlightColor
| CHighlightThickness
| CIPadX
| CIPadY
| CImageBitmap
| CImagePhoto
| CIn
| CIndicatorOn
| CInsertBackground
| CInsertBorderWidth
| CInsertOffTime
| CInsertOnTime
| CInsertWidth
| CJoinStyle
| CJump
| CJustify
| CLMargin1
| CLMargin2
| CLabel
| CLength
| CMaskdata
| CMaskfile
| CMenu
| COffValue
| COffset
| COnValue
| COrient
| COutline
| COutlineStipple
| COverStrike
| CPadX
| CPadY
| CPageAnchor
| CPageHeight
| CPageWidth
| CPageX
| CPageY
| CPalette
| CPostCommand
```

```
| CRMargin
| CRelHeight
| CRelWidth
| CRelX
| CRelY
| CRelief
| CRepeatDelay
| CRepeatInterval
| CResolution
| CRotate
| CScaleCommand
| CScreen
| CScrollCommand
| CScrollRegion
| CSelectBackground
| CSelectBorderWidth
| CSelectColor
| CSelectForeground
| CSelectImageBitmap
| CSelectImagePhoto
| CSelectMode
| CSetGrid
| CShow
| CShowValue
| CSide
| CSliderLength
| CSmooth
| CSpacing1
| CSpacing2
| CSpacing3
| CSplineSteps
| CStart
| CState
| CStipple
| CStretch
| CTabs
| CTags
| CTakeFocus
| CTearOff
| CText
| CTextHeight
| CTextVariable
| CTextWidth
| CThroughColor
| CTickInterval
```

```
        | CTo
        | CUnderline
        | CUnderlinedChar
        | CValue
        | CVariable
        | CVisual
        | CWidth
        | CWindow
        | CWrap
        | CWrapLength
        | CX
        | CXScrollCommand
        | CXScrollIncrement
        | CY
        | CYScrollCommand
        | CYScrollIncrement
;;(* /type *)

let options_any_table = [CAccelerator; CActiveBackground; CActiveBorderWidth; CActiveForegro
;;
type searchSpec =
        Above of tagOrId                (* tk option: above [tagOrId] *)
        | All           (* tk option: all *)
        | Below of tagOrId              (* tk option: below [tagOrId] *)
        | Closest of units * units              (* tk option: closest [units] [units] *)
        | ClosestHalo of units * units * units          (* tk option: closest [units] [units
        | ClosestHaloStart of units * units * units * tagOrId           (* tk option: closes
        | Enclosed of units * units * units * units             (* tk option: enclosed [unit
        | Overlapping of int * int * int * int          (* tk option: overlapping [int] [int
        | Withtag of tagOrId            (* tk option: withtag [tagOrId] *)
;;
type canvasItem =
        Arc_item                (* tk option: arc *)
        | Bitmap_item           (* tk option: bitmap *)
        | Image_item            (* tk option: image *)
        | Line_item             (* tk option: line *)
        | Oval_item             (* tk option: oval *)
        | Polygon_item          (* tk option: polygon *)
        | Rectangle_item                (* tk option: rectangle *)
        | Text_item             (* tk option: text *)
        | User_item of string           (* tk option: [string] *)
        | Window_item           (* tk option: window *)
;;
type icccm =
        DisplayOf of widget             (* tk option: -displayof [widget] *)
```

```
        | ICCCMFormat of string         (* tk option: -format [string] *)
        | ICCCMType of string           (* tk option: -type [string] *)
        | LostCommand of (unit -> unit)         (* tk option: -command [(unit -> unit)] *)
        | Selection of string           (* tk option: -selection [string] *)
;;
type grabStatus =
        GrabGlobal              (* tk option: global *)
        | GrabLocal             (* tk option: local *)
        | GrabNone              (* tk option: none *)
;;
type menuItem =
        Cascade_Item            (* tk option: cascade *)
        | Checkbutton_Item              (* tk option: checkbutton *)
        | Command_Item          (* tk option: command *)
        | Radiobutton_Item              (* tk option: radiobutton *)
        | Separator_Item                (* tk option: separator *)
        | TearOff_Item          (* tk option: tearoff *)
;;
type optionPriority =
        Interactive             (* tk option: interactive *)
        | Priority of int               (* tk option: [int] *)
        | StartupFile           (* tk option: startupFile *)
        | UserDefault           (* tk option: userDefault *)
        | WidgetDefault         (* tk option: widgetDefault *)
;;
type photo =
        ImgFormat of string             (* tk option: -format [string] *)
        | ImgFrom of int * int * int * int              (* tk option: -from [int] [int] [int
        | ImgTo of int * int * int * int                (* tk option: -to [int] [int] [int]
        | Shrink                (* tk option: -shrink *)
        | Subsample of int * int                (* tk option: -subsample [int] [int] *)
        | TopLeft of int * int          (* tk option: -to [int] [int] *)
        | Zoom of int * int             (* tk option: -zoom [int] [int] *)
;;
type widgetElement =
        Arrow1          (* tk option: arrow1 *)
        | Arrow2                (* tk option: arrow2 *)
        | Beyond                (* tk option:  *)
        | Slider                (* tk option: slider *)
        | Trough1               (* tk option: trough1 *)
        | Trough2               (* tk option: trough2 *)
;;
type comparison =
        EQ              (* tk option: == *)
        | GE            (* tk option: >= *)
```

```
        | GT            (* tk option: > *)
        | LE            (* tk option: <= *)
        | LT            (* tk option: < *)
        | NEQ           (* tk option: != *)
;;
type markDirection =
        Mark_Left               (* tk option: left *)
        | Mark_Right            (* tk option: right *)
;;
type textSearch =
        Backwards               (* tk option: -backwards *)
        | Count of textVariable         (* tk option: -count [textVariable] *)
        | Exact         (* tk option: -exact *)
        | Forwards              (* tk option: -forwards *)
        | Nocase                (* tk option: -nocase *)
        | Regexp                (* tk option: -regexp *)
;;
type atomId =
        AtomId of int           (* tk option: [int] *)
;;
type focusModel =
        FocusActive             (* tk option: active *)
        | FocusPassive          (* tk option: passive *)
;;
type wmFrom =
        FromProgram             (* tk option: program *)
        | FromUser              (* tk option: user *)
;;
```

3.4 `bell`: the bell commands

```
value ring : unit ->unit
```

tk invocation: bell

```
value ring_displayof : widget -> unit
```

tk invocation: bell -displayof `widget`

3.5 button: the button widget

```
value create : widget -> options list -> widget
```

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

```
value create_named : widget -> string -> options list -> widget
```

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

```
value configure : widget -> options list -> unit
```

tk invocation: `widget` configure `options list`

```
value configure_get : widget -> string
```

tk invocation: `widget` configure

```
value flash : widget -> unit
```

tk invocation: `widget` flash

```
value invoke : widget -> unit
```

tk invocation: `widget` invoke

3.6 canvas: the canvas widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value addtag : widget -> tagOrId -> searchSpec list -> unit`

tk invocation: `widget` addtag `tagOrId searchSpec list`

`value bbox : widget -> tagOrId list -> int * int * int * int`

tk invocation: `widget` bbox `tagOrId list`

`value canvasx : widget -> units -> float`

tk invocation: `widget` canvasx `units`

`value canvasx_grid : widget -> units -> units -> float`

tk invocation: `widget` canvasx `units units`

`value canvasy : widget -> units -> float`

tk invocation: `widget` canvasy `units`

`value canvasy_grid : widget -> units -> units -> float`

tk invocation: `widget` canvasy `units units`

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_arc : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_bitmap : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value configure_image : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_line : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_oval : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_polygon : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list` `value configure_rectangle : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_text : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value configure_window : widget -> tagOrId -> options list -> unit`

tk invocation: `widget` itemconfigure `tagOrId` `options list`

`value coords_get : widget -> tagOrId -> float list`

tk invocation: `widget` coords `tagOrId`

`value coords_set : widget -> tagOrId -> units list -> unit`

tk invocation: `widget` coords `tagOrId` `units list`

`value create_arc : widget -> units -> units -> units -> units -> options list -> tagOrId`

tk invocation: `widget` create arc `units` `units` `units` `units` `options list`

`value create_bitmap : widget -> units -> units -> options list -> tagOrId`

tk invocation: `widget` create bitmap `units` `units` `options list`

`value create_image : widget -> units -> units -> options list -> tagOrId`

tk invocation: `widget` create image `units` `units` `options list` `value create_line : widget -> units list -> options list -> tagOrId`

tk invocation: `widget` create line `units list` `options list`

`value create_oval : widget -> units -> units -> units -> units -> options list -> tagOrId`

tk invocation: `widget` create oval `units` `units` `units` `units` `options list`

`value create_polygon : widget -> units list -> options list -> tagOrId`

tk invocation: `widget` create polygon `units list` `options list`

`value create_rectangle : widget -> units -> units -> units -> units -> options list -> tagOr`

tk invocation: `widget` create rectangle `units` `units` `units` `units` `options list`

`value create_text : widget -> units -> units -> options list -> tagOrId`

tk invocation: `widget` create text `units` `units` `options list`

`value create_window : widget -> units -> units -> options list -> tagOrId`

tk invocation: `widget` create window `units` `units` `options list`

`value dchars : widget -> tagOrId -> index -> index -> unit`

tk invocation: `widget` dchars `tagOrId` `index` `index`

`value delete : widget -> tagOrId list -> unit`

tk invocation: `widget` delete `tagOrId list` `value dtag : widget -> tagOrId -> tagOrId -> unit`

tk invocation: `widget` dtag `tagOrId` `tagOrId`

`value find : widget -> searchSpec list -> tagOrId list`

tk invocation: `widget` find `searchSpec list`

`value focus : widget -> tagOrId -> unit`

tk invocation: `widget` focus `tagOrId`

`value focus_get : widget -> tagOrId`

tk invocation: `widget` focus

`value focus_reset : widget -> unit`

tk invocation: `widget` focus

`value gettags : widget -> tagOrId -> tagOrId list`

tk invocation: `widget` gettags `tagOrId`

`value icursor : widget -> tagOrId -> index -> unit`

tk invocation: `widget` icursor `tagOrId` `index`

`value index : widget -> tagOrId -> index -> int`

tk invocation: `widget` index `tagOrId` `index` `value insert : widget -> tagOrId -> index -> string -> unit`

tk invocation: `widget` insert `tagOrId` `index` `string`

`value itemconfigure_get : widget -> tagOrId -> string`

tk invocation: `widget` itemconfigure `tagOrId`

`value lower_below : widget -> tagOrId -> tagOrId -> unit`

tk invocation: `widget` lower `tagOrId` `tagOrId`

`value lower_bot : widget -> tagOrId -> unit`

tk invocation: `widget` lower `tagOrId`

`value move : widget -> tagOrId -> units -> units -> unit`

tk invocation: `widget` move `tagOrId` `units` `units`

`value postscript : widget -> options list -> string`

tk invocation: `widget` postscript `options list`

`value raise_above : widget -> tagOrId -> tagOrId -> unit`

tk invocation: `widget` raise `tagOrId` `tagOrId`

`value raise_top : widget -> tagOrId -> unit`

tk invocation: `widget` raise `tagOrId`

```
value scale : widget -> tagOrId -> units -> units -> float -> float -> unit
```

tk invocation: `widget` scale `tagOrId` `units` `units` `float` `float`

```
value scan_dragto : widget -> int -> int -> unit
```

tk invocation: `widget` scan dragto `int` `int`

```
value scan_mark : widget -> int -> int -> unit
```

tk invocation: `widget` scan mark `int` `int`

```
value select_adjust : widget -> tagOrId -> index -> unit
```

tk invocation: `widget` select adjust `tagOrId` `index`

```
value select_clear : widget -> unit
```

tk invocation: `widget` select clear

```
value select_from : widget -> tagOrId -> index -> unit
```

tk invocation: `widget` select from `tagOrId` `index`

```
value select_item : widget -> tagOrId
```

tk invocation: `widget` select item

```
value select_to : widget -> tagOrId -> index -> unit
```

tk invocation: `widget` select to `tagOrId` `index`

```
value typeof : widget -> tagOrId -> canvasItem
```

tk invocation: `widget` type `tagOrId`

```
value xview : widget -> scrollValue -> unit
```

tk invocation: `widget` xview `scrollValue`

```
value xview_get : widget -> float * float
```

tk invocation: `widget` xview

```
value yview : widget -> scrollValue -> unit
```

tk invocation: `widget` yview `scrollValue`

```
value yview_get : widget -> float * float
```

tk invocation: `widget` yview

```
value bind : support__widget -> tagOrId ->
                   (modifier list * xEvent) list -> bindAction -> unit
```

3.7 `checkbutton`: the checkbutton widget

```
value create : widget -> options list -> widget
```

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

```
value create_named : widget -> string -> options list -> widget
```

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value deselect : widget -> unit`

tk invocation: `widget` deselect

`value flash : widget -> unit`

tk invocation: `widget` flash

`value invoke : widget -> unit`

tk invocation: `widget` invoke

`value select : widget -> unit`

tk invocation: `widget` select

`value toggle : widget -> unit`

tk invocation: `widget` toggle

3.8 `clipboard`: the clipboard commands

`value append : icccm list -> string -> unit`

tk invocation: clipboard append `icccm list` – `string`

`value clear : icccm list -> unit`

tk invocation: clipboard clear `icccm list`

3.9 entry: the entry widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value delete_range : widget -> index -> index -> unit`

tk invocation: `widget` delete `index index`

`value delete_single : widget -> index -> unit`

tk invocation: `widget` delete `index`

`value get : widget -> string`

tk invocation: `widget` get

`value icursor : widget -> index -> unit`

tk invocation: `widget` icursor `index`

`value index : widget -> index -> int`

tk invocation: `widget` index `index`

`value insert : widget -> index -> string -> unit`

tk invocation: `widget` insert `index` `string`

`value scan_dragto : widget -> int -> unit`

tk invocation: `widget` scan dragto `int`

`value scan_mark : widget -> int -> unit`

tk invocation: `widget` scan mark `int`

`value selection_adjust : widget -> index -> unit`

tk invocation: `widget` selection adjust `index`

`value selection_clear : widget -> unit`

tk invocation: `widget` selection clear

`value selection_from : widget -> index -> unit`

tk invocation: `widget` selection from `index`

`value selection_present : widget -> bool`

tk invocation: `widget` selection present `value selection_range : widget -> index -> index -> unit`

tk invocation: `widget` selection range `index` `index`

`value selection_to : widget -> index -> unit`

tk invocation: `widget` selection to `index`

`value xview : widget -> scrollValue -> unit`

tk invocation: `widget` xview `scrollValue`

`value xview_get : widget -> float * float`

tk invocation: `widget` xview

`value xview_index : widget -> index -> unit`

tk invocation: `widget` xview `index`

3.10 `focus`: the focus commands

`value displayof : widget -> widget`

tk invocation: focus -displayof `widget`

`value follows_mouse : unit ->unit`

tk invocation: tk_focusFollowsMouse

`value force : widget -> unit`

tk invocation: focus -force `widget`

`value get : unit ->widget`

tk invocation: focus

`value lastfor : widget -> widget`

tk invocation: focus -lastfor `widget`

`value next : widget -> widget`

tk invocation: tk_focusNext `widget`

`value prev : widget -> widget`

tk invocation: tk_focusPrev `widget`

`value set : widget -> unit`

tk invocation: focus `widget`

3.11 `frame`: the frame widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

3.12 grab: the grab commands

`value all_current : unit ->widget list`

tk invocation: grab current

`value current : widget -> widget list`

tk invocation: grab current `widget`

`value release : widget -> unit`

tk invocation: grab release `widget`

`value set : widget -> unit`

tk invocation: grab `widget`

`value set_global : widget -> unit`

tk invocation: grab -global `widget`

`value status : widget -> grabStatus`

tk invocation: grab status `widget`

3.13 imagebitmap: the imagebitmap commands

`value configure : imageBitmap -> options list -> unit`

tk invocation: `imageBitmap` configure `options list`

`value configure_get : imageBitmap -> string`

tk invocation: `imageBitmap` configure

`value create : options list -> imageBitmap`

tk invocation: image create bitmap `options list`

`value create_named : imageBitmap -> options list -> imageBitmap`

tk invocation: image create bitmap `imageBitmap` `options list`

`value delete : imageBitmap -> unit`

tk invocation: image delete `imageBitmap`

`value height : imageBitmap -> int`

tk invocation: image height `imageBitmap`

`value width : imageBitmap -> int`

tk invocation: image width `imageBitmap`

3.14 `imagephoto`: the imagephoto commands

`value blank : imagePhoto -> unit`

tk invocation: `imagePhoto` blank

`value configure : imagePhoto -> options list -> unit`

tk invocation: `imagePhoto` configure `options list`

`value configure_get : imagePhoto -> string`

tk invocation: `imagePhoto` configure

`value copy : imagePhoto -> imagePhoto -> photo list -> unit`

tk invocation: `imagePhoto` copy `imagePhoto` `photo list`

`value create : options list -> imagePhoto`

tk invocation: image create photo `options list`

`value delete : imagePhoto -> unit`

tk invocation: image delete `imagePhoto`

`value get : imagePhoto -> int -> int -> int * int * int`

tk invocation: `imagePhoto` get `int` `int`

`value height : imagePhoto -> int`

tk invocation: image height `imagePhoto`

`value read : imagePhoto -> string -> photo list -> unit`

tk invocation: `imagePhoto` read `string` `photo list`

`value redither : imagePhoto -> unit`

tk invocation: `imagePhoto` redither

`value width : imagePhoto -> int`

tk invocation: image width `imagePhoto`

`value write : imagePhoto -> photo list -> unit`

tk invocation: `imagePhoto` write `photo list`

3.15 `label`: the label widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

3.16 `listbox`: the listbox widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value activate : widget -> index -> unit`

tk invocation: `widget` activate `index` `value bbox : widget -> index -> int * int * int * int`

tk invocation: `widget` bbox `index`

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value curselection : widget -> index list`

tk invocation: `widget` curselection

`value delete : widget -> index -> index -> unit`

tk invocation: `widget` delete `index index`

`value get : widget -> index -> string`

tk invocation: `widget` get `index`

`value get_range : widget -> index -> index -> string list`

tk invocation: `widget` get `index index`

`value index : widget -> index -> int`

tk invocation: `widget` index `index`

`value insert : widget -> index -> string list -> unit`

tk invocation: `widget` insert `index` `string list`

`value nearest : widget -> int -> index`

tk invocation: `widget` nearest `int`

`value scan_dragto : widget -> int -> int -> unit`

tk invocation: `widget` scan dragto `int` `int`

`value scan_mark : widget -> int -> int -> unit`

tk invocation: `widget` scan mark `int` `int`

`value see : widget -> index -> unit`

tk invocation: `widget` see `index`

`value selection_anchor : widget -> index -> unit`

tk invocation: `widget` selection anchor `index`

`value selection_clear : widget -> index -> index -> unit`

tk invocation: `widget` selection clear `index` `index`

`value selection_includes : widget -> index -> bool`

tk invocation: `widget` selection includes `index` `value selection_set : widget -> index -> index -> unit`

tk invocation: `widget` selection set `index index`

`value size : widget -> int`

tk invocation: `widget` size

`value xview : widget -> scrollValue -> unit`

tk invocation: `widget` yview `scrollValue`

`value xview_get : widget -> float * float`

tk invocation: `widget` xview

`value xview_index : widget -> index -> unit`

tk invocation: `widget` xview `index`

`value yview : widget -> scrollValue -> unit`

tk invocation: `widget` yview `scrollValue`

`value yview_get : widget -> float * float`

tk invocation: `widget` yview

`value yview_index : widget -> index -> unit`

tk invocation: `widget` yview `index`

3.17 menu: the menu widget

```
value create : widget -> options list -> widget
```

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

```
value create_named : widget -> string -> options list -> widget
```

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

```
value activate : widget -> index -> unit
```

tk invocation: `widget` activate `index`

```
value add_cascade : widget -> options list -> unit
```

tk invocation: `widget` add cascade `options list`

```
value add_checkbutton : widget -> options list -> unit
```

tk invocation: `widget` add checkbutton `options list`

```
value add_command : widget -> options list -> unit
```

tk invocation: `widget` add command `options list`

```
value add_radiobutton : widget -> options list -> unit
```

tk invocation: `widget` add radiobutton `options list`

```
value add_separator : widget -> unit
```

tk invocation: `widget` add separator

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_cascade : widget -> index -> options list -> unit`

tk invocation: `widget` entryconfigure `index options list`

`value configure_checkbutton : widget -> index -> options list -> unit`

tk invocation: `widget` entryconfigure `index options list`

`value configure_command : widget -> index -> options list -> unit`

tk invocation: `widget` entryconfigure `index options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value configure_radiobutton : widget -> index -> options list -> unit`

tk invocation: `widget` entryconfigure `index options list`

`value delete : widget -> index -> index -> unit`

tk invocation: `widget` delete `index index`

`value entryconfigure_get : widget -> index -> string`

tk invocation: `widget` entryconfigure `index` `value index : widget -> index -> int`

tk invocation: `widget` index `index`

`value insert_cascade : widget -> index -> options list -> unit`

tk invocation: `widget` insert `index` cascade `options list`

`value insert_checkbutton : widget -> index -> options list -> unit`

tk invocation: `widget` insert `index` checkbutton `options list`

`value insert_command : widget -> index -> options list -> unit`

tk invocation: `widget` insert `index` command `options list`

`value insert_radiobutton : widget -> index -> options list -> unit`

tk invocation: `widget` insert `index` radiobutton `options list`

`value insert_separator : widget -> index -> unit`

tk invocation: `widget` insert `index` separator

`value invoke : widget -> index -> string`

tk invocation: `widget` invoke `index`

`value popup : widget -> int -> int -> unit`

tk invocation: tk_popup `widget int int` `value popup_entry : widget -> int -> int -> index -> unit`

tk invocation: tk_popup `widget int int index`

`value post : widget -> int -> int -> unit`

tk invocation: `widget` post `int int`

`value postcascade : widget -> index -> unit`

tk invocation: `widget` postcascade `index`

`value typeof : widget -> index -> menuItem`

tk invocation: `widget` type `index`

`value unpost : widget -> unit`

tk invocation: `widget` unpost

`value yposition : widget -> index -> int`

tk invocation: `widget` yposition `index`

3.18 menubutton: the menubutton widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

3.19 message: the message widget

`value create : widget -> options list -> widget`

> `create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

> `create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

> tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

> tk invocation: `widget` configure

3.20 pack: the pack commands

`value configure : widget list -> options list -> unit`

> tk invocation: pack configure `widget list` `options list`

`value forget : widget list -> unit`

> tk invocation: pack forget `widget list`

`value propagate_get : widget -> bool`

> tk invocation: pack propagate `widget`

`value propagate_set : widget -> bool -> unit`

> tk invocation: pack propagate `widget` `bool`

`value slaves : widget -> widget list`

> tk invocation: pack slaves `widget`

3.21 palette: the palette commands

`value bisque : unit ->unit`

tk invocation: tk_bisque

`value set_background : color -> unit`

tk invocation: tk_setPalette `color`

3.22 place: the place commands

`value configure : widget -> options list -> unit`

tk invocation: place configure `widget options list`

`value forget : widget -> unit`

tk invocation: place forget `widget`

`value info : widget -> string`

tk invocation: place info `widget`

`value slaves : widget -> widget list`

tk invocation: place slaves `widget`

3.23 radiobutton: the radiobutton widget

```
value create : widget -> options list -> widget
```

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

```
value create_named : widget -> string -> options list -> widget
```

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

```
value configure : widget -> options list -> unit
```

tk invocation: `widget` configure `options list`

```
value configure_get : widget -> string
```

tk invocation: `widget` configure

```
value deselect : widget -> unit
```

tk invocation: `widget` deselect

```
value flash : widget -> unit
```

tk invocation: `widget` flash

```
value invoke : widget -> unit
```

tk invocation: `widget` invoke

```
value select : widget -> unit
```

tk invocation: `widget` select

3.24 resource: the resource commands

`value add : string -> string -> optionPriority -> unit`

tk invocation: option add `string string optionPriority`

`value clear : unit ->unit`

tk invocation: option clear

`value get : widget -> string -> string -> string`

tk invocation: option get `widget string string`

`value readfile : string -> optionPriority -> unit`

tk invocation: option readfile `string optionPriority`

3.25 scale: the scale widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list` `value configure_get : widget -> string`

tk invocation: `widget` configure

`value get : widget -> float`

tk invocation: `widget` get

`value get_xy : widget -> int -> int -> float`

tk invocation: `widget` get `int int`

`value identify : widget -> int -> int -> widgetElement`

tk invocation: `widget int int`

`value set : widget -> float -> unit`

tk invocation: `widget` set `float`

3.26 `scrollbar`: the scrollbar widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value activate : widget -> widgetElement -> unit`

tk invocation: `widget` activate `widgetElement` `value activate_get : widget -> widgetElement`

tk invocation: `widget` activate

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value delta : widget -> int -> int -> float`

tk invocation: `widget` delta `int int`

`value fraction : widget -> int -> int -> float`

tk invocation: `widget` fraction `int int`

`value get : widget -> float * float`

tk invocation: `widget` get

`value identify : widget -> int -> int -> widgetElement`

tk invocation: `widget` identify `int int`

`value old_get : widget -> int * int * int * int`

tk invocation: `widget` get

`value old_set : widget -> int -> int -> int -> int -> unit`

tk invocation: `widget` set `int int int int`

`value set : widget -> float -> float -> unit`

tk invocation: `widget` set `float float`

3.27 `selection`: the selection commands

`value clear : icccm list -> unit`

tk invocation: selection clear `icccm list`

`value get : icccm list -> string`

tk invocation: selection get `icccm list`

`value own_get : icccm list -> widget`

tk invocation: selection own `icccm list`

`value handle_set : icccm list -> widget -> (int -> int -> unit) -> unit`

tk invocation: selection handle ¡icccm list¿ ¡widget¿ ¡command¿

`value own_set : icccm list -> widget -> unit`

tk invocation: selection own ¡icccm list¿ ¡widget¿

3.28 `text`: the text widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value bbox : widget -> textIndex -> int * int * int * int`

tk invocation: `widget` bbox `textIndex`

`value compare : widget -> textIndex -> comparison -> textIndex -> bool`

tk invocation: `widget` compare `textIndex` `comparison` `textIndex`

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

`value debug : widget -> bool -> unit`

tk invocation: `widget` debug `bool`

`value delete : widget -> textIndex -> textIndex -> unit`

tk invocation: `widget` delete `textIndex` `textIndex`

`value delete_char : widget -> textIndex -> unit`

tk invocation: `widget` delete `textIndex`

`value dlineinfo : widget -> textIndex -> int * int * int * int * int`

tk invocation: `widget` dlineinfo `textIndex` `value get : widget -> textIndex -> textIndex -> string`

tk invocation: `widget` get `textIndex` `textIndex`

`value get_char : widget -> textIndex -> string`

tk invocation: `widget` get `textIndex`

`value index : widget -> textIndex -> index`

tk invocation: `widget` index `textIndex`

`value insert : widget -> textIndex -> string -> textTag list -> unit`

tk invocation: `widget` insert `textIndex` `string` `textTag list`

`value mark_gravity_get : widget -> textMark -> markDirection`

tk invocation: `widget` mark gravity `textMark`

`value mark_gravity_set : widget -> textMark -> markDirection -> unit`

tk invocation: `widget` mark gravity `textMark` `markDirection`

`value mark_names : widget -> textMark list`

tk invocation: `widget` mark names

`value mark_set : widget -> textMark -> textIndex -> unit`

tk invocation: `widget` mark set `textMark` `textIndex` `value mark_unset : widget -> textMark list -> unit`

tk invocation: `widget` mark unset `textMark list`

`value scan_dragto : widget -> int -> int -> unit`

tk invocation: `widget` scan dragto `int int`

`value scan_mark : widget -> int -> int -> unit`

tk invocation: `widget` scan mark `int int`

`value search : widget -> textSearch list -> string -> textIndex -> textIndex -> index`

tk invocation: `widget` search `textSearch list` – `string textIndex textIndex`

`value see : widget -> textIndex -> unit`

tk invocation: `widget` see `textIndex`

`value tag_add : widget -> textTag -> textIndex -> textIndex -> unit`

tk invocation: `widget` tag add `textTag textIndex textIndex`

`value tag_add_char : widget -> textTag -> textIndex -> unit`

tk invocation: `widget` tag add `textTag textIndex`

`value tag_allnames : widget -> textTag list`

tk invocation: `widget` tag names

`value tag_configure : widget -> textTag -> options list -> unit`

tk invocation: `widget` tag configure `textTag` `options list`

`value tag_delete : widget -> textTag list -> unit`

tk invocation: `widget` tag delete `textTag list`

`value tag_indexnames : widget -> textIndex -> textTag list`

tk invocation: `widget` tag names `textIndex`

`value tag_lower_below : widget -> textTag -> textTag -> unit`

tk invocation: `widget` tag lower `textTag` `textTag`

`value tag_lower_bot : widget -> textTag -> unit`

tk invocation: `widget` tag lower `textTag`

`value tag_nextrange : widget -> textTag -> textIndex -> textIndex -> index * index`

tk invocation: `widget` tag nextrange `textTag` `textIndex` `textIndex`

`value tag_raise_above : widget -> textTag -> textTag -> unit`

tk invocation: `widget` tag raise `textTag` `textTag`

`value tag_raise_top : widget -> textTag -> unit`

tk invocation: `widget` tag raise `textTag` `value tag_ranges : widget -> textTag -> index list`

tk invocation: `widget` tag ranges `textTag`

`value tag_remove : widget -> textTag -> textIndex -> textIndex -> unit`

tk invocation: `widget` tag remove `textTag` `textIndex` `textIndex`

`value tag_remove_char : widget -> textTag -> textIndex -> unit`

tk invocation: `widget` tag remove `textTag` `textIndex`

`value window_configure : widget -> textTag -> options list -> unit`

tk invocation: `widget` window configure `textTag` `options list`

`value window_create : widget -> textIndex -> options list -> unit`

tk invocation: `widget` window create `textIndex` `options list`

`value window_names : widget -> widget list`

tk invocation: `widget` window names

`value xview : widget -> scrollValue -> unit`

tk invocation: `widget` xview `scrollValue`

`value xview_get : widget -> float * float`

tk invocation: `widget` xview

```
value yview : widget -> scrollValue -> unit
```

tk invocation: `widget` yview `scrollValue`

```
value yview_get : widget -> float * float
```

tk invocation: `widget` yview

```
value yview_index : widget -> textIndex -> unit
```

tk invocation: `widget` yview `textIndex`

```
value yview_index_pickplace : widget -> textIndex -> unit
```

tk invocation: `widget` yview -pickplace `textIndex`

```
value yview_line : widget -> int -> unit
```

tk invocation: `widget` yview `int`

```
value tag_bind: support__widget -> textTag ->
                       (modifier list * xEvent) list -> bindAction -> unit
```

3.29 `tkwait`: the tkwait commands

```
value variable : textVariable -> unit
```

tk invocation: tkwait variable `textVariable`

```
value visibility : widget -> unit
```

tk invocation: tkwait visibility `widget`

```
value window : widget -> unit
```

tk invocation: tkwait window `widget`

3.30 toplevelw: the toplevel widget

`value create : widget -> options list -> widget`

`create p options` creates a new widget with parent p. Options are restricted to the widget class subset, and checked dynamically.

`value create_named : widget -> string -> options list -> widget`

`create p name options` creates a new widget with parent p and new patch component name. Options are restricted to the widget class subset, and checked dynamically.

`value configure : widget -> options list -> unit`

tk invocation: `widget` configure `options list`

`value configure_get : widget -> string`

tk invocation: `widget` configure

3.31 winfo: the winfo commands

`value atom : string -> atomId`

tk invocation: winfo atom `string`

`value atom_displayof : widget -> string -> atomId`

tk invocation: winfo atom -displayof `widget string`

`value atomname : atomId -> string`

tk invocation: winfo atomname `atomId`

`value atomname_displayof : widget -> atomId -> string`

tk invocation: winfo atomname -displayof `widget atomId`

`value cells : widget -> int`

tk invocation: winfo cells `widget`

`value children : widget -> widget list`

tk invocation: winfo children `widget`

`value class_name : widget -> string`

tk invocation: winfo class `widget`

`value colormapfull : widget -> bool`

tk invocation: winfo colormapfull `widget`

`value containing : int -> int -> widget`

tk invocation: winfo containing `int int`

`value containing_displayof : widget -> int -> int -> widget`

tk invocation: winfo containing -displayof `widget int int`

`value depth : widget -> int`

tk invocation: winfo depth `widget`

`value exists : widget -> bool`

tk invocation: winfo exists `widget`

`value fpixels : widget -> units -> float`

tk invocation: winfo fpixels `widget units`

`value geometry : widget -> string`

tk invocation: winfo geometry `widget`

`value height : widget -> int`

tk invocation: winfo height `widget`

`value id : widget -> string`

tk invocation: winfo id `widget`

`value interps : unit ->string list`

tk invocation: winfo interps

`value interps_displayof : widget -> string list`

tk invocation: winfo interps -displayof `widget`

`value ismapped : widget -> bool`

tk invocation: winfo ismapped `widget`

`value manager : widget -> string`

tk invocation: winfo manager `widget`

`value name : widget -> string`

tk invocation: winfo name `widget`

`value parent : widget -> widget`

tk invocation: winfo parent `widget`

`value pathname : string -> widget`

tk invocation: winfo pathname `string`

`value pathname_displayof : widget -> string -> widget`

tk invocation: winfo pathname -displayof `widget` `string`

`value pixels : widget -> units -> int`

tk invocation: winfo pixels `widget` `units`

`value pointerx : widget -> int`

tk invocation: winfo pointerx `widget`

`value pointerxy : widget -> int * int`

tk invocation: winfo pointerxy `widget`

`value pointery : widget -> int`

tk invocation: winfo pointery `widget`

`value reqheight : widget -> int`

tk invocation: winfo reqheight `widget`

`value reqwidth : widget -> int`

tk invocation: winfo reqwidth `widget`

`value rgb : widget -> color -> int * int * int`

tk invocation: winfo rgb `widget color`

`value rootx : widget -> int`

tk invocation: winfo rootx `widget`

`value rooty : widget -> int`

tk invocation: winfo rooty `widget`

`value screen : widget -> string`

tk invocation: winfo screen `widget`

`value screencells : widget -> int`

tk invocation: winfo screencells `widget` `value screendepth : widget -> int`

tk invocation: winfo screendepth `widget`

`value screenheight : widget -> int`

tk invocation: winfo screenheight `widget`

`value screenmmdepth : widget -> int`

tk invocation: winfo screenmmdepth `widget`

`value screenmmheight : widget -> int`

tk invocation: winfo screenmmheight `widget`

`value screenvisual : widget -> string`

tk invocation: winfo screenvisual `widget`

`value screenwidth : widget -> int`

tk invocation: winfo screenwidth `widget`

`value server : widget -> string`

tk invocation: winfo server `widget`

`value toplevel : widget -> widget`

tk invocation: winfo toplevel `widget`

`value viewable : widget -> bool`

tk invocation: winfo viewable `widget`

`value visual : widget -> string`

tk invocation: winfo visual `widget`

`value visualsavailable : widget -> string`

tk invocation: winfo visualsavailable `widget`

`value vrootheight : widget -> int`

tk invocation: winfo vrootheight `widget`

`value vrootwidth : widget -> int`

tk invocation: winfo vrootwidth `widget`

`value vrootx : widget -> int`

tk invocation: winfo vrootx `widget`

`value vrooty : widget -> int`

tk invocation: winfo vrooty `widget`

`value width : widget -> int`

tk invocation: winfo width `widget`

`value x : widget -> int`

tk invocation: winfo x `widget`

`value y : widget -> int`

tk invocation: winfo y `widget`

`value contained : int -> int -> widget -> bool`

`contained x y w` returns true if (x,y) is in w

3.32 `wm`: the wm commands

`value aspect_get : widget -> int * int * int * int`

tk invocation: wm aspect `widget`

`value aspect_set : widget -> int -> int -> int -> int -> unit`

tk invocation: wm aspect `widget int int int int`

`value client_get : widget -> string`

tk invocation: wm client `widget`

`value client_set : widget -> string -> unit`

tk invocation: wm client `widget string`

`value colormapwindows_get : widget -> widget list`

tk invocation: wm colormapwindows `widget`

`value colormapwindows_set : widget -> widget list -> unit`

tk invocation: wm colormapwindows `widget widget list`

`value command_clear : widget -> unit`

tk invocation: wm command `widget`

`value command_get : widget -> string list`

tk invocation: wm command `widget`

`value command_set : widget -> string list -> unit`

tk invocation: wm command `widget string list`

`value deiconify : widget -> unit`

tk invocation: wm deiconify `widget`

`value focusmodel_get : widget -> focusModel`

tk invocation: wm focusmodel `widget`

`value focusmodel_set : widget -> focusModel -> unit`

tk invocation: wm focusmodel `widget focusModel`

`value frame : widget -> string`

tk invocation: wm frame `widget`

`value geometry_get : widget -> string`

tk invocation: wm geometry `widget`

`value geometry_set : widget -> string -> unit`

tk invocation: wm geometry `widget string`

`value grid_clear : widget -> unit`

tk invocation: wm grid `widget`

`value grid_get : widget -> int * int * int * int`

tk invocation: wm grid `widget`

`value grid_set : widget -> int -> int -> int -> int -> unit`

tk invocation: wm grid `widget int int int int`

`value group_clear : widget -> unit`

tk invocation: wm group `widget`

`value group_get : widget -> widget`

tk invocation: wm group `widget`

`value group_set : widget -> widget -> unit`

tk invocation: wm group `widget widget`

`value iconbitmap_clear : widget -> unit`

tk invocation: wm iconbitmap `widget`

`value iconbitmap_get : widget -> bitmap`

tk invocation: wm iconbitmap `widget`

`value iconbitmap_set : widget -> bitmap -> unit`

tk invocation: wm iconbitmap `widget bitmap` `value iconify : widget -> unit`

tk invocation: wm iconify `widget`

`value iconmask_clear : widget -> unit`

tk invocation: wm iconmask `widget`

`value iconmask_get : widget -> bitmap`

tk invocation: wm iconmask `widget`

`value iconmask_set : widget -> bitmap -> unit`

tk invocation: wm iconmask `widget` `bitmap`

`value iconname_get : widget -> string`

tk invocation: wm iconname `widget`

`value iconname_set : widget -> string -> unit`

tk invocation: wm iconname `widget` `string`

`value iconposition_clear : widget -> unit`

tk invocation: wm iconposition `widget`

`value iconposition_get : widget -> int * int`

tk invocation: wm iconposition `widget`

`value iconposition_set : widget -> int -> int -> unit`

tk invocation: wm iconposition `widget int int`

`value iconwindow_clear : widget -> unit`

tk invocation: wm iconwindow `widget`

`value iconwindow_get : widget -> widget`

tk invocation: wm iconwindow `widget`

`value iconwindow_set : widget -> widget -> unit`

tk invocation: wm iconwindow `widget widget`

`value maxsize_get : widget -> int * int`

tk invocation: wm maxsize `widget`

`value maxsize_set : widget -> int -> int -> unit`

tk invocation: wm maxsize `widget int int`

`value minsize_get : widget -> int * int`

tk invocation: wm minsize `widget`

`value minsize_set : widget -> int -> int -> unit`

tk invocation: wm minsize `widget int int` `value overrideredirect_get : widget -> bool`

tk invocation: wm overrideredirect `widget`

`value overrideredirect_set : widget -> bool -> unit`

tk invocation: wm overrideredirect `widget` `bool`

`value positionfrom_clear : widget -> unit`

tk invocation: wm positionfrom `widget`

`value positionfrom_get : widget -> wmFrom`

tk invocation: wm positionfrom `widget`

`value positionfrom_set : widget -> wmFrom -> unit`

tk invocation: wm positionfrom `widget` `wmFrom`

`value protocol_clear : widget -> string -> unit`

tk invocation: wm protocol `widget` `string`

`value protocol_set : widget -> string -> (unit -> unit) -> unit`

tk invocation: wm protocol `widget` `string` `(unit -> unit)`

`value protocols : widget -> string list`

tk invocation: wm protocol `widget`

`value resizable_get : widget -> bool * bool`

tk invocation: wm resizable `widget`

`value resizable_set : widget -> bool -> bool -> unit`

tk invocation: wm resizable `widget` `bool` `bool`

`value sizefrom_clear : widget -> unit`

tk invocation: wm sizefrom `widget`

`value sizefrom_get : widget -> wmFrom`

tk invocation: wm sizefrom `widget`

`value sizefrom_set : widget -> wmFrom -> unit`

tk invocation: wm sizefrom `widget` `wmFrom`

`value state : widget -> string`

tk invocation: wm state `widget`

`value title_get : widget -> string`

tk invocation: wm title `widget`

`value title_set : widget -> string -> unit`

tk invocation: wm title `widget` `string`

`value transient_clear : widget -> unit`

tk invocation: wm transient `widget`

`value transient_get : widget -> widget`

tk invocation: wm transient `widget`

`value transient_set : widget -> widget -> unit`

tk invocation: wm transient `widget` `widget`

`value withdraw : widget -> unit`

tk invocation: wm withdraw `widget`

3.33 optionmenu: support for tk_optionMenu

```
value create: widget -> textVariable -> string list -> widget * widget
```

`create parent var options` creates a multi-option menubutton and its associated menu. The option is also stored in the variable. Both widgets (menubutton and menu) are returned

3.34 textvariable: support for Tk -textvariable option

```
type textVariable
```

TextVariable is an abstract type

```
value create : unit -> textVariable
```

Allocation of a TextVariable

```
value   create_temporary : widget -> textVariable
```

Allocation of a textVariable with lifetime associated to widget

```
value   set : textVariable -> string -> unit
```

Setting the value of a TextVariable

```
value   get : textVariable -> string
```

Reading the value of a TextVariable

```
value   name : textVariable -> string
```

Its tcl name

```
value cCAMLtoTKtextVariable : textVariable -> tkArgs
```

Internal conversion function

Bibliography

[1] John K. Ousterhout. *Tcl and the Tk Toolkit.* Number ISBN 0-201-63337-X. Addison-Wesley, 1994.

[2] Brent B. Welch. *Practical Programming in Tcl and Tk.* Number ISBN 0-13-182007-9. Prentice Hall, 1995.

Appendix A

BNF syntax of the interface description language

```
$accept : %entry% $end
TypeName : IDENT
         | WIDGET
Type0 : TYINT
      | TYFLOAT
      | TYBOOL
      | TYCHAR
      | TYSTRING
      | TypeName
Type1 : Type0
      | TypeName LPAREN IDENT RPAREN
      | WIDGET LPAREN IDENT RPAREN
      | OPTION LPAREN IDENT RPAREN
Type2 : Type1
      | Type1 LIST
Type_list : Type2 COMMA Type_list
          | Type2
FType : LPAREN RPAREN
      | LPAREN Type2 RPAREN
      | LPAREN Type_list RPAREN
Type : Type2
     | FUNCTION FType
Arg : STRING
    | Type
    | Template
ArgList : Arg SEMICOLON ArgList
        | Arg
Template : LBRACKET ArgList RBRACKET
Constructor : IDENT Template
AbbrevConstructor : Constructor
```

```
                  | IDENT
Constructors : Constructor Constructors
             | Constructor
AbbrevConstructors : AbbrevConstructor AbbrevConstructors
                   | AbbrevConstructor
Safe :
     | UNSAFE
Command : Safe FUNCTION FType IDENT Template
External : Safe EXTERNAL IDENT STRING
Option : OPTION IDENT Template
       | OPTION IDENT
WidgetComponents :
                 | Command WidgetComponents
                 | Option WidgetComponents
                 | External WidgetComponents
ModuleComponents :
                 | Command ModuleComponents
                 | External ModuleComponents
ParserArity :
            | SEQUENCE
entry : TYPE ParserArity TypeName LBRACE Constructors RBRACE
      | TYPE ParserArity TypeName EXTERNAL
      | SUBTYPE ParserArity OPTION LPAREN IDENT RPAREN LBRACE AbbrevConstructors RBRACE
      | SUBTYPE ParserArity TypeName LPAREN IDENT RPAREN LBRACE AbbrevConstructors RBRACE
      | Command
      | WIDGET IDENT LBRACE WidgetComponents RBRACE
      | MODULE IDENT LBRACE ModuleComponents RBRACE
      | EOF
%entry% : ´\001´ entry
```