

Presenting eti

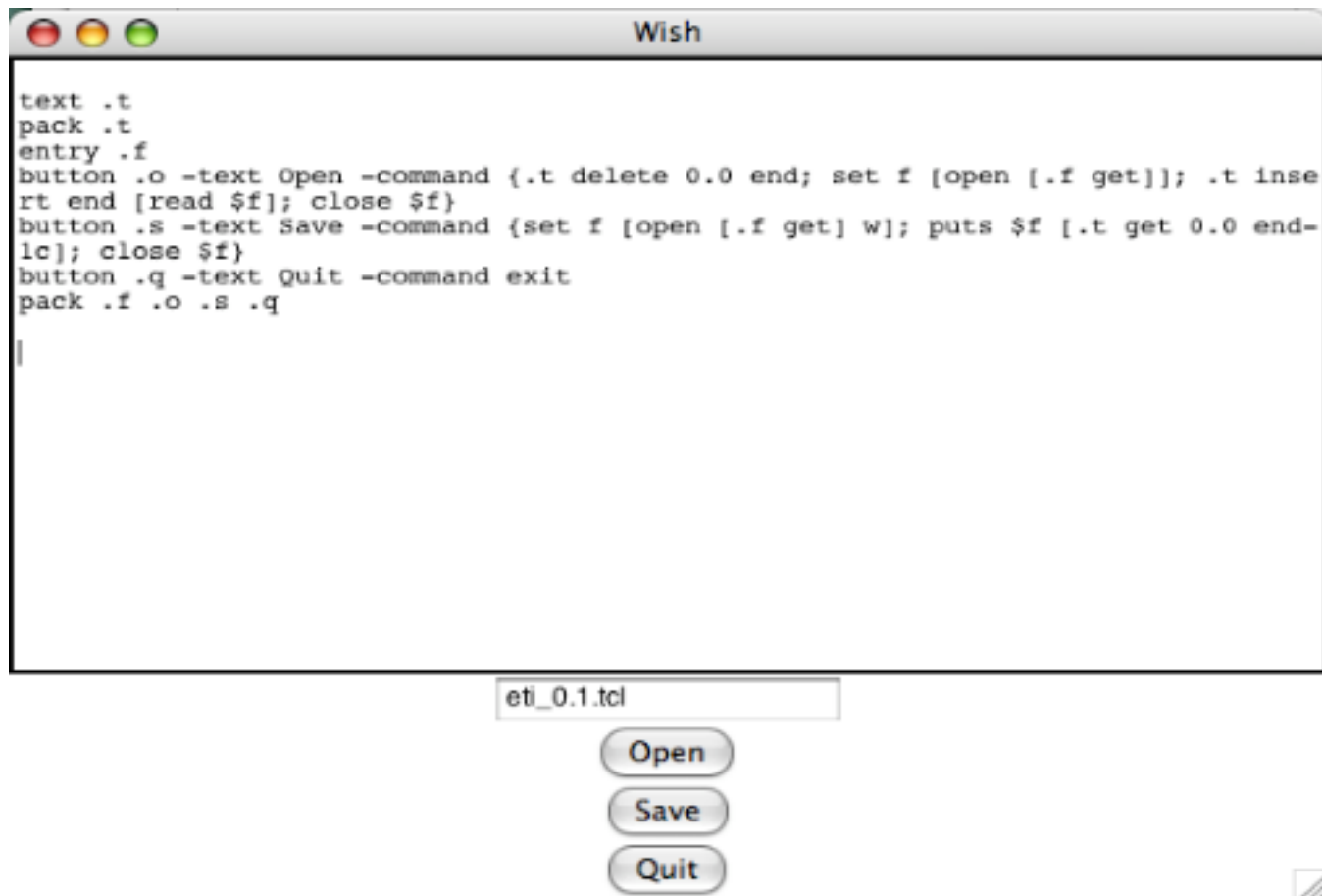
- How i learned to stop worrying and created megawidgets -



Axel Nagelschmidt <http://axn.atspace.org/tcl.html>

Writing a complete editor is possible after one month of learning TCL!

I wrote something like this around the year 1997:



```
text .t
pack .t
entry .f
button .o -text Open -command {.t delete 0.0 end; set f [open [.f get]]; .t insert end [read $f]; close $f}
button .s -text Save -command {set f [open [.f get] w]; puts $f [.t get 0.0 end]; close $f}
button .q -text Quit -command exit
pack .f .o .s .q
```

eti_0.1.tcl

Open

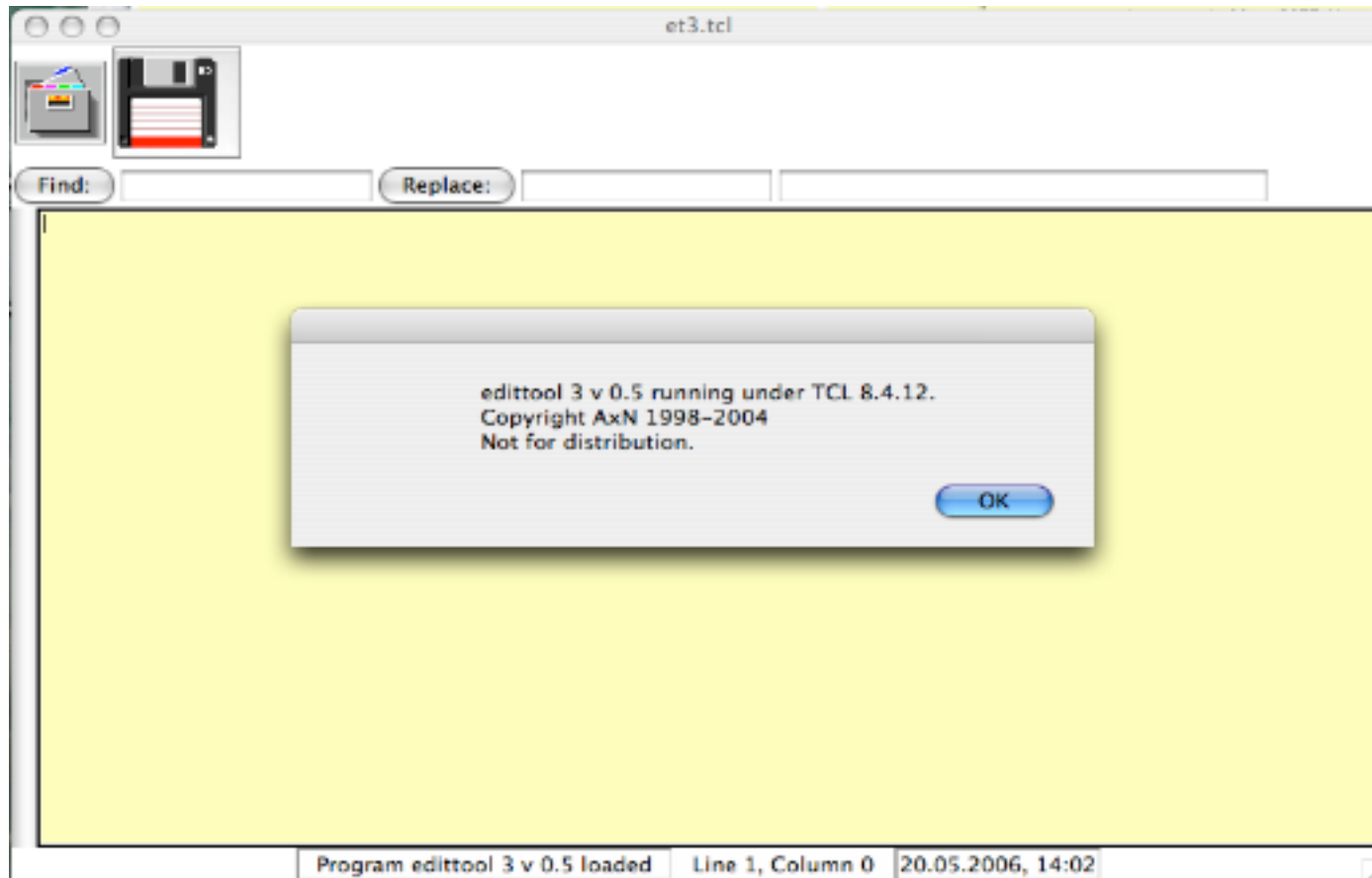
Save

Quit

eti 0.1, nearly feature complete

Even better version 0.3:

- Has menubar (not shown here, screenshot from Mac)
- Options to save / save as / open / close file, toolbar, find and replace
- Options to change font family / size / background colour
- Status bar with status info, cursor position and clock
- Can mail it's content
- Ported to Solaris (because no change necessary)
- No widespread use, probably because never deployed



Usage of object orientation seemed to be a good idea:

- ◇ Have a mini editor for any kind of text. Derive HTML editor, TCL editor etc. from this.
- ◇ Have a set of classes needed for graphic programs, derive different graphical drawings / simulations from this.
- ◇ Specify a set of common used actions to include into the menu or a toolbar
- ◇ Have multiple instances of the same type, so can use a notebook with several tabs
- ◇ incr TCI offering classes and incr TK offering megawidgets like notebook seemed natural

But (1) ...

- ◇ incr TK seemed to be more complicated than necessary
- ◇ incr TK seemed to be slow because of code overhead

Solution 1:

- ◇ create one superclass drawing the toolbar, the notebook and the status bar
- ◇ create one superclass in incr TCL (not TK!) for all instances (called etimodules)
- ◇ have each etimodul draw it's own frame
- ◇ inherit superclass etimodul to take care of common needed methods

But (2) ...

- ◇ creating multiple windows with notebooks needed another superclass
- ◆ inheritance for widgets was not really used
 - ◇ tcledit or diffedit could not really inherit the class miniedit, different count and layout of widgets
- ◇ learned about tile and increasing / ongoing support for Mac OS X

Solution II:

- ◇ recreated some widgets using tile components
- ◇ added menu to change style to any existing tile theme
- ◇ some usage of mkWidgets, because better creation of megawidgets was anticipated

But (3) ...

- ◇ mixing iwidgets with tile widgets was no good idea
- ◇ mkWidgets usable for megawidgets, but not really an OO system
- ◇ mkWidgets not being supported any longer?
- ◇ not much time to try something new

Then ...

- ◇ visited the 5th european TCL User meeting (Thanks to Michael and Holger!)
- ◇ got more involved using tequila, thus making shared apps an easy possibility (Thanks Jean-Claude!)
- ◇ learned about integrating OpenGL to create apps using 3D graphics (Thanks Paul!)
- ◇ learned more about tile (Thanks Rolf!)
- ◇ learned more on creating fullfeatured apps (Thanks Detlef!)
- ◇ learned more on mixing text and graphics to create integrated learning experiences (Thanks Jos!)
- ◇ learned more on tablelist and it's integration into tile (Thanks Csaba!)

Other influences happening in this time:

- ◇ More usage of Java and especially Eclipse at work
- ◇ Our company introducing Lotus Notes, me visiting a developers course
- ◇ Seeing that TCL can control much of windows through twapi
- ◇ .NET and mono appearing in widespread use, new IDEs anjuta and monodevelop
- ◇ Applications becoming more complex, each one developing their (own) makro language
- ◇ Seeing that TCL can integrate calls to graphic libraries like ImageMagic or OpenGL
- ◇ Understanding that integration in big consistent frameworks like Oberon or Squeak makes usage much easier
- ◇ Anticipating to learn that TCL can integrate calls to control OpenOffice.org
- ◇ Learning about snit and it's ongoing support and integration with TCL 8.5
- ◇ Writing more applications as helpers or for database access, needing a usable and robust framework to do so
- ◇ Expecting user interfaces to allow users much more choice and flexibility - "graphical programming"
- ◇ Not wanting to learn a new P*-language every year
- ◇ Seeing that TCL is here to stay, easy to use, easy to teach or use as description language, cross platform

=> Decided to base framework and apps on TCL, integrating good usage examples into an usable IDE

(OK, did so before with Pascal, Oberon and Delphi ...)

Moving from incr TCL to snit

Topic	incr TCL	snit
Header	itcl::class <name>	snit::widget <name>
Inheritance vs. Delegation	inherit <upperclass>	delegate <method> to <otherclass>
Building widgets	frame \$w, delivered as argument	set w \$win, frame already created
Reference to instance	\$this	\$self
Calling own methods	<method>	\$self <method>
declaring method	method <name> <args> <body>	method <name> <args> <body>
Destroying instance	itcl::class delete object \$this	destroy \$self
Overwrite class definition	no, only after deletion of class	yes, like the TCL way
Inherit methods	yes	no, but delegation helps

Rewrote about 5 major classes and the core of the framework in about 3 days.

Need to rewrite toolbar, and some handy megawidgets used from mkWidgets to snit.

First results seem to be very promising.

Skeleton of an etimodule:

```
# _____ note _____  
  
# lib-skeleton here  
  
# _____ pack _____  
  
package require date3  
  
# _____ proc _____  
  
snit::widget skel {  
# delegate unknown method calls to my notebook  
  delegate method * using {redirect %m %s}  
  
  variable w  
  variable title ""  
  variable file ""  
  
  constructor {} {  
    set w $win.t  
    set type skel  
    set file "Untitled [alloc3::uniq_id]"  
    pack [text $w -background #ffffbb] -fill both -expand yes  
    focus $w  
    .....  
  
    $w insert end "This skel was created at [date3::timestamp].\n"  
    $self timer  
  }  
}
```

```
destructor {
    debugsay "destructor skel"
}

method getfile {} {return $file}
method gettitle {} {return $title}

method timer {} {
    incr counter
    $f.t configure -text [format %8d $counter]
    $f.tt configure -text [date3::timestamp]
    after 1000 $self timer
}

method onshow {} {
    $w insert end "This skel was shown at [date3::timestamp].\n"
}

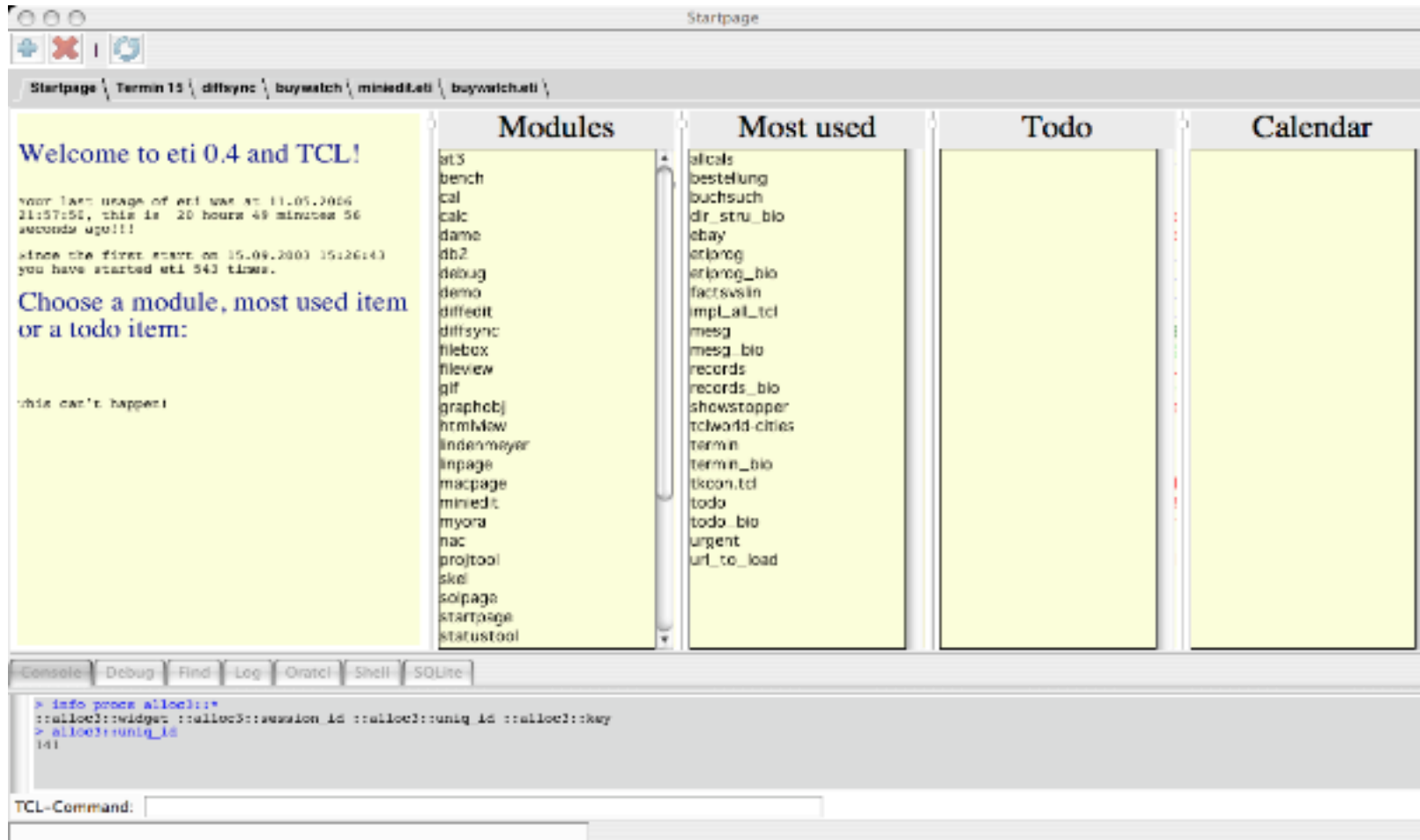
method toolbar {} {
    # use default toolbar or examples below!!!
    $self toolbar_default
    $self toolbar pic $icon3::close {etimodul::call closefile} Tooltiptext
    $self toolbar delim
}

method onhide {} {
    set lasthide [date3::timestamp]
    $w insert end "This skel was hidden at $lasthide.\n"
}

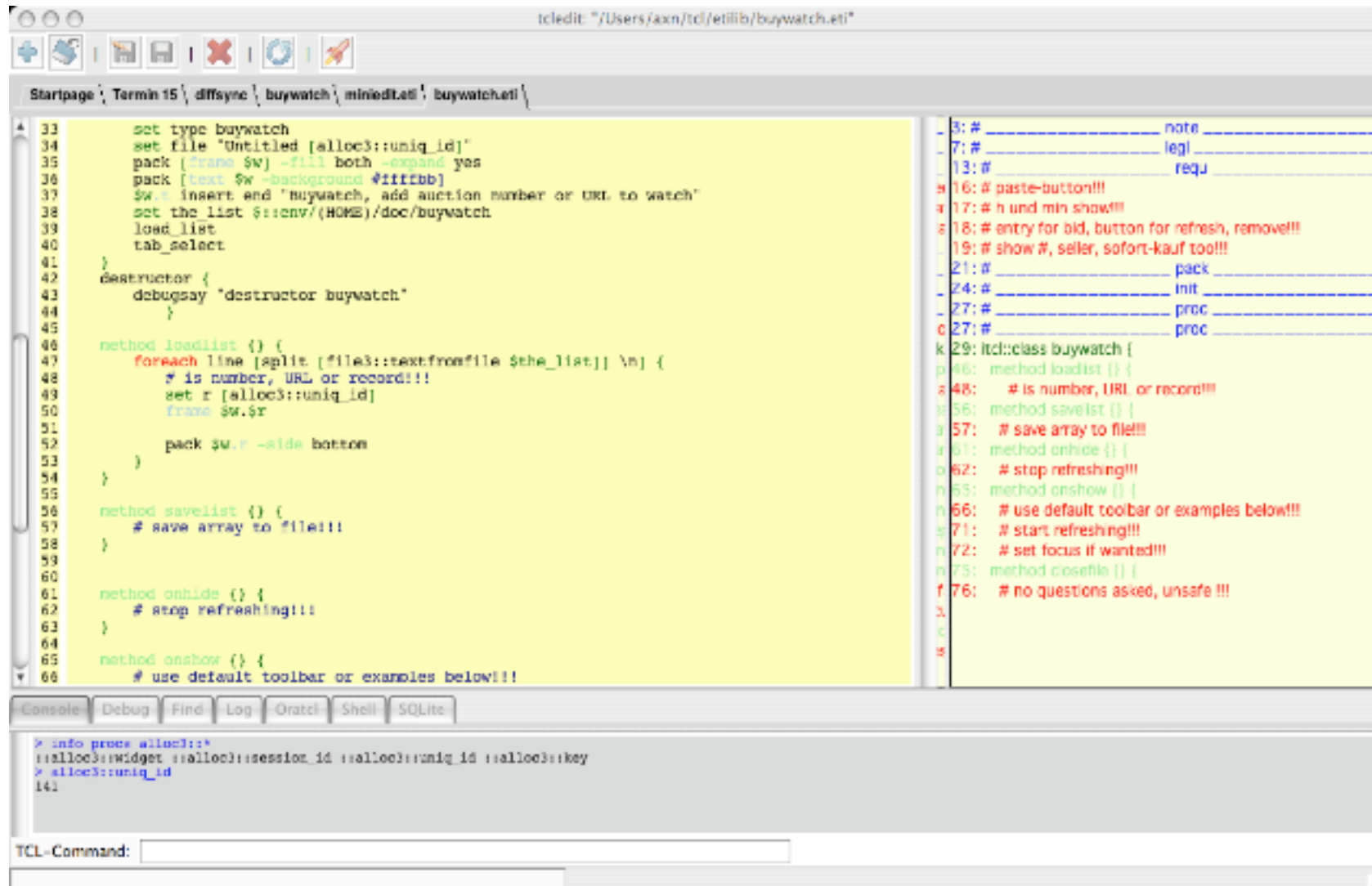
# end of class skel
}
```

Examples of etimodules:

Startpage, allowing to start other modules, showing most used documents, todo and calendar
Older framework for helper applications (terminal, consoles for sqlite and oratcl) shown below



tcledit, editor with syntax colouring based on ctext
Allows working in projects and libraries, highlights requirements and proc / method headers



```
33 set type buywatch
34 set file "Untitled [alloc3::uniq_id]"
35 pack [frame $w] -fill both -expand yes
36 pack [text $w -background #fff0f0]
37 $w insert end "buywatch, add auction number or URL to watch"
38 set the_list $::env/(HOME)/doc/buywatch
39 load_list
40 tab_select
41 }
42 destructor {
43     debugsay "destructor buywatch"
44 }
45
46 method loadlist {} {
47     foreach line [split [file3::textfromfile $the_list] \n] {
48         # is number, URL or record!!!
49         set r [alloc3::uniq_id]
50         frame $w.$r
51
52         pack $w.$r -side bottom
53     }
54 }
55
56 method savelist {} {
57     # save array to file!!!
58 }
59
60
61 method onhide {} {
62     # stop refreshing!!!
63 }
64
65 method onshow {} {
66     # use default toolbar or examples below!!!
67 }
68
```

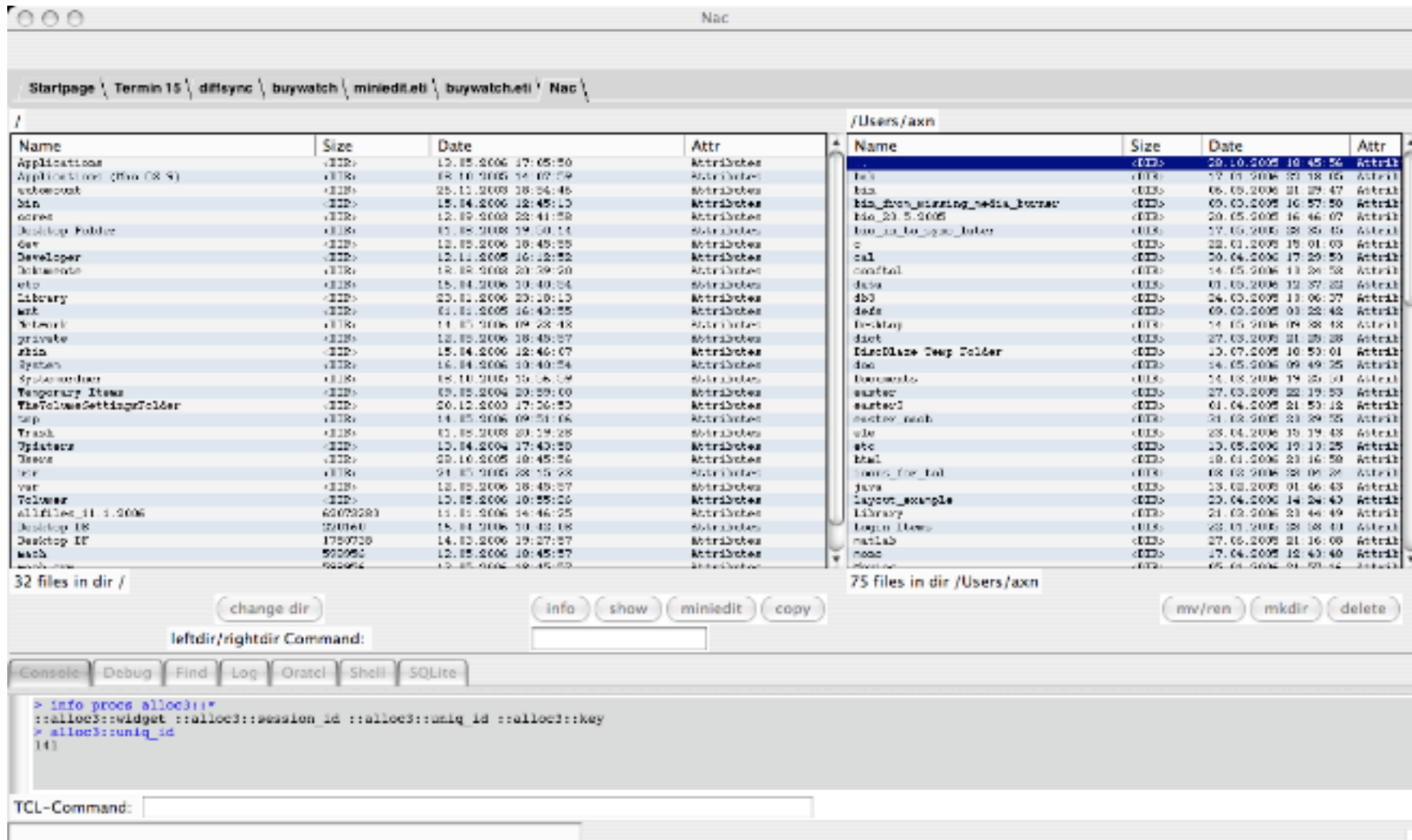
```
3: # ----- note -----
7: # ----- legl -----
13: # ----- requ -----
# 16: # paste-button!!!
# 17: # h und min show!!!
# 18: # entry for bid, button for refresh, remove!!!
# 19: # show #, seller, sofort-kauf too!!!
21: # ----- pack -----
24: # ----- init -----
27: # ----- proc -----
c 27: # ----- proc -----
k 29: itcl::class buywatch {
p 46: method loadlist {} {
# 48: # is number, URL or record!!!
# 56: method savelist {} {
# 57: # save array to file!!!
# 61: method onhide {} {
# 62: # stop refreshing!!!
# 65: method onshow {} {
# 66: # use default toolbar or examples below!!!
# 71: # start refreshing!!!
# 72: # set focus if wanted!!!
# 75: method closefile {} {
f 76: # no questions asked, unsafe !!!
.
c
#
```

Console: Debug Find Log Oratch Shell SQLite

```
> info proc alloc3::*
;alloc3::widget ;alloc3::session_id ;alloc3::uniq_id ;alloc3::key
> alloc3::uniq_id
141
```

TCL-Command:

nac, (not another commander), because this is my main productivity tool
 Using tablelist, but need more intelligent bindings for keyboard operation and cleaning of layout



info, to show information about the application
Thanks again, this is really a tool made possible by the community!

