Tcltk : : CHEAT SHEET and moderate-complexity GUI example (1/3)

Introduction

Tcltk is the R implementation of the Tcl/Tk toolkit for creating GUIs. The GUIs have customizable widgets to gather inputs from the user and return them for use later by R script.

(Tk is a package for creating GUI widgets, and it runs on the programming language Tcl, or Tool Command Language.) Tcl/Tk is cross-platform and has implementations in other programming languages like `tkinter` ("Tk interface") in Python. Tcl/Tk is quite old, and for R, people have largely moved on to alternatives like Shiny. This cheat sheet is for people who have found a reason to still use Tcl/Tk in R.

```
library(tcltk2)  # tcltk is a base package in R but tcltk2 is not.
library(dplyr)
library(stringr)
```

Remember to run `install.packages("tcltk2")`. A few of its features are used in this example.

Base, canvas and frame

```
base <- tktoplevel() # This makes a new window appear on your screen
tkwm.title(base, "Burger builder")
tkwm.geometry(base, "+50+50" ) # Specify the position within your screen
```

Canvases and frames are areas that can hold widgets inside of them. This canvas `cnv` has `base` as its parent, so it will go inside of `base`. The frame `fme` has `cnv` as its parent, and `fme` will be parent to other widgets later.

```
cnv <- tkcanvas(base)
fme <- tkframe(cnv)
```

Making a canvas scrollable

Canvases have a few more features than frames, including being scrollable. If just for holding widgets, we usually use frames. The code below is complicated; just know that it makes the canvas scrollable.

```
scr <- tkscrollbar(base, repeatinterval = 5,
    orient = "vertical", command  =  function(...) tkyview(cnv,...))
xscr <- tkscrollbar(base, repeatinterval = 5,
    orient = "horizontal", command = function(...) tkxview(cnv,...))
tkconfigure(cnv, yscrollcommand = function(...) tkset(scr,...),
    xscrollcommand = function(...) tkset(xscr, ...))
tkbind(base, "<MouseWheel>",  function(...) tkyview(cnv,...)) # Bind the
  mouse wheel to the scroll function
scrollform_resize <- function(frm) {
       canvport <- tkwinfo("parent", frm)
       bbox <- tkbbox(canvport, "all")
       w <- tkwinfo("width", frm)
       tkconfigure(canvport, width = w, scrollregion = bbox,
           yscrollincrement = "0.1i")
}
tkbind(fme, "<Configure>", function () scrollform_resize(fme))
tkcreate(cnv, "window", 0, 0, anchor = "nw", window = fme)
```

Geometry managers

You must apply a geometry manager like `Tkpack` to a widget in order for it to appear
With `Tkpack`, you specify `side` (top/bottom/left/right), `anchor` (n, e, sw, etc.), `fill` (x/y/both), and `expand` (T/F). Packing is easiest for quickly arranging a few widgets. More complex geometries can be achieved by packing widgets into frames and frames into other frames.

The other geometry managers are `tkgrid` (to be used later) and `tkplace` (rarely used)
You can only have one geometry manager per parent (E.g., you cannot mix widgets with `tkpack` and `tkgrid` inside of the same frame)

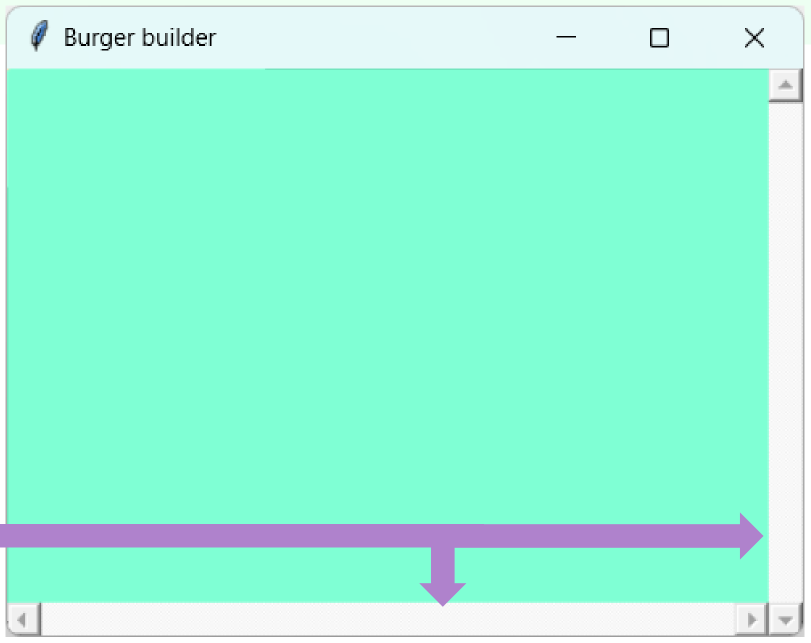
```
tkpack(scr, side = "right", fill = "y", expand = F)
tkpack(xscr, side = "bottom", fill = "x", expand = F)
tkpack(cnv, side = "bottom", fill = "both", expand = T)
```

`tkconfigure`

Use `tkconfigure` to change the options for a widget after it's already been created.

```
tkconfigure(fme, background = "wheat")
tkconfigure(cnv, background = "aquamarine")
```

`fme` is not visible below because no widgets have been added into it yet.



Additional resources on Tcl/Tk

- tcltk2 package documentation
- *A Primer on the R-Tcl/Tk Package by Peter Dalgaard*
- Tcl/Tk Reference Guide, not specific to the R package

Tcltk : : CHEAT SHEET and moderate-complexity GUI example (2/3)

Widgets w/ control variables

Label (to display text)

1a

```
lbl_bun <- tklabel(fme, text = "Select your bun", width = 25, anchor = "w")
lbl_topping <- tklabel(fme, text = "Select your topping(s)", width = 25, anchor = "w")
lbl_sauce <- tklabel(fme, text = "Selected your sauce", width = 25, anchor = "w")
lbl_togo <- tklabel(fme, text = "To-go?")
```

Geometry managers – Tkgrid

Tkgrid is the other geometry manager. You specify which row/column a widget will appear in, as if the frame were a table. This makes large numbers of widgets easier to align.

```
tkgrid(lbl_bun, row = 1, column = 1)
tkgrid(lbl_topping, row = 2, column = 1)
tkgrid(lbl_sauce, row = 4, column = 1, rowspan = 4)
tkgrid(lbl_togo, row = 7, column = 1) # (Rows 3, 5 and 6 will be used later)
```

Combobox (drop-down list)

1b

```
options_for_bun <- c("Classic", "Sesame", "Whole wheat", "Gluten-free")
var_bun <- tclVar("Sesame") # Choose the default here
cmb_bun <- tk2combobox(fme, width = 40, values = options_for_bun, textvariable = var_bun)
```

Most widgets can be assigned a control variable (like `textvariable` above) that holds the selected value. You can get the value of the variable (e.g., with `tclvalue(var_bun)`) even after the GUI is closed.

```
tkgrid(cmb_bun, row = 1, column = 2)
```

Checkbutton (Checkbox)

1c

```
var_togo <- tclVar(FALSE) # Control variable
chk_togo <- tkcheckbutton(fme, variable = var_togo)
tkgrid(chk_togo, row = 7, column = 2)
tktoggle(chk_togo) # You can check/uncheck the box programatically
```

Widgets w/o control variables

Listbox (Multiple select)

1d

Note that `listbox`es and `text` boxes do not have control variables. See later for how to get their values.

```
options_for_topping <- c("Tomatoes", "Lettuce", "Onion", "Bacon", "Pickles", "Pineapple")
lbx_topping <- tk2listbox(fme, values = options_for_topping, selectmode = c("multiple"))
tkgrid(lbx_topping, row = 2, column = 2)
lapply(c(0,1,4), function(x) tkselection.set(lbx_topping, x)) # Activate some default selections
```

Text (Text box for typing)

Advanced: We are going to allow the user to add new text boxes by clicking a button.

```
fme_patty <- tkframe(fme, background = "salmon") # A frame to help organize the patty-related widgets
tkgrid(fme_patty, row = 3, column = 1, columnspan = 2)
lbl_patty <- tklabel(fme_patty, text = "Type your patty choice", width = 25, anchor = "w")
tkgrid(lbl_patty, row = 1, column = 1)

txts_patty <- list() # The text boxes will be stored in a list (each one will be anonymous)
```

But create the first box automatically here:

1e

```
txts_patty[[1]] <- tktext(fme_patty, height = 2, width = 25)
tkinsert(txts_patty[[1]], "0.0", "beef") # Add text to the text box
tkgrid(txts_patty[[1]], row = 1, column = 2)
```

Buttons

Define a function to run when the user clicks "Add a patty".

```
on_add_patty <- function(){
  # First, get how many patties are already there
  size_of_existing_tkgrid <- fme_patty %>% tkgrid.size() %>%
    tclvalue() %>% strsplit(" ") %>% unlist() %>% as.numeric()
  n <- size_of_existing_tkgrid[2] # tkgrid.size() returns (cols+1 , rows+1)
  txts_patty[[n]] <- tktext(fme_patty, height = 2, width = 25) # Add new patty
  tkinsert(txts_patty[[n]], "0.0", paste0("beef", n)) # Add default text
  tkgrid(txts_patty[[n]], row = n, column = 2)
}
```

1f

```
btn_add_patty <- tkbutton(fme_patty, text = "Add a patty",
  command = on_add_patty)
tkgrid(btn_add_patty, row = 1, column = 3)
```

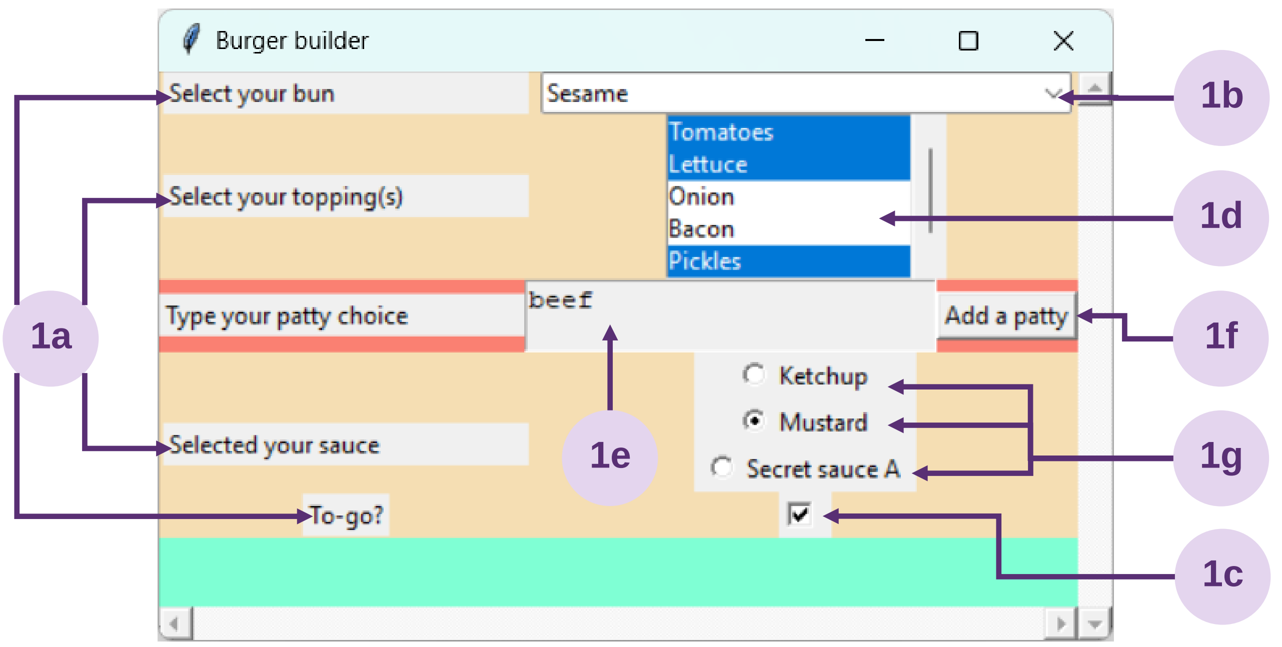
Widgets w/ control variables – Radiobutton

Each `radiobutton` is a separate widget, but we assign them all the same control variable.

```
var_sauce <- tclVar("Mustard")
rbn_ketchup <- tkradiobutton(fme, variable = var_sauce, text = "Ketchup",
  value = "Ketchup", width = 12)
rbn_mustard <- tkradiobutton(fme, variable = var_sauce, text = "Mustard",
  value = "Mustard", width = 12)
rbn_ssa <- tkradiobutton(fme, variable = var_sauce, text = "Secret sauce A",
  value = "Relish", width = 12)
```

1g

```
tkgrid(rbn_ketchup, row = 4, column = 2)
tkgrid(rbn_mustard, row = 5, column = 2)
tkgrid(rbn_ssa, row = 6, column = 2)
```



Tcltk : : CHEAT SHEET and moderate-complexity GUI example (3/3)

Buttons – An OK button

As noted above, some widgets like `text` (the patties) and `listbox` (the toppings) don't have control variables. So, you have to get the values from the widgets manually. Here we do so inside the function `on_ok()`, which is triggered when the user clicks OK.

The values can be saved to `tclArray` objects, so that they can be passed out of `on_ok()` without using global variables.

```
var_patty <- tclArray()
var_selected_values_for_topping <- tclArray()

on_ok <- function(){ # Activates when OK is clicked

  # Get the values from the listbox (toppings)
  var_selected_values_for_topping[[1]] <- tkcurselection(lbx_topping)

  # Get the values from the text boxes (patties)
  # Resort to referring to widgets by their locations in the tkgrid instead of by name
  patties <- tkgrid.slaves(fme_patty, column = 2)
  number_of_patties <- patties %>% tclvalue() %>% str_split(" ") %>% unlist() %>% length()

  for (i in 1:number_of_patties){
    patty <- tkget(tkgrid.slaves(fme_patty, row = i , column = 2), "1.0", "end-1c")
    # This gets the text from beginning to end

    var_patty[[i]] <- patty
  }

  tkdestroy(base) # Close the GUI
}

btn_ok <- tk2button(fme, text = "OK", width = 15, command = on_ok)
tkbind(base,"<Return>", on_ok) # Bind the Enter key to btn_ok
tkgrid(btn_ok, row = 8, column = 1, columnspan = 2)
```

2a

Additional

Some additional widgets that you can use. Check tcltk documentation for more.

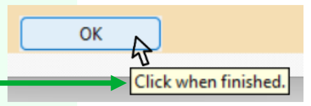
```
# tkscale (fme, variable = var_x) for a sliding scale
# tkentry (fme, textvariable = var_y) for one single line of text
# texttttkspinbox (fme, from = -5, to = 5, textvariable = var_z) for a spinbox
```

Use `tkgrid.configure` to change the `tkgrid` options for an already-gridded widget.

```
for (widget in list(lbl_bun, cmb_bun, lbl_topping, lbx_topping, fme_patty,
  lbl_patty, btn_add_patty, lbl_sauce, lbl_togo, chk_togo, btn_ok)) {
    tkgrid.configure (widget, padx = 5, pady = 5) }
```

Miscellaneous

```
# tk2tip(btn_ok, message = "Click when finished.") to add tooltips to help users
# tkcget(lbl_patty, "-text") %>% tclvalue() to get values of existing widget options
# tkmessageBox(title = "Greeting", message = "Hello") to make a pop-up box
# tkgetSaveFile(filetypes = "{ {Excel} {*.xlsx} }", defaultextension =
".xlsx") %>% tclvalue() to ask the user to specify a save file
```



Retrieving and using the values

Use this line to make the rest of your R script wait until the GUI is closed before running.

```
tkwait.window(base)
```

Remember that these widgets had control variables, so getting their values is straight-forward:

```
bun <- tclvalue(var_bun)
sauce <- tclvalue(var_sauce)
togo <- ifelse(tclvalue(var_togo) == "1", "to-go", "for here")
```

These widgets didn't have control variables, so we saved their values in `tclArray` objects:

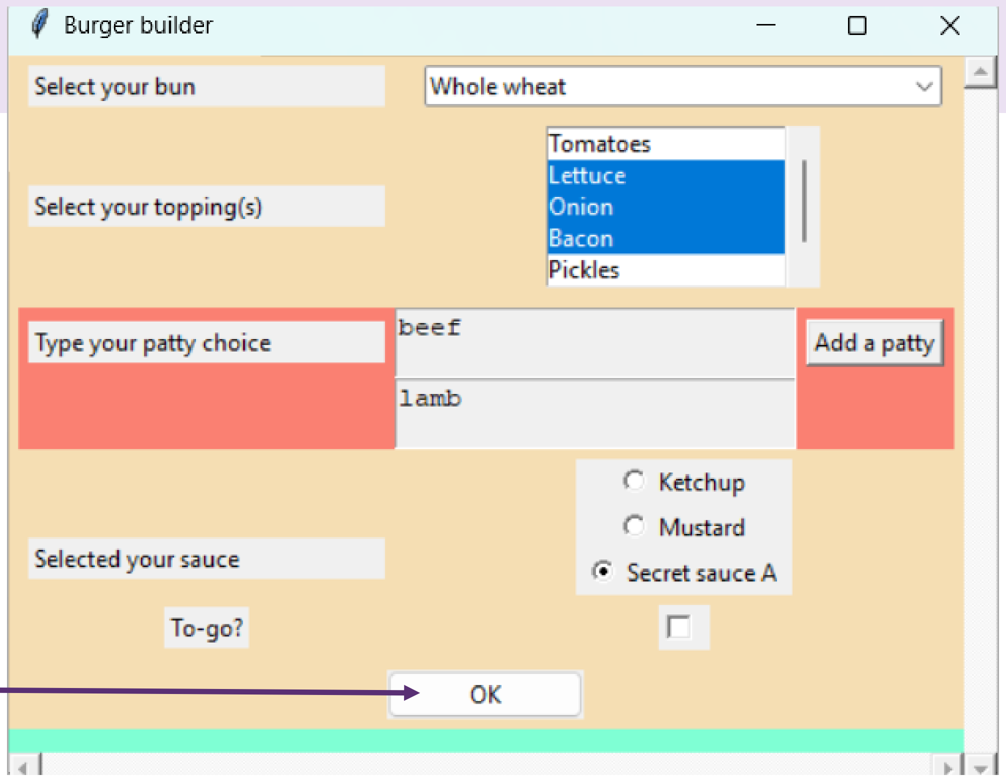
```
toppings_indexes <- var_selected_values_for_topping[[1]] %>%
    tclvalue() %>% stringr::str_split(" ") %>% unlist() %>% as.numeric()
toppings <- options_for_topping [1 + toppings_indexes] %>%
    paste(collapse = ", ") # Have to +1 because listbox uses 0-based indexing

number_of_patties <- length(var_patty)
patty <- list()
for (i in 1:number_of_patties) patty[[i]] <- tclvalue(var_patty[[i]])
patties <- paste(patty, collapse = ", ")
```

Now you can do something with the values.

```
print(paste("The order was a", patties, "burger", "with", toppings, "and",
  sauce, "on a", bun, "bun", togo, sep = " "))
```

```
[1] "The order was a beef, lamb burger with Lettuce,
Onion, Bacon and Relish on a whole wheat bun for here"
```


Burger builder

Select your bun: Whole wheat

Select your topping(s): Tomatoes, Lettuce, Onion, Bacon, Pickles

Type your patty choice: beef / lamb — Add a patty

Selected your sauce: Ketchup / Mustard / [✓ Secret sauce A]

To-go? ☐

OK

2a