在 Linux上运行 Tcl/Tk 编程语言

Run Tcl/Tk on Linux and Windows 10

William Xu
6/10/2022

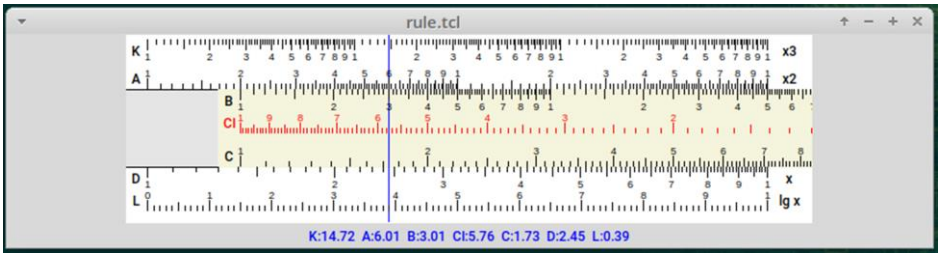
摘要

Tcl/Tk 如同 Basic一样是一种简单的描述性编程语言. 它是一种跨机种的语言, 用简单几句就可以编制专业性的用户图形接口, 比 VC++容易和快捷得多. 本文简述 Tcl/Tk 在 Linux 上运行的方法. 文中所述各样例程序在下列两个机器中测试无误:

A) VM VirtualBox上的 Linux OpenSuSE.

B) Raspberry Pi 4 上的 Linux ubuntu MATE.

在网上找到了一个精美的 Tcl/Tk 程序, 短短几句话就能实现一个真实的计算尺, 见下图. 如用VC++来写, 程序最起码有几頁长.



在 Linux上运行 Tcl/Tk 编程语言

目 录

I. 什么是 Tcl/Tk

1.1 中文描述

中国人的习惯是开门见山所以只要一句话就可以讲清楚:

Tcl/Tk可以分为两部分,Tcl和Tk. Tcl如同 Baisc语言一样提供算法; Tk 提供用户图形接口. (参见 Appendix A)

1.2 英文描述

洋人喜欢兜圈, 搞繁琐哲学, 看下面 Tcl, Tk的分别描述.

1.2.1 关于 Tcl

Tcl is shortened form of **Tool Command Language**. John Ousterhout of the University of California, Berkeley, designed it. It is a combination of a scripting language and its own interpreter that gets embedded to the application, we develop with it.

Tcl was developed initially for Unix. It was then ported to Windows, DOS, OS/2, and Mac OSX. Tcl is much similar to other unix shell languages like Bourne Shell (Sh), the C Shell (csh), the Korn Shell (sh), and Perl.

It aims at providing ability for programs to interact with other programs and also for acting as an embeddable interpreter. Even though, the original aim was to enable programs to interact, you can find full-fledged applications written in Tcl/Tk.

Tcl 是工具命令语言的缩写形式。 加州大学伯克利分校的 John Ousterhout 设计了它。 它是脚本语言和嵌入到应用程序中的自己的解释器的组合，我们用它来开发程序。

Tcl 最初是为 Unix 开发的。 然后它被移植到 Windows、DOS、OS/2 和 Mac OSX。 Tcl 与其他 unix shell 语言非常相似，例如 Bourne Shell (Sh)、C Shell (csh)、Korn Shell (sh) 和 Perl。

它旨在为程序提供与其他程序交互的能力，并充当可嵌入的解释器。 尽管最初的目标是使程序能够交互，但您可以找到用 Tcl/Tk 编写的成熟应用程序。

1.2.2 关于 Tk

Tk refers to Toolkit and it provides cross platform GUI widgets, which helps you in building a Graphical User Interface. It was developed as an extension to Tcl scripting

language by John Ousterhout. Tk remained in development independently from Tcl with version being different to each other, before, it was made in sync with Tcl in v8.0.

Tk 指的是 Toolkit，它提供了跨平台的 GUI 小部件(widgets)，可帮助您构建图形用户界面。 它是由 John Ousterhout 作为 Tcl 脚本语言的扩展开发的。 Tk 保持独立于 Tcl 的开发，版本彼此不同，之前，它是在 v8.0 中与 Tcl 同步的。

II. Tcl/Tk 编程参考资料

关于 Tcl/Tk 编程的问题，都可以在下列网站找到答案:
https://www.tutorialspoint.com/tcl-tk/tcl_environment.htm

III. Tcl/Tk 编程实用的几条命令

3.1 Install Tcl/Tk

以下命令安装 Tcl/Tk: sudo apt-get install tcl tk <CR>
如果安装成功，系统就会产生两条有效命令:
tclsh 和 wish

tclsh 用于执行只有 Tcl 命令的 *.tcl文件; wish 可以执行带有 Tcl和Tk两种命令的 *.tcl 文件.

两种方法运行 *.tcl文件:
A) > tclsh myTclProg.tcl <CR>
B) > wish myTclTkProg.tcl <CR>

附加说明: 如果在 *.tcl文件的开始加了如下语句
package require Tk
tclsh 也能执行带有 Tcl 和 Tk 两种命令的 *.tcl 文件.

3.2 检查Tcl/Tk版本

tclsh <CR>
info patchlevel <CR>
看到显示版本是 8.6.7

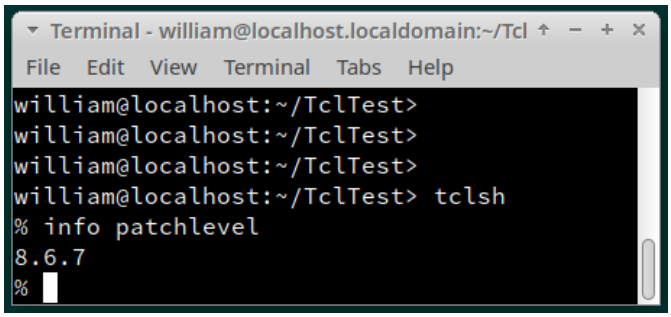


图2. 检查Tcl/Tk版本

3.3 简单测试程序 test.tcl

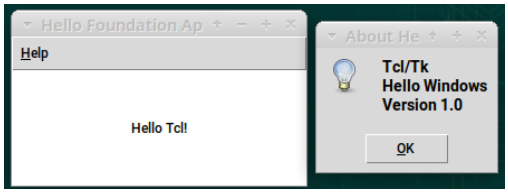


图3. test.tcl

```
#! /bin/env tclsh
package require Tk
#  Create the main message window
message .m -text {Hello Tcl!} -background white
pack .m -expand true -fill both -ipadx 100 -ipady 40

#  Create the main menu bar with a Help-About entry
menu .menubar
menu .menubar.help -tearoff 0
.menubar add cascade -label Help -menu .menubar.help -underline 0
.menubar.help add command -label {About Hello ...} \
   -accelerator F1 -underline 0 -command showAbout

#  Define a procedure - an action for Help-About
proc showAbout {} {
```

```
    tk_messageBox -message "Tcl/Tk\nHello Windows\nVersion 1.0" \
        -title {About Hello}
}

#  Configure the main window
wm title . {Hello Foundation Application}
. configure -menu .menubar -width 200 -height 150
bind . {<Key F1>} {showAbout}
```

3.4 启动测试程序

可以用 tclsh 命令或者 wish 命令启动测试程序. 每个命令都有两种方式可使用. 见下表:

	tclsh 命令	wish 命令
1	tclsh test.tcl <CR>	wish test.tcl <CR>
2	> tclsh <CR> #> source test.tcl <CR>	> wish <CR> #> source test.tcl <CR>

如果运行程序出错, 可参见 Appendix A.

IV. 几个实用的 Tcl/Tk 样例程序

4.1 Select File　file.tcl

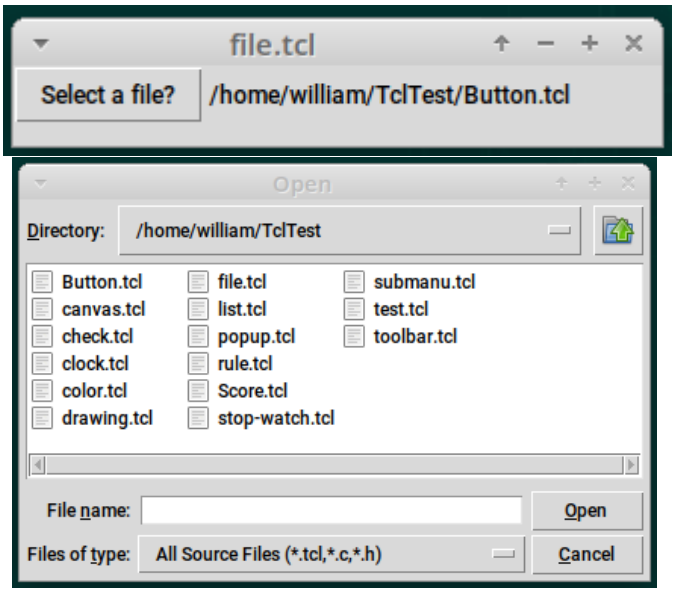


图4. file.tcl

```
#!/usr/bin/env tclsh
package require Tk

set types {
        {"All Source Files"     {.tcl .c .h}    }
        {"Image Files"          {.gif}          }
        {"All files"            *}
}

proc doIt {label} {
    global types
    set file [tk_getOpenFile -filetypes $types -parent .]
    $label configure -text $file
}

label .l -text "No File"
button .b -text "Select a file?" \
        -command "doIt .l"

grid .b -row 0 -column 0
grid .l -row 0 -column 1
```

4.2 Stop-Watch

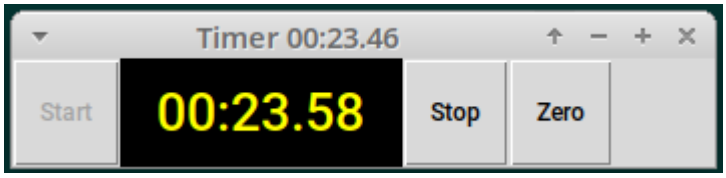


图5. Stop-watch.tcl

```
#! /usr/bin/tclsh
package require Tk

option add *Button.padY 0       ;# to make it look better on Windows
option add *Button.borderWidth 1
#-------------------------------------------------- testing i18n
package require msgcat
namespace import msgcat::mc msgcat::mcset
mcset de Start Los
mcset de Stop  Halt
mcset de Zero  Null
mcset fr Start Allez
mcset fr Stop  Arrêtez
mcset fr Zero  ???
mcset zh Start \u8DD1
mcset zh Stop  \u505C
mcset zh Zero  ???
msgcat::mclocale en ;# edit this line for display language
#------------------------------------------------------------ UI
button .start -text [mc Start] -command Start
label  .time -textvar time -width 9 -bg black -fg yellow -font "Sans 20"
set time 00:00.00
button .stop -text [mc Stop] -command Stop
button .zero -text [mc Zero] -command Zero
set state 0
bind . <Key-space> {
   if {$state} {.stop invoke
   } else {
      .start invoke
   }
}
bind . <Key-0> {
```

```
    .zero invoke
}
eval pack [winfo children .] -side left -fill y
#-------------------------------------------------------- procedures
proc every {ms body} {eval $body; after $ms [namespace code [info level 0]]}

proc Start {} {
    if {$::time eq {00:00.00}} {
        set ::time0 [clock clicks -milliseconds]
    }
    every 20 {
        set m [expr {[clock clicks -milliseconds] - $::time0}]
        set ::time [format %2.2d:%2.2d.%2.2d \
            [expr {$m/60000}] [expr {($m/1000)%60}] [expr {$m%1000/10}]]
        incr ::titleskip
        if {$::titleskip >= 12} {
            wm title . "Timer $::time"
            set ::titleskip 0
        }
    }
    .start config -state disabled
    set ::state 1
}
proc Stop {} {
    if {[llength [after info]]} {
        after cancel [after info]
    }
    .start config -state normal
    set ::state 0
}
proc Zero {} {
    set ::time 00:00.00
    set ::time0 [clock clicks -milliseconds]
}
```

4.3 计算尺 slide-rule.tcl

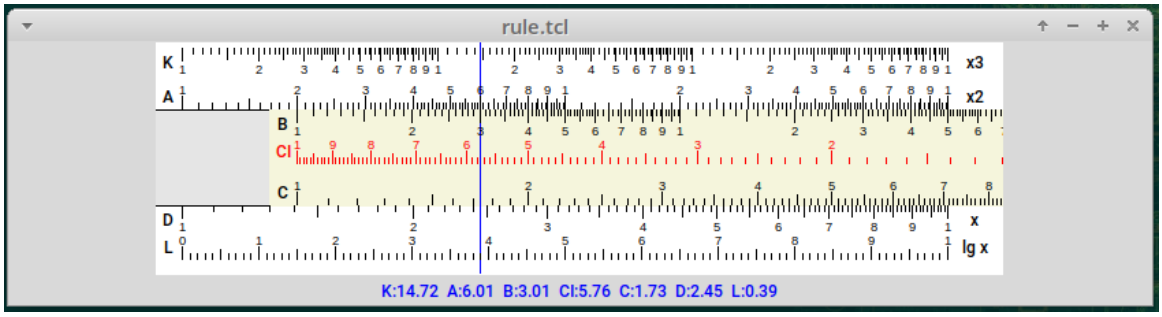


图6. slide-rule.tcl

```
#! /bin/env tclsh
package require Tk

 proc ui {} {
   set width 620
   pack [canvas .c -width $width -height 170 -bg white]
   pack [label .l -textvariable info -fg blue] -fill x
   .c create rect 0 50 $width 120 -fill grey90
   .c create rect 0 50 $width 120 -fill beige -outline beige \
      -tag {slide slidebase}
   .c create line 0 0 0 120 -tag mark -fill blue
   drawScale .c K  x3    10 5    5 log10 1 1000 186.6666667
   drawScale .c A  x2    10 50  -5 log10 1 100 280
   drawScale .c B  x2    10 50   5 log10 1 100 280 slide
   drawScale .c CI 1/x   10 90 -5 -log10 1 10  560 slide
   drawScale .c C  x     10 120 -5 log10 1 10  560 slide
   drawScale .c D  x     10 120  5 log10 1 10  560
   drawScale .c L "lg x" 10 160 -5 by100  0 10   5600
   bind .c <Motion> {.c coords mark %x 0 %x 170; set info [values .c]}
   bind .c <1> {set x %x}
   bind .c <B1-Motion> {%W move slide [expr {%x-$x}] 0; set x %x}
   bind . <Shift-Left>  {.c move slide -1 0; set info [values .c]}
   bind . <Shift-Right> {.c move slide  1 0; set info [values .c]}
   bind . <Left>  {.c move mark -1 0; set info [values .c]}
   bind . <Right> {.c move mark  1 0; set info [values .c]}
}

 proc drawScale {w name label x y dy f from to fac {tag {}}} {
```

```
  set color [expr {[string match -* $f]? "red": "black"}]
  $w create text $x [expr $y+2*$dy] -text $name -tag $tag -fill $color
  $w create text 600 [expr $y+2*$dy] -text $label -tag $tag -fill $color
  set x [expr {[string match -* $f]? 580: $x+10}]
  set mod 5
  set lastlabel ""
  set lastx 0
  for {set i [expr {$from*10}]} {$i<=$to*10} {incr i} {
    if {$i>100} {
      if {$i%10} continue ;# coarser increments
      set mod 50
    }
    if {$i>1000} {
      if {$i%100} continue ;# coarser increments
      set mod 500
    }
    set x0 [expr $x+[$f [expr {$i/10.}]]*$fac]
    set y1 [expr {$i%(2*$mod)==0? $y+2.*$dy: $i%$mod==0? $y+1.7*$dy: $y+$dy}]
    set firstdigit [string index $i 0]
    if {$y1==$y+$dy && abs($x0-$lastx)<2} continue
    set lastx $x0
    if {$i%($mod*2)==0 && $firstdigit != $lastlabel} {
      $w create text $x0 [expr $y+3*$dy] -text $firstdigit \
        -tag $tag -font {Helvetica 7} -fill $color
      set lastlabel $firstdigit
    }
    $w create line $x0 $y $x0 $y1 -tag $tag -fill $color
  }
}

proc values w {
  set x0 [lindex [$w coords slidebase] 0]
  set x1 [lindex [$w coords mark] 0]
  set lgx [expr {($x1-20)/560.}]
  set x [expr {pow(10,$lgx)}]
  set lgxs [expr {($x1-$x0-20)/560.}]
  set xs [expr {pow(10,$lgxs)}]
  set res    K:[format %.2f [expr {pow($x,3)}]]
  append res "  A:[format %.2f [expr {pow($x,2)}]]"
```

```
  append res "  B:[format %.2f [expr {pow($xs,2)}]]"
  append res "  CI:[format %.2f [expr {pow(10,-$lgxs)*10}]]"
  append res "  C:[format %.2f $xs]"
  append res "  D:[format %.2f $x]"
  append res "  L:[format %.2f $lgx]"
}

proc pow10 x {expr {pow(10,$x)}}
proc log10 x {expr {log10($x)}}
proc -log10 x {expr {-log10($x)}}
proc by100  x {expr {$x/100.}}

#--------------------------------
ui
bind . <Escape> {exec wish $argv0 &; exit}
```

4.4 Check Widget check.tcl

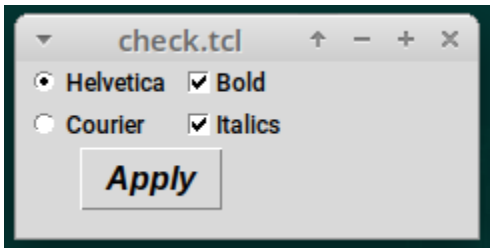


图7. check.tcl

```
#! /bin/env tclsh
package require Tk
set font Helvetica

proc applyIt { } {
   global bold italics font
   if {$bold} {set weight bold} {set weight normal}
   if {$italics} {set slant italic} {set slant roman}
   .b configure -font "-family $font -weight $weight -slant $slant"
}
checkbutton .c1 -text Bold -variable bold   -anchor w
checkbutton .c2 -text Italics -variable italics  -anchor w
radiobutton .r1 -text Helvetica -variable font -value helvetica
```

```
radiobutton .r2 -text Courier   -variable font -value courier

button .b -text Apply \
   -command "applyIt"
applyIt
#The "sticky" option aligns items to the left (west) side
grid .c1 -row 0 -column 1 -sticky w
grid .c2 -row 1 -column 1 -sticky w
grid .r1 -row 0 -column 0 -sticky w
grid .r2 -row 1 -column 0 -sticky w
grid .b -row 2 -column 0 -columnspan 2
```

Appendix A Tk的功能 - 为Tcl增加图形接口

前面已经说过: Tcl如同 Baisc 一样提供算法; Tk 提供用户图形接口. Tcl 主要用于计算. 它提供数字的定义, 必要的计算公式和方法, 比如 loop, if {} then {}, while, 等等.

Tk 提供用户图形接口. 在下面的Table 10-1和Table 10-2 中我们看到 Tk 定义了许多图形命令, 比如 button, menu, checkbox, canvas 等等.

如果在程序运行的时候出现错误:

invalid command “某个Tk 图形命令”

就会猜想到可能 Tk 软件没有 Load 进去. 这个时候就要在程序的开头加上两行字:

```
#! /bin/env tclsh  (OR  #! usr/bin/tclsh )
package require Tk
```

例如在网上下载的一个程序 drawing.tcl运行时出错 (图8)

Error: invalid command name “Canvas”

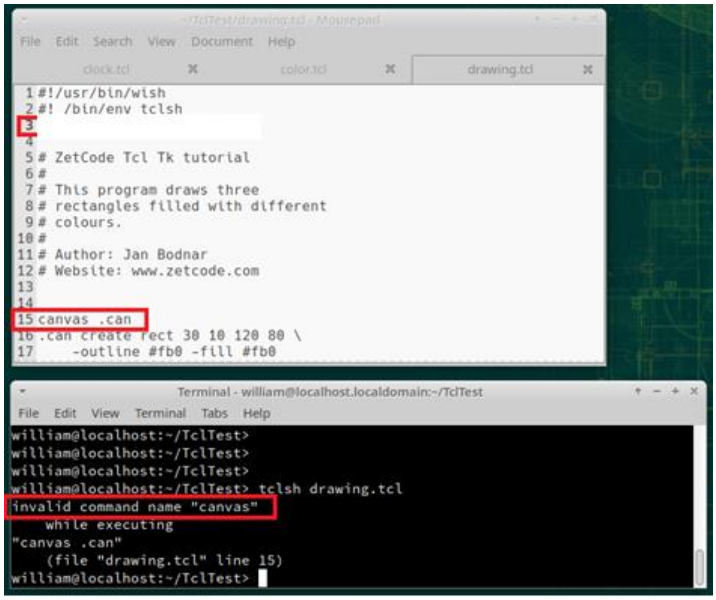


图8. drawing.tcl error: invalid command name “Canvas”

从Table 10-1这表中查出, “Canvas”是一个 Tk 的命令. 因此我们只要加上这一句 package require Tk 就解决问题了. 见图9.

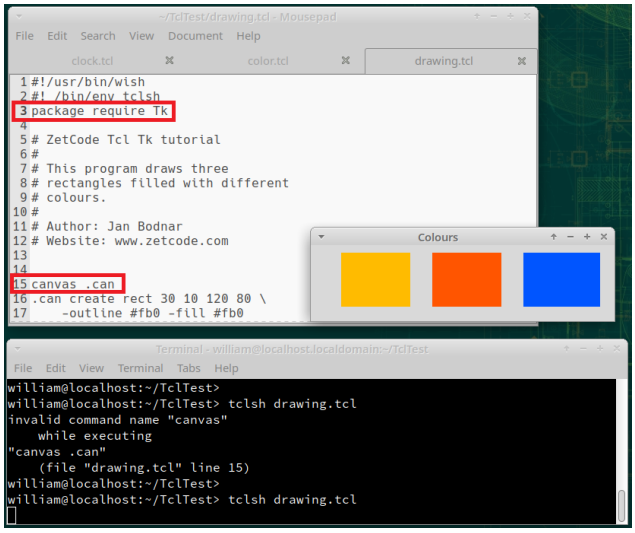


图9. Drawing.tcl

A.1 Drawing.tcl

```
#! /usr/bin/wish
#! /bin/env tclsh
package require Tk

# ZetCode Tcl Tk tutorial
#
# This program draws three
# rectangles filled with different
# colours.
#
# Author: Jan Bodnar
# Website: www.zetcode.com
canvas .can
.can create rect 30 10 120 80 \
   -outline #fb0 -fill #fb0
.can create rect 150 10 240 80 \
   -outline #f50 -fill #f50
.can create rect 270 10 370 80 \
   -outline #05f -fill #05f
pack .can
wm title . "Colours"
wm geometry . 400x100+300+300
```

Appendix B　Tk Widget Commands

下面的 Table 10-1 和 Table 10-2 两个表系从 Book “Practical Programming in Tcl and Tk” 中抄录而来.

B.1 Tk Widget Creation Commands

Table 10–1 Tk widget-creation commands

Command	Pg.	Description
button	145	Create a command button.
checkbutton	149	Create a toggle button that is linked to a Tcl variable.
radiobutton	149	Create one of a set of radio buttons that are linked to one variable.
menubutton	153	Create a button that posts a menu.
menu	153	Create a menu.
canvas	227	Create a canvas, which supports lines, boxes, bitmaps, images, arcs, text, polygons, and embedded widgets.
label	165	Create a read-only, one-line text label.
entry	180	Create a one-line text entry widget.
message	167	Create a read-only, multi-line text message.
listbox	183	Create a line-oriented, scrolling text widget.
text	212	Create a general purpose text widget.
scrollbar	172	Create a scrollbar that can be linked to another widget.
scale	169	Create a scale widget that adjusts the value of a variable.
frame	163	Create a container widget that is used with geometry managers.
toplevel	163	Create a frame that is a new top level X window.

B.2 Tk Widget Manipulation Commands

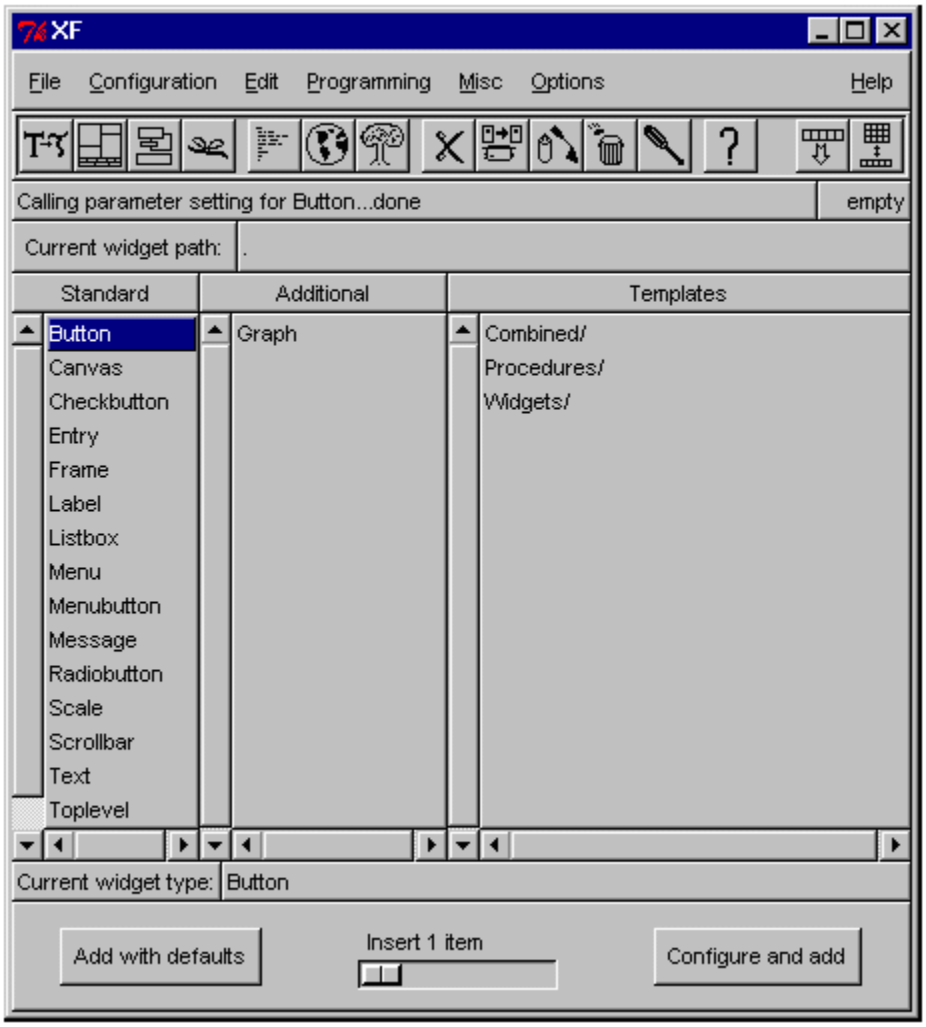
Table 10–2 Tk widget-manipulation commands

Command	Pg.	Description
`after`	259	Execute a command after a period of time elapses.
`bell`	176	Ring the X bell device.
`bind`	133	Bind a Tcl command to an X event.
`bindtags`	134	Create binding classes and control binding inheritence.
`clipboard`	255	Manipulate the X clipboard .
`destroy`	200	Delete a widget.
`fileevent`	260	Associate Tcl commands with file descriptors.
`focus`	195	Control the input focus.
`grab`	197	Steal the input focus from other widgets.
`image`	281	Create and manipulate images.
`lower`	132	Lower a window in the stacking order.
`option`	325	Access the Xresources database.
`pack`	130	The packer geometry manager.
`place`	130	The placer geometry manager.
`raise`	132	Raise a window in the stacking order.
`selection`	254	Manipulate the X PRIMARY selection.
`send`	261	Send a Tcl command to another Tk application.
`tk`	313	Query internal Tk state (e.g., the color model)
`tkerror`	87	Handler for background errors.
`tkwait`	198	Block awaiting an event.
`update`	200	Update the display by going through the event loop.
`winfo`	308	Query window state.
`wm`	303	Interact with the window manager.

Appendix C　XF – 有趣的Tcl/Tk-IDE

非常有趣的 IDE, 帮你建立Tcl/Tk 程序.

XF is a GUI builder for TCL/TK
https://web.archive.org/web/20070901150259/http://home.nycap.rr.com/dlabelle/xf/xf.html



C.1 Overview

XF is one of the original GUI builders for TCL/TK. It was developed in 1993 by Sven Delmas. Since I used **XF** so much at work and home, in 1998 I decided to adopt **XF**. Since then, I have provided bug fixes and integrated support for some of the BLT widgets into the application.
XF is a fairly complete development environment for the TCL/TK scripting language. However, much has happened to the TCL/TK language since the early

1990s. TCL/TK has a large number of new features that didn't exist at the time XF was first created. Eventually, I felt that a brand new design was better than trying to modify the existing XF code.

This new design has taken the form of the TKproE program and I will no longer actively maintain XF itself.. Feel free to check out TKproE. I think you will be pleased.

You can still download the last version of XF using the links below,

C.2 Download the latest version - XF 4.3

- Read description of version 4.3 bug fixes and enhancements
- Installation instructions for version 4.3
- Download XF4.3 zip file (for Windows)
- Download XF4.3 compressed tar file (for UNIX)

https://web.archive.org/web/20061218060238/http://home.nycap.rr.com/dlabelle/xf/download/xf43.zip

我下载的是 XF4.3 zip file (xf43.zip) for Windows. Unzip 以后把所有文件放在 c:/xf 目录下面. 运行 XF 程序只要调用 xf/src/xfmain.tcl 这个文件就可以了. 见下图.

```
Type:   > wish c:/xf/src/xfmain.tcl <CR>
          或者
        > wish c:/xf/src/xfmain.tcl my-new.tcl <CR>
```

注: 已经在Windows 10上面安装了 “Tcl/Tk for Windows”.
见 Appendix E Install Tcl/Tk for Windows

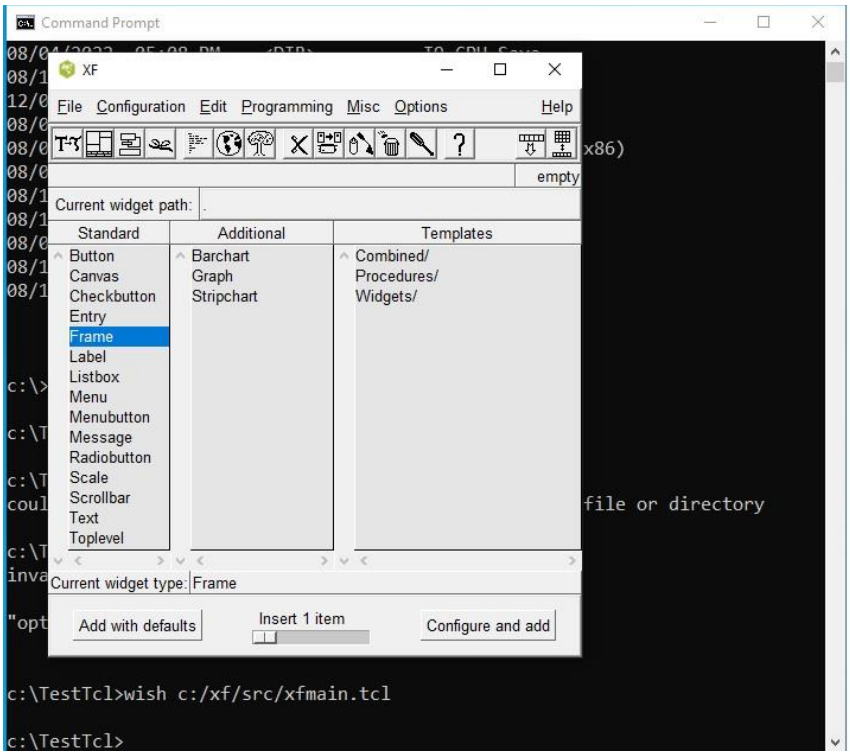


图10. 运行 XF IDE

Appendix D TKproE – 另外一个Tcl/Tk-IDE

这是另外一个与 XF 相似的 Tcl/Tk IDE.

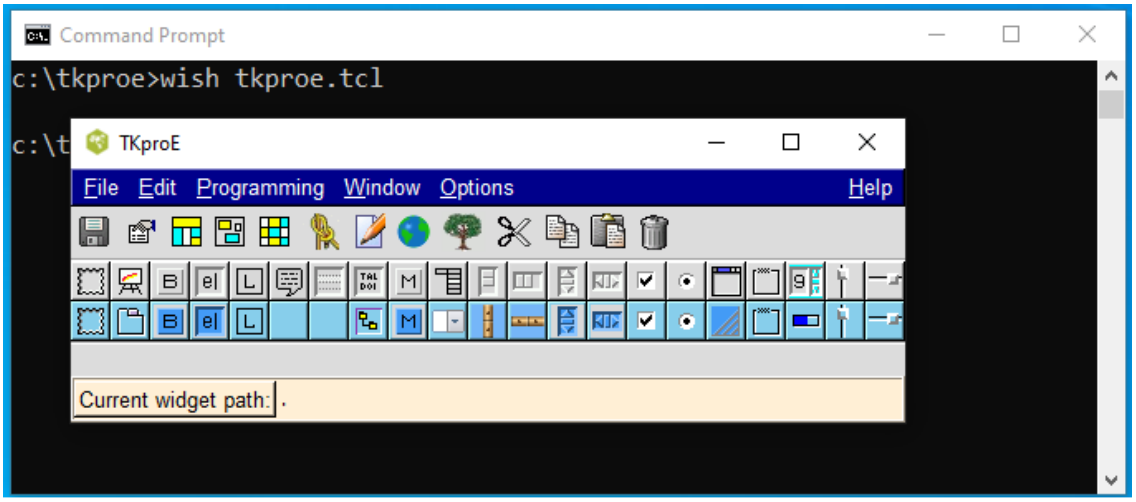


图11. 运行 TKproE IDE

D.1 TKproE 程序下载: (tkproe2_20.zip)

https://sourceforge.net/projects/tkproe/files/tkproe/TKproE%202.30/

D.2 TKproE 安装程序 on Windows 10

A) Unzip tkproe2_20.zip

B) Copy Unzip tkproe2_20 目录下面所有的文件 -> C:/tkproe

C) Run TKproE IDE: Type wish c:/tkproe/tkproe.tcl <CR>

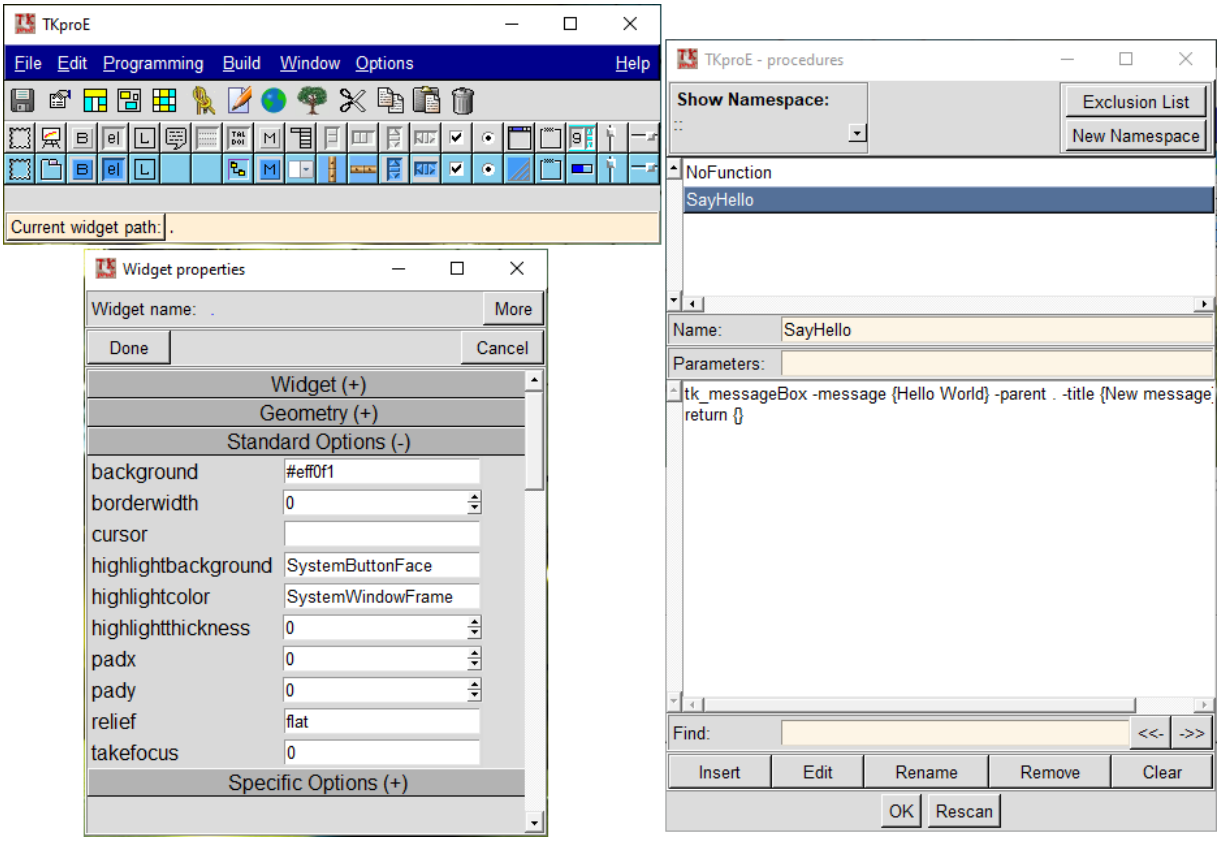
D.3 TKproE v3.2 Home Page

https://tkproe.dengensys.com/

D.4 Description

TCL/TK Programming Environment

TKproE is an integrated program development environment for the TCL/TK scripting language. TKproE, itself, is completely written in the TCL/TK language.



TKproE supports the rapid development of sophisticated graphical user interfaces. TKproE takes advantage of TK, a widget set that is accessible through TCL, a very efficient interpreted programming language. With TKproE the user can build or modify application programs while they are running. This makes it possible to test program changes immediately without the cost of compiling the application.

TKproE is an Open Source product.
TKproE 2.30 requires TCL/TK 8.6 or later.
There are no license fees associated with TKproE. See the TKproE license.

D.5 TKproE v 1.2 Home Page

https://web.archive.org/web/20070903215250/http://tkproe.sourceforge.net/

TKproE

TCL/TK Programming Environment

hosted by: SOURCEFORGE.net

Description

Used for TCL/TK program development

TKproE is an integrated program development environment for the TCL/TK scripting language. TKproE, itself, is completely written in the TCL/TK language.

TKproE supports the rapid development of sophisticated graphical user interfaces. TKproE takes advantage of TK, a widget set that is accessible through TCL, a very efficient interpreted programming language. With TKproE the user can build or modify application programs while they are running. This makes it possible to test program changes immediately without the cost of compiling the application.

TKproE is an Open Source product.

TKproE 1.2 requires TCL/TK 8.4 or later.

TKproE consists of a single TCL script file that can run on Linux, UNIX or

Windows operating systems.

TKproE was designed as a replacement for the XF application. This latter application was originally developed by Sven Delmas over a decade ago. Since then, I adopted XF and have tried to keep it usable with new revisions of the TCL/TK language. However, much has happened to the TCL/TK language since the early 1990s. TCL/TK has a large number of new features that didn't exist at the time XF was first created. Eventually, I felt that a brand new design was better than trying to modify the existing XF code.

No license fees

There are no license fees associated with TKproE See the TKproE license .

Availability

The same TKproe installation package can be used on Linux, UNIX and Windows95/98/NT/2000/XP. Documentation is included in the distribution.

History - current revision = 1.2

Changes for version 1.2

1. The name of the current file being edited is now displayed in the title bar of the TKproE main window.
2. Fixed the code to prevent the addition of extra blank lines at the end of certain saved procedures.
3. Added some support for saving pure TCL megawidgets.
4. The geometry management of paned window children is now saved.
5. Fixed a few errors related to saving windows and tags for text widgets.
6. The global variable window now sorts variable regardless of case.
7. Increased the number of toplevel window attributes that are now saved. (almost all of them)
8. Namespaces that existed at the start of TKproE are now preserved when loading a new project.
9. The program now saves pack, place and grid propagate information.
10. Added support for configuring BLT graph widgets. (Use the BLTgraph.tcl template to add the first graph to a project.)
11. Added project type as a general option. TKproE will not attempt to save TK components for projects identified as "TCL only".

Review detailed history

How to use TKproE

Read the TKproE documentation

Downloads

The latest downloads of TKproE are available from http://sourceforge.net/projects/tkproe/

Appendix E Install Tcl/Tk for Windows

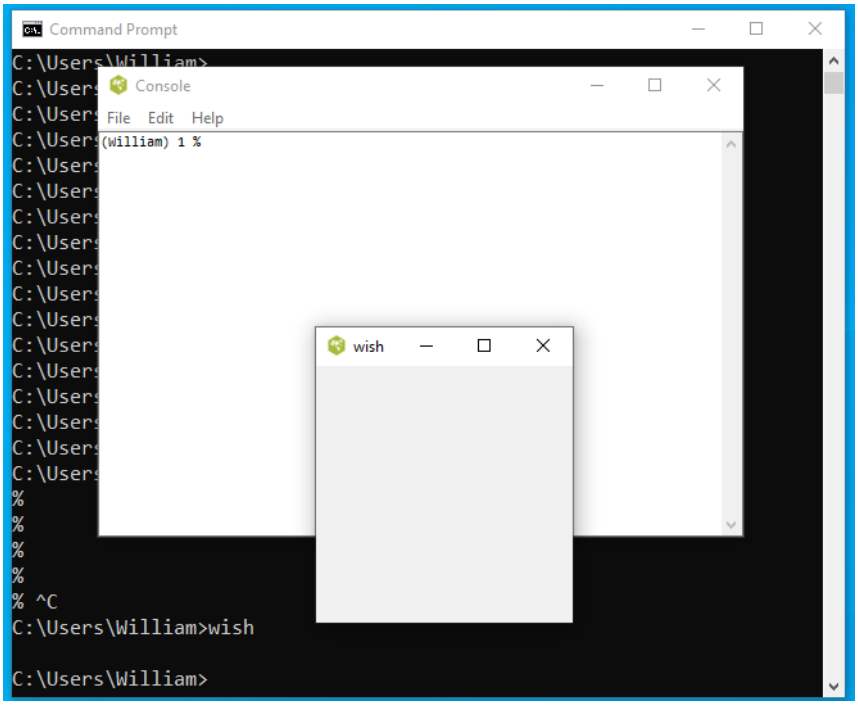
E.1 如何在Windows 10 上装 Tcl/Tk

A) Download ActiveTcl-8.6.

https://www.activestate.com/products/tcl/

B) Install ActiveTcl-8.6 on Windows 10

https://www.tutorialspoint.com/tcl-tk/tcl_environment.htm#:~:text=Installation%20on%20Windows,following%20the%20on%20screen%20instructions.



Appendix F　其他有趣 Tcl/Tk 程序

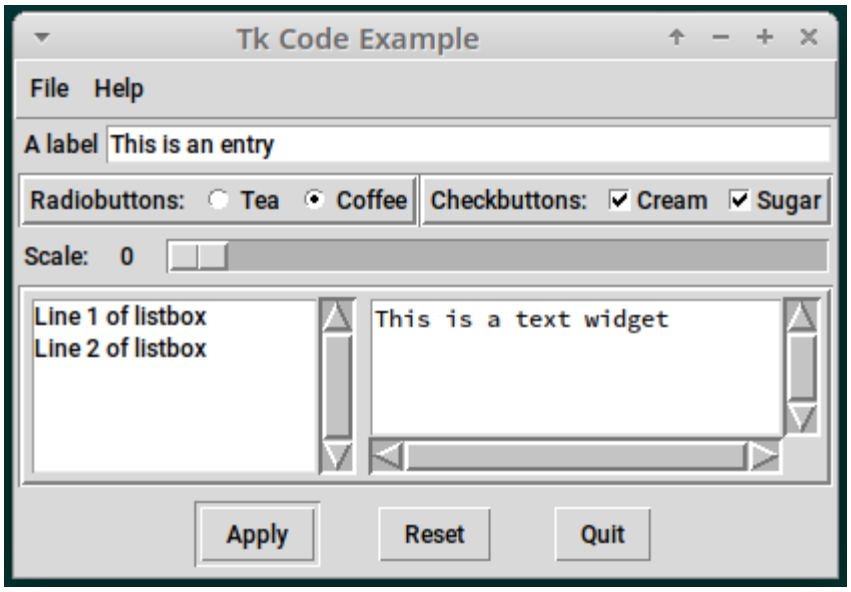


图10. TkFile.tcl

F.1 TkFile.tcl

```
#!/opt/tcl/bin/sytcl
#!/opt/tcl/bin/tcl
package require Tk

wm withdraw .
set w [toplevel .t]
wm title .t {Tk Code Example}
set m [menu $w.menubar -tearoff 0]
$m add cascade -label File -menu [menu $m.file]
$m.file add command -label Quit -command exit
$m add cascade -label Help -menu [menu $m.help]
$m.help add command -label Index -command {puts Sorry}
$w configure -menu $m
set f [frame $w.f1]
pack [label $f.label -text {A label}] -side left
pack [entry $f.entry] -side left -fill x -expand true
$f.entry insert 0 {This is an entry}
```

```
pack $f -fill x -padx 2 -pady 2
set f [frame $w.f2]
pack [frame $f.rg -relief groove -bd 3] -side left -fill x -expand true
pack [label $f.rg.lbl -text Radiobuttons:] -side left
pack [radiobutton $f.rg.b1 -text Tea -variable choice -value 1] -side left
pack [radiobutton $f.rg.b2 -text Coffee -variable choice -value 0] -side left
pack [frame $f.cg -relief groove -bd 3] -side left -fill x -expand true
pack [label $f.cg.lbl -text Checkbuttons:] -side left
pack [checkbutton $f.cg.b1 -text Cream] -side left
pack [checkbutton $f.cg.b2 -text Sugar] -side left
pack $f -fill x -padx 2 -pady 2
set f [frame $w.f3]
pack [label $f.lbl -text Scale:] -side left
pack [label $f.val -textvariable scaleval -width 4] -side left
pack [scale $f.scl -variable scaleval -orient horizontal -from 0 \
-to 10 -showvalue false] -side left -fill x -expand true
pack $f -fill x -padx 2 -pady 2
set f [frame $w.f4 -relief groove -bd 3]
pack [frame $f.lf] -side left -fill both -padx 3 -pady 3
pack [listbox $f.lf.lb -yscrollcommand "$f.lf.sb set" -height 4] \
-side left -fill both -expand true
pack [scrollbar $f.lf.sb -command "$f.lf.lb yview"] \
-side left -fill y
$f.lf.lb insert end {Line 1 of listbox} {Line 2 of listbox}
pack [frame $f.tf] -side left -fill both -expand true -padx 3 -pady 3
grid columnconfigure $f.tf 0 -weight 1
grid rowconfigure $f.tf 0 -weight 1
grid [text $f.tf.tx -yscrollcommand "$f.tf.sy set" -height 4 -width 25 \
-xscrollcommand "$f.tf.sx set"] -column 0 -row 0 -sticky nsew
grid [scrollbar $f.tf.sy -command "$f.tf.tx yview"] \
-column 1 -row 0 -sticky ns
grid [scrollbar $f.tf.sx -command "$f.tf.tx xview" -orient horizontal] \
-column 0 -row 1 -sticky ew
$f.tf.tx insert end {This is a text widget}
pack $f -fill both -expand true -padx 2 -pady 2
set f [frame $w.f5]
button $f.b1 -text Apply -default active -command {puts $scaleval}
button $f.b2 -text Reset -default normal -command {set scaleval 0}
button $f.b3 -text Quit -default normal -command exit
```

```
pack $f.b1 $f.b2 $f.b3 -padx 10 -side left
pack $f -pady 2
```

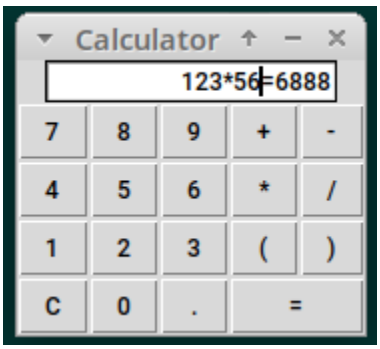


图11. 计算器

F.2 Calculater.tcl

```
package require Tk
wm title . Calculator
grid [entry .e -textvar e -just right] -columnspan 5
bind .e <Return> =
set n 0
foreach row {
  {7 8 9 + -}
  {4 5 6 * /}
  {1 2 3 ( )}
  {C 0 . = }
} {
  foreach key $row {
    switch -- $key {
      =     {set cmd =}
      C     {set cmd {set clear 1; set e ""}}
      default {set cmd "hit $key"}
    }
    lappend keys [button .[incr n] -text $key -command $cmd]
  }
  eval grid $keys -sticky we ;#-padx 1 -pady 1
  set keys [list]
}
grid .$n -columnspan 2 ;# make last key (=) double wide
proc = {} {
  regsub { =.+} $::e "" ::e ;# maybe clear previous result
```

```
  if [catch {set ::res [expr [string map {/ *1.0/} $::e]]}] {
     .e config -fg red
  }
  append ::e = $::res
  .e xview end
  set ::clear 1
}
proc hit {key} {
  if $::clear {
     set ::e ""
     if ! [regexp {[0-9().]} $key] {set ::e $::res}
     .e config -fg black
     .e icursor end
     set ::clear 0
  }
  .e insert end $key
}
set clear 0
focus .e         ;# allow keyboard input
wm resizable . 0 0
```

F.3 Tcl/Tk Fonts

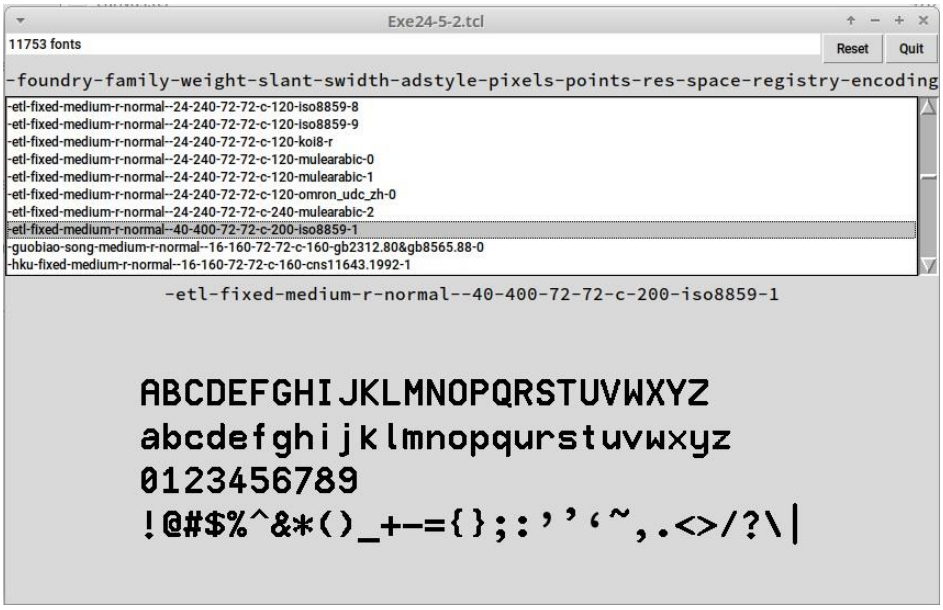


图12. Tcl/Tk Fonts

该程序非常有意思，利用 X-Window 命令‘xlsfonts’显示所有的 fonts，供用户选择，然后显示这个 font 的所有字母. 要注意的是必须保证在你现有的机器上‘xlsfonts’这条命令能够执行.

在xterm上 Type: xlsfonts <CR>，应该能看到下图的结果.

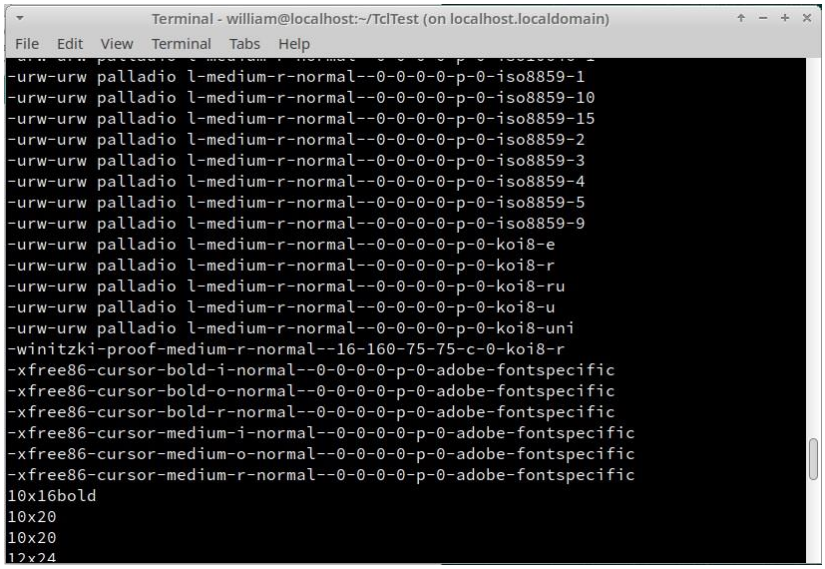


图12. Unix X Window Command ‘xlsfonts’

如果这条命令不能执行那么就可以安装这条命令. 如果是 Open SuSE Linux，可以用如下的命令安装:

sudo zipper install xlsfonts <CR>

TclFonts.tcl

```
#!/import/tcl/bin/wish+
package require Tk

# The menus are big, so position the window
# near the upper-left corner of the display
wm geometry . +30+30

# Create a frame and buttons along the top
frame .buttons
pack .buttons -side top -fill x
button .buttons.quit -text Quit -command exit
button .buttons.reset -text Reset -command Reset
```

```
pack .buttons.quit .buttons.reset -side right

# An entry widget is used for status messages
entry .buttons.e -textvar status -relief flat
pack .buttons.e -side top -fill x
proc Status { string } {
        global status
        set status $string
        update idletasks
}

# So we can see status messages
tkwait visibility .buttons.e

# Set up the menus. There is one for each
# component of a font name, except that the two resolutions
# are combined and the avgWidth is suppressed.
frame .menubar
set font(comps) {foundry family weight slant swidth \
adstyle pixels points res res2 \
space avgWidth registry encoding}
foreach x $font(comps) {
        # font(component) lists all possible component values
        # font(cur,component) keeps the current component values
        set font(cur,$x) *
        set font($x) {}
        # Trim out the second resolution and the average width
        if {$x == "res2" || $x == "avgWidth"} {
                continue
        }

        # The border and highlight thickness are set to 0 so the
        # button texts run together into one long string.
        menubutton .menubar.$x -menu .menubar.$x.m -text -$x \
        -padx 0 -bd 0 -font fixed \
        -highlightthickness 0
        menu .menubar.$x.m
        pack .menubar.$x -side left
```

```
        # Create the initial wild card entry for the component
        .menubar.$x.m add radio -label * \
        -variable font(cur,$x) \
        -value * \
        -command [list DoFont]
}


# Use traces to patch up the suppressed font(comps)
trace variable font(cur,res2) r TraceRes2
proc TraceRes2 { args } {
        global font
        set font(cur,res2) $font(cur,res)
}
trace variable font(cur,avgWidth) r TraceWidth
proc TraceWidth { args } {
        global font
        set font(cur,avgWidth) *
}
# Mostly, but not always, the points are 10x the pixels
trace variable font(cur,pixels) w TracePixels
proc TracePixels { args } {
        global font
        catch {
         # Might not be a number
         set font(cur,points) [expr 10*$font(cur,pixels)]
        }
}



# Create a listbox to hold all the font names
frame .body
set font(list) [listbox .body.list \
-setgrid true -selectmode browse \
-yscrollcommand {.body.scroll set}]
scrollbar .body.scroll -command {.body.list yview}
pack .body.scroll -side right -fill y
pack .body.list -side left -fill both -expand true
```

```
# Clicking on an item displays the font
bind $font(list) <ButtonRelease-1> [list SelectFont $font(list) %y]

# Use the xlsfonts program to generate a
# list of all fonts known to the server.
Status "Listing fonts..."
if [catch {open "|xlsfonts *"} in] {
        puts stderr "xlsfonts failed $in"
        exit 1
}

set font(num) 0
set numAliases 0
set font(N) 0
while {[gets $in line] >= 0} {
        $font(list) insert end $line
        # fonts(all,$i) is the master list of existing fonts
        # This is used to avoid potentially expensive
        # searches for fonts on the server, and to
        # highlight the matching font in the listbox
        # when a pattern is specified.
        set font(all,$font(N)) $line
        incr font(N)
        set parts [split $line -]
        if {[llength $parts] < 14} {
                # Aliases do not have the full information
                lappend aliases $line
                incr numAliases
        } else {
                incr font(num)
                # Chop up the font name and record the
                # unique font(comps) in the font array.
                # The leading - in font names means that
                # parts has a leading null element and we
                # start at element 1 (not zero).
                set i 1
                foreach x $font(comps) {
                set value [lindex $parts $i]
```

```
                incr i
                if {[lsearch $font($x) $value] < 0} {
                        # Missing this entry, so add it
                        lappend font($x) $value
                        }
                }
        }
}


# Fill out the menus
foreach x $font(comps) {if {$x == "res2" || $x == "avgWidth"} {
        continue
}
foreach value [lsort $font($x)] {
        if {[string length $value] == 0} {
        set label (nil)
        } else {
                set label $value
        }
        .menubar.$x.m add radio -label $label \
        -variable font(cur,$x) \
        -value $value \
        -command DoFont
        }
}
Status "Found $font(num) fonts and $numAliases aliases"


# This label displays the current font
label .font -textvar font(current) -bd 5 -font fixed

# A message displays a string in the font.
set font(msg) [message .font(msg) -aspect 1000 -borderwidth 10]
set font(sampler) "
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqurstuvwxyz
0123456789
!@#$%^&*()_+-=[]{};:'"`~,.<>/?\\|
```

```
"
set font(errormsg) "
(No matching font)
"
# font Now pack the main display
pack .menubar -side top -fill x
pack .body -side top -fill both -expand true
pack .font $font(msg) -side top


proc DoFont { } {
        global font
        set font(current) {}
        foreach x $font(comps) {
                append font(current) -$font(cur,$x)
                }
        SetFont
}

proc SelectFont { list y } {
        # Extract a font name from the listbox
        global font
        set ix [$font(list) nearest $y]
        set font(current) [$font(list) get $ix]
        set parts [split $font(current) -]
        if {[llength $parts] < 14} {
                foreach x $font(comps) {
                set font(cur,$x) {}
                }
        } else {
                set i 1
                foreach x $font(comps) {
                        set value [lindex $parts $i]
                        incr i
                        set font(cur,$x) $value
                        }
        }
        SetFont
}
```

```
proc SetFont {} {
	global font
	# Generate a regular expression from the font pattern
	regsub -all -- {‹nil›} $font(current) {} font(current)
	regsub -all -- {\*} $font(current) {[^-]*} pattern
	for {set n 0} {$n < $font(N)} {incr n} {
		if [regexp -- $pattern $font(all,$n)] {
			$font(msg) config -font $font(current) \
			-text $font(sampler)
			catch {$font(list) select clear \
			[$font(list) curselection]}
			$font(list) select set $n
			$font(list) see $n
			return
		}
	}
	$font(msg) config -text $font(errormsg)
}

proc Reset {} {
	global font
	foreach x $font(comps) {
	set font(cur,$x) *
	}
	DoFont
	Status "$font(num) fonts"
}

Reset
```