User-friendliness through dynamic menus and L10n

TK À LA CARTE

CARSTEN ZERBST

If a program wants to be friendly to its users then it ought to be able to speak their language. Localisation comes as part of Tcl/Tk's basic package and dynamic menus make applications intuitive for their users.

Menus are one of the most important components of any graphical interface. Hidden behind them, in most cases, are large chunks of the application logic. This instalment of our Tcl/Tk series describes how menus can be generated and which options the toolkit offers programmers. Another aspect is the internationalisation and localisation of Tk applications. All of this is possible within the standard core of Tcl/Tk, no additional packages are required.

The "menu" command in Tk generates all sorts of menus, from entire menu bars to sub-menus or pop-up menus. Like any other widget, menus can be arranged using the layout managers grid, pack or place. However, it is better and simpler to attach the menu bar to a window:

```
. configure -menu .menu
```

This example attaches the menu ".menu" to the main window in the form of a menu bar. As shown in figures 1 to 3, Tk arranges the menu bar according to the relevant styleguide, as appropriate to the operating system on which the application is running. For example, the help menu under Linux is aligned to the right, while under Windows it is left-aligned, following the other menus. On the Macintosh the menu bar of the current application is located at the top of the desktop.

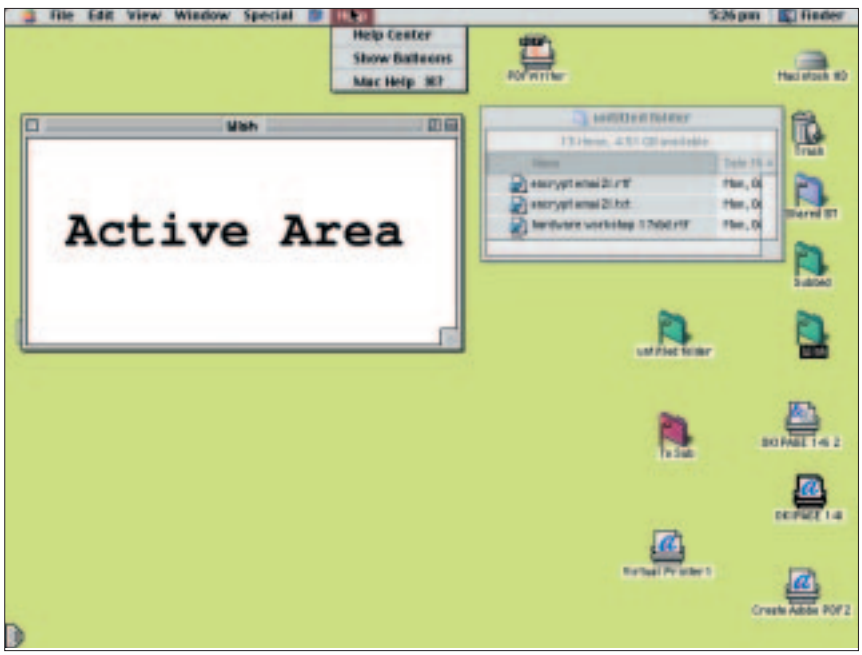

Figure 1: MacOS 9 doesn't display menus in the respective window, but always at the top of the screen. Tk keeps to this convention; even tear-off menus are implemented.

Menus within menus

A menu bar contains a number of sub-menus. These sub-menus contain the actual entries, such as commands, radio or check buttons or further sub-menus. The entries can be attached and edited with the menu widget methods "add", "configure" and "delete". The example in Listing 1 generates a menu bar and then attaches three sub-menus.

Menu labels are more important than many programmers realise. The choice of words and terminology requires a great deal of thought. On the one hand there is limited space available (a maximum of two to three words), on the other hand descriptions should be very precise and unambiguous. Instead of text, menus can display images, as in this help menu (see figure 3).

Many users want to keep their fingers on the keyboard while they are working, rather than constantly having to switch to using the mouse. They open menus using the [Alt]-key and another letter key, which is often shown as underlined in menus. In our example this method has been implemented for the Edit and the Help menu. The option "-underline *position*" tells Tk which letter to underline. This letter is also used to access the sub-menu. Tk creates the relevant bindings automatically.

Tearing off menus

Some applications use so-called tear-off menus,

Listing 1: Creating menus with Tk

```
#!/bin/sh
# Example of menus with Tk
exec wish $0 $@
# Creating a menu bar
menu .menu
# ... and linking it with the main window
. configure -menu .menu
# The sub-menus
# Simple sub-menu
.menu add cascade -label File -menu .menu.file
# Sub-menu with shortcut <ALT-E>
.menu add cascade -label Edit -menu .menu.edit -underline 0
# Sub-menu with shortcut <ALT-H>
.menu add cascade -label Help -menu .menu.help -underline 0
# The File menu
menu .menu.file
# ... with entries for commands
.menu.file add command -label Open
# ... and separation
.menu.file add separator
.menu.file add command -label Exit -command "exit" -accelerator "^q"
# Binding for Exit
bind . <Control-q> exit
# The Help menu (not tear-off)
menu .menu.help -tearoff 0
.menu.help add command -bitmap info
# The (context-sensitive) Edit menu
menu .menu.edit -postcommand menuEdit
set context normal
# The procedure for creating the Edit menu
proc menuEdit {} {
 .menu.edit delete 0 end
 .menu.edit add command -label "Dependant command" -state $::context
 .menu.edit add separator
 .menu.edit add radiobutton -label On -value normal -variable ::context
 .menu.edit add radiobutton -label Off -value disabled -variable ::context
}
# Active area with different cursor
frame .a
grid .a
label .a.b -font {Sans 30 bold} -cursor crosshair -text "Active area"
grid .a.b -padx 25 -pady 25
# Binding for opening the pop-up menu
bind .a.b <Button-3> {
  tk_popup .menu.edit %X %Y
}
```

that is menus which can be separated from the main application. A prominent example of this is the Gimp. By default tearing off is also possible for Tk menus, however, it can be prevented with the option "-tearoff 0".

Context-sensitive menus can make an application more user-friendly. Depending on the current environment (the context) they only offer those commands whose use would be sensible at the moment. All other commands are only shown by Tk in grey. This tells the user which commands are available and which of those he is currently able to use.

The current state of a menu entry is determined by the option "-state". A command is usually "normal", but in its "disabled" state it is greyed out and cannot be selected.

Context-sensitive menus

But how do menus become context-sensitive? The simplest way is for the program to re-create them every time they are opened. This is where the menu widget option "-postcommand *command*" is useful: this command is called by Tk before it opens a menu. Generating a menu normally takes less than a millisecond, the delay should therefore not cause any problem. The individual menu entries are simply created by the script in the relevant mode. However, this simple method fails to work with tear-off menus which are always open and only change their state when the menu is re-opened from the menu bar.

Pop-up menus can be just as useful as context-sensitive menus. They are normally accessed via the right mouse button and provide the users with important commands in specific areas of the application which are only relevant there. The command "tk_popup *menu x y*" is used to open a pop-up menu at any point within an application.

The example in Listing 1 creates an active area (see also figure 3) in which the right mouse button opens the pop-up menu. No additional menu is required for this, the existing Edit menu is simply re-used.

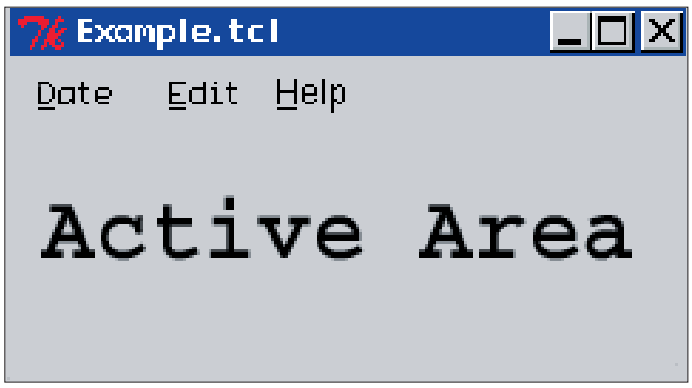

Figure 2: The menu bar under Windows does not stand out from the rest of the application if the program does not have its own background colour. The underlined letters are used as shortcuts for accessing the menus.

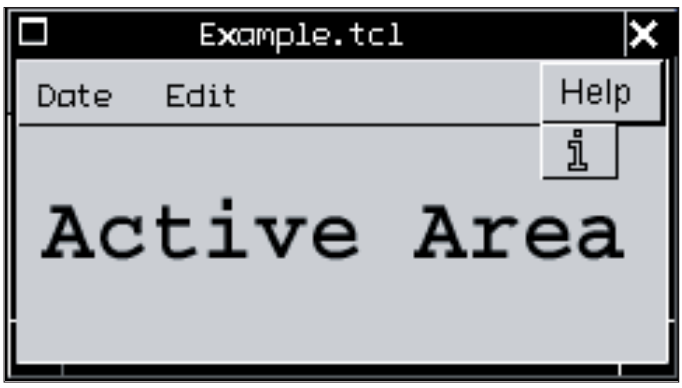

Figure 3: Tk menus under Linux come in the familiar Motif look – the menu bar has a raised 3-D effect. The help menu is always at the right-hand side of the menu bar.

Listing 2: Localisation with Tk

```
#!/bin/sh
# Example for the localisation
# of Tcl applications ➥
exec wish $0 $@

package require msgcat

# Selecting the system settings
msgcat::mclocale $env(LANG)

# Loading the language catalogs
msgcat::mcload [pwd]

# Assembling the GUIs
label .l -text [msgcat::mc “Default is %s”➥
$env(LANG)]
grid .l
label .l2 -text [msgcat::mc date 10 15]
grid .l2
button .b -text [msgcat::mc “Exit”] -command➥
exit
grid .b -padx 5 -pady 5

wm geometry . 220x75
wm title . “Deutsch”
#wm title . “English”
```

Keyboard shortcuts

The more frequently you work with an application the more likely you are to start wishing for shortcuts. The pop-up menus we have just described will help a bit. But rather than fighting your way through different menus every time, a shortcut is a much quicker way of getting to where you want to go.

In order for users to be able to learn these shortcuts quickly it makes sense to put them next to the relevant menu entry. This can be done using the option “-accelerator *shortcut*”. Programmers must be careful, however, not to use standard shortcuts for their own purposes. For example, users expect [Ctrl]+[q] to exit an application.

Unlike the “-underline” option, the binding for shortcuts has to be specially created using the “bind” command. If you have a number of such commands, it’s also worth having a look at the powerful virtual events.

Tk multilingual

Applications don’t only display text in menus but also in many other areas. Since not all users can necessarily be expected to speak English, interfaces should be available in several languages. That is called localisation or L10n. Under Tcl the Msgcat package is the most suitable for this purpose. It is part of the normal core of Tcl and will also be used for system messages in the forthcoming Tcl 8.4. Msgcat uses catalogues, which contain translations for different labels or outputs.

Listing 2 shows a simple example of using the Msgcat package. The translation catalogue for German must be held in the file “de.msg” (see Listing 3), the English messages are contained in “en.msg” (see Listing 4). The example starts off by setting the desired language (locale) with “msgcat::mclocale *locale*”. Locales are described using country codes according to ISO 639. The user setting is normally found the environment variable “LANG”.

The language catalogues have to be loaded next. This is done with the command “msgcat::mcload *directory*”. The specified directory should contain one file per language with the relevant translation. The file names consist of the country codes and the extension “.msg”. These files contain a number of commands of the type “msgcat::mcset *language*” “*original*” “*translation*”.

A catalogue for every language

These files are very easy to create with the special editor MSGedit. Within the localised application itself every character string for a message or a label simply has to be replaced with a call to

Listing 3: German messages

```
msgcat::mcset de “Default is
%s” ➥
“Systemeinstellung ist %s”
msgcat::mcset de date ➥
“Beispieldatum %2\$i.%1\$i
(Tag, Monat)”
msgcat::mcset de “Exit”
“Beenden”
```

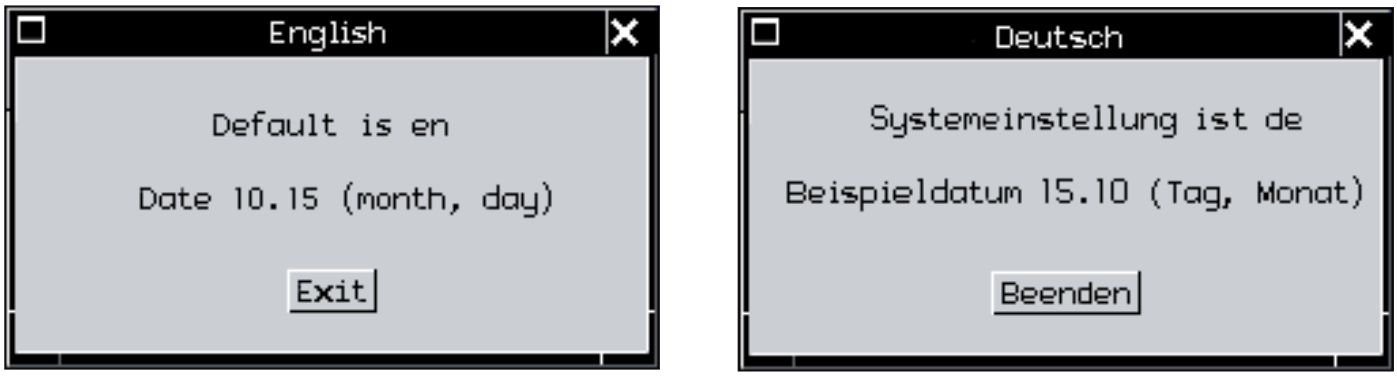

Figure 4: Another aspect of localisation is that in US English the month comes before the day, but the reverse is true for UK English, where the day comes before the month.

Info

Country codes according to ISO 639: ***http://www.oasis-open.org/cover/iso639a.html***
MSGedit: ***http://www.tu-harburg.de/~skfcz/tcltk.html***
Balloon help: ***http://purl.org/thecliff/tcl/wiki/534.html***
Tcllib, BWidgets: ***http://sourceforge.net/projects/tcllib/***
OSCON-Tcl-Papers: ***ftp://ftp.oreilly.com/pub/conference/os2001/tcl_papers***
Gnocl: ***http://www.dr-baum.net/gnocl/***
Agenda web browser: ***http://www.psnw.com/~alcald/tiny_tcl_web_browser.html***
Etlinux: ***http://www.etlinux.org/***

Listing 4: English catalogue

```
msgcat::mcset en date "Date?
%i.%i (month, day)"
```

"[msgcat::mc "*original*"]". The command returns the translation from the specified catalogue. If no translation is found "msgcat:mc" returns the original character string so that the application remains useable. The catalogue for the original language is therefore almost empty (see Listing 4).

However, in many cases this is not enough: messages often consist of a fixed and a dynamic part. Translations can therefore contain formatting commands, even to the point where the order of the dynamic components can differ between languages. In the example this feature is used to output the date in the right order, once as Day, Month and once as Month, Day (see figure 4).

Depending on the environment variable "LANG" the application starts with an English ("en") or a German interface ("de"). In the tcsh this is set using "setenv LANG en", in bash with "LANG=en". An export may also be required.

The features introduced here allow you to create pretty user-friendly interfaces. In addition, there are, for instance, help balloons, also called tooltips. How to create these is described at the purl.org website, the BWidgets already contain an appropriate widget.

After this excursion to the surface the next article in our Tcl/Tk series will describe how you can create visualisations with VTK.

The author

Carsten Zerbst is a member of staff at Hamburg-Harburg Technical University. Apart from researching service integration on board ships, he investigates Tcl in all its forms. He is looking for new tasks in a Unix/Linux environment.

News from the Tcl world

Despite the summer break, a lot has happened since the last instalment of our Tcl/Tk series. OSCON has just finished. Naturally this conference also had a Tcl track. The papers for the presentations can be found at the O'Reilly FTP site. Among the topics covered was spoken language as application input and output. The focus, however, was the integration of Tcl in applications for ECAD, one of the Tcl domains.

Tcl distribution from Active State

Just in time for OSCON Active State have published their own Tcl distribution. The package, a full 31MB when unpacked, contains important extensions along with the actual interpreter, for example [incr Tcl], TclX, Expect, Tcllib and Tkcon. Furthermore, the package is supplied with an installation program and is currently likely to be the simplest way to an all-inclusive installation of Tcl/Tk.

You can, of course, still get the individual components free on the net. Special mention must go to Tcllib and the BWidgets. Both are available from Sourceforge in a joint project. Tcllib consists of several modules for everyday problems, such as Base64 encoding, MIME handling or POP3 and FTP access. Like the BWidgets, Tcllib is written in Tcl itself and does not require compilation.

The BWidgets are a useful collection of widgets that are normally missing in Tk (see figure 5). All new widgets have been built using existing Tk-Widgets (so-called mega-widgets). The package contains, among other things, a combobox, a tree-widget, a notebook widget and a spinbox. It also supports drag & drop.

Great plans

The Tcl core team is currently discussing the integration of new widgets and additional options in Tk. In many cases implementations already exist. Another, very controversial discussion deals with the integration of [incr Tcl] into normal Tcl.

A further topic under discussion is theming support for Tk; in this respect Tk is well behind Gtk or Qt. Anyone still wanting to use themes with Tcl at the moment can resort to Peter Baum's Gnocl which allows the use of Gtk and Gnome widgets.

PDA-capable applications

Alexander Caldwell has written a small browser and an email client especially for the Linux PDA Agenda. Both applications are geared towards the fairly limited capabilities of PDAs. The use of interpreter languages makes a lot of sense especially in this environment: instead of several applications running, just one interpreter at runtime is sufficient. Scripts can also be stored in a compressed format, with scripts normally being able to be compressed significantly more than binary programs. In Etlinux Tcl scripts even control the entire boot process.

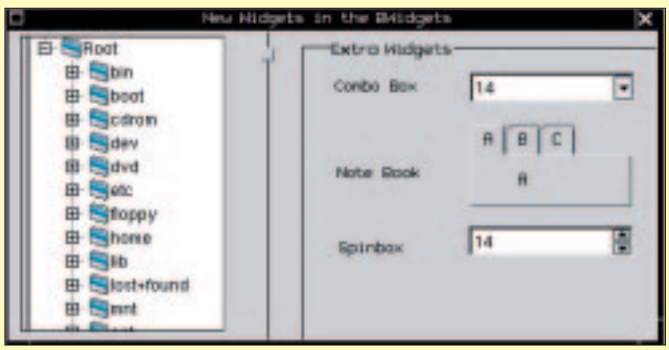

Figure 5: BWidgets contain new widgets for Tk. Tree, Paned Window, Combobox, Notebook and Spinbox are standard features of many modern interfaces.