The Qi/tk Companion

© Copyright Mark Tarver, 2007.

Chapter 1

Installing Qi/tk

1.1 The Development of Qi/Tk

Qi/tk is an integration of two languages – Qi and the **TCL/tk** language developed by **John Ousterhout**. TCL is short for **T**ool **C**ommand **L**anguage and tk is short for **t**ool **k**it. TCL itself is a fairly straightforward procedural language that can be used for **scripting** (i.e. writing interactive programs to animate web pages) as well as for writing ordinary programs. TCL connects to tk which is a facility for building **graphical user interfaces** (or **GUI**s pronounced 'gooeys') by using tk commands. Basically tk commands create graphical objects or **widgets** such as buttons and menus which, when used, activate procedures written in TCL.

TCL/tk is sometimes grafted on to other languages to provide a graphical component to them and Qi/tk is no exception. But Qi/tk hides many of the awkward syntactic features of TCL/tk. Hence no knowledge of TCL/tk is needed to program in Qi/tk, and the style will feel less procedural than programming in TCL/tk. However many of the function names in Qi/tk are borrowed from tk commands, and so if you are a TCL/tk programmer then Qi/tk should not feel too strange to you. The graphical interface builder in Qi/tk is completely type secure and the graphics programs can be type checked just like any Qi program.

The Qi/tk programming system first ran in 2002 for Qi 1.4 and was used at SUNY for teaching in that same year. The first Qi/tk manual was written in December 2002. It was intended at some time to be published but at that time Qi was still just a personal tool. The system was revised in 2004, 2007 and 2008. Development took second place to several other concerns, principally the completion of Qi itself.

1.2 How to Install Qi/tk

Qi/tk is available from the Qi library in the downloads section of Lambda Associates (www.lambdassociates.org). It includes an installation program for TCL/tk. If you do not have TCL/tk on your computer then run the installation program tcl832 in the folder Qi-tk which installs TCL/tk for Windows. Alternatively you may want to download the latest version of TCL/tk from http://www.tcl.tk/software/tcltk/ and run that.

When TCL/tk is installed you might need to configure the file tcl-lisp.lsp by setting the constant *tk-prog* to the path where the executable TCL/tk resides. The default is (DEFCONSTANT qi-tk::*tk-prog* "C:/PROGRAM FILES/TCL/BIN/WISH83.EXE")

Change the path if necessary to where TCL/tk lives on your computer. To install Qi/tk, copy your executable image of Qi to the Qi/tk folder and start Qi. Then enter (LOAD "install-qi-tk.lsp").

This will save an executable version of Qi/tk. To run it click on the exectable file Qi/tk. When you activate Qi/tk, the following display will appear (figure 1.1).

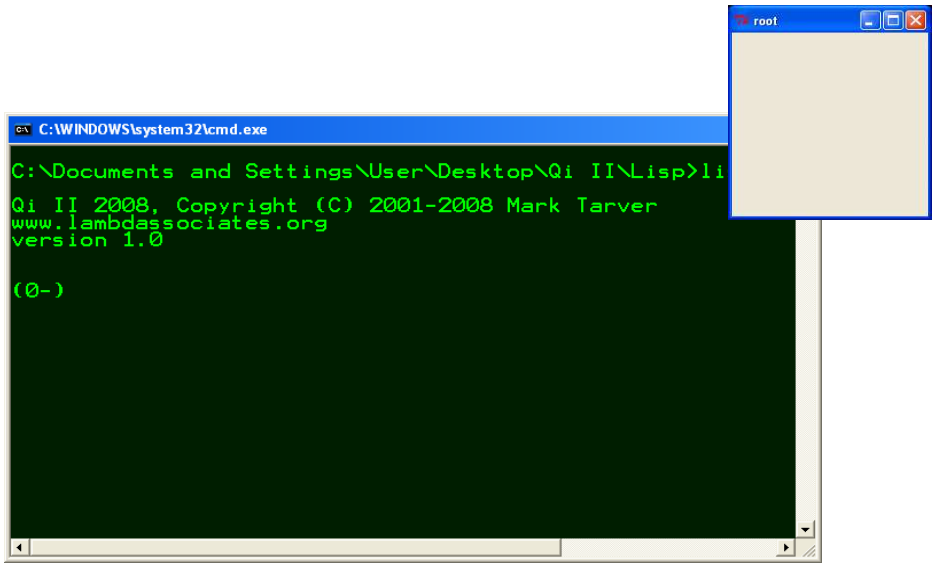


Figure 1.1 The command wndow and the root window

The large window is the Qi command window for the read-evaluate-print loop. The small grey window is the root window for the Tk graphical interface. Qi/tk has a command driven GUI builder. Widgets (graphical objects such as buttons or menus) are built by Qi/tk commands.

1.2 How to Talk to the OS and Foreign Processes in Qi/tk

The functions in tcl-lisp.lsp give Qi/tk the capacity to talk to the operating system and receive and transmit data from foreign processes . The function system communicates a command embedded in a string to the OS.

(9+) system
system : (string --> string)

The command is executed by the OS and the response returned as a string. Thus (system "dir") will, in Windows, return a string containing the listing of the current directory.

The function open-process-stream of type string --> [string] --> stream creates a bidirectional stream between Qi and a foreign process F. The first argument gives the path to the executable file that initiates F and the second argument

gives a list of strings that are passed as arguments to that executable file. That list may be empty. The result returned is a Lisp **bidirectional stream**.

A bidirectional stream is like a wiretap on two telephones. We can tap into each of the phones to listen to what is being said. We can also (and here is the bidirectional bit) talk down the line to each of the phones. Hence we can pass instructions through our wiretap from one process to the other. Apparently one process is talking directly to the other, but in fact the conversation is routed via our wiretap. Since the two processes may run in different languages, our wiretap must translate the language of each process to fit the understanding of the other. When this works, the result is that two apparently foreign processes communicate and collaborate with each other.

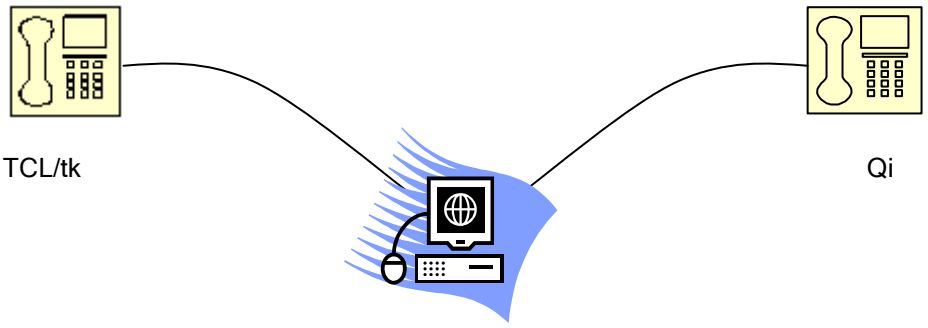


Figure 1.2 The wiretap model of a birectional stream

Example 1

```
(set *tk* (open-process-stream
                "C:/PROGRAM FILES/TCL/BIN/WISH83.EXE" ["-name" "root"]))
```

creates a bidirectional stream between Qi and the process initiated by the command "C:/PROGRAM FILES/TCL/BIN/WISH83.EXE -name root."

The result is a stream which is stored in the global *tk*. By tapping this stream we can listen to and send data from Qi to TCL/tk and from TCL/tk to Qi. In fact the process in example 1.1 is initiated and stored when Qi/tk is run, so you do not have to create it. return-tk-stream of type A --> stream will return the TCL/tk stream that Qi/tk reads. The argument to return-tk-stream is ignored.

Qi/tk contains 2 important functions for listening to and sending data down a bidirectional stream. The relevant functions are talk and listen.

talk is used to send a string to TCL/tk as a command line. This string is executed by TCL/tk, but the response is not returned and the empty list is returned.

Example 2

(talk "bell" (return-tk-stream _)) sends the command to sound the bell to TCL/tk. talk does not wait for the response but simply returns the empty list.

listen waits for a response from the stream. The output returned by process on the other end of the stream (i.e. the process foreign to Qi) is itself returned as a string. The first input to listen is a number measuring in seconds that times out the function if no input from the stream is received. In this case the empty string "" is returned.

Example 3

(6+)(do (talk "puts {34}" (return-tk-stream _)) (listen 1 (return-tk-stream _)))
"34" : string

sends the command for TCL/tk to print 34 and waits for at most 1 second to receive a response. The response is heard and "34" is returned. Rather more convenient is talk-and-listen which combines both functions.

(7+) (talk-and-listen "puts {34}" 1 (return-tk-stream _)))
"34" : string

The function flush-stream of type stream --> [A] will flush away any input from the foreign process that is 'stuck' in the stream waiting for Qi to read it.

Though Qi/tk can be driven from talk and listen, part of the purpose of Qi/tk is to act as a buffer between the user and the lower level details of these transactions. Specifically Qi/tk does the following:

1. Builds a barrier of abstraction between the user and TCL/tk so that learning TCL/tk syntax is not necessary.
2. Avoids the need for fiddly string processing and allows programming in a functional style with Qi list syntax.
3. Provides type security for graphical programs.
4. Provides a top level for running graphics and the read-evaluate-print loop concurrently.

The next chapters are concerned with learning how to use these features.

Chapter 2

Widgets

2.1 Attributes and Values

Though TCL/tk is a complex and powerful language, Qi/tk represents the tk toolset through a simple model - the **attribute-value model**. Widgets are entities with attributes and these attributes have values. Thus a window has the attribute **-height** which gives the height of the window and the value of this attribute is a number which gives the height of the window in pixels. The type discipline of Qi/tk is there to ensure that attributes are given values of the right type; thus a window cannot be given a string as the value of its height attribute. Widgets are differentiated, not just by their appearance on the screen, but by the attributes they can sustain and the values that these attributes may take.

There are three fundamental functions for controlling the creation and appearance of widgets.

1. make!
2. set!
3. get!

The make! function is a polyadic function that receives

1. a symbol, which becomes the name of a widget,
2. a symbol which describes what sort of widget it is supposed to be (button, text box etc) and
3. a series of attributes and values.

There are three stages for creating and using widgets under Qi/tk.

1. The widgets are created by make!.
2. The widgets are placed on the screen by calling the window manager.
3. We interact with the widgets.

The set! and get! functions allow the value of an attribute to be changed or accessed. Thus (set! .b -text "bye") changes the **attribute** text on the button .b to the **value** "bye". The expression (get! .b -text) returns whatever the value is attached to the text attribute of .b.

Here is a simple example; custom suggests we create a button that prints *Hello World*. We place the text *Friendly* on the button by placing the widget option –text "Friendly" as an option in the list. The command attribute allows us to attach commands to a button. A first attempt might be

```
(make! .b1 button -command (output "Hello World~%") -text "Friendly")
```

We will see shortly that this is not quite right.

To make our button appear, we need to declare it to the window manager. The simplest way of invoking the manager is through the **pack** function.

This function takes a list of widgets to place on the screen and a series of options that determine how these widgets are placed. If this options list is empty then **pack** chooses a default packing arrangement by stacking the widgets from top to bottom in the order in which they appear in the list.

Example 4

Here we pack a button.

```
(9-) (pack .b1)
packed
```

If we click on the button shown in example 4, nothing happens because in common with Qi, nearly all Qi/tk functions are strictly evaluating. This means that the expression (output "Hello World~%") is evaluated when the button .b1 is created and not when it is pressed. In fact, if type checking is enabled, the expression (make! .b1 button -command (output "Hello World~%") -text "Friendly") will raise a type error.

Qi/tk expects that the value of the command attribute will be a lazy object because the point of having a command is to evaluate it only when the widget is activated. To enable this we freeze the command value by the function freeze which creates a lazy object. The correct version should have been

(make! .b1 button -command **(freeze** (output "Hello World~%")) -text "Friendly")

We can fix this by overwriting the value of the command attribute using set!.

(set! .b -command (freeze (output "Goodbye World~%")))

2.2 Error Messages from tk

Once a widget is created using make!, it cannot be recreated without first being destroyed. We can arrange for error messages to be relayed from TCL/tk. Attempting to do so will generate an error message from tk.

Example 5

```
(5-) (make! .b button)
.b

(6-) (make! .b button)
window name "b" already exists in parent.b
```

By default, Qi/tk does not relay error messages from tk. This default can be cancelled by the command (echo-tk +). (echo-tk -) restores the default.

2.3 Attributes of Buttons

Figure 2.2 gives the attributes of buttons and their potential values.

Attribute	Type	Value
-height	measure	height of the button*
-width	measure	width of the button*
-background	colour	the background colour of the button*
-foreground	colour	the colour of the text on the button*
-relief	relief	flat, raised, or sunken depending on how the widget is to be displayed (default raised in buttons).
-image	image	the image displayed on the widget
-font	font	the font used to display text
-state	state	the widget is active, normal or disabled (default normal)
-default	state	whether the button is displayed as active, normal or disabled (default normal)
-activebackground	colour	the colour of the button when it is clicked
-activeforeground	colour	the colour of the text on the button when it is clicked
-cursor	cursor	the appearance of the cursor when over the button (see appendix III for a list of cursor values)
-borderwidth	measure	the width of the interior border within the the button
-disabledforeground	colour	the foreground colour when disabled
-anchor	anchor	the text begins in one of the eight compass points elative to the button (n, s, e, w, ne, se,sw, nw).

-justify	justification	of the text, either right left or centre
-padx	number	x-axis padding around the text
-pady	number	y-axis padding around the text.
-underline	number	underlines the nth character of the text starting from 0.
-wraplength	number	the maximum line length N.
-highlightthickness	measure	the thickness of the line around a button when it has input focus.
-highlightbackground	colour	the background colour of the button when it has input focus
-highlightforeground	colour	the foregroundcolour of the button when it has input focus
-repeatdelay	number	controls delay before autorepeat
-repeatinterval	number	controls interval between repeats
-text	string	the text on the button

Figure 2.2 The Attributes of Buttons and their Types

-background or **-bg** determines the colour of the button and **-foreground** or **-fg** determines the colour of the text on the button. **-activebackground** and **-activeforeground** determine the colours that appear on the button when it is **active** (when the cursor is over the button and the button will do something if clicked). **-disabledforeground** gives the foreground colour of the button when it is disabled.

The **-state** attribute determines the state of the button; the possible values are **normal**, **active** and **disabled**. **normal** is the default state. **disabled** means that the button will not work if clicked. **active** causes the button to display in its active colours.

The thickness of the line around the button is controlled by **-highlightthickness.** **-padx** and **-pady** add padding to the button in the y direction (vertically) and the x direction (horizontally). These control the space between the text on the button and the perimeter of the button. **-underline** takes a natural number input *n* and underlines the *n*th character of the string displayed by **-text**. Numbering starts from 0, so that the first character of the text is numbered by 0.

-justify determines how the text on the button is displayed when there is more than one line of text; a justification value is either **right**, **left** or **centre**. **-wraplength** determines the maximum number of characters that will be fitted on a line before the the text begins a new line. Lines are broken only at new lines in the text. The option **-anchor n/s/e/w/nw/ne/sw/se/center** anchors the text on the button relative to the button itself.

The default unit of measure for height, width etc. is pixels. This can be changed to inches, centimeters, millimeters or print units[1] by placing **i**, **c**, **m** or **p** as a suffix

[1] A print unit is 1/72nd of an inch.

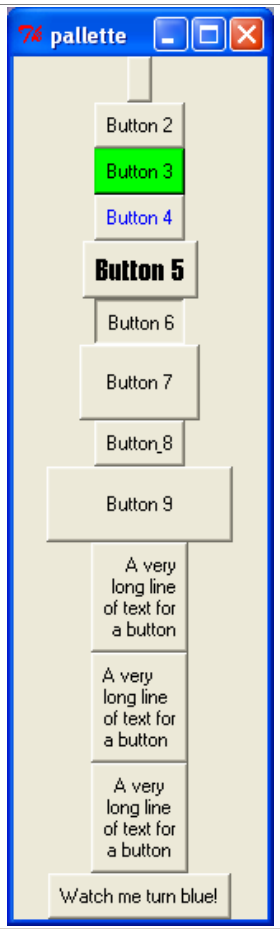
to the number. Thus **(make! .b6 button [-width 3c -height 1c])** creates a button which is 3 cm wide and 1cm high. All objects which are numbers or numbers attached to metrics like **i** and **c** are recognized as belonging to the type **measure** in Qi/tk.

-relief specifies the 3-D effect desired for the widget. Acceptable values are **raised**, **sunken**, **flat**, **ridge**, **solid**, and **groove**. The value indicates how the interior of the widget should appear relative to its exterior; for example, **raised** means the interior of the widget should appear to protrude from the screen, relative to the exterior of the widget.

The option **-repeatdelay** is found for all buttons. It is followed by a natural number that specifies in number of milliseconds a button or key must be held down before it begins to auto-repeat the associated command. **-repeatinterval** followed by a natural number that specifies in number of milliseconds the interval between each autorepeat.

Example 6

```
(make!  .b1 button)
(make!  .b2  button -text "Button 2")
(make!  .b3  button
 -text "Button 3" -background "green" )
(make!  .b4  button
-text "Button 4" -foreground "blue" )
(make!  .b5 button
  -text "Button 5" -font "Impact" )
(make! .b6 button
   -text "Button 6" -relief sunken)
(make!  .b7 button
    -text "Button 7" -highlightthickness 10)
(make!  .b8 button
  -text "Button 8" -underline 6)
(make!  .b9 button
  -text "Button 9" -padx 30 -pady 10)
(make!  .b10 button
 -text "A very long line of text for a button"
    -justify right -wraplength 50)
(make!  .b11 button
 -text "A very long line of text for a button"
  -justify left -wraplength 50)
(make!  .b12 button
  -text "A very long line of text for a button"
   -justify center -wraplength 50)

(pack .b1 .b2 .b3 .b4 .b5 .b6 .b7 .b8 .b9 .b10
                    .b11 .b12)
```



-image receives an image which is displayed on the button. The image must have been created with the image function (see next chapter). Typically, if the image option is specified then it overrides other options that specify a bitmap or extual value to display in the widget. The image option may be reset to an empty string to re-enable a bitmap or text display. **-font** determines the font used to display the text on the button. **-cursor** specifies the mouse cursor to be used for the widget. The value may have any of the forms listed in appendix III.

Example 6 shows a palette of exotic buttons and the resulting window.

2.4 Dynamic Linking

The set! and get! functions allow us to set the attributes of one button to the values of another. Thus the following commands.

```
(make! .b button -text "hi")
(make! .c button -text (get! .b -text))
(pack .b .c)
```

create two buttons .b and .c where the text of .c is set to the text of .b.

However if we then change the text of .b ….

```
(set! .b -text "bye")
```

… the text attached to .c stays the same.

Dynamic linking using the dynamic function allows us to state that the text attached to .c should always be the same as that of .b

```
(make! .c button -text (dynamic (get! .b -text)))
```

Now (set! .b -text "bye") causes the text on .c to change to 'bye'.

The dynamic function can be wrapped around any value to make the association a dynamic one. The dynamic value may be a function of any number of other values attached to other widgets. Dynamic associations may be removed by overwriting the dynamic value by a non-dynamic one. Thus

```
(set! .c -text "ciao")
```

will assign a non-dynamic value to the text attribute of .c, overwriting any previous dynamic value.

2.5 Check Buttons and Radio Buttons

A check button is similar to an ordinary button except that it carries around with it a binary internal state which toggles on/off according to whether the user has clicked on it. The default (pre-clicked) state is off. The check button appears as

a small box which carries a tick when it is in the 'on' state. The following program creates three check buttons.

Example 7

Here we create a list of check buttons; the resulting widget is shown alongside.

```
(make! .cb1
                checkbutton
                -command (freeze (toggle .cb1)))

(make! .cb2
                 checkbutton
                -command (freeze (toggle .cb2)))

(make! .cb3
                checkbutton
                -command (freeze (toggle .cb3))
                 -text "hi")

(set *cb1* false)
(set *cb2* false)
(set *cb3* false)

(define toggle
   .cb1 -> (set *cb1* (not (value *cb1*)))
   .cb2 -> (set *cb2* (not (value *cb2*)))
   .cb3  -> (set *cb3* (not (value *cb3*))))

(pack .cb1 .cb2 .cb3)
```

Loading the program in concurrent mode allows us to change the value of a global

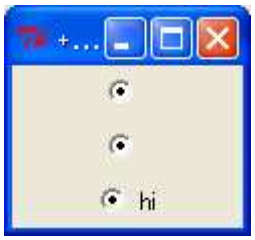
```
[4+] (value *cb2*)
false : boolean

[5+] \Here we first click on .cb2 as shown in picture and then type ...\
(value *cb2*)
true : boolean
```

TCL/tk arranges to query the internal state associated with a check button, but it is easier and faster to arrange for this to be done in Qi as shown in this program. If we want to use radiobuttons instead of checkbuttons, we use **radiobutton** instead of **checkbutton** in the program above (see below). The options for radiobuttons are the same as checkbuttons. Using radiobuttons instead of checkbuttons gives the display in figure 3.2.

Example 8

Here we create a list of radio buttons.

```
(make! .rb1 radiobutton)
(make! .rb2 radiobutton)
(make! .rb3 radiobutton -text "hi")
(pack .rb1 .rb2 .rb3)
```



2.6 Labels

Labels are placed within GUIs as titles or guides to the objects contained therein. They have the many of the same attributes as buttons (figure 2.3) except that, since they cannot be clicked, they do not have commands associated with them.

Attribute	Type	Value
-height	measure	height of the label*
-width	measure	width of the label*
-background	colour	the background colour of the label*
-foreground	colour	the colour of the text on the label*
-relief	relief	flat, raised, or sunken depending on how the widget is to be displayed (default raised in labels).
-image	image	the image displayed on the widget
-font	font	the font used to display text
-cursor	cursor	the appearance of the cursor when over the label
-borderwidth	measure	the width of the interior border within the the label
-anchor	anchor	the text begins in one of the eight compass points elative to the label (n, s, e, w, ne, se,sw, nw).
-justify	justification	of the text, either right left or centre
-padx	measure	x-axis padding around the text
-pady	measure	y-axis padding around the text.
-underline	number	underlines the nth character of the text starting from 0.
-wraplength	number	the maximum line length N.
-text	string	the text on the label

Figure 2.3 The Attributes of Labels

2.7 Entries

Entries are active areas of text where the user can type his or her input. Example 9 shows a program that builds a grid of five labels and five entries. (We will cover the analog-grid function in the next chapter).

Example 9

```
(make! .l0 label -text "Name")
(make! .l1 label -text "Street Number")
(make! .l2 label -text "City")
(make! .l3 label -text "State/County")
(make! .l4 label -text "Country")
(make!  .name entry -width 20)
(make!  .address1 entry -width 20)
(make!  .address2 entry -width 20)
(make!  .address3 entry -width 20)
(make!  .address4 entry  -width 20)
(put-entry .address4 "USA")

(analog-grid [[.l0 .name]
                [.l1 .address1]
                [.l2 .address2]
                [.l3 .address3]
                [.l4 .address4]]
        -padx 10 -pady 10)
```

The function get-entry takes the name of an entry and returns the value found in it as a string. Thus suppose I type my name into the .name entry. The expression (get-entry .name) now returns "Mark Tarver". put-entry takes an entry and a string and inserts the string in the entry.

Figure 2.4 shows the attributes of entries. The **-show** option followed by a single symbol <symbol> means that the information entered to the entry will be displayed as a series of <symbol>s. Thus **-show *** will mean that every character typed to the entry will appear on the screen as a *. This is useful in creating entries that receive confidential information such as passwords.

The command **-icursor** <position> shows where the cursor is to appear when the entry is selected. The <position> is given by a natural number.

Attribute	Type	Value
-background	colour	the background colour of the entry
-foreground	colour	the colour of the text on the entry
-relief	relief	relief; flat, raised, or sunken depending on how the widget is to be displayed (default raised in entries).
-font	font	the font used to display text
-cursor	cursor	the appearance of the cursor when over the entry
-borderwidth	measure	the width of the interior border within the the entry
-justify	justification	of the text, either right left or centre
-highlightthickness	measure	the thickness of the line around a entry when it has input focus
-highlightbackground	color	the colour of the entry when it has input focus
-icursor	number	the initial position of the cursor
-show	symbol	data entry is shown as a list of symbols

Figure 2.4 The Attributes of Entries

2.8 TextBoxes

Text boxes differ from entries in allowing multiple lines of text to be entered but share the same attributes. The functions get-textbox and put-textbox retrieve and insert text into a textbox.

Example 10

Here we create a textbox.

```
(make! .t1 textbox)
(pack .t1)
```

Figure 2.5 shows the attributes of textboxes.

Attribute Tag	Slot Type	Slot Value
-background	colour	the background colour of the textbox*
-foreground	colour	the colour of the text on the textbox*
-relief	relief	relief; flat, raised, or sunken depending on how the widget is to be displayed (default raised in entries).
-font	font	the font used to display text
-cursor	cursor	the appearance of the cursor when over the textbox
- borderwidth	measure	the width of the interior border within the the textbox.
-justify	justification	of the text, either right left or centre
-highlightthickness	measure	the thickness of the line around a textbox when it has input focus.
-highlightbackground	color	the colour of the textbox when it has input focus
-icursor	number	the initial position of the cursor
-show	symbol	data entry is shown as a list of symbols
-height	number	the height of the textbox in characters
-width	number	the width of the textbox in characters
-wrap	none/char/word	no, character or word wrapping
-spacing1	number	adds n spaces above each line
-spacing2	number	adds n spaces between all lines but wrapped lines
-spacing3	number	adds spacing after each line
-state	enabled/disabled	whether enabled (default) or otherwise
-xscroll	boolean	if true places scrollbar in x axis
-yscroll	boolean	if true places scrollbar in y axis

Figure 2.5 The Attributes of TextBoxes

2.9 Images

Images can be used as backgrounds for many kinds of widget. Images are based on gif or bmp (bitmap) files. Attributes allow the image file and display height and width to be specified.

Attribute	Type	Value
-file	string	the path of the image file
-height	measure	the height of the image
-width	measure	the width of the image

Figure 2.6 The Attributes of Images and their Types

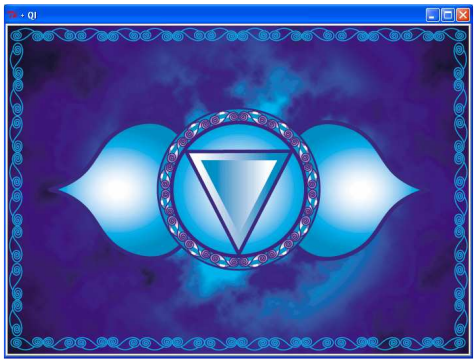
To create an image the image function is used. image takes a symbol which becomes the name of the image, an image type (either photo (.gif) or bitmap (.bmp)), and a string giving the path to the file where the image is kept.

figure 3.2., a .gif file is used and the name of the .gif file is given in the string argument to the **-file** attribute. Images are placed on buttons using the **-image** attribute.

Example

This creates a very large button which is the same size as the image!

```
(make!  eye image
                -file "third_eye.gif")
(make!  .third-eye button
              -image eye)
(pack .third-eye)
```



Using the height and width attributes on the button constrains the image to fit to the size of the button rather than conversely. However the image is not shrunk to the button but simply chopped down to size and what appears may not resemble the image.

Example

Here the image ic ropped to fit the button

```
(set! .third-eye -height 20)
(set! .third-eye -width 30)
```

An alternative approach is to shrink the image to the button by specifying the size of the image

```
(make! eye image -file "third_eye.gif" -height 20 -width
30)
(make! .third-eye button -image eye)
(pack .third-eye)
```

This gives a better result through the shrunk image does not closely resemble the original image. This is because of the disparity between the size of the shrunken image and the size of the original. In general, small simple images work best for buttons, though larger images can be used with larger widgets.

(set! .third-eye -image "") will remove the image from the button.

2.10 Fonts

The font argument creates a font. The first argument is a symbol which is the chosen name for the font. The second argument is a list of attribute-values. drawn from the list in figure 2.7.

Attribute	Type	Value
family	family	the font family
-size	number	the size of the font
-weight	thickness	either **bold** or **normal**
-slant	boolean	whether slanted
-underline	boolean	whether underlined
-overstrike	boolean	whether overstruck

Figure 2.7 The Attributes of Fonts

The next example creates a font and uses it on a button.

Example

```
(make! fancy font
      -family "Imprint MT Shadow"
      -size 30
      -weight bold
      -slant italic
      -overstrike true
      -underline true)

(make!  .fun button
                  -text "demo" -font fancy)
 (pack .fun)
```

The list of font families varies according to the platform of the computer. Appendix II gives those used under Windows XP.

Chapter 3

The Window Manager

3.1 Creating Windows

When Qi/tk is called, by default there is a single graphics window which is the **root window**. The root window is the default window to which all graphical objects are placed unless directed otherwise. All other windows created depend on the root window in that, if the root window is killed, then these windows are killed too. Also your command window will become psychotic.

In general it is not a great idea to place graphical objects in the root window. The root window is the logical place to place the programming environment of Qi/tk and Qi/tk allows you to do exactly this. Hence unless you are actually building your own programming environment, then the root window should be left blank.

Creating a window is easy.

(make! .mywindow window)

creates a window shown below. By default the name of the window appears on the window bar.

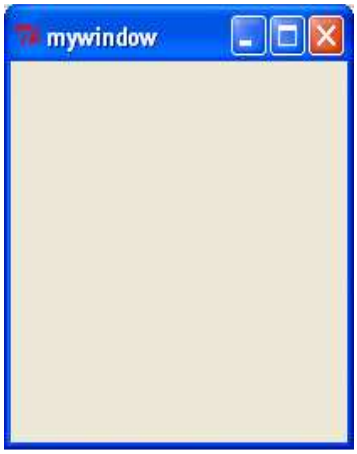


Figure 3.1 Creating a Window

When the name of the window is used within the name of a widget, it signifies that the widget is to be packed into that window. Thus these instructions

```
(make! .mywindow.hello button -text "Hello")
(make! .mywindow.goodbye -text "Goodbye")
(pack .mywindow.hello .mywindow.goodbye)
```

give the following window

The mywindow window is the parent widget to the widgets within it. The name of .mywindow.hello relative to that window is .hello.

The windows options are listed in figure 3.2

Attribute Tag	Slot Type	Slot Value
-height	measure	height of the window*
-width	measure	width of the window*
-background	colour	the background colour of the window*
-cursor	cursor	the appearance of the cursor when over the window

Figure 3.2 The Attributes of Windows

Here we create a violently yellow window.

```
(make! .mywindow window   -height 100  -width  200 -background "yellow")
```

Figure 3.3 Creating a Yellow Window

3.2 Dialog Windows

Dialog windows are windows that pop up according to the flow of control within a Qi/tk program and disable the other GUIs until the user replies to them. We start with a series of predefined dialog windows called **convenience dialog windows**. The most important command for invoking a dialog window is **messageBox.**

Dialog windows are windows that pop up according to the flow of control within a Qi/tk program and disable the other GUIs until the user replies to them. The most important command for invoking a dialog window is **messagebox.**

The **messagebox** function takes a list of options and produces a message box. Depending on the nature of the user interaction, a string is returned. Figure 3.4 shows the attributes of messageboxes and 3.5 gives some examples.

Attribute Tag	Attribute Type	Slot Value
-type	messageboxtype	abortretryignore ok okcancel retrycancel yesno yesnocancel
-message	string	the message in the box
-title	string	the title of the messagebox
-icon	icon	question or warning

Figure 3.4 The attribute tags of message boxes and the types of their values

Expression	Message Box
(messagebox) returns "ok" after user input.	
(messagebox -message "File deleted") returns "ok" after user input.	

```
(messagebox
 -message "Are you sure?"
  -title "Deleting vital file!"
  -type yesnocancel
  -icon warning)
```

returns "yes" or "no" or "cancel" after user input.

Figure 3.5 Some message boxes

opencolour activates a dialog window that enables the user to choose a colour from a colour palette. The function returns an rgb value that represents the colour selected.

```
(2+) (opencolour)
"#8080c0" : rgb
```

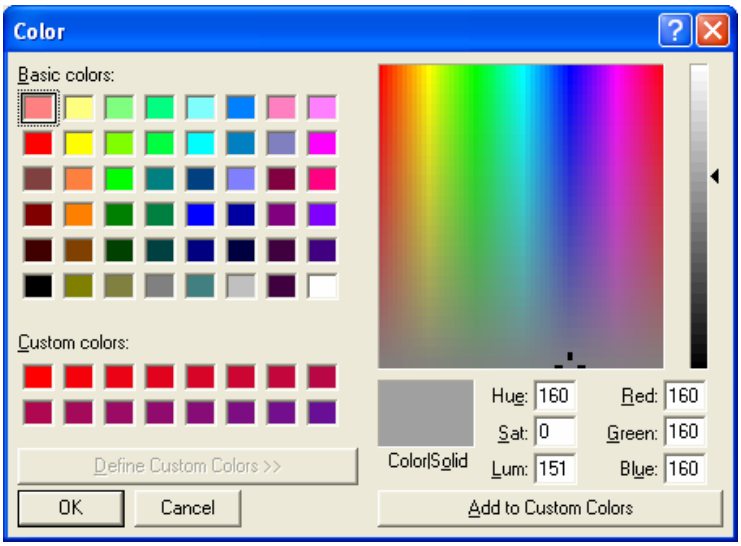


Figure 3.6 Using a Palette to Choose a Colour

opencolour returns an error if Cancel is chosen.[2]

openfile and **savefile** both receive options and display the resident directory. Selecting a file returns the path name of that file as a string. savefile allows the

[2] Note if you want to trap this error then consult the Qi library for type secure error trapping.

user to create a new filename which is returned as a string. "0" is returned if Cancel is chosen.

(3+) (openfile)
"C:/Documents and Settings/User/My Documents/Computer Science/Languages/Qi/Qi-tk 7.2/debug.txt" : string

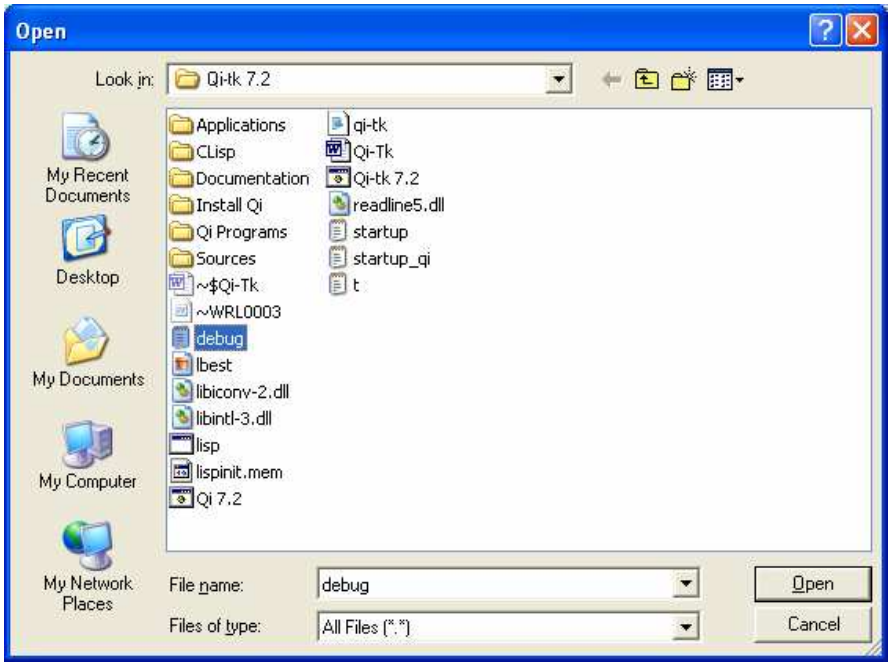


Figure 3.7 Opening a File in a Dialog Window

The options to these widgets are given in figure 3.8.

Attribute Tag	Slot Type	Slot Value
-defaultextension	string	extension to the file
-initialdir	string	the directory displayed
-background	colour	the background colour of the window
-initialfile	string	the default name in the pop up window
-message	string	message in the client area

Figure 3.8 Attriibutes to the getOpen dialog windows

-defaultextension specifies a string that will be appended to the filename if the user enters a filename without an extension. The default value is the empty string, which means no extension will be appended to the filename in any case.[3]

[3] This option is ignored on the Macintosh platform, which does not require extensions to filenames.

-initialdir specifies that the files in directory should be displayed when the dialog pops up. If this parameter is not specified, then the files in the current working directory are displayed. If the parameter specifies a relative path, the return value will convert the relative path to an absolute path.[4]

-initialfile specifies a filename to be displayed in the dialog when it pops up.[5] **-message** specifies a message to include in the client area of the dialog.[6] **-title** specifies a string to display as the title of the dialog box. If this option is not specified, then a default title is displayed.

3.3 Packing

The Qi/tk window manager controls the way that widgets are placed on the screen. The three essential commands needed to master the window manager are **pack**, **analog-grid** and **frame**.

By default, **pack** always packs vertically from top to bottom, separating the buttons by a fixed space. We can change this. The program below packs a line of buttons from left to right illustrated in figure 3.9.

(pack .b1 .b2 .b3 .b4 .b5 -side left)

Figure 3.9 Creating a line of buttons arranged left to right

The permissible values to **-side** are **left, right, top** and **bottom**. **top** stacks the successive buttons one on top of the other (the default). **bottom** places them underneath each.

Suppose we want to space the buttons; the **-padx** option will do this, spacing buttons in the x-axis direction (figure 7). To create this effect we change the previous pack command to (pack .b1 .b2 .b3 .b4 .b5 -side left -padx 20). The spacing is measured in pixels.

[4] This option may not always work on the Macintosh. This is not a bug. Rather, the General Controls control panel on the Mac allows the end user to override the application default directory.
[5] This option is ignored on the Macintosh platform.
[6] This is only available on the Macintosh, and only when Navigation Services are installed.

3.10 Spacing buttons using -padx

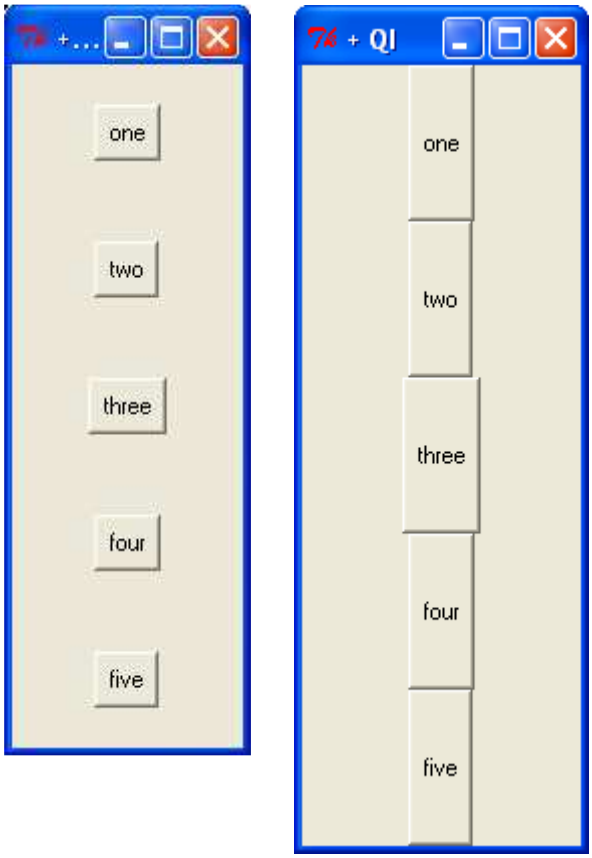


Figure 3.11 Spaced buttons, vertically aligned.

If we use (pack .b1 .b2 .b3 .b4 .b5 -side top -pady 20) then we obtain the packing shown in the lefthand window in in figure 3.11.

There are two types of padding; internal and external. External padding controls the space between the widgets. Internal padding controls the space between the text on the widget and the edges of the widget. The expression (pack .b1 .b2 .b3 .b4 .b5 -ipady 25 -padx 50) packs the buttons vertically (the default) with a generous internal padding in the y axis (see the window in the right hand of figure 3.11).

We can tell the packer to pack the widgets so that they fill either the x axis or the y axis of the window or both. The option **-fill x/y/both** does this. This command packs five buttons so that they fill the x axis (figure 3.12).

(pack .b1 .b2 .b3 .b4 .b5 -fill x)

Figure 3.12 Buttons Padded to the x Axis

The list of packing options is given in figure 3.13.

Attribute	Slot Type	Action
-side	must be one of left, right, top, bottom	Packs the widgets accordingly (left to right … etc.)
-padx	number	Spaces them in the x axis
-pady	number	Spaces them in the y axis
-ipadx	number	Internal padding – x axis
-ipady	number	Internal padding – y axis
-fill	x or y	Fills in the x or y axis

Figure 3.13 Packing Options

A very useful *Qi*/tk command (not found in TCL/tk) is analog-grid which can be used in place of pack. This enables the creation of lists of lists of text widgets where the list structure mirrors the layout of the widgets. Thus suppose we want to create 4 buttons in a square; then instead of using **pack** we can write.

(analog-grid [[.b1 .b2] [.b3 .b4]] -padx 10 -pady 10)

The items in each list are displayed horizontally, left to right and stacked vertically with the required padding to produce the effect in figure 3.14.

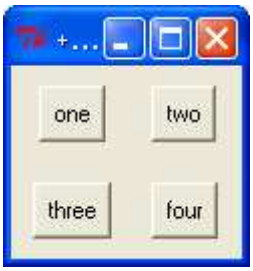


Figure 3.14 A square of buttons arranged using analog-grid.

A lower level command is **grid** which gives you more control over the widget location. The syntax is (grid <widget> <column> <row>…<options>…). Where <column> and <row> are natural numbers giving the column and row position of the widget. Numbering starts from 0. The option **-columnspan** N stretches the widget over N columns.

3.4 Frames

The **frame** command enables a set of widgets to be grouped into an object called a **frame**. Once a frame has been created, it can be packed or placed like any other widget. The effect of packing a frame is to place all its constituent widgets in the position determined by the packing command.

The power of frames is that they allow us to manipulate complex collections of widgets as if they were a single entity. This is particularly useful when the GUI we want to build has many widgets laid out in different ways. The widget layouts in *Qi*/tk work best when the layout follows a single format. But when we want to mix formats then we need to use frames.

Here is a short program showing the use of frames. We create two frames and four buttons.

```
(make! .f1 frame)
(make! .f2 frame)

(make! .f1.b1 button -text "inside frame f1")
(make! .f1.b2 button -text "also inside frame f1")
(make! .f2.b3  button  -text "inside frame f2")
(make! .f2.b4 button -text "also inside frame f2")
```

The identifiers for the buttons, .f1.b1 .f1.b2 .f2.b3 .f2.b4, show that the first two buttons are embedded in the frame .f1 and the second two in the frame .f2. The purpose of the dot is to isolate the name of the frame (thus neither .f1b2 or f1b2 would place the button in the .f1 frame). The buttons are now created.

To pack the buttons we

(a) Pack them in their frames.
(b) Pack the frames themselves.

The first two commands pack the buttons in the frames.

(pack .f1.b1 .f1.b2 -side left)
(pack .f2.b3 .f2.b4)
The final command packs the frames

(pack .f1 .f2)

producing the result in figure 3.15.

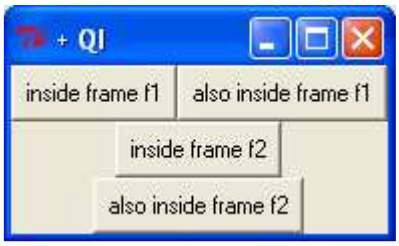


Figure 3.15 Using Frames

Figure 3.5 shows a Qi/tk program that arranges two labels and two buttons in a grid.

(make! .b1 button -text "one")
(make! .b2 button -text "two")
(make! .l1 label -text "Print 1")
(make! .l2 label -text "Print 21")
(analog-grid [[.l1 .b1] [.l2 .b2]] -padx 10 -pady 10)

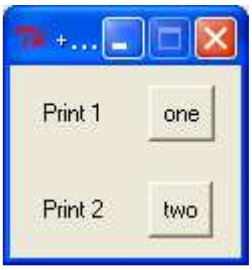


Figure 3.16 Labels with Buttons

Chapter 4

Aspect and Control

This chapter groups together a miscellany of topics which relate to program features that are not widgets and are not created with the make! command and cannot be packed. The topics include aspects of widgets such as colour; the creation of sound; the position and appearance of the cursor and the binding of keys to actions.

4.1 Colours and Sounds

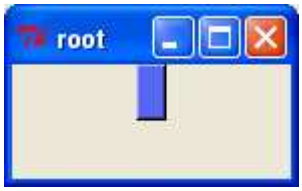
(bell) sounds the bell.

Qi/tk inherits 760 preset colours from TCL/tk. The full list of these colours and their RGB (red-green-blue) values is given in appendix I. Every preset colour is given by a string and every preset colour is recognized as belonging to the type colour.

An object of type **rgb** is a 6 digit hexadecimal number expressed as a string. An RGB number can also be used as a colour and in fact all objects of type **rgb** are recognised as objects of type colour. Here are two lines that create a blue button.

Example

This is a purple button created using an RGB value.




```
(make!  .b1 button -background "#5a67f2")
(pack .b1)
```

colour? and **rgb?** recognise colours and rgb objects.

4.2 Downloading Web Pages into Qi/tk

The function url downloads a web page as a string.

Example

The command (url "http://www.lambdassociates.org/" 10) will download the corresponding web page as a string. If no site is found at the URL, then the string corresponding to an Error 404 message is downloaded. The numeric argument is a timeout in seconds - if no response is found after the timeout has expired then the process aborts with the empty string.

Note that some web pages may crash Qi/tk if the resident Lisp is running the ASCII character set and the web page characters are not ASCII.

4.3 Key Bindings

Key bindings allow you to bind the behaviour of a key to a procedure. There are two functions for doing this; **bindkey**.

bindkey is a 3-place function that takes a symbol, an event which is either a a list of keystrokes taken from the following set[7]

Alt Control Delete Shift Lock Meta Double Triple Button-1 Button-2 Button-3 Button-4 Button-5 a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

The final argument is a lazy object. **key?** Recognizes keys.

Example

The expression

```
(bindkey all [Control x] (freeze (print ctrl+x)))
```

creates a key binding that causes ctrl+x to be printed whenever the Control key and the x key are pressed together in any widget of the GUI. If the first argument is the name of a widget then the binding only works when the cursor is in that widget.

4.4 User Defined Attributes

Attributes are signaled by symbols prefaced by -; -height, -width etc. These attributes are all built in to TCL/tk. User-defined attributes are prefaced by +. These slots can be filled in any way by the user and their values changed by set! and get! in the usual way.

This function creates a playing card widget which stores all of the information needed to use it in a program.

```
(define make-card
   S Card Image -> (make! S button -relief flat -image Image +card  Card))

(make! as image -file "cards/as.gif")
(make-card .as (@p 14 s) as)
(pack .as)
```

Typing (get! .as +card) will deliver (@p 14 s).

[7] The Button-n events relate to the buttons on the mouse.

Chapter 5

Drawing in Qi/tk

5.1 Graphical Objects

Graphical objects require a canvas to be created on which they are drawn. The command for drawing objects is **draw**. This is a polyadic function that takes

1. A canvas.
2. A shape
3. A list of coordinates.
4. A series of optional attribute-value pairs.

The possible shapes are **line**, **arc**, **polygon**, **rectangle** and **oval**.

A canvas is created by the make! function.

```
(make! .c canvas)
(pack .c)
```

The attributes of canvases are as follows.

Attribute	Type	Value
-fg	colour	the foreground colour
-bg	colour	the background colour
-foreground	colour	the foreground colour
- background	colour	the background colour
-width	number	whether underlined
-height	number	whether overstruck
-cursor	cursor	the cursor value
-image	image	ihe background image

Figure 5.1 The Attributes of Fonts

The coordinates of the line are given with 0,0 being taken as the top left hand corner of the screen. The list of numbers should be of even length with at least 4 elements. The first element of each pair of numbers in the coordinate list refers to the horizontal axis and the second to the vertical axis. If there are more than 4 numbers, then the extra numbers continue the line from its last coordinate. This command creates a line linking coordinates 0, 0; 100, 100 and 125, 150.

```
(draw  .c  line  [0 0 100 100 125 150])
```

The origin of the line begins at the top left hand corner of the canvas and the line extends downwards (figure 3.1).

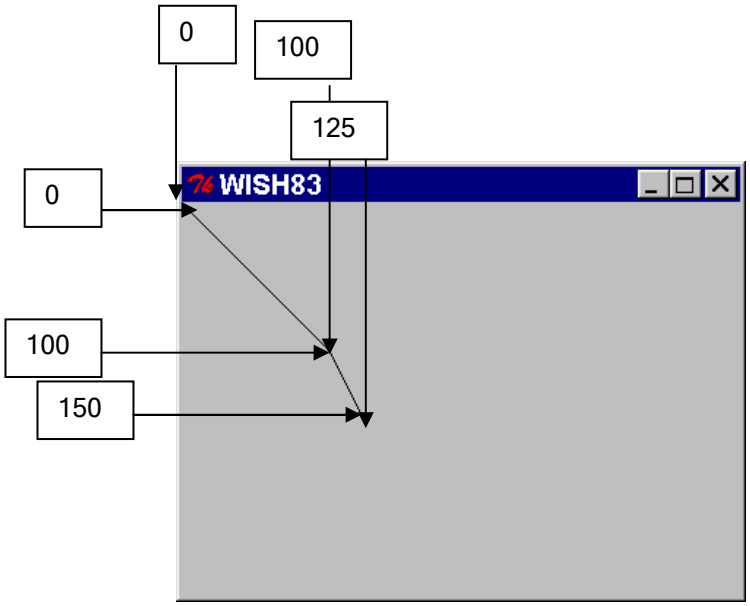


Figure 5.1 Drawing a line on a canvas

We can clean the canvas (figure 3.2) of these items by the command.

(wipe all .c)

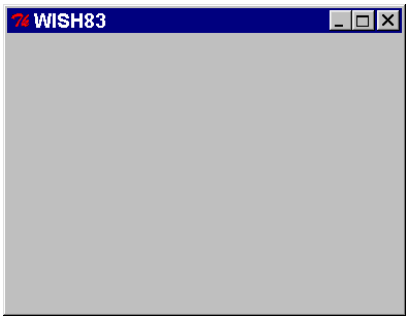


Figure 5.2 After wiping the canvas

We add another thicker line to the canvas and add a final blue line by the commands

(draw .c line [100 100 100 200] -width 10)

(draw .c line [50 50 250 300] -width 5 -fill "blue")

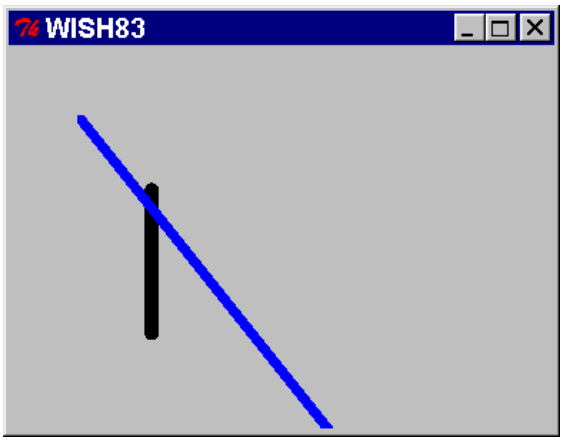


Figure 5.3. Drawing a blue line

(draw .c rectangle [50 50 100 150])

produces the canvas in figure 19.

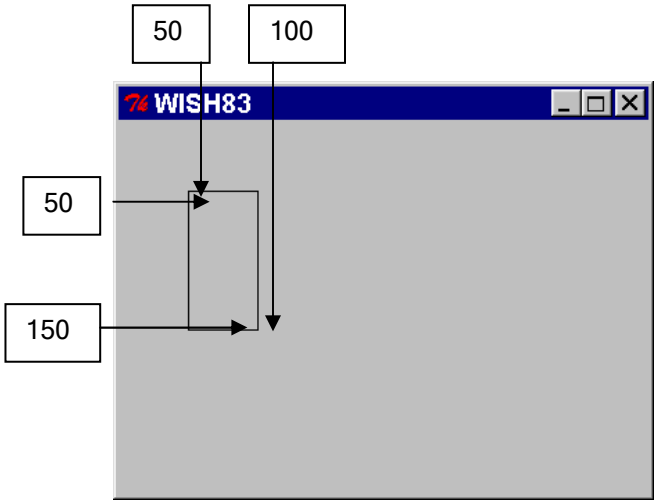
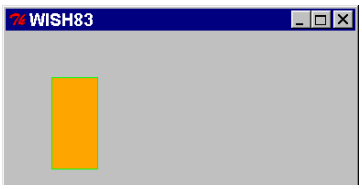


Figure 5.4 Drawing a rectangle

-outline "green" -fill "orange" will produce a rectangle with a green border and an orange fill (figure 5.5).




```
(wipe all .c)
(draw .c rectangle [50 50 100 150] -outline "green" -fill "orange")
```

Figure 5.5 A Colored Rectangle

The next statement produces an arc.

```
(wipe all .c)
(draw .c  arc [50 50 150 150])
```

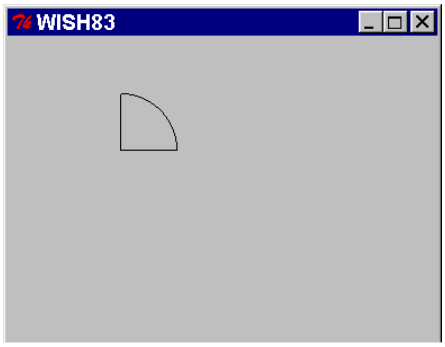


Figure 5.6 An arc

The next program produces a circle (figure 22).

```
(wipe all .c)
(draw .c oval [50 50 150 150])
```

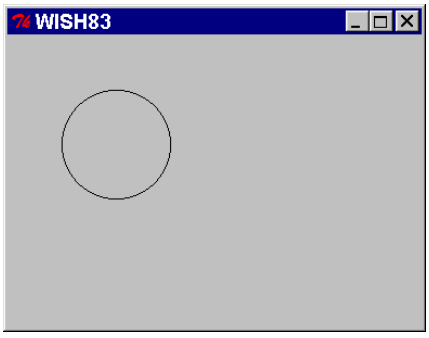


Figure 5.7 Drawing a circle.

The next command produces a polygon. The line borders of the polygon are in red with a 10 pixel thickness and the interior is orange (figure 23). The logic of polygon is that of line, except the computer produces a closed shape by joining the first and last points of the polygon

```
(wipe all .c)
(draw .c polygon  [220 10 250 10 250 30 210 50 220 80]
    -outline "red" -fill "orange" -width 10)
```

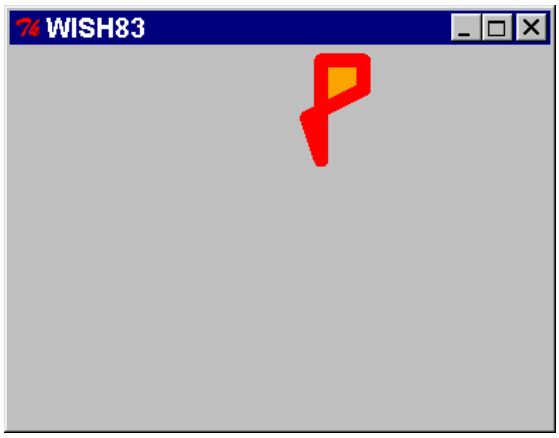


Figure 5.8 Drawing a polygon

5.2 Tagging Figures

Individual figures may be tagged by including the option -tag *Tag* to the fraw function where *Tag* can be any symbol or natural number. The effect is to name the figure. Using the tag name as the first argument to **wipe** causes only those objects which share the tag to be wiped from the canvas.

The function **getf!** takes a canvas, a tag for a figure and an attribute of that figure and returns the value of that attribute. **setf!** takes a canvas, a tag for a figure and an attribute of that figure, a value V for that attribute and sets the attribute of all the figures with that tag to V.

Appendix I

Colour Names and RGB Values in Qi/tk

alice blue	240	248	248
AliceBlue	240	248	248
antique white	250	235	235
AntiqueWhite	250	235	235
AntiqueWhite1	255	239	239
AntiqueWhite2	238	223	223
AntiqueWhite3	205	192	192
AntiqueWhite4	139	131	131
aquamarine	127	255	255
aquamarine1	127	255	255
aquamarine2	118	238	238
aquamarine3	102	205	205
aquamarine4	69	139	139
azure	240	255	255
azure1	240	255	255
azure2	224	238	238
azure3	193	205	205
azure4	131	139	139
beige	245	245	245
bisque	255	228	228
bisque1	255	228	228
bisque2	238	213	213
bisque3	205	183	183
bisque4	139	125	125
black	0	0	0
blanched almond	255	235	235
BlanchedAlmond	255	235	235
blue	0	0	0
blue violet	138	43	43
blue1	0	0	0
blue2	0	0	0
blue3	0	0	0
blue4	0	0	0
BlueViolet	138	43	43
brown	165	42	42
brown1	255	64	64
brown2	238	59	59
brown3	205	51	51
brown4	139	35	35
burlywood	222	184	184
burlywood1	255	211	211
burlywood2	238	197	197
burlywood3	205	170	170
burlywood4	139	115	115

cadet blue	95	158	158
CadetBlue	95	158	158
CadetBlue1	152	245	245
CadetBlue2	142	229	229
CadetBlue3	122	197	197
CadetBlue4	83	134	134
chartreuse	127	255	255
chartreuse1	127	255	255
chartreuse2	118	238	238
chartreuse3	102	205	205
chartreuse4	69	139	139
chocolate	210	105	105
chocolate1	255	127	127
chocolate2	238	118	118
chocolate3	205	102	102
chocolate4	139	69	69
coral	255	127	127
coral1	255	114	114
coral2	238	106	106
coral3	205	91	91
coral4	139	62	62
cornflower blue	100	149	149
CornflowerBlue	100	149	149
cornsilk	255	248	248
cornsilk1	255	248	248
cornsilk2	238	232	232
cornsilk3	205	200	200
cornsilk4	139	136	136
cyan	0	255	255
cyan1	0	255	255
cyan2	0	238	238
cyan3	0	205	205
cyan4	0	139	139
dark blue	0	0	0
dark cyan	0	139	139
dark goldenrod	184	134	134
dark gray	169	169	169
dark green	0	100	100
dark grey	169	169	169
dark khaki	189	183	183
dark magenta	139	0	0
dark olive green	85	107	107
dark orange	255	140	140
dark orchid	153	50	50
dark red	139	0	0
dark salmon	233	150	150
dark sea green	143	188	188
dark slate blue	72	61	61
dark slate gray	47	79	79
dark slate grey	47	79	79

dark turquoise	0	206	206
dark violet	148	0	0
DarkBlue	0	0	0
DarkCyan	0	139	139
DarkGoldenrod	184	134	134
DarkGoldenrod1	255	185	185
DarkGoldenrod2	238	173	173
DarkGoldenrod3	205	149	149
DarkGoldenrod4	139	101	101
DarkGray	169	169	169
DarkGreen	0	100	100
DarkGrey	169	169	169
DarkKhaki	189	183	183
DarkMagenta	139	0	0
DarkOliveGreen	85	107	107
DarkOliveGreen1	202	255	255
DarkOliveGreen2	188	238	238
DarkOliveGreen3	162	205	205
DarkOliveGreen4	110	139	139
DarkOrange	255	140	140
DarkOrange1	255	127	127
DarkOrange2	238	118	118
DarkOrange3	205	102	102
DarkOrange4	139	69	69
DarkOrchid	153	50	50
DarkOrchid1	191	62	62
DarkOrchid2	178	58	58
DarkOrchid3	154	50	50
DarkOrchid4	104	34	34
DarkRed	139	0	0
DarkSalmon	233	150	150
DarkSeaGreen	143	188	188
DarkSeaGreen1	193	255	255
DarkSeaGreen2	180	238	238
DarkSeaGreen3	155	205	205
DarkSeaGreen4	105	139	139
DarkSlateBlue	72	61	61
DarkSlateGray	47	79	79
DarkSlateGray1	151	255	255
DarkSlateGray2	141	238	238
DarkSlateGray3	121	205	205
DarkSlateGray4	82	139	139
DarkSlateGrey	47	79	79
DarkTurquoise	0	206	206
DarkViolet	148	0	0
deep pink	255	20	20
deep sky blue	0	191	191
DeepPink	255	20	20
DeepPink1	255	20	20
DeepPink2	238	18	18

DeepPink3	205	16	16
DeepPink4	139	10	10
DeepSkyBlue	0	191	191
DeepSkyBlue1	0	191	191
DeepSkyBlue2	0	178	178
DeepSkyBlue3	0	154	154
DeepSkyBlue4	0	104	104
dim gray	105	105	105
dim grey	105	105	105
DimGray	105	105	105
DimGrey	105	105	105
dodger blue	30	144	144
DodgerBlue	30	144	144
DodgerBlue1	30	144	144
DodgerBlue2	28	134	134
DodgerBlue3	24	116	116
DodgerBlue4	16	78	78
firebrick	178	34	34
firebrick1	255	48	48
firebrick2	238	44	44
firebrick3	205	38	38
firebrick4	139	26	26
floral white	255	250	250
FloralWhite	255	250	250
forest green	34	139	139
ForestGreen	34	139	139
gainsboro	220	220	220
ghost white	248	248	248
GhostWhite	248	248	248
gold	255	215	215
gold1	255	215	215
gold2	238	201	201
gold3	205	173	173
gold4	139	117	117
goldenrod	218	165	165
goldenrod1	255	193	193
goldenrod2	238	180	180
goldenrod3	205	155	155
goldenrod4	139	105	105
gray	190	190	190
gray0	0	0	0
gray1	3	3	3
gray2	5	5	5
gray3	8	8	8
gray4	10	10	10
gray5	13	13	13
gray6	15	15	15
gray7	18	18	18
gray8	20	20	20
gray9	23	23	23

gray10	26	26	26
gray11	28	28	28
gray12	31	31	31
gray13	33	33	33
gray14	36	36	36
gray15	38	38	38
gray16	41	41	41
gray17	43	43	43
gray18	46	46	46
gray19	48	48	48
gray20	51	51	51
gray21	54	54	54
gray22	56	56	56
gray23	59	59	59
gray24	61	61	61
gray25	64	64	64
gray26	66	66	66
gray27	69	69	69
gray28	71	71	71
gray29	74	74	74
gray30	77	77	77
gray31	79	79	79
gray32	82	82	82
gray33	84	84	84
gray34	87	87	87
gray35	89	89	89
gray36	92	92	92
gray37	94	94	94
gray38	97	97	97
gray39	99	99	99
gray40	102	102	102
gray41	105	105	105
gray42	107	107	107
gray43	110	110	110
gray44	112	112	112
gray45	115	115	115
gray46	117	117	117
gray47	120	120	120
gray48	122	122	122
gray49	125	125	125
gray50	127	127	127
gray51	130	130	130
gray52	133	133	133
gray53	135	135	135
gray54	138	138	138
gray55	140	140	140
gray56	143	143	143
gray57	145	145	145
gray58	148	148	148
gray59	150	150	150

gray60	153	153	153
gray61	156	156	156
gray62	158	158	158
gray63	161	161	161
gray64	163	163	163
gray65	166	166	166
gray66	168	168	168
gray67	171	171	171
gray68	173	173	173
gray69	176	176	176
gray70	179	179	179
gray71	181	181	181
gray72	184	184	184
gray73	186	186	186
gray74	189	189	189
gray75	191	191	191
gray76	194	194	194
gray77	196	196	196
gray78	199	199	199
gray79	201	201	201
gray80	204	204	204
gray81	207	207	207
gray82	209	209	209
gray83	212	212	212
gray84	214	214	214
gray85	217	217	217
gray86	219	219	219
gray87	222	222	222
gray88	224	224	224
gray89	227	227	227
gray90	229	229	229
gray91	232	232	232
gray92	235	235	235
gray93	237	237	237
gray94	240	240	240
gray95	242	242	242
gray96	245	245	245
gray97	247	247	247
gray98	250	250	250
gray99	252	252	252
gray100	255	255	255
green	0	255	255
green yellow	173	255	255
green1	0	255	255
green2	0	238	238
green3	0	205	205
green4	0	139	139
GreenYellow	173	255	255
grey	190	190	190
grey0	0	0	0

grey1	3	3	3
grey2	5	5	5
grey3	8	8	8
grey4	10	10	10
grey5	13	13	13
grey6	15	15	15
grey7	18	18	18
grey8	20	20	20
grey9	23	23	23
grey10	26	26	26
grey11	28	28	28
grey12	31	31	31
grey13	33	33	33
grey14	36	36	36
grey15	38	38	38
grey16	41	41	41
grey17	43	43	43
grey18	46	46	46
grey19	48	48	48
grey20	51	51	51
grey21	54	54	54
grey22	56	56	56
grey23	59	59	59
grey24	61	61	61
grey25	64	64	64
grey26	66	66	66
grey27	69	69	69
grey28	71	71	71
grey29	74	74	74
grey30	77	77	77
grey31	79	79	79
grey32	82	82	82
grey33	84	84	84
grey34	87	87	87
grey35	89	89	89
grey36	92	92	92
grey37	94	94	94
grey38	97	97	97
grey39	99	99	99
grey40	102	102	102
grey41	105	105	105
grey42	107	107	107
grey43	110	110	110
grey44	112	112	112
grey45	115	115	115
grey46	117	117	117
grey47	120	120	120
grey48	122	122	122
grey49	125	125	125
grey50	127	127	127

grey51	130	130	130
grey52	133	133	133
grey53	135	135	135
grey54	138	138	138
grey55	140	140	140
grey56	143	143	143
grey57	145	145	145
grey58	148	148	148
grey59	150	150	150
grey60	153	153	153
grey61	156	156	156
grey62	158	158	158
grey63	161	161	161
grey64	163	163	163
grey65	166	166	166
grey66	168	168	168
grey67	171	171	171
grey68	173	173	173
grey69	176	176	176
grey70	179	179	179
grey71	181	181	181
grey72	184	184	184
grey73	186	186	186
grey74	189	189	189
grey75	191	191	191
grey76	194	194	194
grey77	196	196	196
grey78	199	199	199
grey79	201	201	201
grey80	204	204	204
grey81	207	207	207
grey82	209	209	209
grey83	212	212	212
grey84	214	214	214
grey85	217	217	217
grey86	219	219	219
grey87	222	222	222
grey88	224	224	224
grey89	227	227	227
grey90	229	229	229
grey91	232	232	232
grey92	235	235	235
grey93	237	237	237
grey94	240	240	240
grey95	242	242	242
grey96	245	245	245
grey97	247	247	247
grey98	250	250	250
grey99	252	252	252
grey100	255	255	255

honeydew	240	255	255
honeydew1	240	255	255
honeydew2	224	238	238
honeydew3	193	205	205
honeydew4	131	139	139
hot pink	255	105	105
HotPink	255	105	105
HotPink1	255	110	110
HotPink2	238	106	106
HotPink3	205	96	96
HotPink4	139	58	58
indian red	205	92	92
IndianRed	205	92	92
IndianRed1	255	106	106
IndianRed2	238	99	99
IndianRed3	205	85	85
IndianRed4	139	58	58
ivory	255	255	255
ivory1	255	255	255
ivory2	238	238	238
ivory3	205	205	205
ivory4	139	139	139
khaki	240	230	230
khaki1	255	246	246
khaki2	238	230	230
khaki3	205	198	198
khaki4	139	134	134
lavender	230	230	230
lavender blush	255	240	240
LavenderBlush	255	240	240
LavenderBlush1	255	240	240
LavenderBlush2	238	224	224
LavenderBlush3	205	193	193
LavenderBlush4	139	131	131
lawn green	124	252	252
LawnGreen	124	252	252
lemon chiffon	255	250	250
LemonChiffon	255	250	250
LemonChiffon1	255	250	250
LemonChiffon2	238	233	233
LemonChiffon3	205	201	201
LemonChiffon4	139	137	137
light blue	173	216	216
light coral	240	128	128
light cyan	224	255	255
light goldenrod	238	221	221
light goldenrod yellow	250	250	250
light gray	211	211	211
light green	144	238	238
light grey	211	211	211

light pink	255	182	182
light salmon	255	160	160
light sea green	32	178	178
light sky blue	135	206	206
light slate blue	132	112	112
light slate gray	119	136	136
light slate grey	119	136	136
light steel blue	176	196	196
light yellow	255	255	255
LightBlue	173	216	216
LightBlue1	191	239	239
LightBlue2	178	223	223
LightBlue3	154	192	192
LightBlue4	104	131	131
LightCoral	240	128	128
LightCyan	224	255	255
LightCyan1	224	255	255
LightCyan2	209	238	238
LightCyan3	180	205	205
LightCyan4	122	139	139
LightGoldenrod	238	221	221
LightGoldenrod1	255	236	236
LightGoldenrod2	238	220	220
LightGoldenrod3	205	190	190
LightGoldenrod4	139	129	129
LightGoldenrodYellow	250	250	250
LightGray	211	211	211
LightGreen	144	238	238
LightGrey	211	211	211
LightPink	255	182	182
LightPink1	255	174	174
LightPink2	238	162	162
LightPink3	205	140	140
LightPink4	139	95	95
LightSalmon	255	160	160
LightSalmon1	255	160	160
LightSalmon2	238	149	149
LightSalmon3	205	129	129
LightSalmon4	139	87	87
LightSeaGreen	32	178	178
LightSkyBlue	135	206	206
LightSkyBlue1	176	226	226
LightSkyBlue2	164	211	211
LightSkyBlue3	141	182	182
LightSkyBlue4	96	123	123
LightSlateBlue	132	112	112
LightSlateGray	119	136	136
LightSlateGrey	119	136	136
LightSteelBlue	176	196	196
LightSteelBlue1	202	225	225

LightSteelBlue2	188	210	210
LightSteelBlue3	162	181	181
LightSteelBlue4	110	123	123
LightYellow	255	255	255
LightYellow1	255	255	255
LightYellow2	238	238	238
LightYellow3	205	205	205
LightYellow4	139	139	139
lime green	50	205	205
LimeGreen	50	205	205
linen	250	240	240
magenta	255	0	0
magenta1	255	0	0
magenta2	238	0	0
magenta3	205	0	0
magenta4	139	0	0
maroon	176	48	48
maroon1	255	52	52
maroon2	238	48	48
maroon3	205	41	41
maroon4	139	28	28
medium aquamarine	102	205	205
medium blue	0	0	0
medium orchid	186	85	85
medium purple	147	112	112
medium sea green	60	179	179
medium slate blue	123	104	104
medium spring green	0	250	250
medium turquoise	72	209	209
medium violet red	199	21	21
MediumAquamarine	102	205	205
MediumBlue	0	0	0
MediumOrchid	186	85	85
MediumOrchid1	224	102	102
MediumOrchid2	209	95	95
MediumOrchid3	180	82	82
MediumOrchid4	122	55	55
MediumPurple	147	112	112
MediumPurple1	171	130	130
MediumPurple2	159	121	121
MediumPurple3	137	104	104
MediumPurple4	93	71	71
MediumSeaGreen	60	179	179
MediumSlateBlue	123	104	104
MediumSpringGreen	0	250	250
MediumTurquoise	72	209	209
MediumVioletRed	199	21	21
midnight blue	25	25	25
MidnightBlue	25	25	25
mint cream	245	255	255

MintCream	245	255	255
misty rose	255	228	228
MistyRose	255	228	228
MistyRose1	255	228	228
MistyRose2	238	213	213
MistyRose3	205	183	183
MistyRose4	139	125	125
moccasin	255	228	228
navajo white	255	222	222
NavajoWhite	255	222	222
NavajoWhite1	255	222	222
NavajoWhite2	238	207	207
NavajoWhite3	205	179	179
NavajoWhite4	139	121	121
navy	0	0	0
navy blue	0	0	0
NavyBlue	0	0	0
old lace	253	245	245
OldLace	253	245	245
olive drab	107	142	142
OliveDrab	107	142	142
OliveDrab1	192	255	255
OliveDrab2	179	238	238
OliveDrab3	154	205	205
OliveDrab4	105	139	139
orange	255	165	165
orange red	255	69	69
orange1	255	165	165
orange2	238	154	154
orange3	205	133	133
orange4	139	90	90
OrangeRed	255	69	69
OrangeRed1	255	69	69
OrangeRed2	238	64	64
OrangeRed3	205	55	55
OrangeRed4	139	37	37
orchid	218	112	112
orchid1	255	131	131
orchid2	238	122	122
orchid3	205	105	105
orchid4	139	71	71
pale goldenrod	238	232	232
pale green	152	251	251
pale turquoise	175	238	238
pale violet red	219	112	112
PaleGoldenrod	238	232	232
PaleGreen	152	251	251
PaleGreen1	154	255	255
PaleGreen2	144	238	238
PaleGreen3	124	205	205

PaleGreen4	84	139	139
PaleTurquoise	175	238	238
PaleTurquoise1	187	255	255
PaleTurquoise2	174	238	238
PaleTurquoise3	150	205	205
PaleTurquoise4	102	139	139
PaleVioletRed	219	112	112
PaleVioletRed1	255	130	130
PaleVioletRed2	238	121	121
PaleVioletRed3	205	104	104
PaleVioletRed4	139	71	71
papaya whip	255	239	239
PapayaWhip	255	239	239
peach puff	255	218	218
PeachPuff	255	218	218
PeachPuff1	255	218	218
PeachPuff2	238	203	203
PeachPuff3	205	175	175
PeachPuff4	139	119	119
peru	205	133	133
pink	255	192	192
pink1	255	181	181
pink2	238	169	169
pink3	205	145	145
pink4	139	99	99
plum	221	160	160
plum1	255	187	187
plum2	238	174	174
plum3	205	150	150
plum4	139	102	102
powder blue	176	224	224
PowderBlue	176	224	224
purple	160	32	32
purple1	155	48	48
purple2	145	44	44
purple3	125	38	38
purple4	85	26	26
red	255	0	0
red1	255	0	0
red2	238	0	0
red3	205	0	0
red4	139	0	0
rosy brown	188	143	143
RosyBrown	188	143	143
RosyBrown1	255	193	193
RosyBrown2	238	180	180
RosyBrown3	205	155	155
RosyBrown4	139	105	105
royal blue	65	105	105
RoyalBlue	65	105	105

RoyalBlue1	72	118	118
RoyalBlue2	67	110	110
RoyalBlue3	58	95	95
RoyalBlue4	39	64	64
saddle brown	139	69	69
SaddleBrown	139	69	69
salmon	250	128	128
salmon1	255	140	140
salmon2	238	130	130
salmon3	205	112	112
salmon4	139	76	76
sandy brown	244	164	164
SandyBrown	244	164	164
sea green	46	139	139
SeaGreen	46	139	139
SeaGreen1	84	255	255
SeaGreen2	78	238	238
SeaGreen3	67	205	205
SeaGreen4	46	139	139
seashell	255	245	245
seashell1	255	245	245
seashell2	238	229	229
seashell3	205	197	197
seashell4	139	134	134
sienna	160	82	82
sienna1	255	130	130
sienna2	238	121	121
sienna3	205	104	104
sienna4	139	71	71
sky blue	135	206	206
SkyBlue	135	206	206
SkyBlue1	135	206	206
SkyBlue2	126	192	192
SkyBlue3	108	166	166
SkyBlue4	74	112	112
slate blue	106	90	90
slate gray	112	128	128
slate grey	112	128	128
SlateBlue	106	90	90
SlateBlue1	131	111	111
SlateBlue2	122	103	103
SlateBlue3	105	89	89
SlateBlue4	71	60	60
SlateGray	112	128	128
SlateGray1	198	226	226
SlateGray2	185	211	211
SlateGray3	159	182	182
SlateGray4	108	123	123
SlateGrey	112	128	128
snow	255	250	250

snow1	255	250	250
snow2	238	233	233
snow3	205	201	201
snow4	139	137	137
spring green	0	255	255
SpringGreen	0	255	255
SpringGreen1	0	255	255
SpringGreen2	0	238	238
SpringGreen3	0	205	205
SpringGreen4	0	139	139
steel blue	70	130	130
SteelBlue	70	130	130
SteelBlue1	99	184	184
SteelBlue2	92	172	172
SteelBlue3	79	148	148
SteelBlue4	54	100	100
tan	210	180	180
tan1	255	165	165
tan2	238	154	154
tan3	205	133	133
tan4	139	90	90
thistle	216	191	191
thistle1	255	225	225
thistle2	238	210	210
thistle3	205	181	181
thistle4	139	123	123
tomato	255	99	99
tomato1	255	99	99
tomato2	238	92	92
tomato3	205	79	79
tomato4	139	54	54
turquoise	64	224	224
turquoise1	0	245	245
turquoise2	0	229	229
turquoise3	0	197	197
turquoise4	0	134	134
violet	238	130	130
violet red	208	32	32
VioletRed	208	32	32
VioletRed1	255	62	62
VioletRed2	238	58	58
VioletRed3	205	50	50
VioletRed4	139	34	34
wheat	245	222	222
wheat1	255	231	231
wheat2	238	216	216
wheat3	205	186	186
wheat4	139	126	126
white	255	255	255
white smoke	245	245	245

WhiteSmoke	245	245	245
yellow	255	255	255
yellow green	154	205	205
yellow1	255	255	255
yellow2	238	238	238
yellow3	205	205	205
yellow4	139	139	139
YellowGreen	154	205	205

Appendix II

Font Families in Qi/tk

"System" "Terminal" "Fixedsys" "Roman" "Script" "Modern" "Small Fonts"
"MS Serif" "WST_Czec" "WST_Engl" "WST_Fren" "WST_Germ" "WST_Ital"
"WST_Span" "WST_Swed" "Courier" "MS Sans Serif" "Marlett" "Arial" "Arial CE"
"Arial CYR" "Arial Greek" "Arial TUR" "Arial Baltic" "Courier New"
"Courier New CE" "Courier New CYR" "Courier New Greek" "Courier New TUR"
"Courier New Baltic" "Lucida Console" "Lucida Sans Unicode"
"Times New Roman" "Times New Roman CE" "Times New Roman CYR"
"Times New Roman Greek" "Times New Roman TUR"
"Times New Roman Baltic" "Wingdings" "Symbol" "Verdana" "Arial Black"
"Comic Sans MS" "Impact" "Georgia" "Franklin Gothic Medium"
"Palatino Linotype" "Tahoma" "Trebuchet MS" "Webdings" "Estrangelo Edessa"
"Gautami" "Latha" "Mangal" "MV Boli" "Raavi" "Shruti" "Tunga" "Sylfaen"
"Microsoft Sans Serif" "Agency FB" "Arial Narrow" "Arial Rounded MT Bold"
"Blackadder ITC" "Bodoni MT" "Bodoni MT Black" "Bodoni MT Condensed"
"Book Antiqua" "Bookman Old Style" "Bradley Hand ITC" "Calisto MT"
"Castellar" "Century Gothic" "Century Schoolbook" "Copperplate Gothic Bold"
"Copperplate Gothic Light" "Curlz MT" "Edwardian Script ITC" "Elephant"
"Engravers MT" "Eras Bold ITC" "Eras Demi ITC" "Eras Light ITC"
"Eras Medium ITC" "Felix Titling" "Forte" "Franklin Gothic Book"
"Franklin Gothic Demi" "Franklin Gothic Demi Cond" "Franklin Gothic Heavy"
"Franklin Gothic Medium Cond" "French Script MT" "Garamond" "Gigi"
"Gill Sans MT Ext Condensed Bold" "Gill Sans MT" "Gill Sans MT Condensed"
"Gill Sans Ultra Bold" "Gill Sans Ultra Bold Condensed"
"Gloucester MT Extra Condensed" "Goudy Old Style" "Goudy Stout"
"Haettenschweiler" "Imprint MT Shadow" "Lucida Sans"
"Lucida Sans Typewriter" "MS Outlook" "Maiandra GD" "Monotype Corsiva"
"OCR A Extended" "Palace Script MT" "Papyrus" "Perpetua"
"Perpetua Titling MT" "Pristina" "Rage Italic" "Rockwell" "Rockwell Condensed"
"Rockwell Extra Bold" "Script MT Bold" "Tw Cen MT" "Tw Cen MT Condensed"
"Wingdings 2" "Wingdings 3" "Bookshelf Symbol 7" "MS Reference Sans Serif"
"MS Reference Specialty" "Tw Cen MT Condensed Extra Bold" "Berling Antiqua"
"Bookdings" "Frutiger Linotype"

Appendix III

Cursors in Qi/tk

X_cursor arrow based_arrow_down based_arrow_up boat bogosity bottom_left_corner bottom_right_corner bottom_side bottom_tee box_spiral center_ptr circle clock coffee_mug cross cross_reverse crosshair diamond_cross dot dotbox double_arrow draft_large draft_small draped_box exchange fleur gobbler gumby hand1 hand2 heart icon iron_cross left_ptr left_side left_tee leftbutton ll_angle lr_angle man middlebutton mouse pencil pirate plus question_arrow right_ptr right_side right_tee rightbutton rtl_logo sailboat sb_down_arrow sb_h_double_arrow sb_left_arrow sb_right_arrow sb_up_arrow sb_v_double_arrowshuttle sizing spider spraycan star target tcross top_left_arrow top_left_corner top_right_corner top_side top_tee trek ul_angle umbrella ur_angle watch xterm

Appendix IV

The Type Theory of Qi/tk

The type theory for Qi/tk occupies over 27,000 lines of Common Lisp if printed out in full. If you wish to preclude this type theory for a given application (see FPQi for **preclude**) the use **preclude-qi-tk (**below).

analog-grid

Widgets : [[widget]]; Options : (widget-options pack);
(analog-grid Widgets …. Options …) : [[widget]];

bindkey
(symbol → (list key) --> (lazy A) --> symbol)

colour?
(A --> boolean?)

cursor?
(A --> boolean)

destroy-widget
(widget --> symbol)

dynamic
(A --> A)

echo-tk
(symbol --> boolean)

flush-stream
(stream --> (list A))

get!
(Widget --> ((attribute Widget A) --> A))

getf!
(canvas --> tag --> (attribute Widget A) --> A)

gets?
(symbol --> symbol --> boolean)

get-entry
(entry --> string)

get-textbox
(text --> string)

grid

Widget : widget;
Row : number;
Column : number;
Options : (widget-options grid);
(grid Widget Row Column … Options …) : widget;

The options for grid are the same as pack except that -columnspan <number> is an option.

key?
(A --> boolean)

listen
(number --> stream --> string)

make!

where Widget ∈ {button, checkbutton, radiobutton, label, entry, text
image, font, canvas, window}

S : Widget >> Options : (widget-options Widget);
(make! S Widget … Options …) : Widget;

( ) : : (widget-options Widget);

Attribute : (attribute Widget A); Value : A; Options : (widget-options Widget);
(Attribute Value … Options ….) : (widget-options Widget);

The axioms for attributes follow the descriptions given in previous chapters e.g. -bg is an attribute of buttons whose value is a colour. In sequent calculus

---
-bg : (attribute button colour);

These axioms are too numerous to be listed here and they can be easily inferred from the tables in the previous chapters.

pack

Widgets : widgets; Options : (widget-options pack);
(pack …. Widgets …. Options …) : symbol;

( ) : widgets;

Widget : widget; Widgets : widgets;
(Widget …. Widgets …) : widgets;

The options to pack are those listed in the chapter on window management. Any object created by make! (other than fonts or images) counts as a widget. Qi/tk recognises all the following types as subtypes of the type widget.

button, checkbutton, radiobutton, label, entry, text, canvas, window

B : button if B is created by make! as a button etc.

preclude-qi-tk (given either + or - enables or disables the qi-tk type system).
(symbol --> (list symbol))

put-entry
(entry --> string --> string)

put-textbox
(text --> string --> string)

return-tk-stream
(A --> stream)

set!
(Widget --> (attribute Widget A) --> A --> A)

setf!
(canvas --> tag --> (attribute Widget A) --> A --> A)

sleep (causes the system to pause for N seconds)
(number --> (list A))

system
(string --> string)

talk
(string --> stream --> (list A))

talk-and-listen
(string --> number --> stream --> string)

widget?
(A --> boolean)

wipe
(symbol --> canvas --> (list A))

url
(string --> number --> string)