# Accelerating Tk Development with Wize 3.1

Peter MacDonald

PDQ Interfaces Inc.

September 14, 2010

## 1. Introduction

If Tk is to realize growing adoption, it must offer developers a compelling reason to use it. It may seem ironic, but a key issue limiting Tk's growth is complexity. Simple demos may be straightforward, but scaling up to a full-blown applications is a different matter.

Wize attempts to reduce complexity by offering a development environment that lets programmers do more, with less, by providing:

- Code and data validation.
- Abstract GUI creation (layouts, scrollbars, tooltips, bindings, etc).
- Powerful widgets (TreeView, Tabset and shaped widgets).
- Hierarchical Tcl data (Tree).
- A common base of support components.

Wize is built on Tcl/Tk 8.5.9 and Blt 2.5, both of which have been modified extensively. The Wize binary is complemented with a package of Tcl support code (Mod/Gui) that backfills commonly required functionality. Mod deals with everything from tooltips and image management, to debugging, packaging and deployment.

---

## 2. Validation

Wize validation involves checking Tcl code for syntax, call arguments and types when run with:

```
  wize -Wall script.tcl
```

Wizes validation capabilities are based on **extern** and **type** definitions which provides declarations for all built-ins, eg:

```
  extern incr {varName {amount 1}} {Int var Int} I "Increment the value of a variable"
  extern source {file args} {. {vopts ?-encoding type?} .} I "Evaluate a file or resource as a Tcl script"
```

## 2.1 Checking Performed

Validation performs the following checks:

- All code in proc bodies is compiled, including nested switch/if/while blocks.
- Syntax errors are detected, eg. unbalanced braces and quotes.
- Commands called without a preceding proc definition or extern.
- Parameters to static calls are checked for count (and possibly types).
- Virtually all calls to **builtin commands** are validated
- Detection of missing upvar, variable or global statements.
- Data access to all static elements in _ array are checked for pre-initialization.
- A **Declare** statement to specify other arrays to check.
- Any extern argument of **type** code or 'expr is compiled.

## 2.2 How Tcl is Validated

Validation in Tcl is challenging because the language is highly dynamic. For example, standard Tcl does not normally compile a proc until it is first invoked. Even then, the sub-eval blocks such as while, if and switch are not compiled until they themselves are actually executed.

While this lazy evaluation is a plus in production, it makes detecting problems during development difficult.

Wize overcomes this by providing the option -Wall to forces Tcl code to compile as it is being sourced. Then, in the resulting compilation phase, extensive checks are performed to identify problems.

Errors or warnings from checking are output in a form similar to gcc warning messages.

## 2.3 How Tk is Validated

Validation of Tk code presents a additional challenges because Tk widgets are normally created as **object-commands**. Sub-commands are then accessed via the object/widget-path. Eg:

```
text .t
.t insert end "ABC"
.t delete 1.1
```

Unfortunately, the use of object/path presents the compile phase with no effective way to perform checking. Maintaining such code afterwards is also problematic. Lastly, text-editors can not effectively provide command completion for Tk calls. This last is truly annoying as Tk widgets are responsible for the vast majority of all command options in Tcl.
To address this, Wize refactors the **Tk widgets** to create a Module command-namespace per widget in ::Tk. Eg:

```
namespace eval ::Tk::Text {
    namespace ensemble create; # ...
    extern insert {win pos str args} I
    extern delete {win pos args} I
    # ...
}
# note: above ensembles get imported from ::Tk to ::
Text new .t
Text insert .t end "ABC"
Text delete .t 1.1
```

Code written in this way can be checked be Wize, and it allows editors (like **Ted**) to support argument completion.

## 2.4 Array Validation

Elements in arrays can be validated by using Declare. This will report use of any element not initialized, eg.

```
variable pc
Declare pc Array

array set pc { a 1 b 2}

proc foo {args} {
    variable pc
    set pc(c) 1;  # Warns that "c" is uninitialized.
}
```

## 2.5 Tod Validation

**Tod** is a simple object extension used in Gui. Wize checks array references to $() for elements not initialized in _, eg.

```
namespace eval ::foo {
  variable _
  array set _ { a 1 b 2 }

  proc sub {_ args} {
    upvar $_ {}
    set (c) $(b);  # Warning is issued for var 'c' undefined.
    $_ bar 1;      # Warning issued for proc 'bar' undefined.
    $_ sub
  }

  # ...
}
```

Note that the dispatch call (eg. bar) is also validated.

---

## 3. An Introduction to Gui

**Gui** simplifies the creation of resilient Tk user interfaces using a model similar to that of HTML Markup/CSS/Javascript:

| HTML | GUI |
|---|---|
| *Markup* | *Layout* a nested Tcl list with tags based on Tk class names |
| *CSS* | *Styles* a definition language based on pattern matching rules. |
| *Javascript* | *Tcl* contained in the **script** tag. |

### 3.1 Layout

A GUI layout specifies a hierarchical set of tags containing attributes and content-values. Tags are usually just the Tk class name. After the tag can be a +/-: the + flag is used to indicate a child sub-tree. Lastly are the attributes to modify the layout, such as pack positioning and scroll-bars.

Here is a simple GUI layout:

```
{Toplevel + -title "Simple Editor"} {
    {Text - -pos * -scroll *} {}
    {Frame + -pos _ -subpos l} {
        Button Save
        Button Load
        Button Quit
        {Entry - -id status -pos *l} {}
    }
}
```

### 3.2 Styles

**Styles** are used in a layout to abstract the use of Tk options such as colors, fonts and images. This avoids hard-coding options which is convenient in small applications, but in larger applications tends to lead to excess complexity. Styles also apply options fault tolerantly such that errors become warnings that are seen only at development time (ie. with -Wall).

```
{Toplevel + -title "Simple Editor"} {

    {style} {
        Button  { -bg DarkKhaki }
        .save   { -bg DarkGreen }
```

```
        .txtwin { -bg Khaki }
        Entry.status { -bg LightGray -state disabled }
        Toplevel {
          @defimages { bled greenball }
        }
        .bsave { -image ^bled -compound left }
    }

    {Text - -id txtwin -pos * -scroll *} {}
    {Frame + -pos _ -subpos l} {
        {Button - -id save -id bsave} Save
        Button Load
        Button Quit
        {Entry - -id status -pos *l} {}
    }
}
```

In a style definition, the dot-prefix patterns will match -id attribute names, while title-case patterns will match tags/widget-class names.

Note that we can define images once in the Toplevel using @deficons and then apply them with image lookups using^.

## 3.3 Script

Unless prototyping is the end goal, an application usually requires at least some code. This is added with a script tag. Using an -id attribute will setup variables to dereference widgets from within code

- (w,NAME) - The widget.
- (v,NAME) - The -variable or -textvariable.

eg.

```
{script} {
    array set _ { file "" }

    proc Quit {_} { ::Delete $_ }

    proc Load {_} {
        upvar $_ {}
        set fn [tk_getOpenFile]
        if {$fn == ""} return
        Text delete $(w,txtwin) 1.0 end
        Text insert $(w,txtwin) end [*fread $fn]
        set (file) $fn
        set (v,status) "[mc {Loaded file}]: $(file)"
```

```
    }

    proc Save {_} {
        upvar $_ {}
        if {$(file) == ""} { set (file) [tk_getOpenFile] }
        if {$(file) == ""} return
        *fwrite $(file) [Text get $(w,txtwin) 1.0 end]
        set (v,status) "[mc {Saved file}]: $(file)"
    }
}

{Toplevel + -title "Simple Editor"} {
    {Text - -id txtwin -pos * -scroll *} {}
    {Frame + -pos _ -subpos l} {
        Button Save
        Button Load
        Button Quit
        {Entry - -id status -pos *l} {}
    }
}
```

Note that in the above Tk code is written using the widget class command (.ie Text). This is the mechanisim which allows code to be **validated**. Note also the use of **Tod** $_, in providing simple object-like functionality..

## 3.4 Dialogs and Menus

An application can define dialogs using Toplevel and Menu.

```
{Toplevel + -id tlinput -ns Input} {
    {Entry - -pos _} {}
    Button Ok
}

{Menu + -label Main} {
    {menu + -label File} { x Open x Save }
    {menu + -label Edit} { x Copy x Paste }
}

{Menu + -id mpop -label IO} {
    x Read
    x Write
}
{Toplevel +} {
    style {
        .txt {
            @bind { <Control-g> !tlinput <3> !mpop }
        }
```

```
    }
    {Text - -pos * -id txt } {}
}
```

The following rules apply:

- The first defined Toplevel with no id or the id main will be the main window.
- The first defined Menu with no id or an id of mainmenu will be used as the toplevel menu.

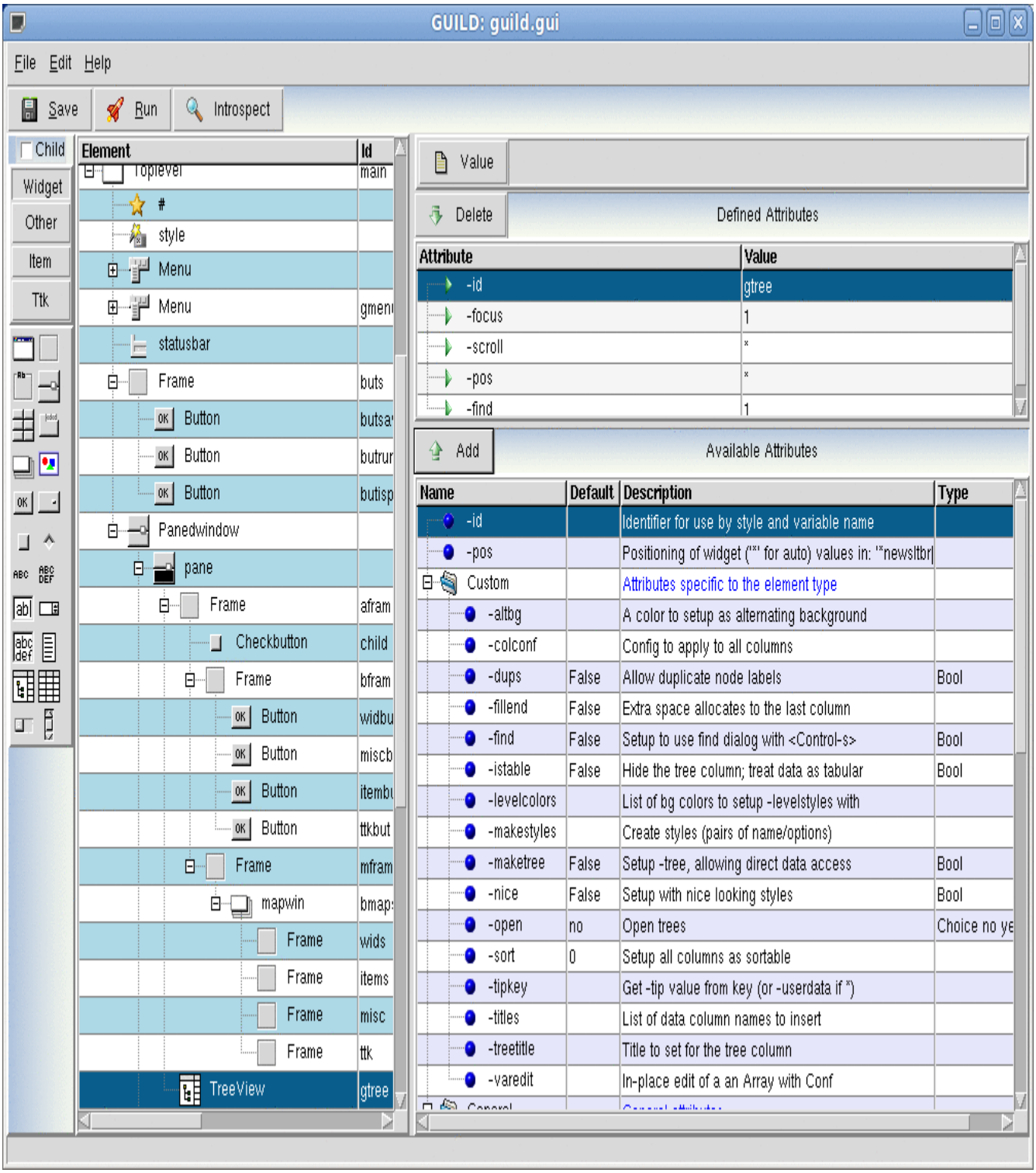
The main Toplevel can use a @bind style to trigger opening Dialogs or Menus. (or use Tk::gui::toplevel from the program).

---

## 4. GUILD - the GUI Layout Designer

Guild is a **GUI** layout editor for .gui files.
While Gui files can easily be hand edited, it's not a convenient way to learn which attributes are available for which tags. Guild custom tree editor uses introspection to display which attributes are available for a tag.

Here is a screenshot of Guild in action:


GUILD: guild.gui
File Edit Help
Save
Run
Introspect
Child
Widget
Other
Item
Ttk
Element
Id
Toplevel
main
#
style
Menu
Menu
gmen
statusbar
Frame
buts
Button
butsa
Button
butrur
Button
butisp
Panedwindow
pane
Frame
afram
Checkbutton
child
Frame
bfram
Button
widbu
Button
miscb
Button
itembu
Button
ttkbut
Frame
mfram
mapwin
bmap
Frame
wids
Frame
items
Frame
misc
Frame
ttk
TreeView
gtree
Value
Delete
Defined Attributes
Attribute
Value
-id
gtree
-focus
1
-scroll
*
-pos
*
-find
1
Add
Available Attributes
Name
Default
Description
Type
-id
Identifier for use by style and variable name
-pos
Positioning of widget ("" for auto) values in: ""newsltbr
Custom
Attributes specific to the element type
-altbg
A color to setup as alternating background
-colconf
Config to apply to all columns
-dups
False
Allow duplicate node labels
Bool
-fillend
False
Extra space allocates to the last column
-find
False
Setup to use find dialog with <Control-s>
Bool
-istable
False
Hide the tree column; treat data as tabular
Bool
-levelcolors
List of bg colors to setup -levelstyles with
-makestyles
Create styles (pairs of name/options)
-maketree
False
Setup -tree, allowing direct data access
Bool
-nice
False
Setup with nice looking styles
Bool
-open
no
Open trees
Choice no ye
-sort
0
Setup all columns as sortable
-tipkey
Get -tip value from key (or -userdata if *)
-titles
List of data column names to insert
-treetitle
Title to set for the tree column
-varedit
In-place edit of a an Array with Conf

## 4.1 Starting Guild

Guild can be started using:

```
wize / Gui/Guild ?file.gui?
```

When started with no file, it prompts for the name of a file. If no file is selected, it asks to insert the application template.

A running application can be modified using <Control-Alt-Shift-2> and selecting Open in Gui Builder from the menu. There you can examine or edit the Gui, save changes, etc.

## 4.2 Using Guild

Elements can then be inserted by clicking the button icons on the left hand side. This will insert a tag element at the current level, or as a child if Child is enabled. Also, some elements have dialogs.

There is a right-click menu for moving tags around, allowing entire tag trees to be Cut and Pasted.

On the right, attributes can be selected and added with 'Add and then edited. Similarly they be selected and removed with Delete.

## 4.3 Styles and Scripts

There is currently no style or script editing dialogs. Instead you just click on Value and an editor pops up.

A better way is to just use your normal editor on the .tcl file and then add to the bottom:

```
Tk::gui::create { include myfile.gui }
```

For styles it is best to execute or run the program and use <Control-Alt-Shift-2> to test out configuration options before adding to rules.

## 4.4 No Style

Applications can be run with style disabled via the Guild menu File/Run-NoStyle.

## 4.5 XML

Applications can be saved as XML via the Guild menu File/Save-As-XML.

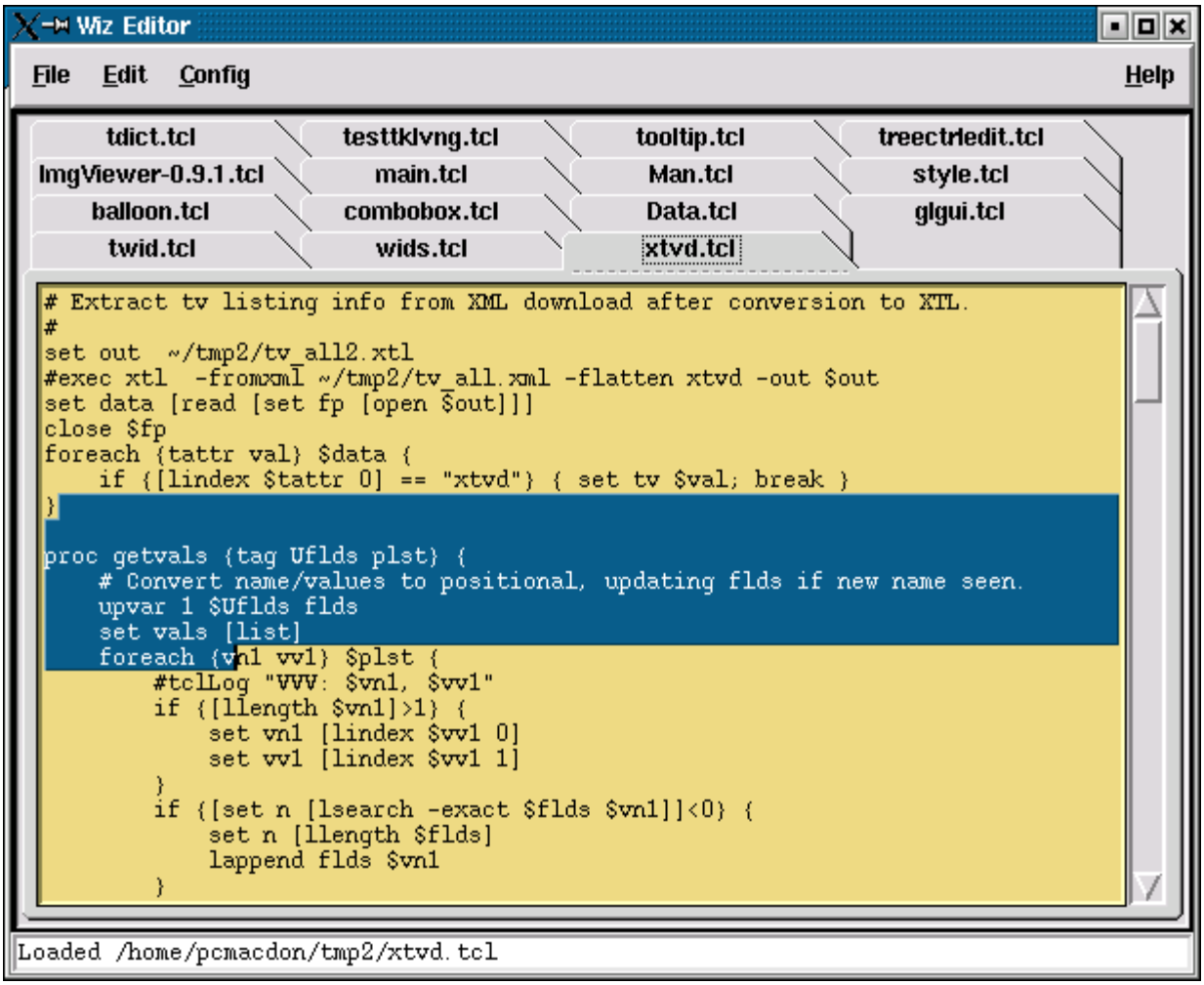
---

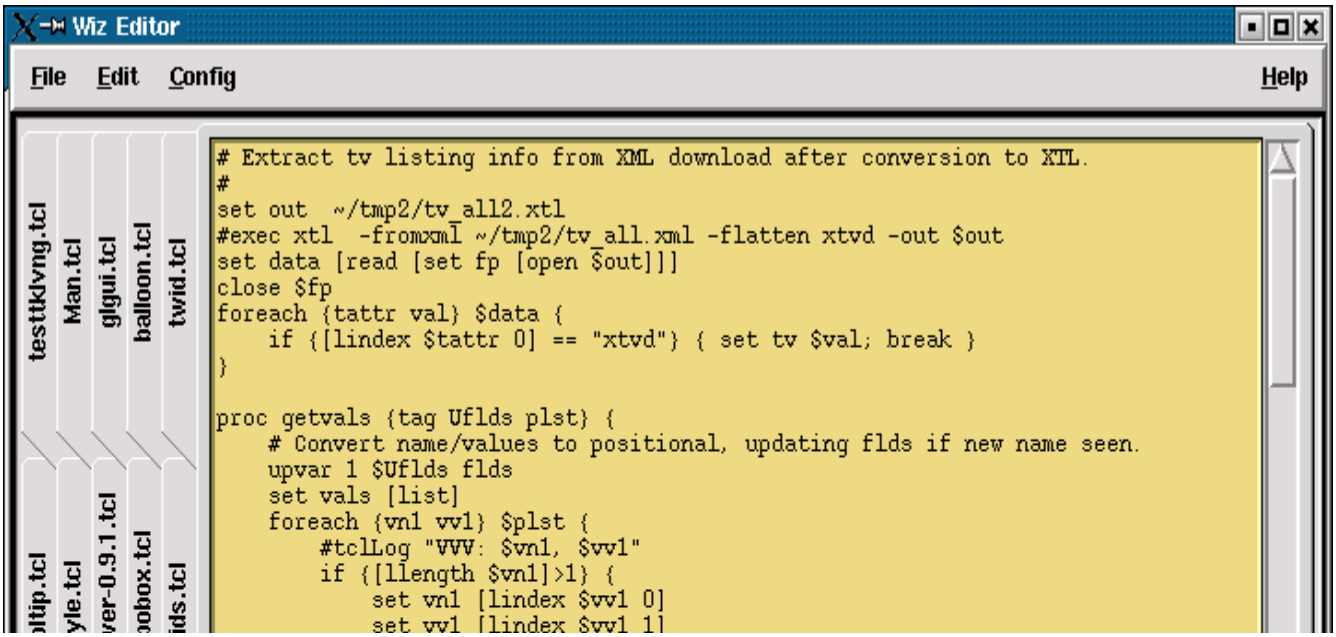
# 5. Tabset

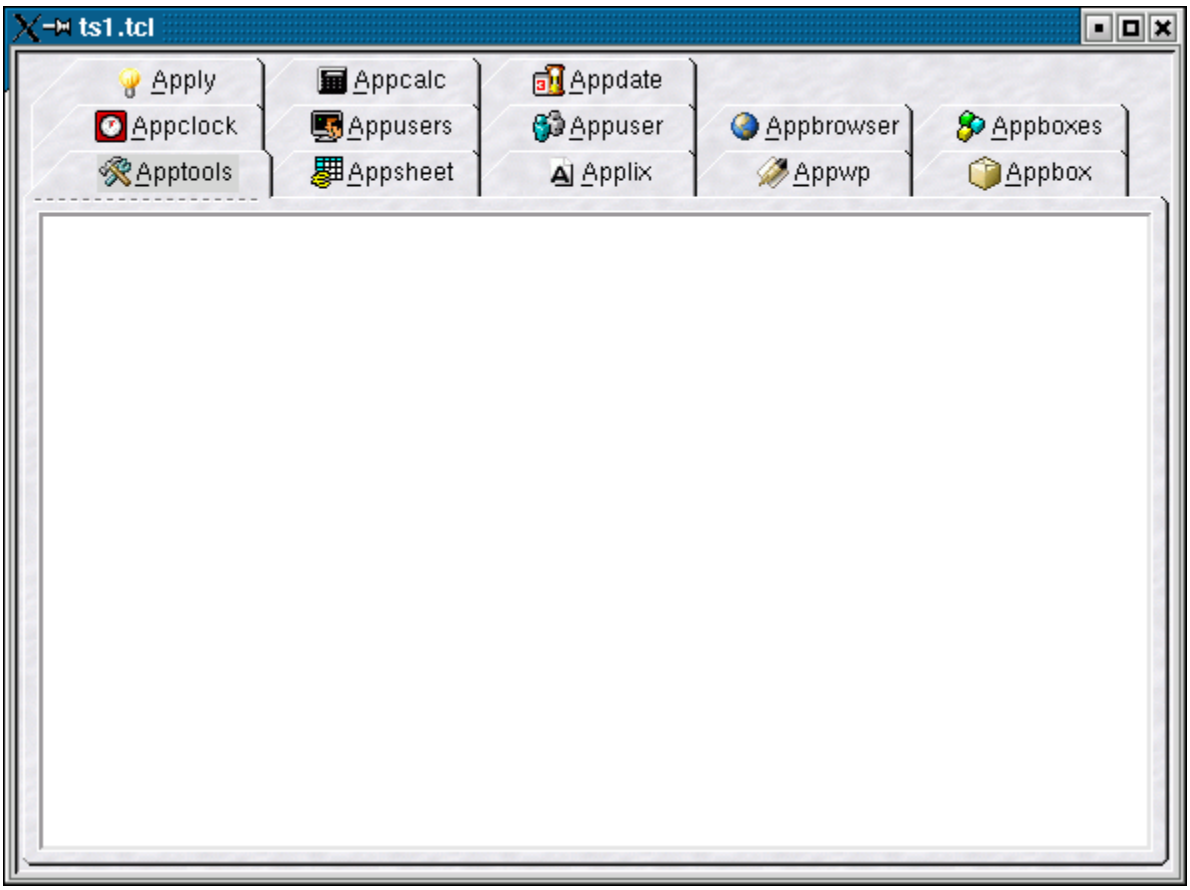
**Tabset** is a notetab widget that includes the following features:

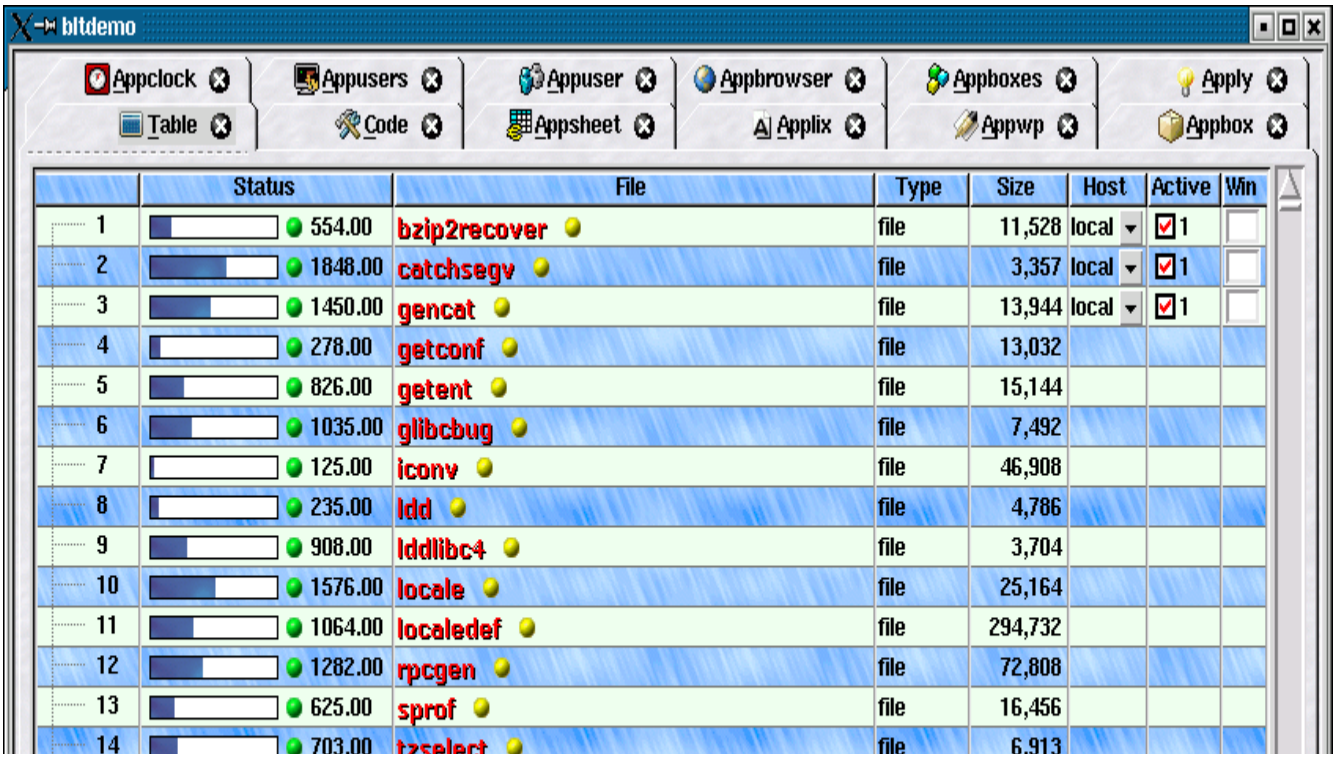
- Tear-off any number of tab-panes.
- Tab slant: left, right, both, or none.
- Tab side: top, bottom, left, or right.
- Rotate tab text labels.
- Drop shadow text support.
- Background image tiling.
- Secondary (right-side) tab image eg. a close button.
- Widget side-images for both left and right sides.
- Tabs use symbolic names to simplify programming.

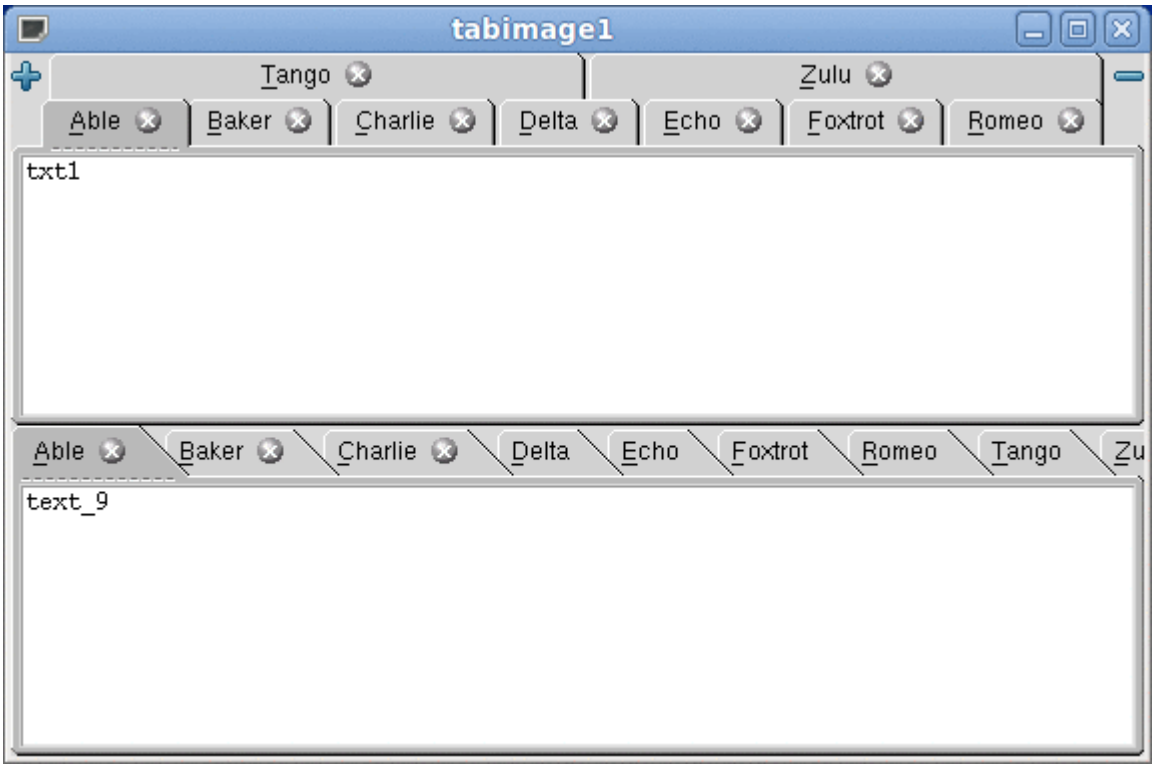
Here are some screenshots:


Wiz Editor
File
Edit
Config
Help
tdict.tcl
testtklvng.tcl
tooltip.tcl
treectrledit.tcl
ImgViewer-0.9.1.tcl
main.tcl
Man.tcl
style.tcl
balloon.tcl
combobox.tcl
Data.tcl
glgui.tcl
twid.tcl
wids.tcl
xtvd.tcl
# Extract tv listing info from XML download after conversion to XTL.
#
set out ~/tmp2/tv_all2.xtl
#exec xtl -fromxml ~/tmp2/tv_all.xml -flatten xtvd -out $out
set data [read [set fp [open $out]]]
close $fp
foreach {tattr val} $data {
if {[lindex $tattr 0] == "xtvd"} { set tv $val; break }
}
proc getvals {tag Uflds plst} {
# Convert name/values to positional, updating flds if new name seen.
upvar 1 $Uflds flds
set vals [list]
foreach {vn1 vv1} $plst {
#tclLog "VVV: $vn1, $vv1"
if {[llength $vn1]>1} {
set vn1 [lindex $vv1 0]
set vv1 [lindex $vv1 1]
}
if {[set n [lsearch -exact $flds $vn1]]<0} {
set n [llength $flds]
lappend flds $vn1
}
Loaded /home/pcmacdon/tmp2/xtvd.tcl


Wiz Editor
File
Edit
Config
Help
testtklvng.tcl
Man.tcl
glgui.tcl
balloon.tcl
twid.tcl
# Extract tv listing info from XML download after conversion to XTL.
#
set out ~/tmp2/tv_all2.xtl
#exec xtl -fromxml ~/tmp2/tv_all.xml -flatten xtvd -out $out
set data [read [set fp [open $out]]]
close $fp
foreach {tattr val} $data {
if {[lindex $tattr 0] == "xtvd"} { set tv $val; break }
}
proc getvals {tag Uflds plst} {
# Convert name/values to positional, updating flds if new name seen.
upvar 1 $Uflds flds
set vals [list]
foreach {vn1 vv1} $plst {
#tclLog "VVV: $vn1, $vv1"
if {[llength $vn1]>1} {
set vn1 [lindex $vv1 0]


ts1.tcl
Apply
Appcalc
Appdate
Appclock
Appusers
Appuser
Appbrowser
Appboxes
Apptools
Appsheet
Applix
Appwp
Appbox


bltdemo
Appclock
Appusers
Appuser
Appbrowser
Appboxes
Apply
Table
Code
Appsheet
Applix
Appwp
Appbox
Status
File
Type
Size
Host
Active
Win
1 554.00 bzip2recover file 11,528 local 1
2 1848.00 catchsegv file 3,357 local 1
3 1450.00 gencat file 13,944 local 1
4 278.00 getconf file 13,032
5 826.00 getent file 15,144
6 1035.00 glibcbug file 7,492
7 125.00 iconv file 46,908
8 235.00 ldd file 4,786
9 908.00 lddlibc4 file 3,704
10 1576.00 locale file 25,164
11 1064.00 localedef file 294,732
12 1282.00 rpcgen file 72,808
13 625.00 sprof file 16,456
14 703.00 tzselect file 6,913




---

## 6. TreeView

**TreeView** is a full featured hierarchical table/**tree** widget that can handle 10s of thousands of rows.

Here is an example that displays a list of files:

```
pack [treeview .t]
foreach i [glob *] {
    .t insert end [list $i]
}
```

Note we use [list] because by default TreeView insert treats a key as a list.

Here is a TreeView screen shot:


## 6.1 Features

Here is a list of TreeView features:

- Auto-sizing column widths and row heights.
- Hide/move columns or nodes.
- Sortable by columns or sub-trees.
- External data storage (in a **blt::tree**).
- Multiple TreeViews can share all a tree.
- Easy to use dynamic loading (for sub-trees).
- Support for multiple style types, including:
  - textbox: text cell with optional images.
  - checkbox: a boolean value.
  - combobox: a multi-choice value.
  - barbox: numeric value with progress bar.
  - windowbox: arbitrary embedded windows.
- Styles can be applied to cols, rows and/or cells
- The -altstyle option for alternating rows (bgcolor, etc).
- The -levelstyles option for per-level styles.
- Background image-tile: widget, columns, and cell-styles.
- Drop shadow text.
- Powerful builtin cell editing.

- Dual mode display: flat and tree.

## 6.2 Data Addressing

TreeView provides methods for updating data elements. It also supports accessing dict sub-elements using arraynotation:

```
set t .t
pack [treeview $t] -fill both -expand y
$t column insert end X Y
$t insert end A -data {X 1 Y 2}
$t insert end B -data {X 3 Y "a 1 b 2"}

$t entry incr 0->A X [$t entry get 0->B X]
$t entry set 0->A Y 3
$t entry incr 0->B Y(a) 9
```

## 6.3 Data Trees

TreeView data is stored externally within a tree. This also supports creating a data tree command which is attached to TreeView, eg.

```
*tree new t = {
      = Age Salary
      Managers {
            = Age Salary Title
            Tina 29 10000 President
            Tom  28 8000 VP
      }
      Staff {
            # Inherit the titles of parent ie. "Age Salary".
            Mary 10 6000
            Sam  10 6000
      }
}
pack [treeview .t -tree $t -width 600 -height 600] -fill both -expand y
eval .t col insert end [lsort [$t keys nonroot]]
.t open all
puts [$t incr 0->Managers->Tina Age]
```

See **Tree** for more details.

## 6.4 Changing the Key Delimiter

TreeViews insert expects a list key unless overriden with an explicit delimiter character. For example, the following displays files in a tree down to 2 directory levels:

```
pack [treeview .t -autocreate 1 -separator /] -fill both -expand y
```

```
foreach i [glob  */*] {
     .t insert end $i
}
.t open [.t find -name CVS -istree]; # Open all CVS dirs.
```

## 6.5 Demand Loading

Data can be demand loaded into a treeview tree as it becomes visible or scrolls into view, eg.

```
pack [treeview .t] -fill both -expand y
set t [tree create]
foreach i {A B C} {
   .t col insert end $i -fillcmd [list FillMe $t $i]
}

proc FillMe {t col id} {
   return $col$id
}

$t populate 10000
.t conf -tree $t
```

One use for this is to load the rowids for an **sqlite** database table, and then loading data rows on demand.

## 6.6 Automatics Styles

TreeView makes it easy to apply a style to given depth levels automatically. For example, the following applies lev1 to all toplevel nodes, and lev2 to all nodes of depth 2.

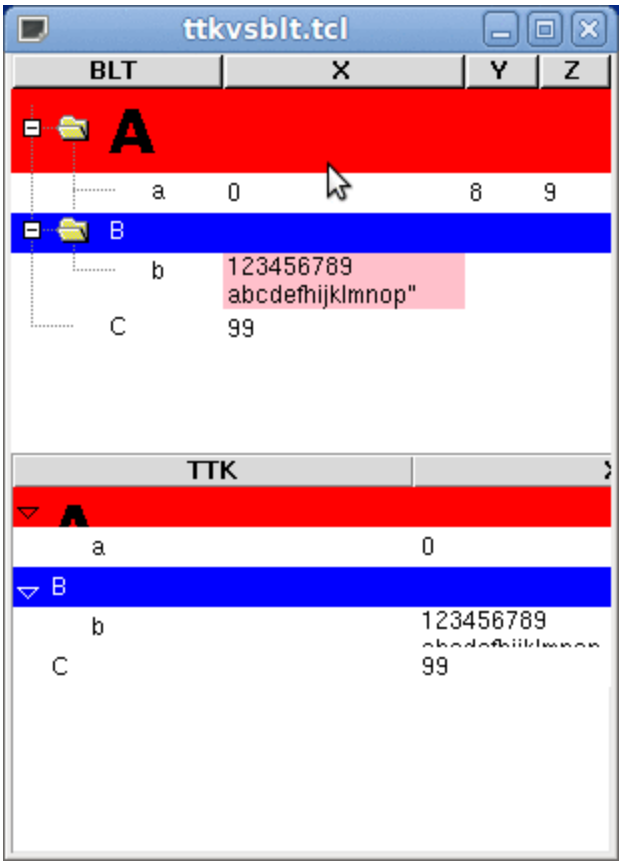
```
   .t style create textbox lev1 -bg LightBlue
   .t style create textbox lev2 -bg LightGreen
   .t conf -levelstyles {lev1 lev2}
```

Alternating row colors is another common effect used in tables. However, for trees the style has to be reapplied everytime a subtree of odd length is opened or closed. The following code snipet shows how TreeView does automatically in TreeView:

```
   .t style create textbox alt -bg LightBlue
   .t conf -altstyle alt
```

## 6.7 TreeView vs TTK

For basic usage, Blt TreeView provides a programming model that is similar to Ttk Treeview. However, TreeView provides more comprehensive autosizing support.




Here is the code:

```
proc TtkTree {t} {
   pack [Treeview new $t] -fill both -expand y
   Treeview conf $t -columns "X Y Z"
   foreach i {X Y Z} { Treeview heading $t $i -text $i }
   Treeview insert $t {} end -id A -text A -tags A -open 1
   Treeview insert $t A end -id a -text a -tags Aa -values {0 8 9}
   Treeview insert $t {} end -id B -text B -tags B -open 1
   Treeview insert $t B end -id b -text b -values [list "123456789\nabcdefhijklmnop"]
   Treeview insert $t {} end -id C -text C -values 99
   Treeview tag conf $t A -font "Verdana -30 bold" -background red
   Treeview tag conf $t B -background Blue -foreground White
   Treeview heading $t #0 -text TTK
}

proc BltTree {t} {
   pack [TreeView new $t] -fill both -expand y
   foreach i {X Y Z} { TreeView column insert $t end $i }
   TreeView insert $t end A -isopen 1 -font "Verdana -30 bold" -bg red
```

```
    TreeView insert $t end "A a" -data "X 0 Y 8 Z 9"
    TreeView insert $t end B -style B -isopen 1
    TreeView insert $t end "B b" -styles "X xb" -data [list X
"123456789\nabcdefhijklmnop"]
    TreeView insert $t end C -data {X 99}
    TreeView style conf $t B -bg Blue -fg white
    TreeView style conf $t xb -bg pink
    TreeView column conf $t #0 -title BLT
 }

 wm geom . 300x400
 eval BltTree .s
 eval TtkTree .t
```

---

## 7. Shaped Buttons

blt::tile:: includes a collection of widgets (button checkbutton radiobutton label) that extend Tk to add shape support. The main use for this is shaped buttons, however, any widget can have a shaped frame by packing it into a label.

blt::tile widgets support the following options (in addition to the standard Tk ones):

| Option | Description |
|---|---|
| -innerbg | The background color inside of the shape. |
| -innertile | The tile image for inside of the shape. |
| -activetile | The tile image when state is active. |
| -disabledtile | The tile image when state is disabled. |
| **-shape** | The button shape, one of: rounded, tube or oval. |
| -radius | For rounded buttons, the radius of the corner curves. |
| -splinesteps | Steps to use in smoothing (same as the canvas polygon). |
| -outline | Color of shape outline (same as the canvas polygon). |
| -linewidth | Width of the outline (same as canvas polygon *-width*). |
| -shadow | Drop shadow support for text |
| **-winshadow** | Drop shadow support for shape |
| -rotate | Support for rotating text in degrees, eg. 90, 180. |
| -checksize | Specify the size of check/radio button indicator. |
| **-icons** | Give a list of 0, 2 or 3 images to use for the indicators. |
| **-bdimage** | A border image that resizes to fit the widget |
| -bdhalo | The number of pixels to preserve in -bdimage |

Here is the shapedbutton.tcl example that defines a large number of shaped buttons, all packed in a single toplevel shaped label:

The above can be run using: wize / Gui/Shapedbutton.

## 7.1 Shape selection: -shape

The -shape option supports shaped buttons/labels. Three shapes are available: rounded, tube, and oval. In addition, you can:

- set -splinesteps to 1 for geometric shapes
- set -splinesteps to 0 for a square.
- set the button outline color with -outline
- use -radius with rounded to sharpen corners.
- Use -winshadow to give shapes 3D relief.

## 7.2 Indicator Images: -icons

The -icons option lets you use a single statement to override the default indicators used for check and radio buttons. It takes 3 image values: normal, selected, and tristate. Indicators can be globally overriden with:

```
option add *Checkbutton.icons [list $imgnormal $imgcheck $imgtristate]
```

This is easier than setting the 5 options -image, -selectimage, -tristateimage, -indicatoron and -compound. It also leaves -image available for the user.

## 7.3 Window Shadow: -winshadow

The -winshadow option adds a drop shadow to a button/label. It takes 3 arguments that describe a color gradient:color1 color2 width. The shapedbutton.tcl screenshot above demonstrates the results.

## 7.4 Border Image: -bdimage

A border image is an image that is dynamically expanded/resized (with borders preserved) to fit the current size of the widget. The image simply provides decoration for the outside of the widget rectangle. Normally 16 pixels of the border are preserved, but -bdhalo can change this. (Note -bdimage is incompatible with -shape.)

Following is an example with a bunch of buttons using -bdimage:


And here is the code:

```
#!/usr/bin/env wize

set bdimg [image create photo -data {
        R0lGODlhQABAAPcAAHx+fMTCxKSipOTi5JSSlNTS1LSytPTy9IyKjMzKzKyq
        rOzq7JyanNza3Ly6vPz6/ISChMTGxKSmpOTm5JSWlNTW1LS2tPT29IyOjMzO
        zKyurOzu7JyenNze3Ly+vPz+/OkAKOUA5IEAEnwAAACuQACUAAFBAAB+AFYd
        QAC0AABBAAB+AIjMAuEEABINAAAAAHMgAQAAAAAAAAAAAKjSxOIEJBIIpQAA
        sRgBMO4AAJAAAHwCAHAAAAUAAJEAAHwAAP+eEP8CZ/8Aif8AAG0BDAUAAAJEA
        AHwAAIXYAOfxAIESAHwAAABAMQAbMBZGMAAAIEggJQMAIAAAAAAAfqgaXESI
        5BdBEgB+AGgALGEAABYAAAAAAACsNwAEAAAMLwAAAH61MQBIAABCM8B+AAAU
        AAAAAAAApQAAsf8Brv8AlP8AQf8Afv8AzP8A1P8AQf8AfgAArAAABAAADAAA
        AACQDADjAAASAAAAAACAAADVABZBAAB+ALjMwOIEhxINUAAAANIgAOYAAIEA
        AHwAAGjSAGEEABYIAAAAAEoBB+MAAIEAAHwCACABAJsAAFAAAAAAAGjJAGGL
        AAFBFgB+AGmIAAAQAABHAAB+APQoAOE/ABIAAAAAAADQAADjAAASAAAAAPiF
        APcrABKDAAB8ABgAGO4AAJAAqXwAAHAAAAUAAJEAAHwAAP8AAP8AAP8AAP8A
        AG0pIwW3AJGSAHx8AEocI/QAAICpAHwAAAA0SABk6xaDEgB8AAD//wD//wD/
        /wD//2gAAGEAABYAAAAAAC0/AHj5AASEgAAAAA01gBkWACDTAB8AFf43PT3
        5IASEnwAAOAYd+PuMBKQTwB8AGgAEGG35RaSEgB8AOj/NOL/ZBL/gwD/fMkc
        q4sA5UGpEn4AAIg02xBk/0eD/358fx/4iADk5QASEgAAAALnHABkAACDqQB8
        AMyINARkZA2DgwB8fBABHL0AAEUAqQAAAIAxKOMAPxIwAAAAAIScAOPxABIS
        AAAAAIIAnQwA/0IAR3cAACwAAAAAQABAAAAI/wA/CBxIsKDBgwgTKlzIsKFD
        gxceNnxAsaLFixgzUrzAsWPFCw8kDgy5EeQDkBxPolypsmXKlx1hXnS48UEH
        CwooMCDAgIJOCjx99gz6k+jQnkWR9lRgYYDJkAk/DlAgIMICkVgHLoggQIPT
        ighVJqBQIKvZghkoZDgA8uDJAwk4bDhLd+ABBmvbjnzbgMKBuoA/bKDQgC1F
        gW8XKMgQOHABBQsMI76wIIOExo0FZIhM8sKGCQYCYA4cwcCEDSYPLOgg4Oro
        uhMEdOB84cCAChReB2ZQYcGGkxsGFGCgGzCFCh1QH5jQIW3xugwSzD4QvIIH
        4s/PUgiQYcCG4BkC5P/ObpaBhwreq18nb3Z79+8Dwo9nL9I8evjWsdOX6D59
```

```
        fPH71Xeef/kFyB93/sln4EP2Ebjegg31B5+CEDLUIH4PVqiQhOABqKFCF6qn
        34cHcfjffCQaFOJtGaZYkIkUuljQigXK+CKCE3po40A0trgjjDru+EGPI/6I
        Y4co7kikkAMBmaSNSzL5gZNSDjkghkXaaGIBHjwpY4gThJeljFt2WSWYMQpZ
        5pguUnClehS4tuMEDARQgH8FBMBBBExGwIGdAxywXAUBKHCZkAIoEEAFp33W
        QGl47ZgBAwZEwKigE1SQgAUCUDCXiwtQIIAFCTQwgaCrZeCABAzIleIGHDD/
        oIAHGUznmXABGMABT4xpmBYBHGgAKGq1ZbppThgAG8EEAW61KwYMSOBAApdy
        pNp/BkhAAQLcEqCTt+ACJW645I5rLrgEeOsTBtwiQIEElRZg61sTNBBethSw
        CwEA/Pbr778ABywwABBAgAAG7xpAq6mGUUTdAPZ6YIACsRKAAbvtZqzxxhxn
        jDG3ybbKFHf36ZVYpuE5oIGhHMTqcqswvyxzzDS/HDMHEiiggQMLDxCZXh8k
        BnEBCQTggAUGGKCB0ktr0PTTTEfttNRQT22ABR4EkEABDXgnGUEn31ZABglE
        EEAAWaeN9tpqt832221HEEECW6M3wc+Hga3SBgtMODBABw00UEEBgxdO+OGG
        J4744oZzXUEDHQxwN7F5G7QRdXxPoPkAnHfu+eeghw665n1vIKhJBQUEADs=
}]

namespace import -force ::blt::tile::*
option add *highlightThickness 0
option add *Label.borderWidth 4
option add *Label.bdImage $bdimg
font conf TkDefaultFont -family Verdana -size 15 -weight bold
set pad 5

pack [frame .f -bg white] -fill x
foreach m {File Edit Commands Settings Help } {
    pack  [button .f.b$m -bdimage $bdimg -text $m] -side left
}

pack [label .l2] -fill x -side bottom
pack [label .l1] -fill both -expand y

text .l1.t -height 12 -bd 0
pack .l1.t -padx $pad -pady $pad -fill both -expand y

entry .l2.e -bd 0
pack .l2.e -padx $pad -pady $pad -fill x

.l1.t insert end "Here is a Text widget packed into a blt::tile::label "
.l1.t insert end "using -bdimage\nto provide shaped borders"
.l2.e insert end "ditto with an entry widget..."
```

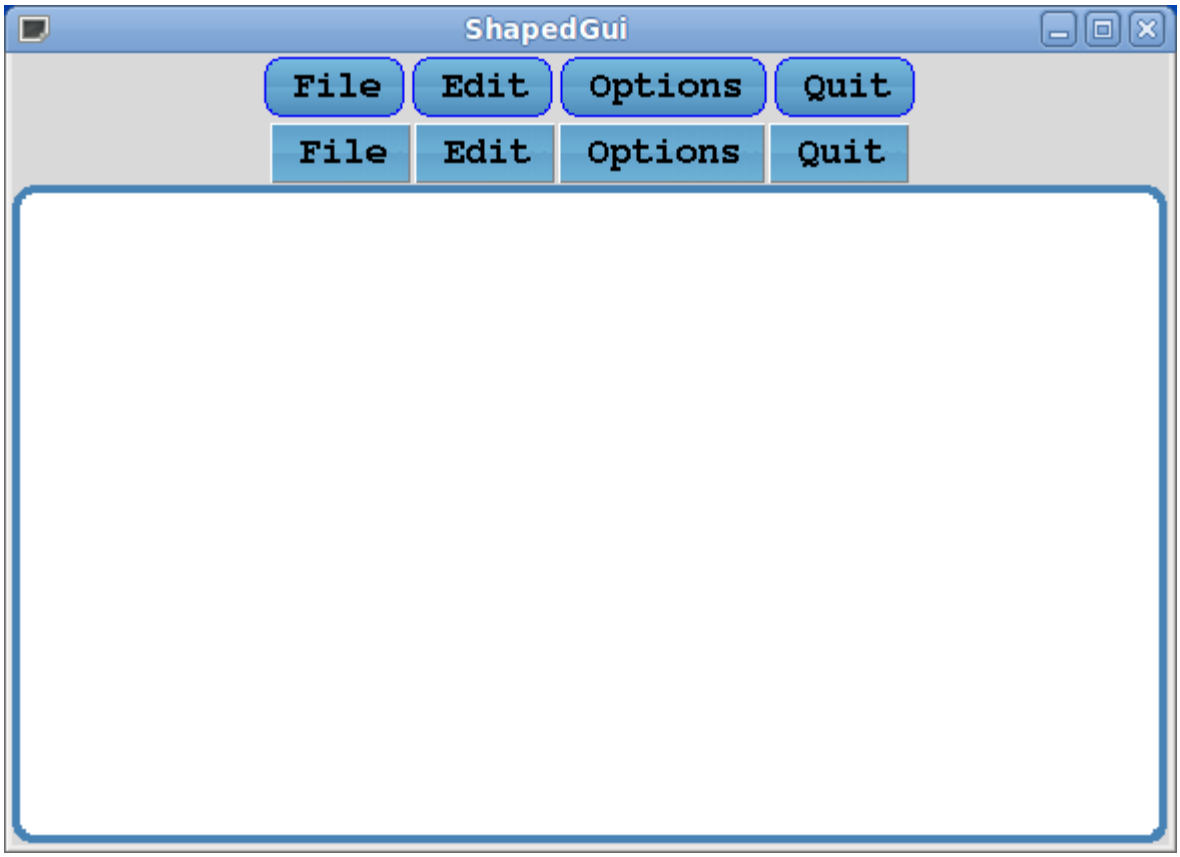
## 7.5 Shaped in Gui

The use of shape widgets can be enabled in **Gui** by using -blt, either in options or attributes.

```
{options - -blt 1} {}

{style} {
  Toplevel {
```

```
    @defgradients {
        bspl { SkyBlue SteelBlue -width 60 -height 10 -type split -rotate 90 }
        spl! { SteelBlue SkyBlue -width 33 -height 10 -type split -rotate 90 }
    }
  }
  Button { -font "Courier -18 bold" }
 .txtfr { -shape rounded -innerbg White -outline SteelBlue -linewidth 4}
 .txtwin { -bd 0 -highlightth 0 }
 @bspl { -bdimage ^bspl -bdhalo -1 }
 @spl  { -shape rounded -innertile ^spl! -outline Blue}
}
{Toplevel +} {
   {Frame + -subpos l -subattr {-gid spl}} {
     Button File Button Edit Button Options Button Quit
   }
   {Frame + -subpos l -subattr {-gid bspl}} {
     Button File Button Edit Button Options Button Quit
   }
   {Frame + -blt 1 -id txtfr} {
     {Text + -id txtwin} {}
   }
}
```

In the above, Button implicitly uses blt, while Frame requires -blt to override the tk::frame with blt::label. Here is the screenshot:




---

## 8. Gradients

Gradient images are widely used within applications and web pages to enhance appearance. Wize has built-in capabilities to generate on-the-fly, complex gradient images. This feature (provided via the Blt sub-command **winop image gradient**) is particularly useful when used with **Gui** @defgradients.

### 8.1 Options

The general form is:

winop image gradient image leftcolor rightcolor ?options...?

where options are:

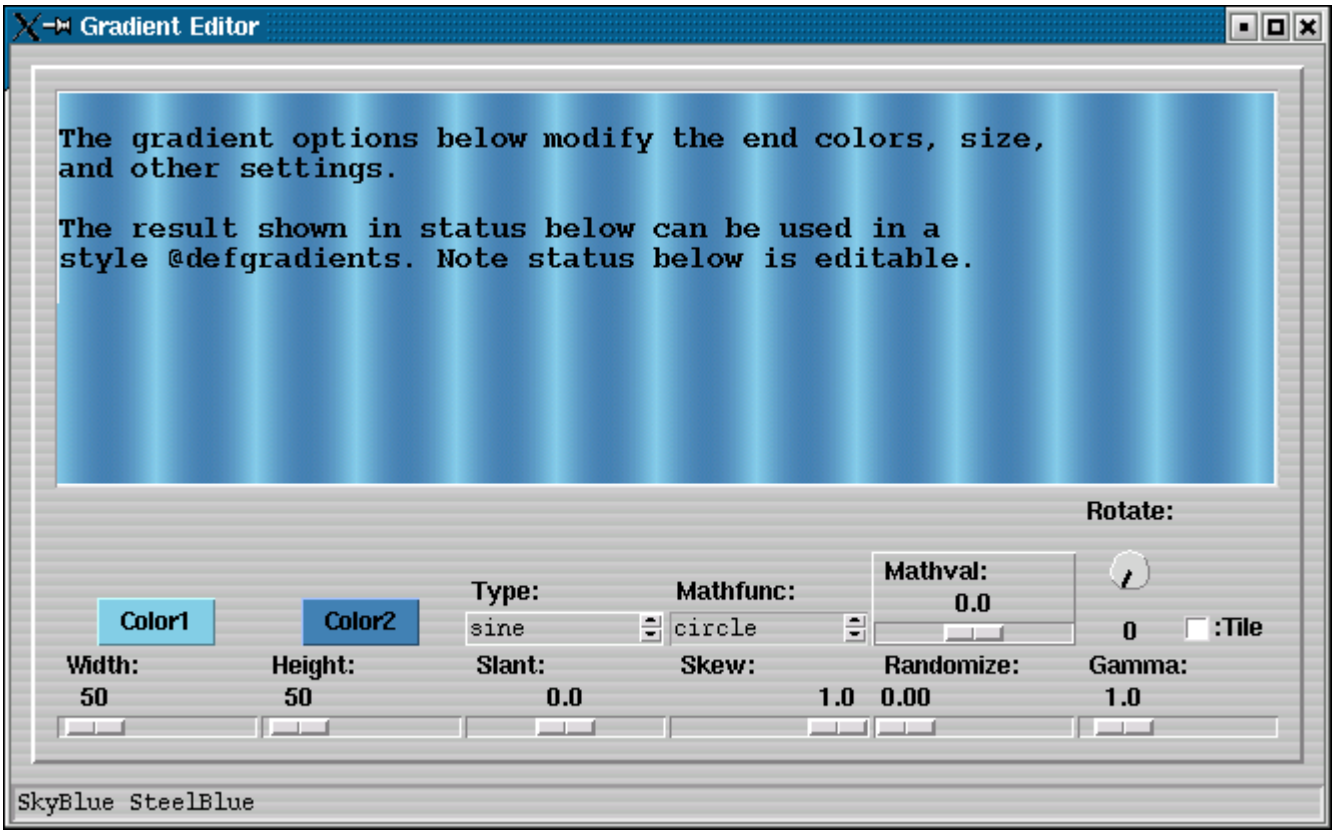
-type halfsine|sine|linear|rectangular|radial|blank
Set the type of gradient. The default is sine.

-skew N
The skew determines the initial fraction of the image that the gradient occupies, after which only rightcolor is used. The skew must be > 0 and <= 1.0 and has a default value of 1.0 (ie. not skewed).
-slant N
Make the gradient slant where a value of 1.0 slants at 45 degrees. The value must be between -100.0 and 100.0.
-curve N
Curve the gradient by passing the Y position to a function (see -func) scaled with the given value. The value must be between -100.0 and 100.0 (typically 1.0).
-func X
Function to use with -curve. The default value is sin. The value must be one of: sin cos tan sinh cosh tanh asin acos atan log log10 exp sqrt rand circle.
-rand N
Add small random purturbations to gradient to avoid striation lines. The value must be between 0.0 and 0.1.

## 8.2 User Interface

There is a **user interface** for exploring the options of gradients:

```
wize / Gui/Gradient
```



## 8.3 Gradients in Styles

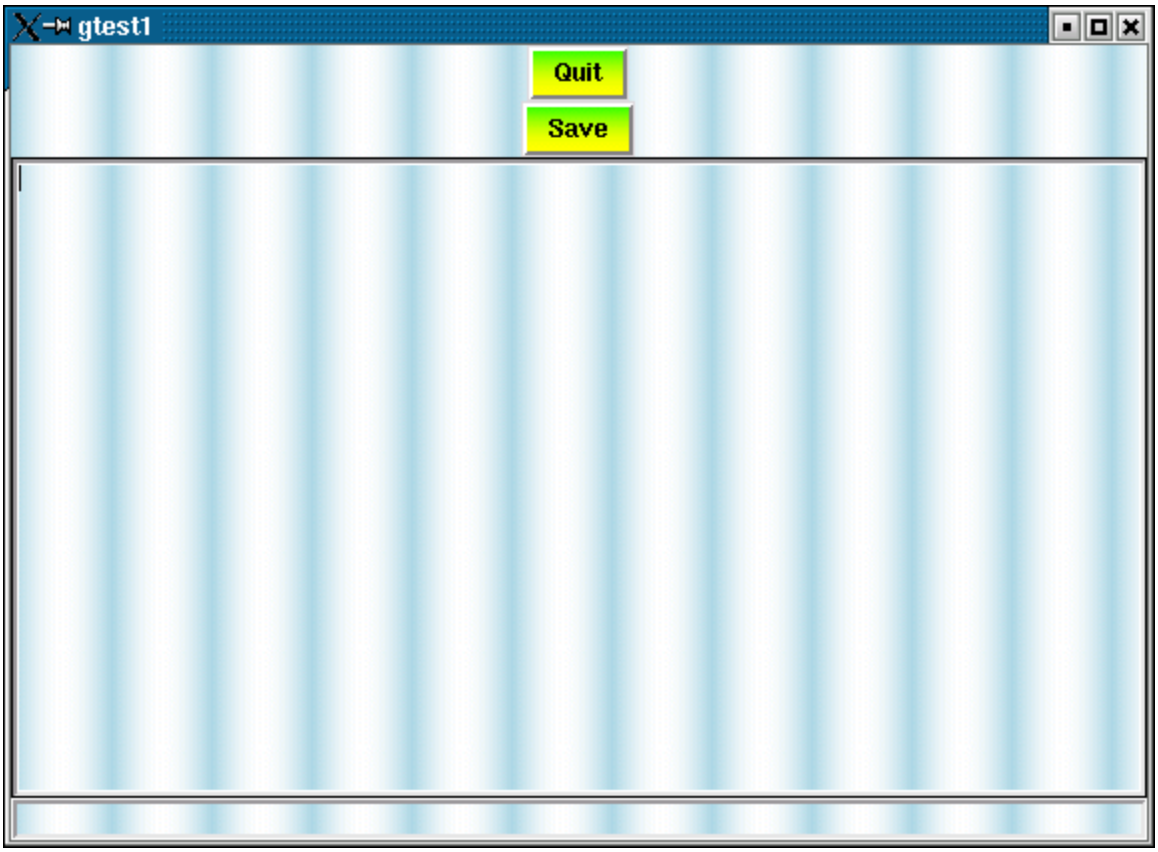
The easiest way to use gradient is with the **Gui Styles** @defgradients macro.

Note that @defgradients support options like -rotate, -tile and -gamma.

Here is a simple gui application using gradient styles.

```
# "gtest.gui"
style {
    Toplevel {
        @defgradients {
            mybg {LightBlue White}
            butbg! {Green Yellow -rotate 90}
        }
        *tile ^mybg
    }
    Button { -tile ^butbg! }
}
{Toplevel +} {
    {Button} Quit
    {Button} Save
    {Text - -pos *} {}
    {Entry - -pos _} {}
}
```

When run, this looks like:

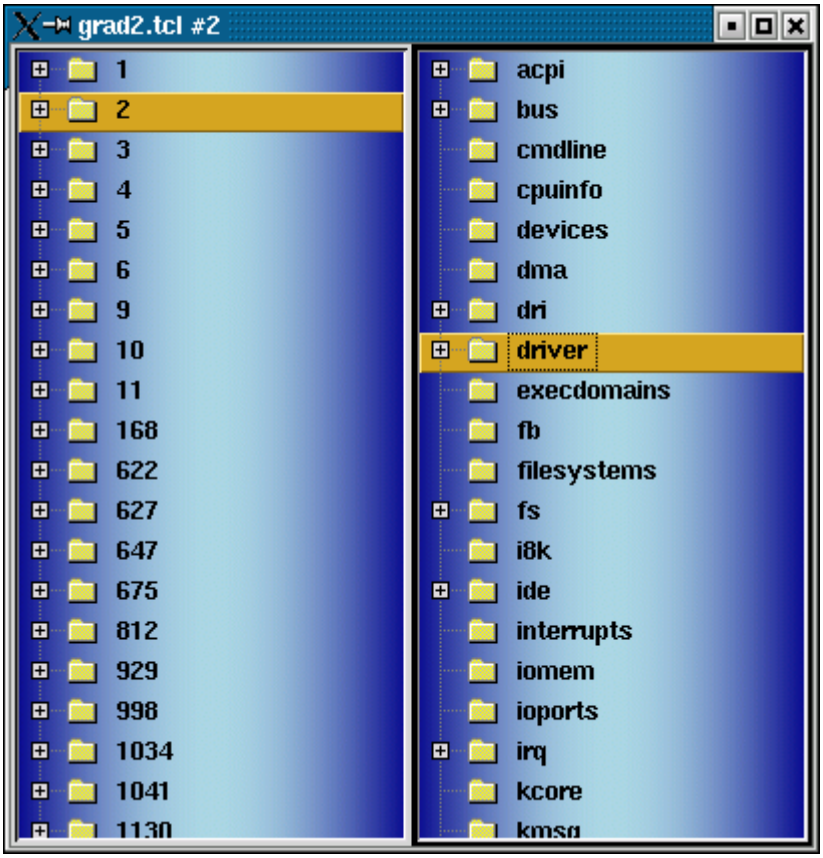


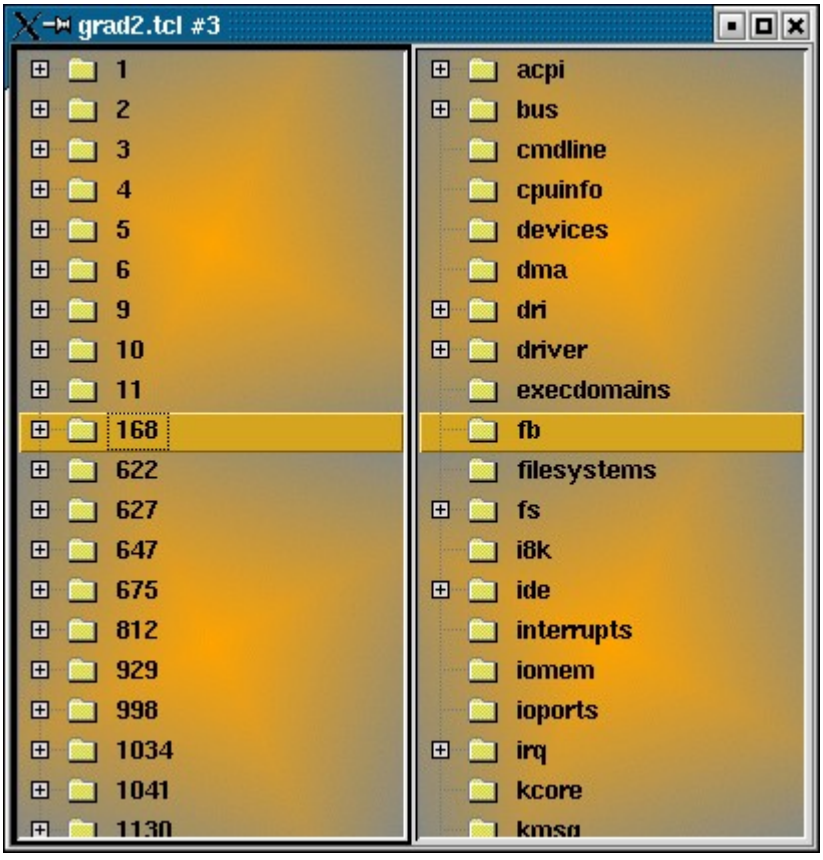

Note tiled image names containing a "!" will use a tile origin from the current window, rather than the toplevel.

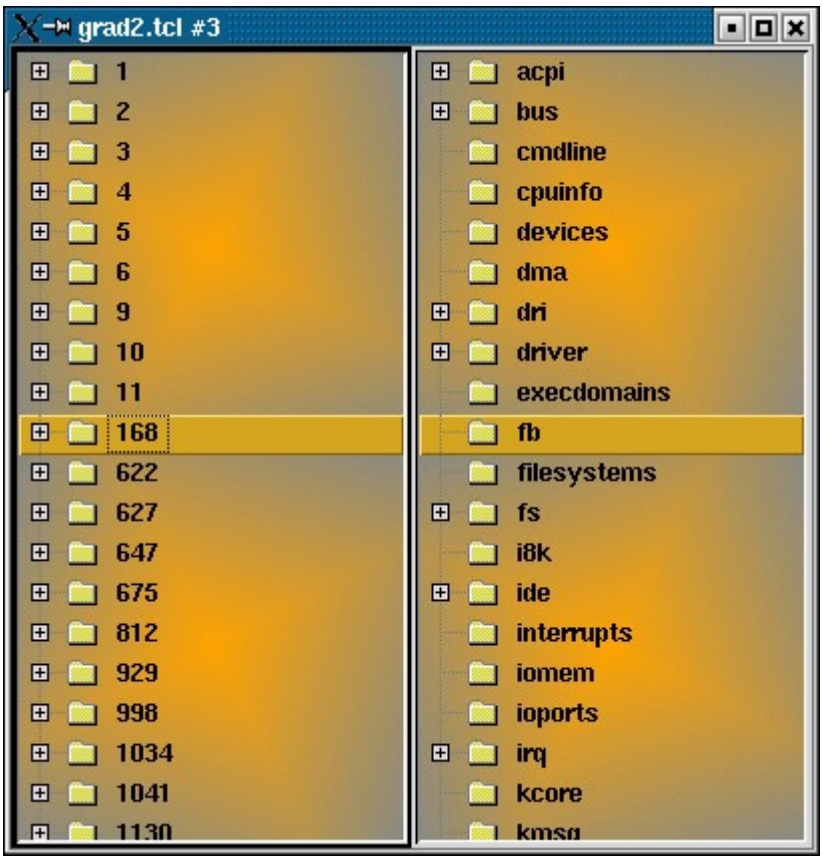
Documentation is available in the gradient sub-command of the **Winop manpage**.

## 8.4 More Examples

Here are a few gradient examples:


grad2.tcl #2
1
2
3
4
5
6
9
10
11
168
622
627
647
675
812
929
998
1034
1041
1130
acpi
bus
cmdline
cpuinfo
devices
dma
dri
driver
execdomains
fb
filesystems
fs
i8k
ide
interrupts
iomem
ioports
irq
kcore
kmsg


grad2.tcl #3
1
2
3
4
5
6
9
10
11
168
622
627
647
675
812
929
998
1034
1041
1130
acpi
bus
cmdline
cpuinfo
devices
dma
dri
driver
execdomains
fb
filesystems
fs
i8k
ide
interrupts
iomem
ioports
irq
kcore
kmsg


grad2.tcl #3
1
2
3
4
5
6
9
10
11
168
622
627
647
675
812
929
998
1034
1041
1130
acpi
bus
cmdline
cpuinfo
devices
dma
dri
driver
execdomains
fb
filesystems
fs
i8k
ide
interrupts
iomem
ioports
irq
kcore
kmsg


The following script can be used to generate the above images.

```
#!/usr/bin/env wize
# grad2.tcl:  demonstrates gradient tiled background generation, eg:
#
#   wize grad2.tcl -g sine -s DarkBlue -e LightBlue
#   wize grad2.tcl -g rectangular -s Orange -e  LightSlateGray
#   wize grad2.tcl -h 20

array set p { -h 200 -w 200 -s DarkGreen -e White -g radial}
array set p $argv
set img [image create photo -width $p(-w) -height $p(-h)]
winop image gradient $img $p(-s) $p(-e) -type $p(-g)

switch -- $p(-g) {
    sine - radial {}
    default {
       set img4 [image create photo]
       winop image mirror $img $img4 tile
       set img $img4
    }
```

```
}

# Create a couple of widgets with tiled background.
option add *font [eval font create [font actual {Helvetica -12 bold}]]
pack [treeview .t -tile $img -scrolltile 1] -fill both -expand y -side left
pack [treeview .t2 -tile $img] -fill both -expand y -side left
if {![file isdirectory [set dir /proc]]} { set dir "" }
foreach i [lsort -dictionary [glob -nocomplain $dir/*]] {
   set it [file tail $i]
   set isdir [file isdirectory $i]
   if {[string is integer $it]} {
       .t insert end $it -forcetree $isdir
   } else {
       .t2 insert end $it -forcetree $isdir
   }
}
foreach tt {.t .t2} {
   $tt conf -selectbackground GoldenRod
   $tt conf -nofocusselectbackground GoldenRod
   $tt conf -selectrelief raised
}
```

---

# 9. Tree

The **Blt** extension provides Tcl with a complex **tree data** structure, eg.

```
   set t [tree create]
   foreach i {Able Baker Charlie} { $t insert 0 -label $i }
   $t set 0->Able X 1 Y 2
   $t incr 0->Able X
```

## 9.1 Dict/Array Keys

Keys in a tree may store a dict that is accessed using an array-like notation, eg.

```
   $t insert 0 -label Harry -data {X 1 Y "a 1 b 2"}
   $t incr 0->Harry Y(a)
```

## 9.2 Static Tree.

Preloaded data trees are quite simple to define with the wize *tree command. Each line represents one row of data with the first token being the key. Subtrees are defined if the last element contains newlines. Titles fields are specified with a leading equals =. Here is an example:

```
*tree new t = {
    = Age Salary
    Managers {
        Tina 29 10000
        Tom  28 8000
    }
    Staff {
        Mary 10 6000
        Sam  10 6000
    }
 }
```

Trees are useful because of their ease of update and access:

```
*tree new t = {
    Vendors {
        = Id Status Products
        NA {
            Oracle 888001 active
            MS     888002 active
        }
        SA {
            Pemex 888008 disabled
            Snapon {
                = Class Items
                pipes {single double twin}
                tools {spanners sockets wrenches}
                wire  { 10 12 14 16 18 }
            }
        }
        Europe {
            Finetix 888009 active { pipes {single twin} wire  { 10 12 14 16 18 } }
        }
    }
}

pack [treeview .t -tree $t -width 600 -height 600] -fill both -expand y
eval .t col insert end [lsort [$t keys nonroot]]
.t open all

puts [$t get 0->Vendors->NA->Oracle]
puts [$t incr 0->Vendors->NA->Oracle Id 0.5]
puts [$t find -top 0->Vendors -name 888* -glob -key Id]
```

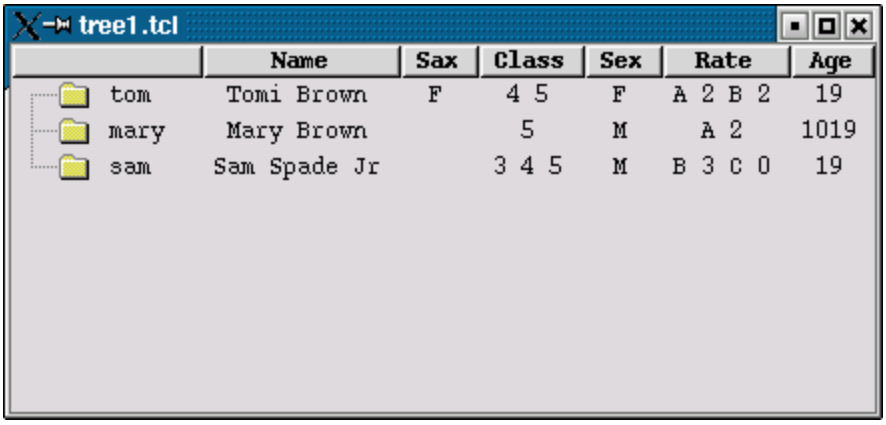
## 9.3 Flat Tree Example

The following loads a table of data into a tree, then updates it. (See also **Tables**)

```
variable Users {
      tom  { Name "Tom Brown"  Sex M Age 19  Class {4 5} Rate {A 1 B 2}}
```

```
     mary { Name "Mary Brown" Sex F Age 16  Class {5}   Rate {A 2}}
     sam  { Name "Sam Spade"  Sex M Age 19  Class {3 4} Rate {B 3}}
}

# Load it.
set t [tree create]
foreach {l d} $Users {
    $t insert end -label $l -data $d  -tags $l
}

# Update it.
$t update   tom       Sex F   Name "Tomi Brown" Age 21
$t append   sam       Name " Jr"
$t lappend  sam       Class 5
$t incr     mary      Age
$t update   tom       Rate(A) 2
$t set      tom       Sax F
$t set      sam       Rate(C) 0
$t incr     0->mary   Age;   # Address via label instead of tag.

# Display it.
pack [treeview .t -tree $t] -fill both -expand y
eval .t column insert end [$t keys all]
```

Note: nodes can be addressed using the form 0->LABEL. Tags can also be used to simplify indexing.

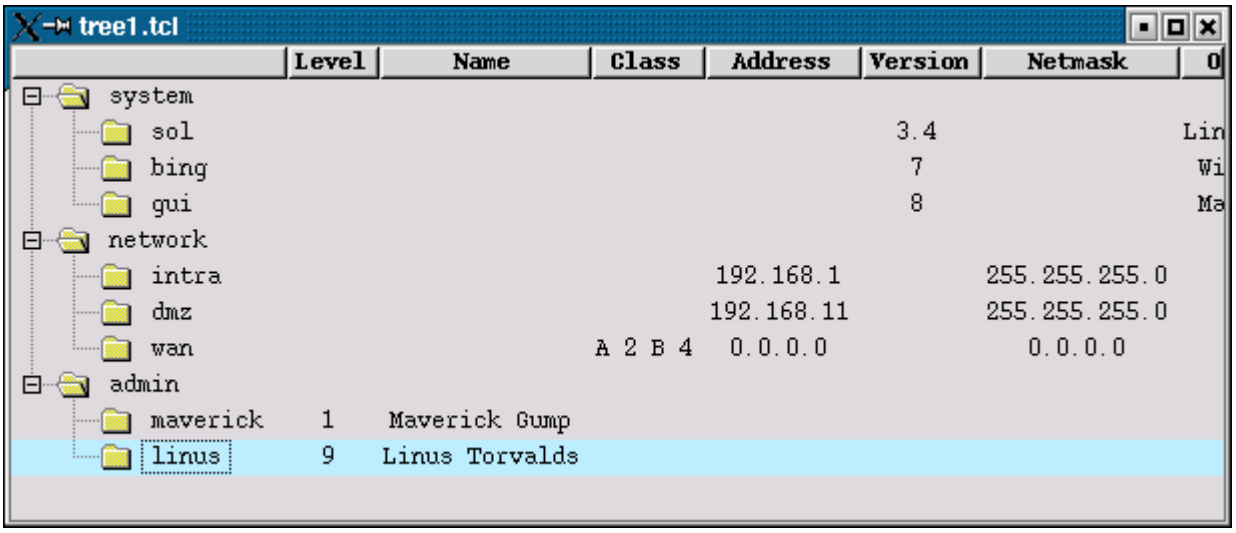



## 9.4 Nested Tree Example

The following example loads data into a nested tree. (See **Trees**)

```
variable Info {
     system {
```

```
          sol  { OS Linux Version 3.4 }
          bing { OS Win   Version 7 }
          gui  { OS Mac   Version 8 }
     }
     network {
          intra { Address 192.168.1  Netmask 255.255.255.0 }
          dmz   { Address 192.168.10 Netmask 255.255.255.0 }
          wan   { Address 0.0.0.0    Netmask 0.0.0.0 Class {A 1 B 4}}
     }
     admin {
          sully    { Name "Sully Van Damme"  Level 3 }
          maverick { Name "Maverick Gump"    Level 1 }
     }
}

# Load it.
set s [tree create]
foreach {n vals} $Info {
     set ind [$s insert end -label $n -tags .$n]
     foreach {l d} $vals {
          $s insert $ind -label $l -data $d -tags .$n.$l
     }
}

# Do queries.
$s update   .network.dmz   Address 192.168.11
$s update   .network.wan   Class(A) 2

set old [$s get  .system.bing]
$s update   .system.bing   OS Linux Version 3.4
eval $s set .system.bing   $old; # ROLLBACK!

$s insert   .admin -label linus -data { Name "Linus Torvalds" Level 9 }
$s delete   .admin.sully


pack [treeview .s -tree $s -width 600] -fill both -expand y
eval .s column insert end [$s keys all]
.s open all
```




## 9.5 Label & Tags

Nodes can be referenced using the label relative to the root, eg:

```
$s update   0->system->bing   OS Linux Version 3.4
```

However, label indexing has several limitations.

If a duplicate labels exists in the same parent the first match is quietly used. And care must be used to avoid labels with spaces, leading integers, or the names of builtins like nextnode, or firstchild (unless quoted).

Another way is to use the index command, which suppors label path lookups, eg:

```
$s update [$s index {system bing}]  OS Linux
```

Using tags however is simpler, and when used with a **tag trace** avoids duplicates.

## 9.6 Enums

A tree can be used as a simple enum by simply setting keys in node 0.

```
set t [tree create]
$t set 0   apple 1 orange 2 banana 3
puts [$t get 0]       ;  # "apple 1 orange 2 banana 3"
puts [$t names 0]     ;  # "apple orange banana"
puts [$t values 0]    ;  # "1 2 3"
puts [$t get 0 apple] ;  # "1"
```

Multiple enums are also easily defined:

```
set t [tree create]
$t set 0  fruit { apple 1 orange 2 banana 3 }
$t set 0  veggy { pea 1 bean 2 cabbage 3 }

puts [$t get 0 fruit(apple)] ;  # "1"
puts [$t get 0 veggy(bean)]  ;  # "2"
```

Alternatively, create each enum in its own node:

```
set t [tree create]
$t insert end -tags fruit -data { apple 1 orange 2 banana 3 }
$t insert end -tags veggy -data { pea 1 bean 2 cabbage 3 }

puts [$t get fruit apple] ;  # "1"
puts [$t get veggy bean]  ;  # "2"
```

If using > 21 keys per node, see **9.13 Key Hashing**.

## 9.7 With

Tree supports the with statement for accessing key data via an array. On entry it copies key values into an array variable, and on completion copies them back out. Eg:

```
$t with s .system.sol {
	$t with b .system.bing {
		set s(OS)	$b(OS)
		set s(Version) $b(Version)
	}
}
```

See **TreeWith** for more details.

## 9.8 Traces

Tree supports setting traces on nodes or notifiers on the tree. See **TreeTrace** for details.

## 9.9 Performance

Performance is generally quite good.

## 9.10 Tree Iterators

The following **tree** commands iterate over a tag:

| Name | Description |
|---|---|
| appendi | Append strings to key value. |
| incri | Increment a key value. |

| | |
|---|---|
| keys | Return keys for one or more nodes. |
| lappendi | Append list element to key value. |
| modify | Change data value for existing key. |
| set | Set/create data value for key. |
| sum | Sum values for a key field |
| vecdump | Dump values to a vector |
| vecload | Load values from a vector |
| with | Assign keys value to an array and eval |

## 9.11 Code Validation

wize supports **validation** of tree commands thus enabling static checking of tree code. To use this requires writing code using the tree object as data rather than as command. Thus the first example would be rewritten as:

```
tree op update  $t tom Sex F Name "Tomi Brown" Age 19
tree op append  $t sam     Name " Jr"
tree op lappend $t sam     Class 5
tree op incr    $t mary    Age
tree op update  $t tom     Rate(A) 2
tree op set     $t tom     Sax F
tree op set     $t sam     Rate(C) 0
tree op incr    $t 0->mary Age
```

The most important use is probably for with, eg.

```
    tree op with $t .system.bing b {
        set s(OS) LX
        set a b c
    }
```

to detect scripting errors.

## 9.12 Data Validation

See **Struct** for one approach to data validation.

## 9.13 Key Hashing

For nodes with 21 or fewer keys, tree remembers the order of key creation. Nodes with more than 21 keys will automatically change over to hash-table based key storage. One side-affect of this is that it alters the order of key iteration, which can change the results from get/names/values. That's because list-based storage preserves the order in which keys are added, whereas a hash-based storage has an undetermined order. This can be overcome by creating the tree with a large -keyhash size (eg. 1000000).

For example, the following sets keys from a list and avoids being hashed:

```
set t [tree create -keyhash [llength $lst]]
set n -1
foreach i $lst {
   $t set 0   $i [incr n]
}
puts [$t names 0 ; # outputs the original $lst.
```

Note that adding just one more key will cause the above to switch to hashing and thus scramble the lst order.

---

## 10. Ted - The Editor

**Ted** is a tabbed editor written using **Gui**. It provides several key functions, the most important of which is **completion**for Tcl and Tk commands and Tk subcommands. For example, we can type the following:

```
TreeView e
```

and in the status line note there are two matching subcommands: entry and edit. By adding an n the editor shows the matching entry which typing <Tab> will complete.
If we then type:

```
TreeView entry conf $w $id -
```

and hit <Control-space>, we get list of all the known options.
This can greatly simplify the job of writing Tk code. It virtually eliminates the need to memorize hundreds of subcommands or their thousands of options.

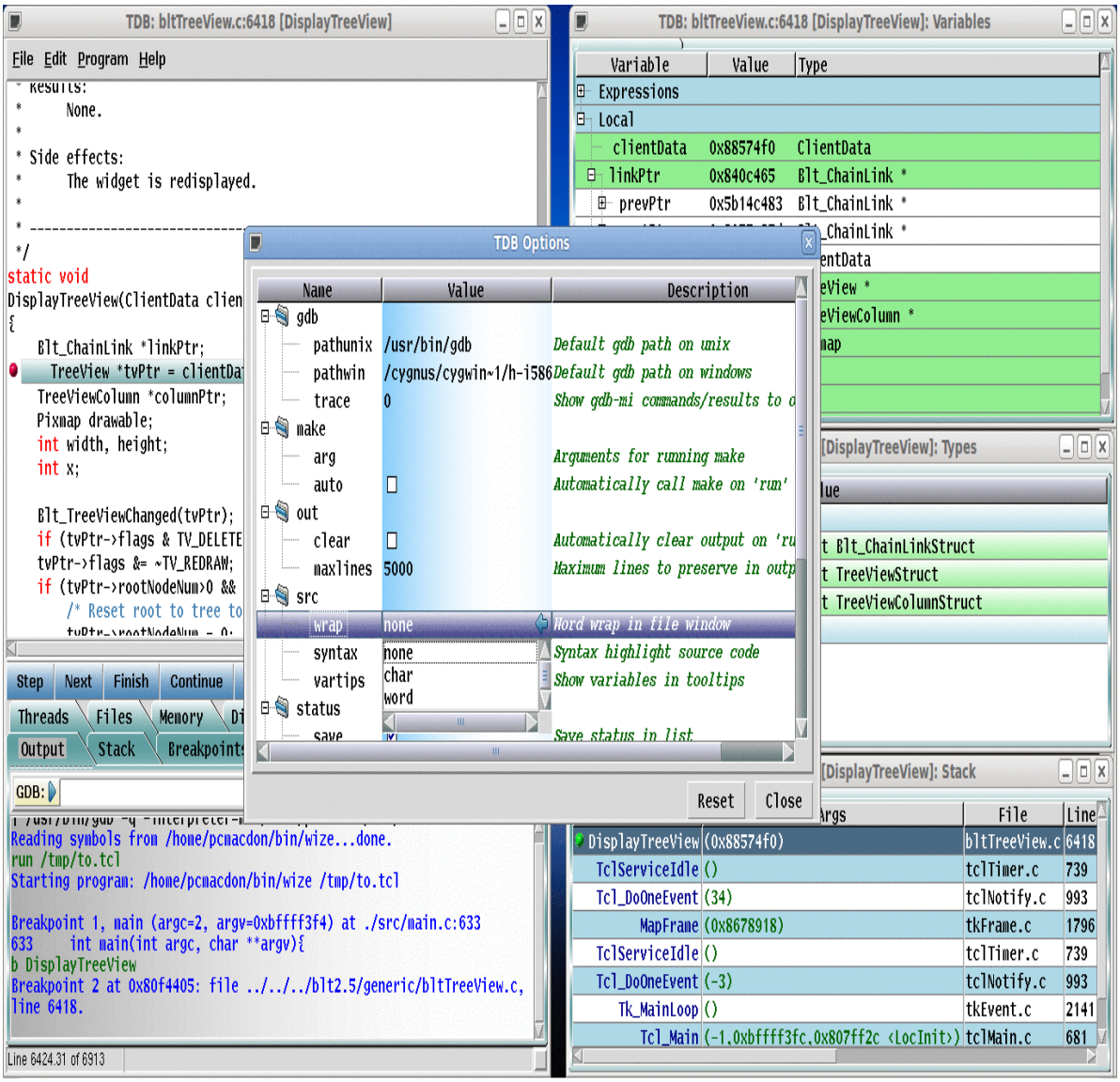
Despite this power, Ted is a fairly simple application. It derives much of it's functionality by hooking into the Tcl implementation of Gui.

---

## 11. Tdb - A GDB Frontend

Tdb is a Gui frontend for GDB written in **Wize** and **Gui**. It provide a compact but powerful interface that exposes most of GDBs features using MI. Unlike other such frontends, Tdb does not use a C parser to decode MI because it maps MI output directly to a Tcl list. This fact allows Tdb to be developed and distributed in pure script form.

## 11.1 Screenshot




## 11.2 Features

Tdb provides the following features:

- A Stack browser.
- A Variable tree inspector.
- A Types tree inspector.

- Files and Functions tree with searches.
- Memory, Registers, Threads and Disassembly.
- A GDB help tree browser with searches.
- A GDB options tree browse and modify.
- Direct access to the GDB interface.

Tdb is fast, and provides most navigation just by double-clicking.

- Double clicking in the Stack tab will return to that point of execution.
- Double clicking in the Variable tab will go to the declaration.
- Double clicking in the Types tab will go to the type definition, etc.

The implementation source is about 5K lines of which 4K lines are validated Tcl code, and 600 lines are GUI specification. Of the latter, the layout and style code are about 50/50 or about 300 lines each.

The total size of tdb.zip is about 35K.

### 11.3 Running Tdb

Tdb can be executed thus:

```
wize tdb.zip myprog arg ....
# or
wize / Apps/Tdb myprog arg ...
```

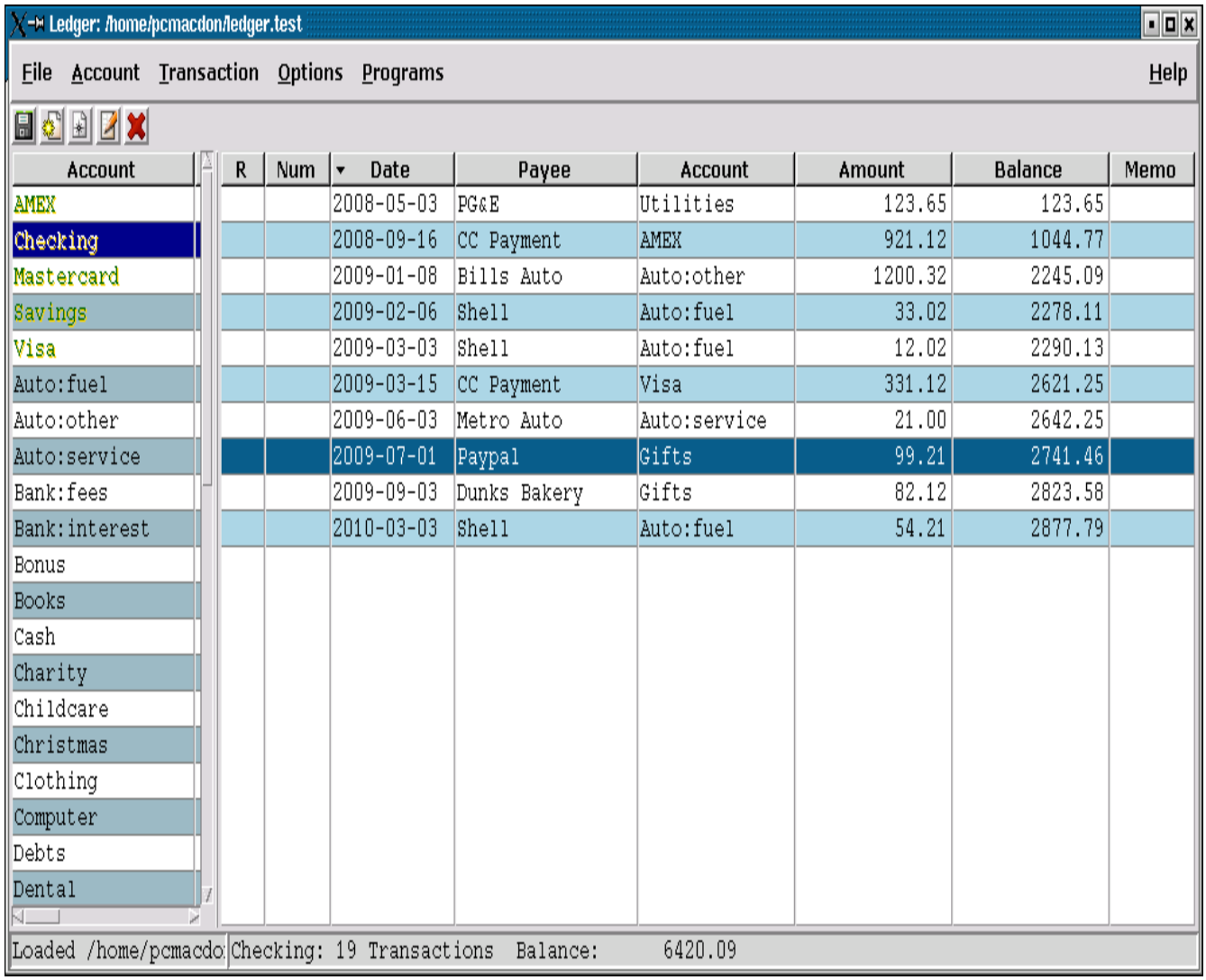
---

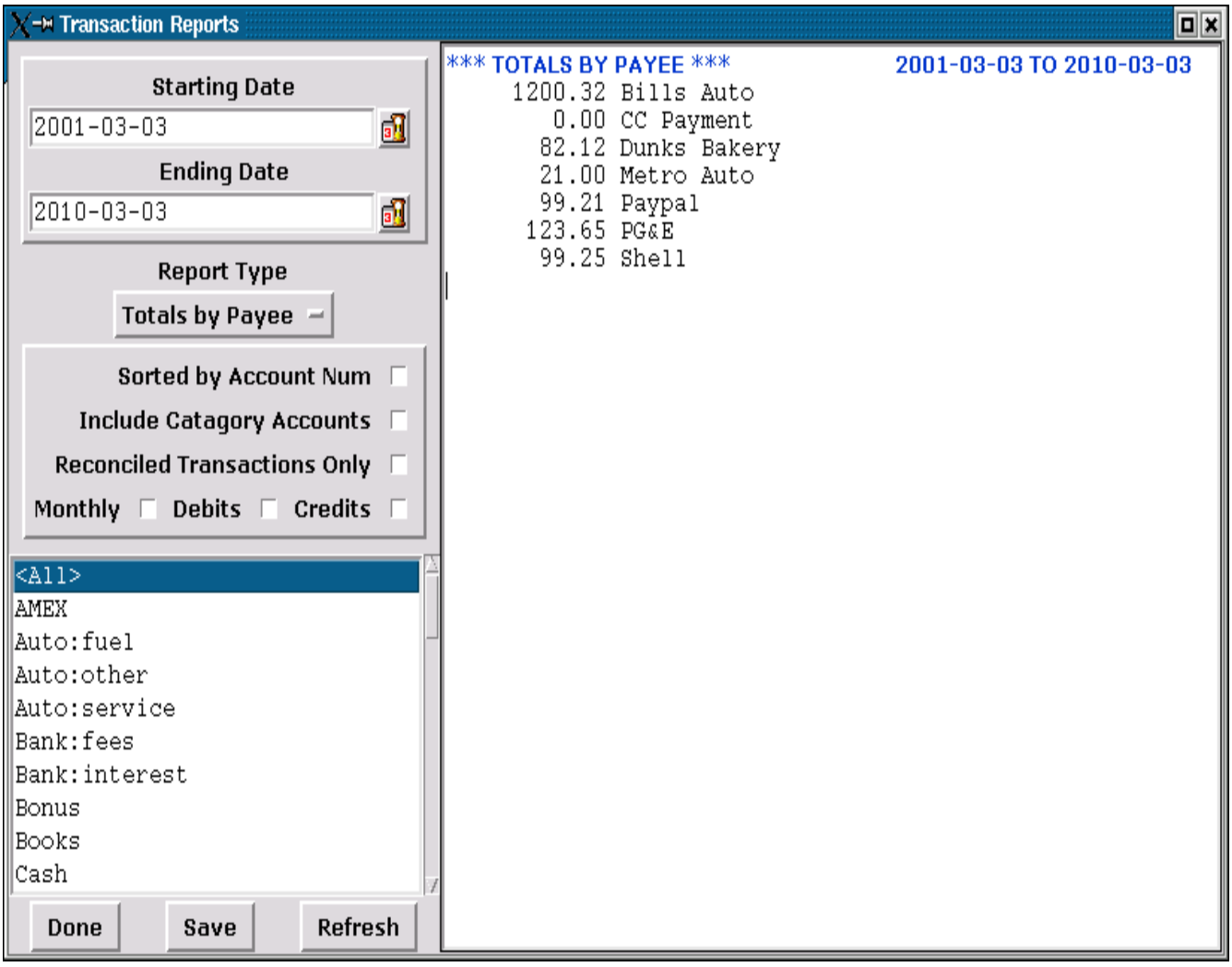
## 12. Ledger

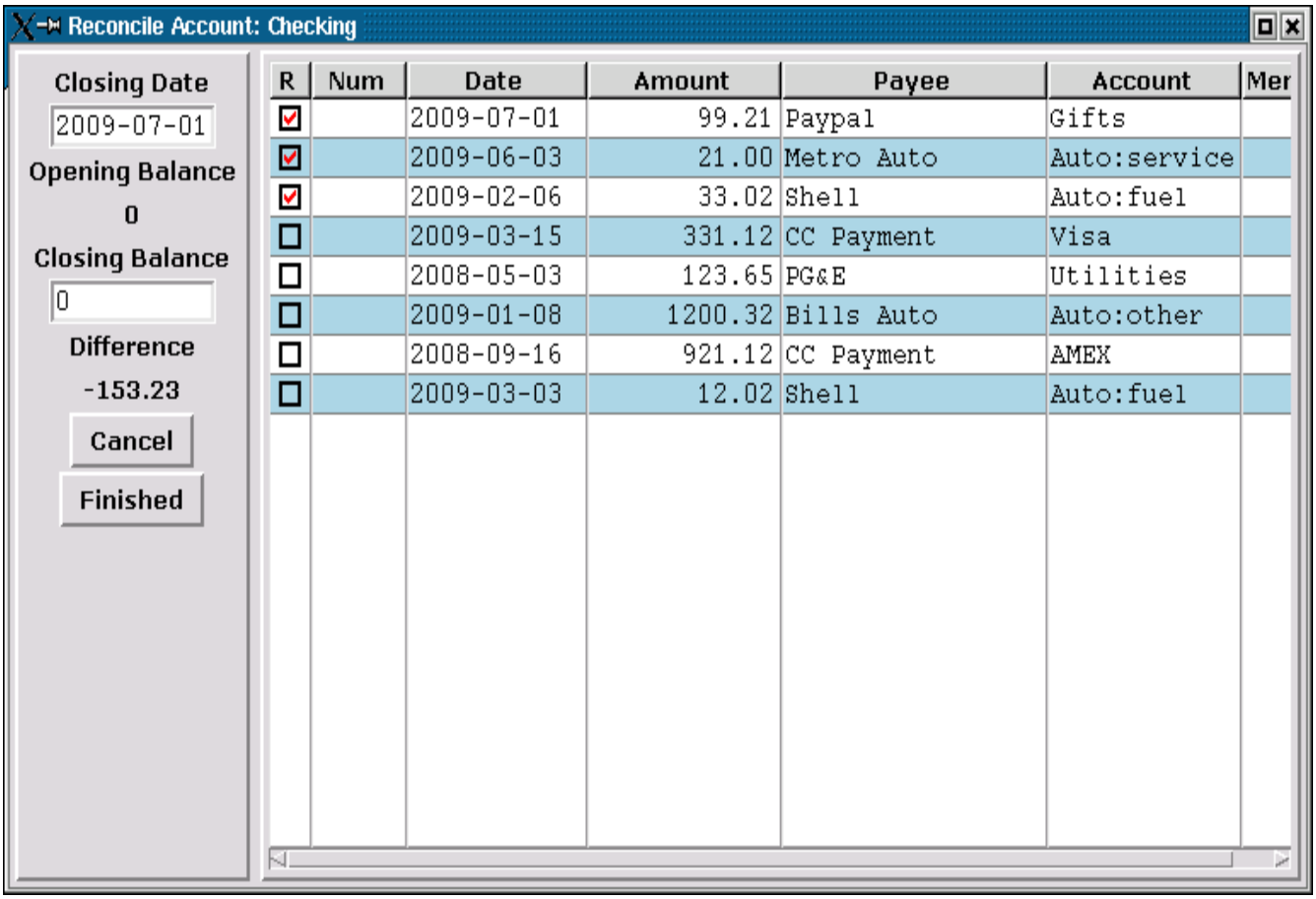
Ledger is **Gui** based personal finance application featuring:

- Fast and easy use with auto-completions.
- Reconciliation and report dialogs.
- Import/export QIF transactions/accounts.
- Uses double-entry accounting.
- Handles 10's of thousands of transactions with ease.
- Stores data as plain UTF8 text.
- Supports RCS and CVS for backup-on-save.
- Near zero dependancies (implemented in a single .tcl file).

Here are some screenshots:


Ledger: /home/pcmacdon/ledger.test
File Account Transaction Options Programs Help
Account
AMEX
Checking
Mastercard
Savings
Visa
Auto:fuel
Auto:other
Auto:service
Bank:fees
Bank:interest
Bonus
Books
Cash
Charity
Childcare
Christmas
Clothing
Computer
Debts
Dental
R Num Date Payee Account Amount Balance Memo
2008-05-03 PG&E Utilities 123.65 123.65
2008-09-16 CC Payment AMEX 921.12 1044.77
2009-01-08 Bills Auto Auto:other 1200.32 2245.09
2009-02-06 Shell Auto:fuel 33.02 2278.11
2009-03-03 Shell Auto:fuel 12.02 2290.13
2009-03-15 CC Payment Visa 331.12 2621.25
2009-06-03 Metro Auto Auto:service 21.00 2642.25
2009-07-01 Paypal Gifts 99.21 2741.46
2009-09-03 Dunks Bakery Gifts 82.12 2823.58
2010-03-03 Shell Auto:fuel 54.21 2877.79
Loaded /home/pcmacdo
Checking: 19 Transactions Balance: 6420.09


Transaction Reports
Starting Date
2001-03-03
Ending Date
2010-03-03
Report Type
Totals by Payee
Sorted by Account Num
Include Catagory Accounts
Reconciled Transactions Only
Monthly
Debits
Credits
<All>
AMEX
Auto:fuel
Auto:other
Auto:service
Bank:fees
Bank:interest
Bonus
Books
Cash
Done
Save
Refresh
*** TOTALS BY PAYEE ***
2001-03-03 TO 2010-03-03
1200.32 Bills Auto
0.00 CC Payment
82.12 Dunks Bakery
21.00 Metro Auto
99.21 Paypal
123.65 PG&E
99.25 Shell




## 12.1 Running It

Ledger requires **Wize** and is run like so:

```
wize ledger.tcl
```

or the builtin version can be run with:

```
wize / Gui/Ledger
```

## 12.2 Data Storage

Since Ledger uses **Tree** saving and restoring data simply uses the sub-commands dump and restore.

## 12.3 Multiple Books

Multiple sets of accounts can be managed using:

```
wize / Gui/Ledger -dir ~/work
```

If -dir is not given it defaults to ~/ledger.

## 12.4 Exploring/Debugging

As with all Wize applications, you can use <Control-Alt-Shift-2> to **explore** it. Select aclist_1 from the "Vars" menu to examine the accounts data, or xaction_1 to examine transaction data. Or use Introspect to examine the entire program state.

## 12.5 Un-implemented Features

- Scheduled transactions.
- Budgets and investments.
- Multiple currencies.
- Charts, graphs, etc.
- Bank download/sync.

---

# 13. Top

Top is a GUI interface to the Unix text based system monitoring facility top. Its purpose is to exercise some key features of Wize, including:

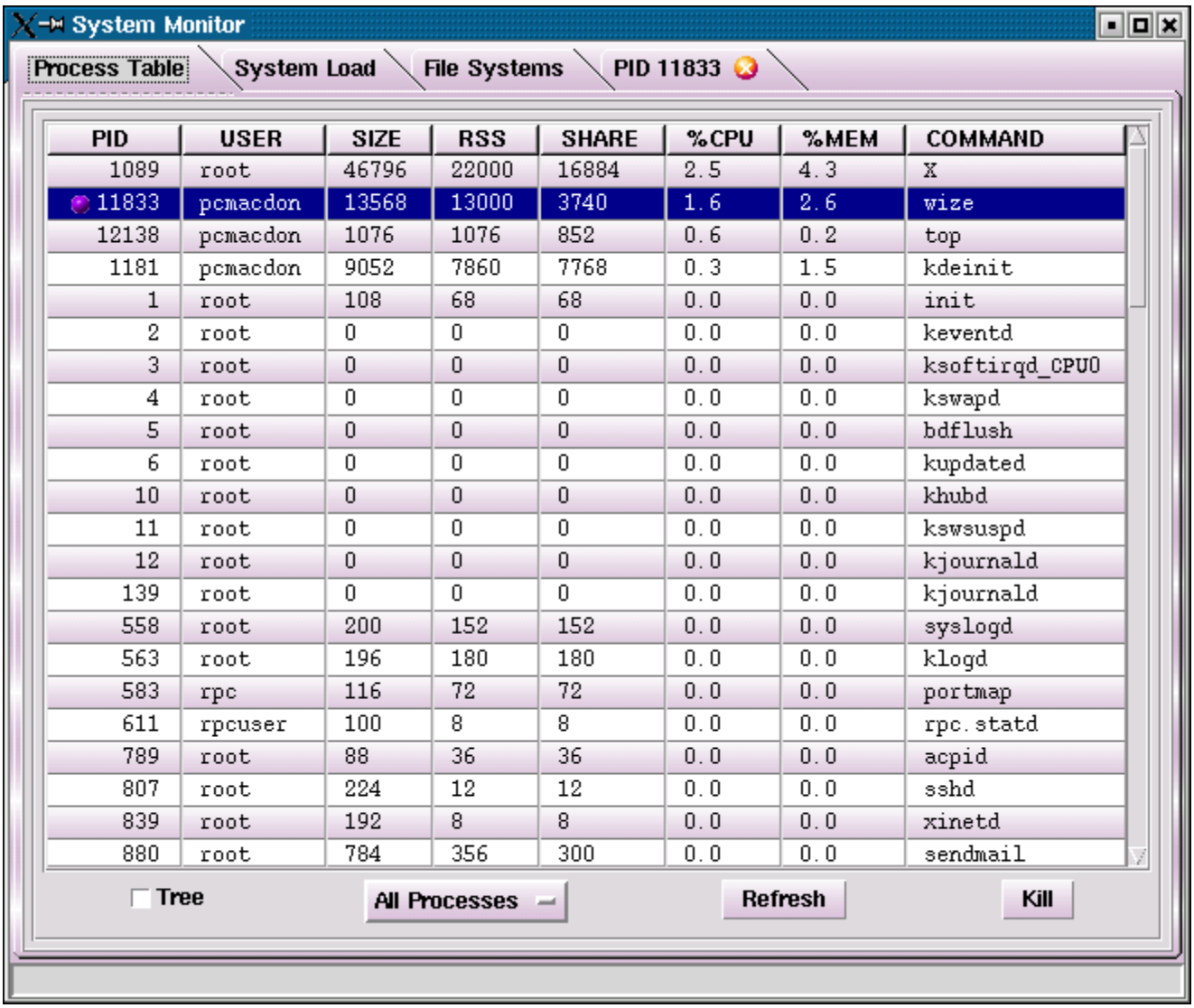
- demonstrate the ease of using **Gui**.
- repeatedly insert/delete data from a TreeView widget.
- make extensive use of **Styles**.

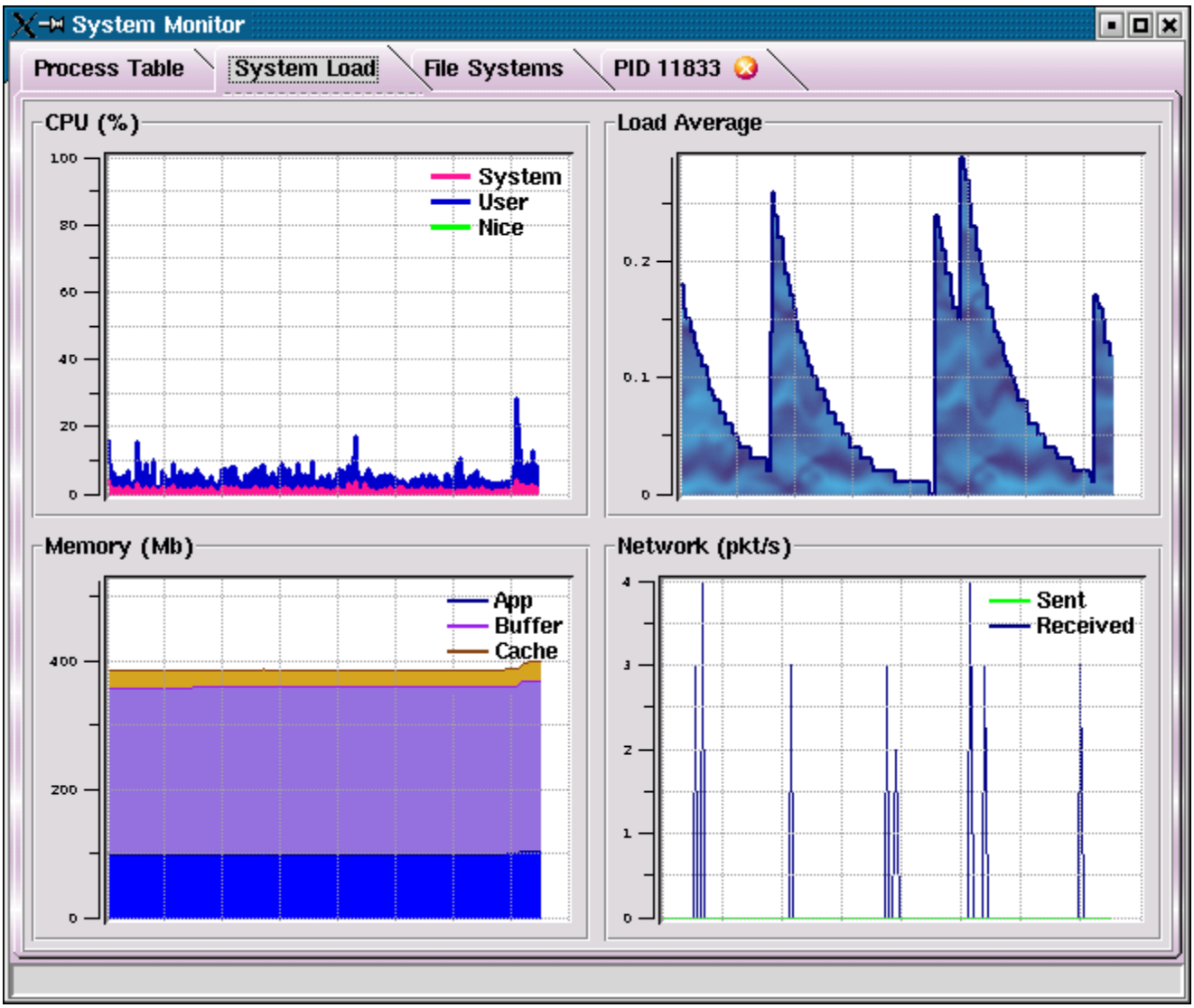
Top can be invoked with:

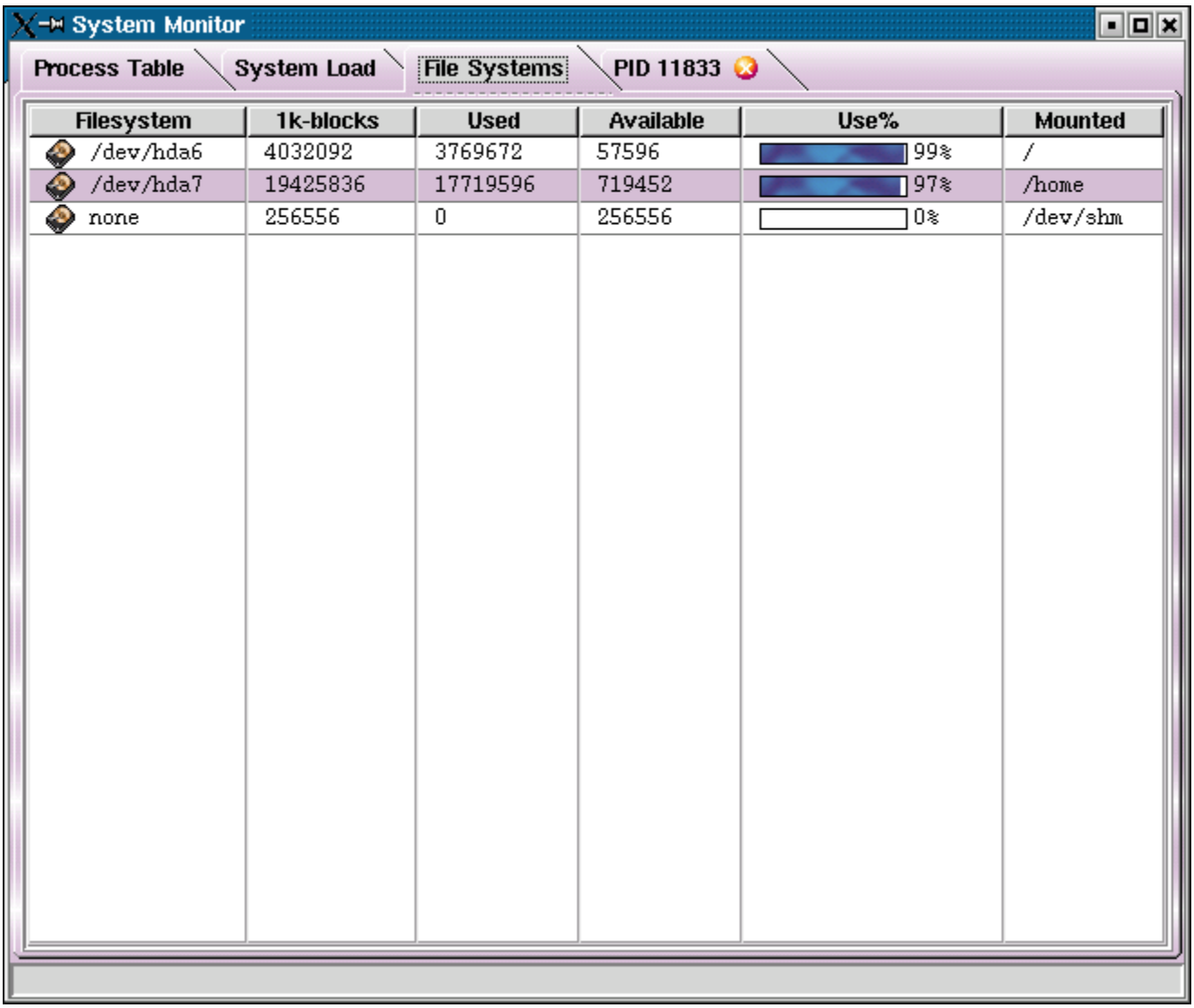
```
wize / Gui/Top
```

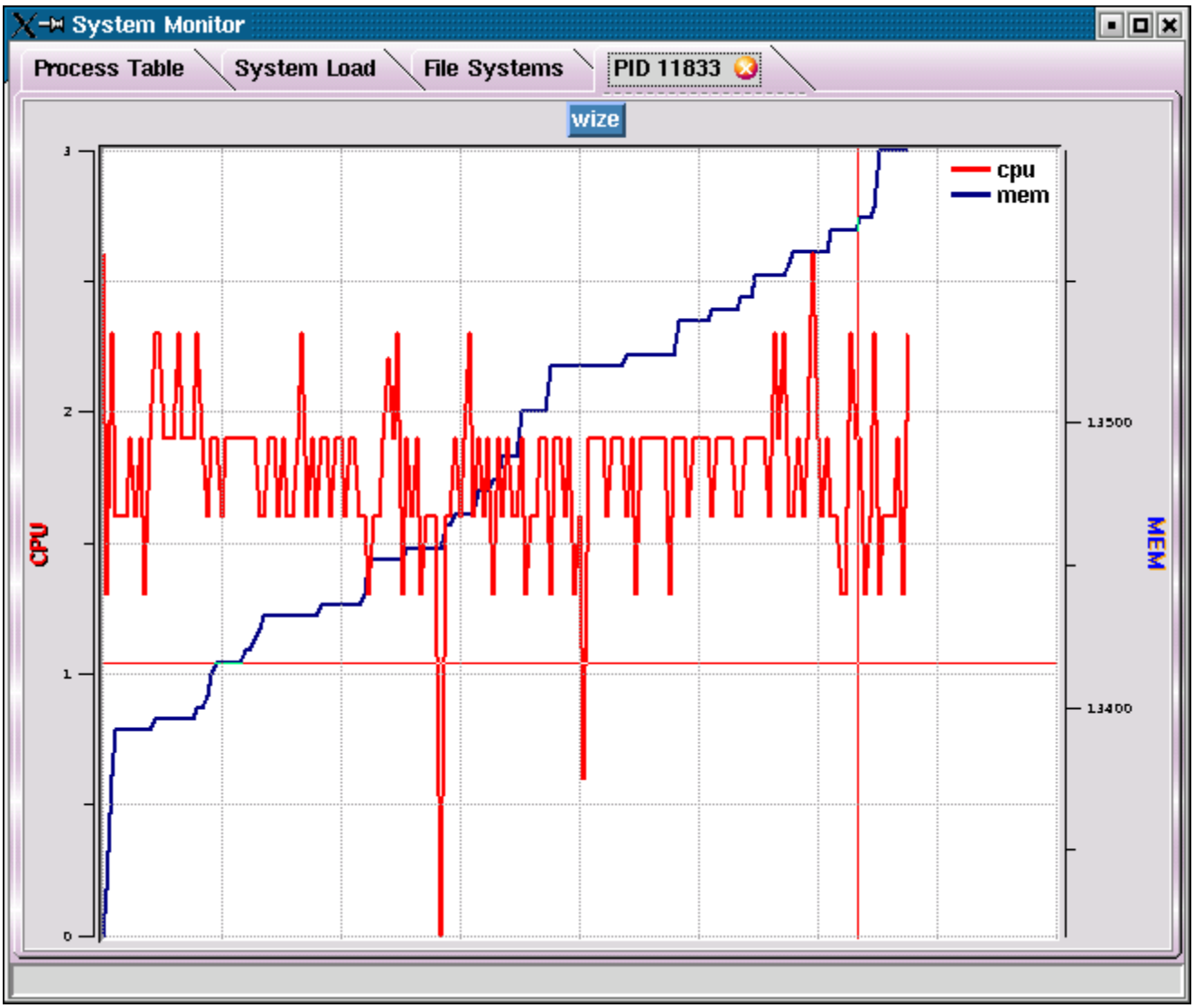
Top has 3 main tabs, plus optional per-PID monitors.

Here are some screenshots:


System Monitor
Process Table
System Load
File Systems
PID 11833
PID USER SIZE RSS SHARE %CPU %MEM COMMAND
1089 root 46796 22000 16884 2.5 4.3 X
11833 pcmacdon 13568 13000 3740 1.6 2.6 wize
12138 pcmacdon 1076 1076 852 0.6 0.2 top
1181 pcmacdon 9052 7860 7768 0.3 1.5 kdeinit
1 root 108 68 68 0.0 0.0 init
2 root 0 0 0 0.0 0.0 keventd
3 root 0 0 0 0.0 0.0 ksoftirqd_CPU0
4 root 0 0 0 0.0 0.0 kswapd
5 root 0 0 0 0.0 0.0 bdflush
6 root 0 0 0 0.0 0.0 kupdated
10 root 0 0 0 0.0 0.0 khubd
11 root 0 0 0 0.0 0.0 kswsuspd
12 root 0 0 0 0.0 0.0 kjournald
139 root 0 0 0 0.0 0.0 kjournald
558 root 200 152 152 0.0 0.0 syslogd
563 root 196 180 180 0.0 0.0 klogd
583 rpc 116 72 72 0.0 0.0 portmap
611 rpcuser 100 8 8 0.0 0.0 rpc.statd
789 root 88 36 36 0.0 0.0 acpid
807 root 224 12 12 0.0 0.0 sshd
839 root 192 8 8 0.0 0.0 xinetd
880 root 784 356 300 0.0 0.0 sendmail
Tree
All Processes
Refresh
Kill


System Monitor
Process Table
System Load
File Systems
PID 11833
CPU (%)
100
80
60
40
20
0
System
User
Nice
Load Average
0.2
0.1
0
Memory (Mb)
400
200
0
App
Buffer
Cache
Network (pkt/s)
4
3
2
1
0
Sent
Received

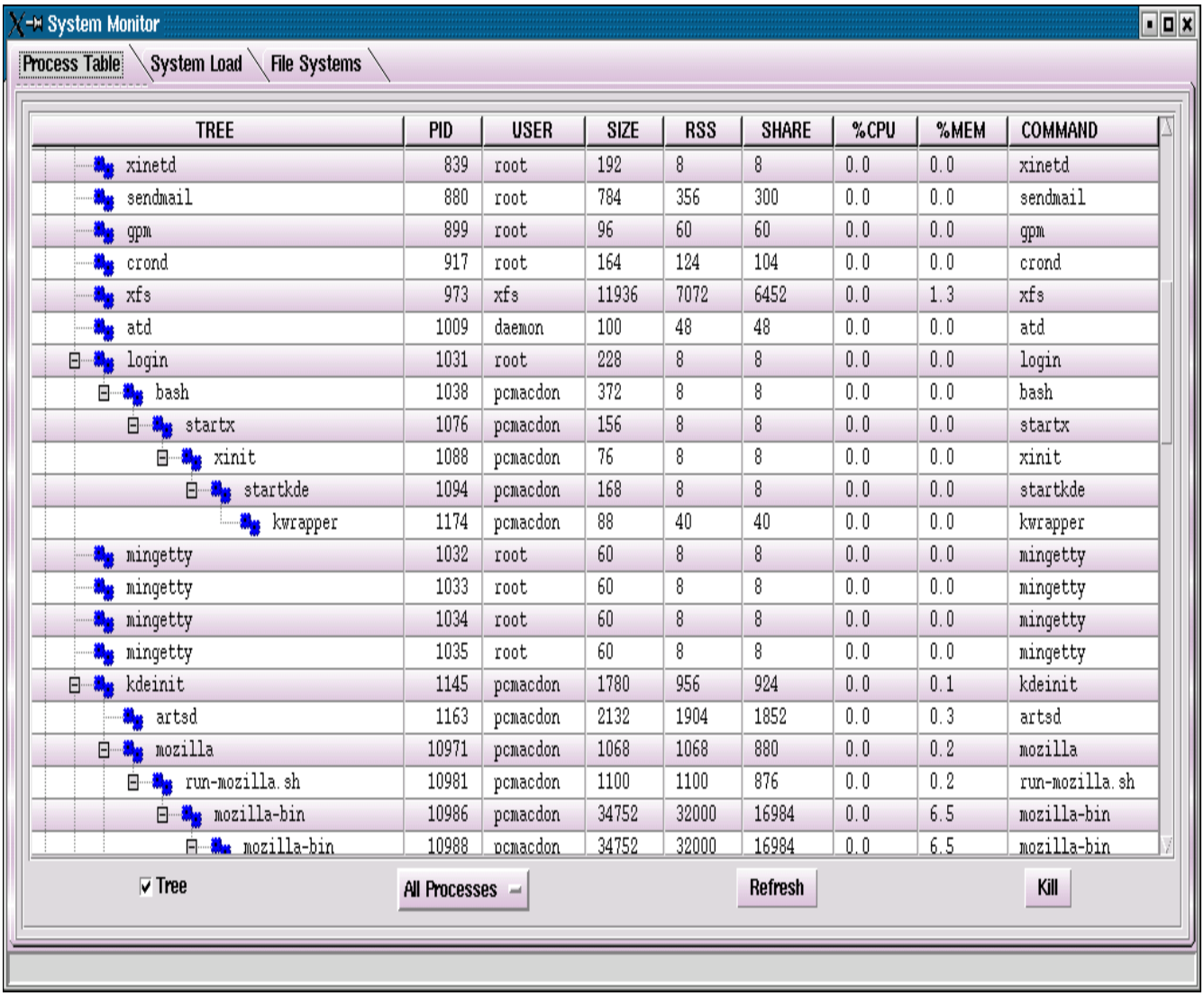

System Monitor
Process Table
System Load
File Systems
PID 11833
Filesystem
1k-blocks
Used
Available
Use%
Mounted
/dev/hda6
4032092
3769672
57596
99%
/
/dev/hda7
19425836
17719596
719452
97%
/home
none
256556
0
256556
0%
/dev/shm



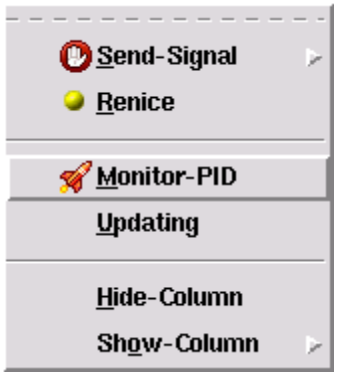

## 13.1 Process Table

Displays a list of all processes running on the system.

There are options for displaying only a subset of processes, as well as changing the display mode to tree.




Using right-click gives a menu that allows monitoring specific PIDs, Renicing a process or sending signals to a process. It can also show or hide columns.



## 13.2 System Load

Displays 4 graphs:

- CPU% - cumultive CPU used by running processes.
- Memory - cumultive Memory used by processes.
- Load Average - the average load factor.
- Network - network activity

## 13.3 File Systems

This displays usage by file system.

## 13.4 PID Monitor

PID Monitor collects and graphs information about a single process. To close the tab, left click on the red cross.

---

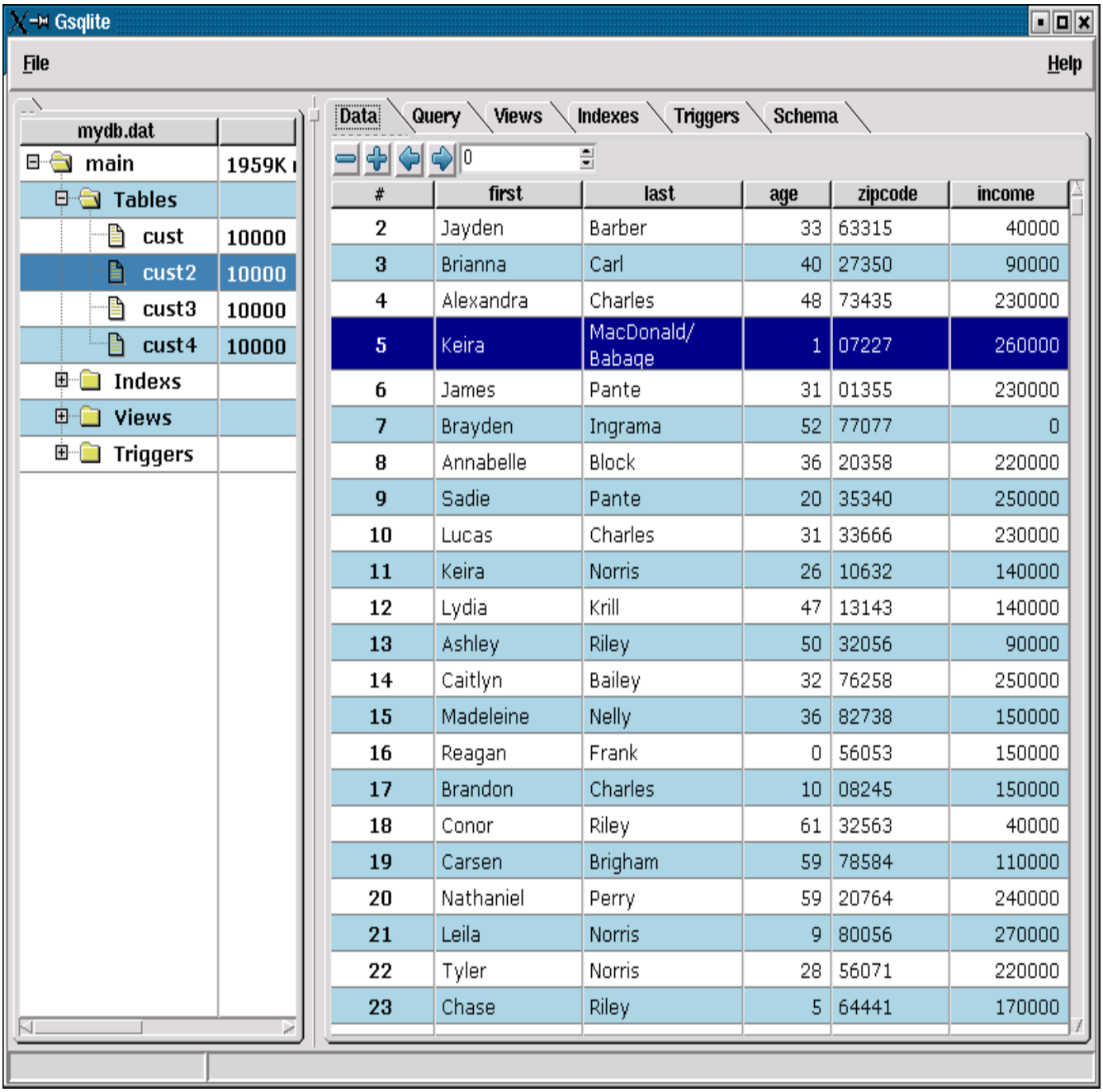
# 14. Gsqlite

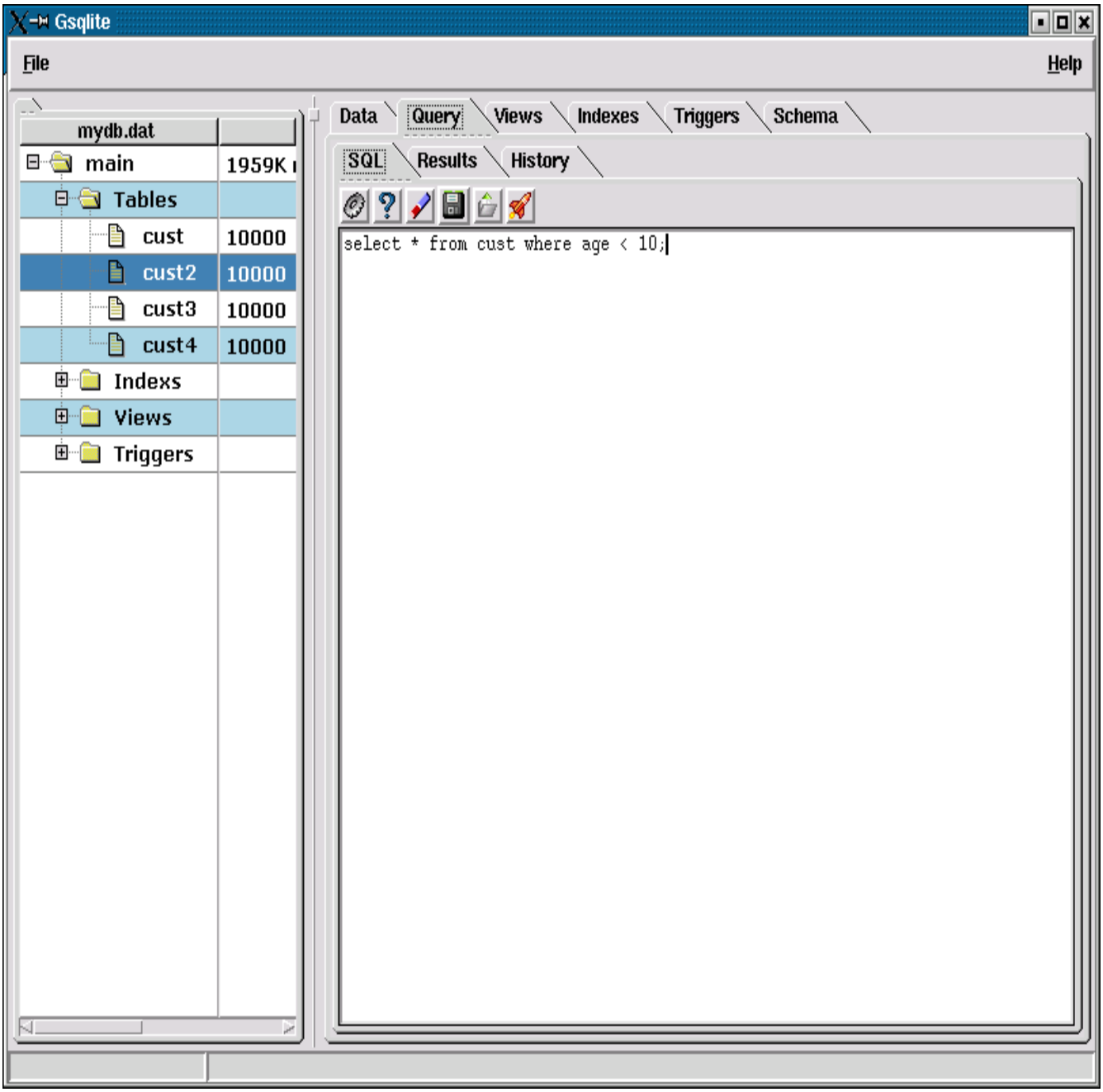
Gsqlite is a user interface for **Sqlite**. It is modeled somewhat after Sqlite Studio, but it's main purpose is to demonstrate how **Gui** can enable single file applications.

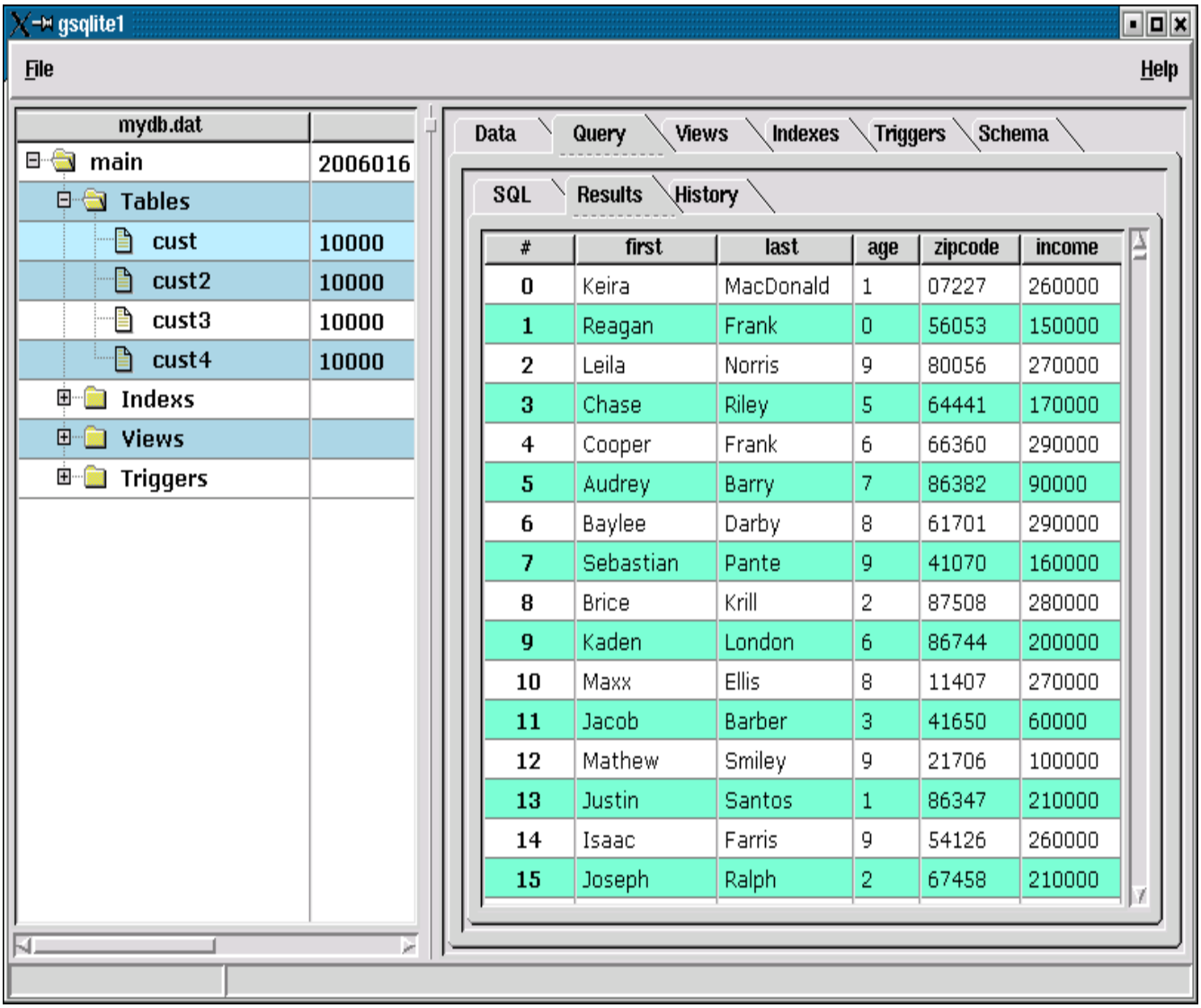
You can run Gsqlite from **Wize**, eg:

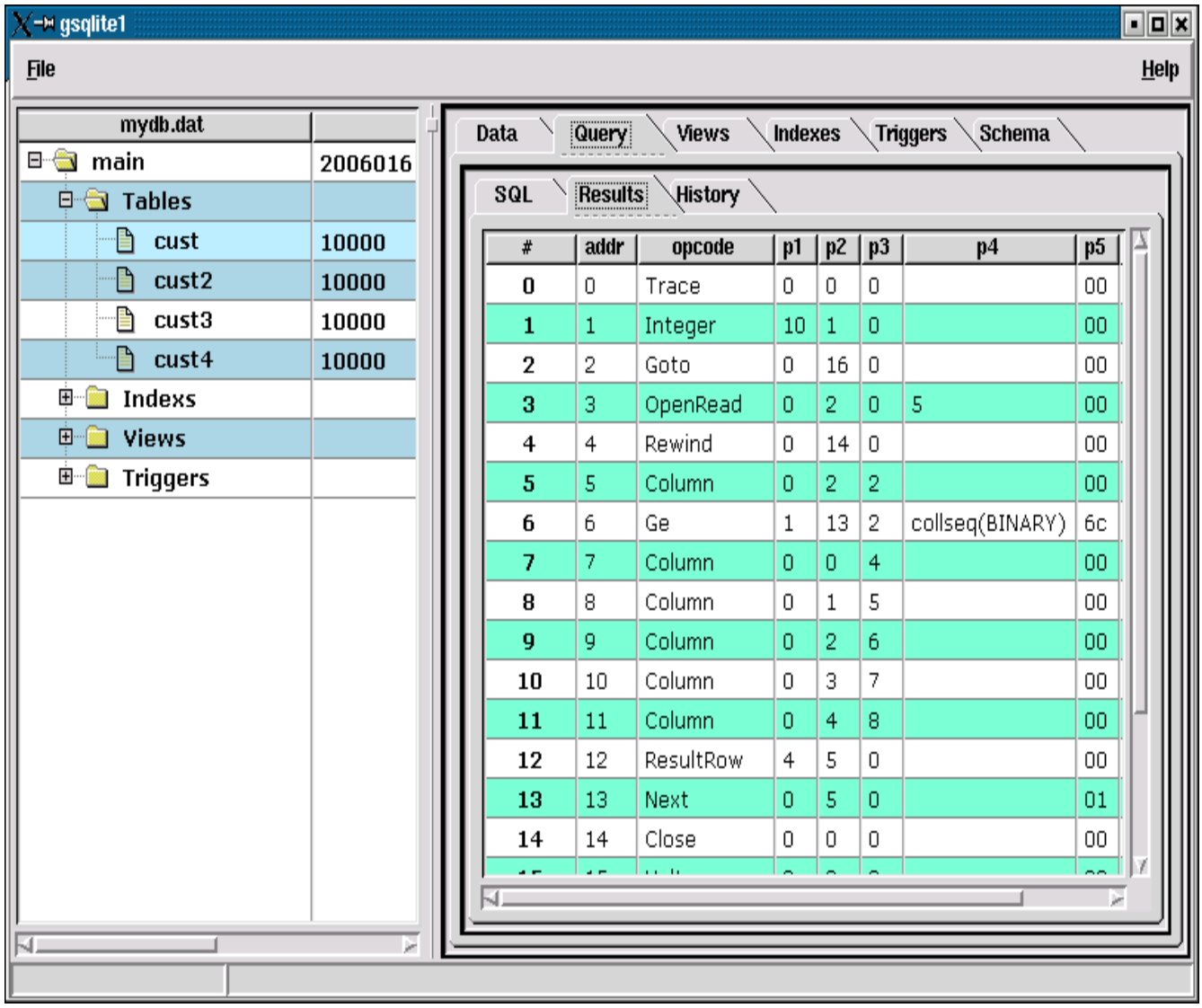
```
wize / Mod/Gsqlite mydata.db
```

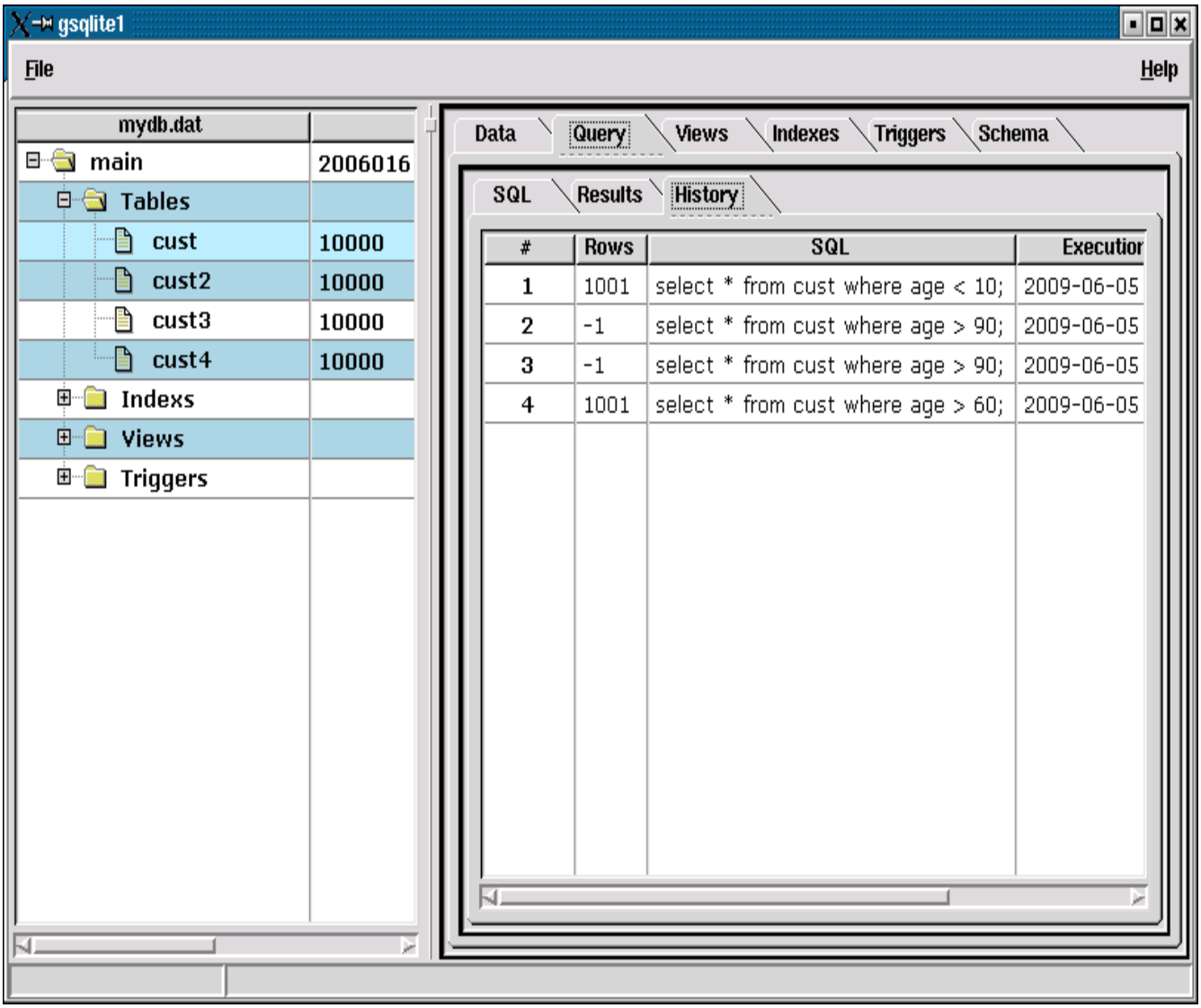
There is currently no documentation other than a few screenshots:


Gsqlite
File
Help
mydb.dat
main
1959K
Tables
cust
10000
cust2
10000
cust3
10000
cust4
10000
Indexs
Views
Triggers
Data
Query
Views
Indexes
Triggers
Schema
0
# first last age zipcode income
2 Jayden Barber 33 63315 40000
3 Brianna Carl 40 27350 90000
4 Alexandra Charles 48 73435 230000
5 Keira MacDonald/ Babage 1 07227 260000
6 James Pante 31 01355 230000
7 Brayden Ingrama 52 77077 0
8 Annabelle Block 36 20358 220000
9 Sadie Pante 20 35340 250000
10 Lucas Charles 31 33666 230000
11 Keira Norris 26 10632 140000
12 Lydia Krill 47 13143 140000
13 Ashley Riley 50 32056 90000
14 Caitlyn Bailey 32 76258 250000
15 Madeleine Nelly 36 82738 150000
16 Reagan Frank 0 56053 150000
17 Brandon Charles 10 08245 150000
18 Conor Riley 61 32563 40000
19 Carsen Brigham 59 78584 110000
20 Nathaniel Perry 59 20764 240000
21 Leila Norris 9 80056 270000
22 Tyler Norris 28 56071 220000
23 Chase Riley 5 64441 170000


Gsqlite
File
Help
mydb.dat
main
1959K
Tables
cust
10000
cust2
10000
cust3
10000
cust4
10000
Indexs
Views
Triggers
Data
Query
Views
Indexes
Triggers
Schema
SQL
Results
History
select * from cust where age < 10;

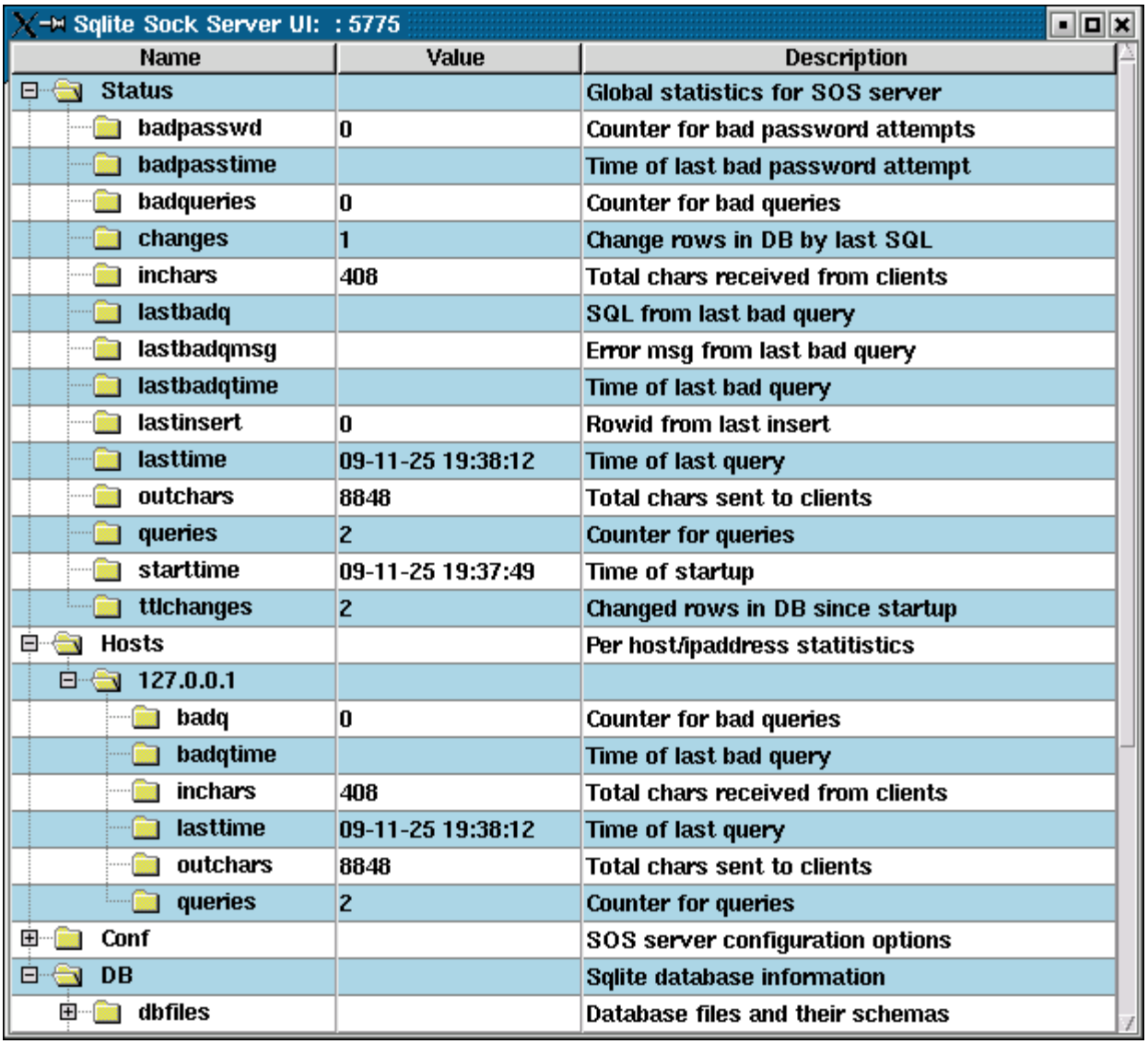

gsqlite1
File
Help
mydb.dat
main
2006016
Tables
cust
10000
cust2
10000
cust3
10000
cust4
10000
Indexs
Views
Triggers
Data
Query
Views
Indexes
Triggers
Schema
SQL
Results
History
# first last age zipcode income
0 Keira MacDonald 1 07227 260000
1 Reagan Frank 0 56053 150000
2 Leila Norris 9 80056 270000
3 Chase Riley 5 64441 170000
4 Cooper Frank 6 66360 290000
5 Audrey Barry 7 86382 90000
6 Baylee Darby 8 61701 290000
7 Sebastian Pante 9 41070 160000
8 Brice Krill 2 87508 280000
9 Kaden London 6 86744 200000
10 Maxx Ellis 8 11407 270000
11 Jacob Barber 3 41650 60000
12 Mathew Smiley 9 21706 100000
13 Justin Santos 1 86347 210000
14 Isaac Farris 9 54126 260000
15 Joseph Ralph 2 67458 210000


gsqlite1
File
Help
mydb.dat
main
2006016
Tables
cust
10000
cust2
10000
cust3
10000
cust4
10000
Indexs
Views
Triggers
Data
Query
Views
Indexes
Triggers
Schema
SQL
Results
History
# addr opcode p1 p2 p3 p4 p5
0 0 Trace 0 0 0 00
1 1 Integer 10 1 0 00
2 2 Goto 0 16 0 00
3 3 OpenRead 0 2 0 5 00
4 4 Rewind 0 14 0 00
5 5 Column 0 2 2 00
6 6 Ge 1 13 2 collseq(BINARY) 6c
7 7 Column 0 0 4 00
8 8 Column 0 1 5 00
9 9 Column 0 2 6 00
10 10 Column 0 3 7 00
11 11 Column 0 4 8 00
12 12 ResultRow 4 5 0 00
13 13 Next 0 5 0 01
14 14 Close 0 0 0 00



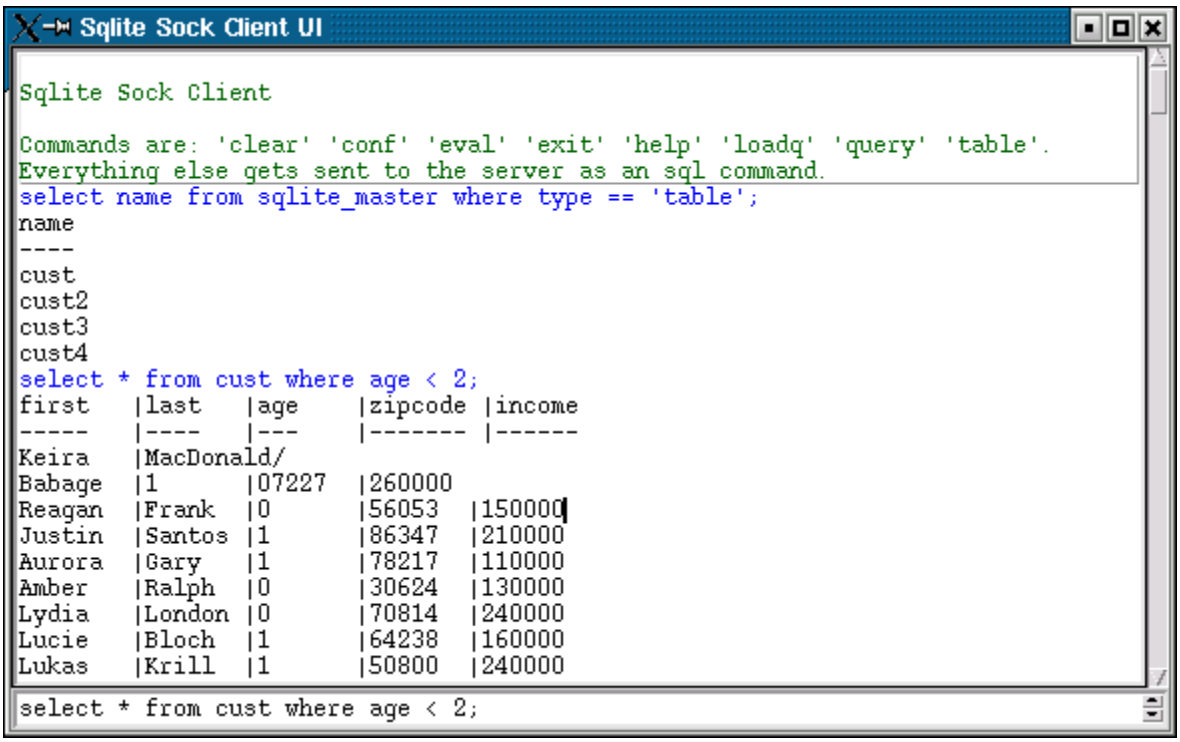

## 14.1 Sqlite Server

Gsqlite can run **SOS** in sqlite server mode. This is launched from the File menu.




## 14.2 Sqlite Client

Gsqlite can run **SOS** in sqlite client mode. This is launched from the File menu.




---

## 15. ProgressBar

ProgressBar demonstrates simple extension tags in **Gui**. ProgressBar is implemented using a canvas, either via an attribute to Canvas, or the ProgressBar tag, eg.

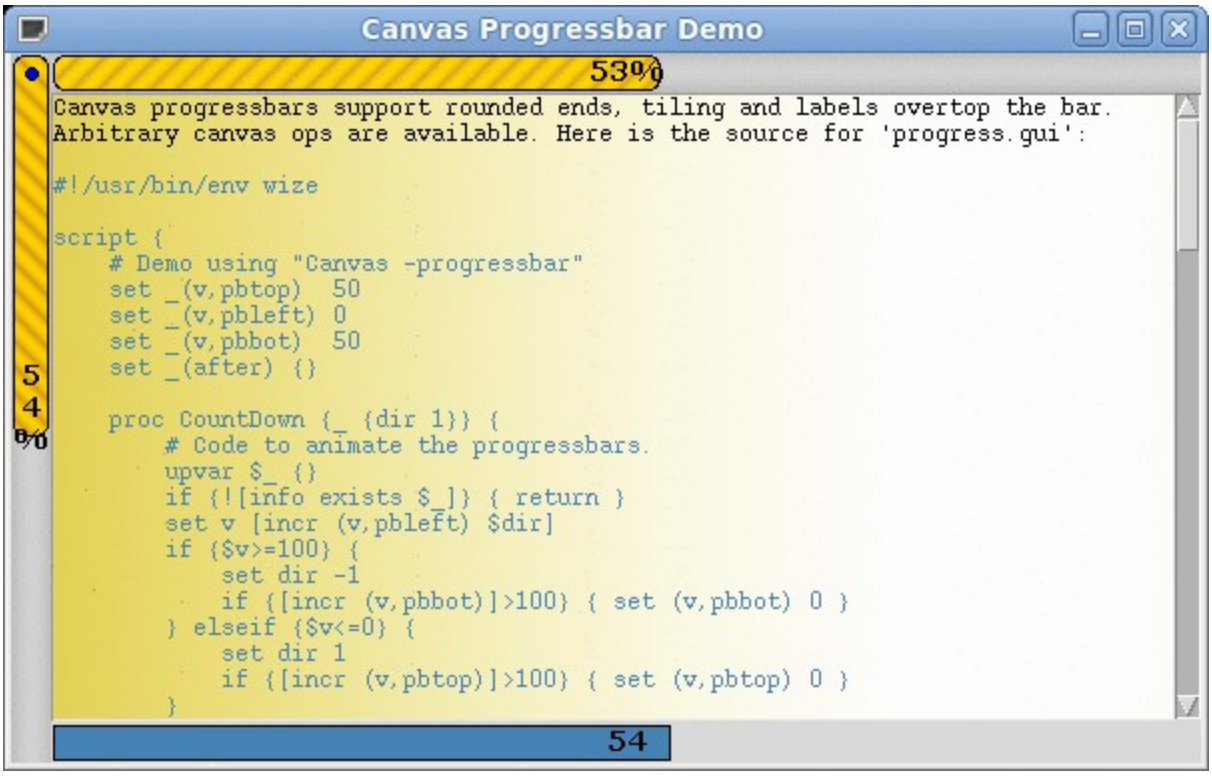
```
{Canvas - -pos _ -progressbar 1} {}
{ProgressBar} {}
```

Note that this is much easier than defining a mega-widget as it does not require a defining programmer methods such as configure, cget, etc.

### 15.1 Example

In this example we use Gui to define some progress bars, where:

- The left hand bar races up and down.
- The left hand bar tile image pulses.
- Mouse over the blue dot turns it red.
- Clicking the dot pauses/resumes.




## 15.2 Demo Source

Here is the source for the demo. See the gui/extattrs.tcl source for the implementation.

```
Canvas progressbars support rounded ends, tiling and labels overtop the bar.
Arbitrary canvas ops are available. Here is the source for 'progress.gui':

#!/usr/bin/env wize

script {
    # Demo using "Canvas -progressbar"
    set _(v,pbtop)  50
    set _(v,pbleft) 0
    set _(v,pbbot)  50
    set _(after) {}

    proc CountDown {_ {dir 1}} {
        # Code to animate the progressbars.
        upvar $_ {}
        if {![info exists $_]} { return }
        set v [incr (v,pbleft) $dir]
        if {$v>=100} {
            set dir -1
            if {[incr (v,pbbot)]>100} { set (v,pbbot) 0 }
```

```
        } elseif {$v<=0} {
            set dir 1
            if {[incr (v,pbtop)]>100} { set (v,pbtop) 0 }
        }
        set (after) [after 30 [list [namespace current]::CountDown $_ $dir]]
    }

    proc StartStop {_} {
        # Start/stop countdown.
        upvar $_ {}
        if {$(after) != {}} {
            after cancel $(after)
            set (after) {}
            return
        }
        CountDown $_
    }

    proc Main {_} {
        # Program entry point.
        upvar $_ {}
        variable pd
        Text insert $(w,text_1) end "Canvas progressbars support rounded ends, tiling
and labels overtop the bar.\n"
        Text insert $(w,text_1) end "Arbitrary canvas ops are available. Here is the
source for 'progress.gui':\n\n"
        Text insert $(w,text_1) end $pd(gui) code
        set c $(w,pbleft)
        # Create a round button to reset start/stop.
        Canvas create oval $c {7 7 13 13} -fill Blue -width 1 -outline Black -tags o
        $c bind o <Enter> "$c itemconf 3 -fill red; $c conf -cursor hand2"
        $c bind o <Leave> "$c itemconf 3 -fill blue; $c conf -cursor {}"
        $c bind o <1> "$_ StartStop"
        CountDown $_
    }

    proc Cleanup {_} {
        # Program cleanup.
        upvar $_ {}
        *catch {after cancel $(after)}
        exit; # Exit cause this is just a demo.
    }

}

style {
    # "Style overrides for -progressbar attrs: creates image tiles, rounded, etc"
```

```
    Toplevel {
        @defgradients {
            slan  {#daa520 #ffd700 -width 13 -height 13 -slant 1.0}
            slanp {#daa520 #ffd700 -width 13 -height 13 -slant 1.0 -rotate 90}
            chal1 {#bebebe #d3d3d3 -width 20 -height 15 -rotate 90}
            chal2 {#bebebe #d3d3d3 -width 20 -height 15}
            tbg { Khaki #ffffff -width 1000 -height 6 -gamma .5}
        }
        @deffonts {
            bfnt {Verdana -14 bold}
        }

        @imgpulse { slanp }
        *highlightThickness 0
    }
    Text { -tile ^tbg -padx 0 -pady 0 @tags {code {-foreground SteelBlue} } }
    .pbtop {
        -tile ^chal1
        @@ { -progressbar {-bartile ^slan  -font ^bfnt -round 1 -suffix %}}
    }
    .pbleft {
        -tile ^chal2
        @@ { -progressbar {-bartile ^slanp -font ^bfnt -round 1 -suffix % -vertical
1 }}
    }
}

{Toplevel + -title "Canvas Progressbar Demo"} {
    {Canvas - -id pbleft -pos |l -progressbar {-vertical 1}} {}
    {Frame + -pos *l} {
        {Canvas - -id pbtop -pos _ -progressbar 1} {}
        {Text - -pos * -scroll *} {}
        {ProgressBar - -id pbbot -font ^bfnt -pos _} {}
    }
}
```

---

## 16. Running Wize

Wize offer a lot of flexibility for packaging and running application scripts, eg.

```
wize               ; # Run interactive shell (then type: console show)
wize file.tcl       ; # Run a script
wize file.gui        ; # Run gui script
wize file.zip        ; # Mount zip file and run main.tcl/.gui
wize file.zip:       ; # Mount zip file and browse for script.
```

```
wize file.zip:x.tcl    ; # Mount zip file and run x.tcl
wize file.so           ; # Load dll, then mount and run main.tcl
wize file.so:          ; # Load dll, mount, and browse,
```

Wize treats any .zip/.so file as a **wizapp**. ie. it looks for main.tcl in the top directory (or single subdirectory). Alternatively, a .tcl or .gui file of the same prefix as the .zip file will be used. If found, it is executed.

To browse instead, just append a colon.

## 16.1 Relayed Links to Wize

A wize executable can use a file link to run a .zip file indirectly. For example, suppose you've developed a Tcl application in the subdirectory foo (and it contains a foo/main.tcl). And assume that wize is located in ~/bin. You can create a new foo command using:

```
zip -r ~/bin/foo.zip foo/
ln -s ~/bin/wize ~/bin/foo
```

See **Admin** if you don't have zip on your system, or can't use ln (eg. on Windows).

## 16.2 Command-line Eval

Tcl can be evaluated from the command-line via:

```
wize /zvfs/wiz/eval.tcl 'pack [button .b -text Hello-World]'
```

Wize can also for run applications via http, eg:

```
wize http://pdqi.com/w/Download/hangman.zip
```

Note, this will download hangman.zip to the curent directory and then runs it.

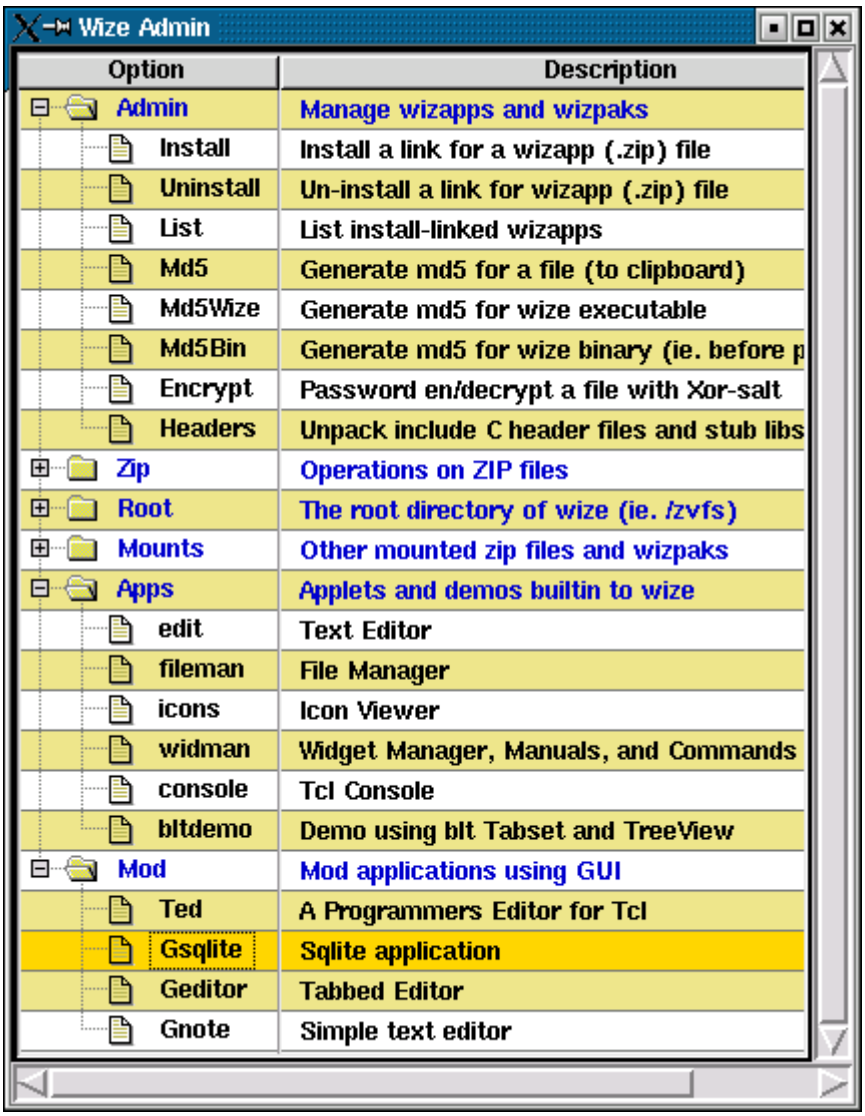
---

# 17. Wize Admin

Wize comes with a builtin administrative interface invocable from the command-line via:

```
wize /
```

Here is a screenshot:




The admin interface gives access to many of the features and applications within Wize, further described below.
You can also run many Admin commands directly from the command-line, eg:

```
wize / Zip/Unzip foo.zip dstdir/
```

## 17.1 Admin

The Admin entry gives access to commands for installing, listing and verifying wize components.

## 17.2 Zip

The Zip entry gives access to commands for managing .zip files.

## 17.3 Root

The Root entry displays the wize builtin filesystem.

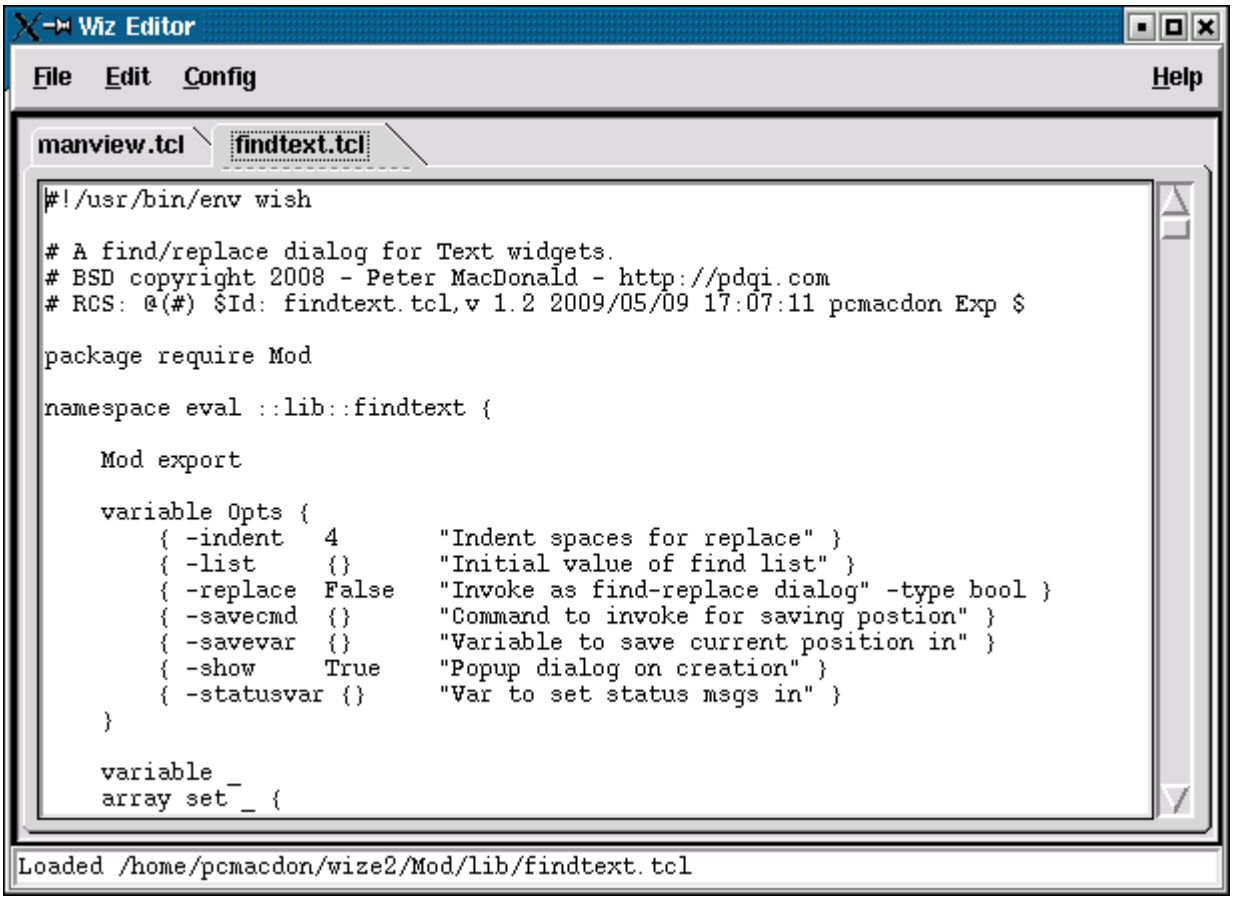
## 17.4 Mounts

Mounts shows all mounted wizpaks, as well as any .zip files manually mounted via Zip/Mount.

## 17.5 Apps

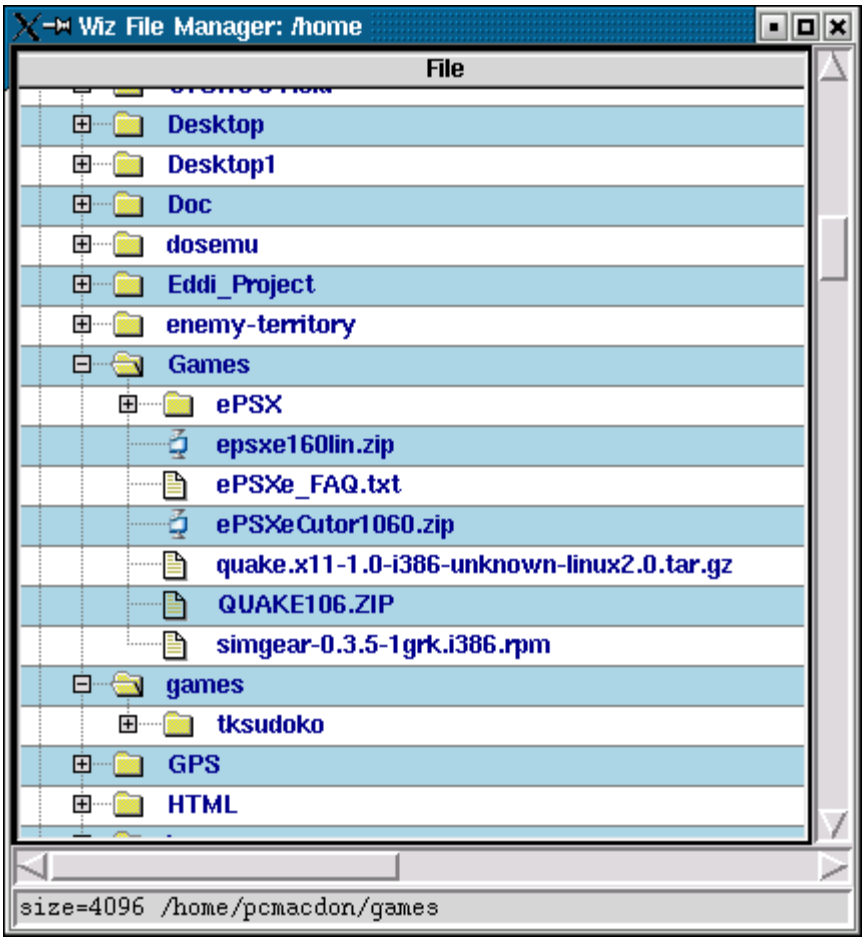
The Apps entry contains a number of builtin example applications for Wize that you can run and examine. Source for these can be browsed from "wize /" or from CVS: **http://wize.cvs.sourceforge.net/viewvc/wize/wize2/Mod/wiz**

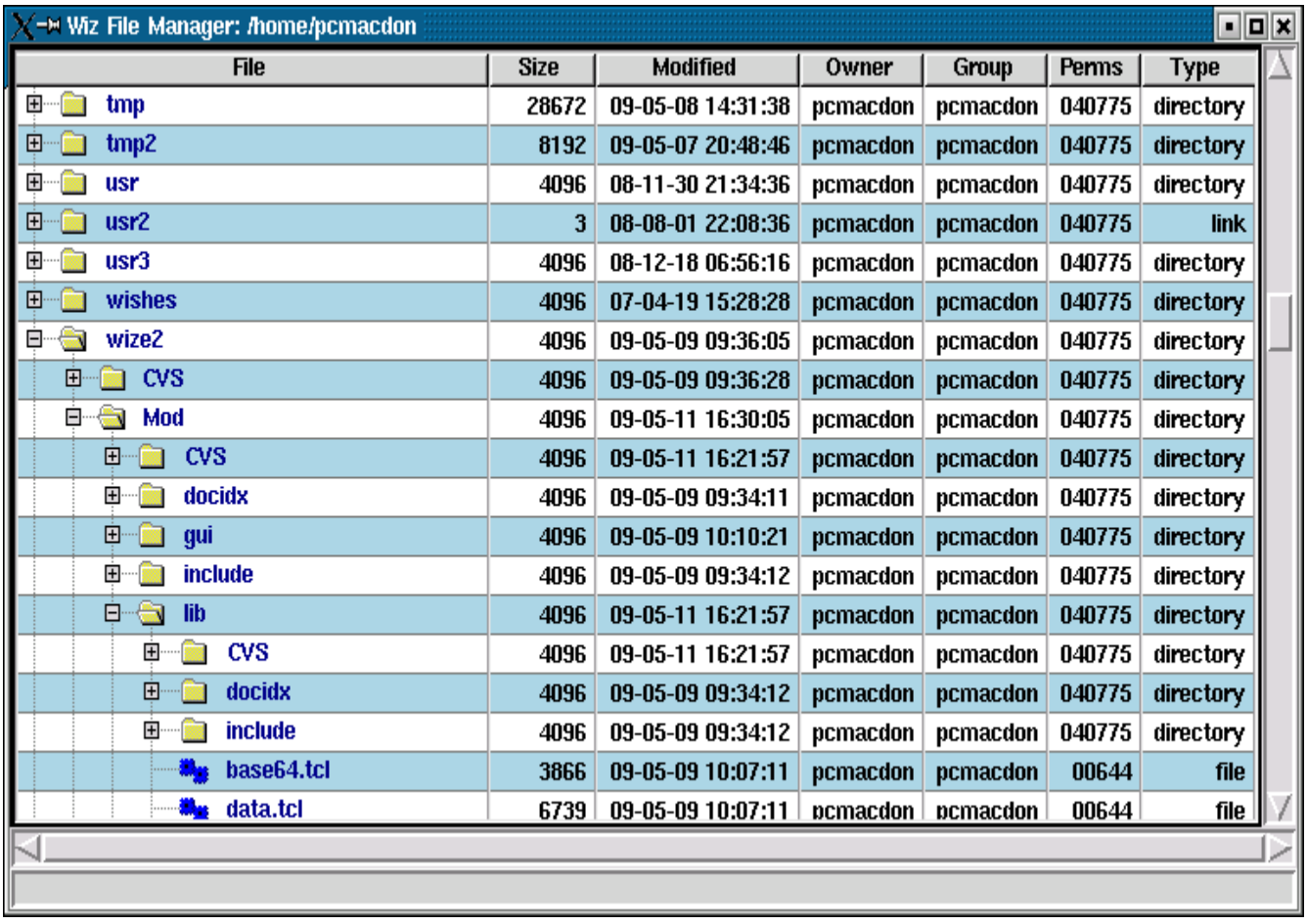
Edit is a very simple editor. eg.

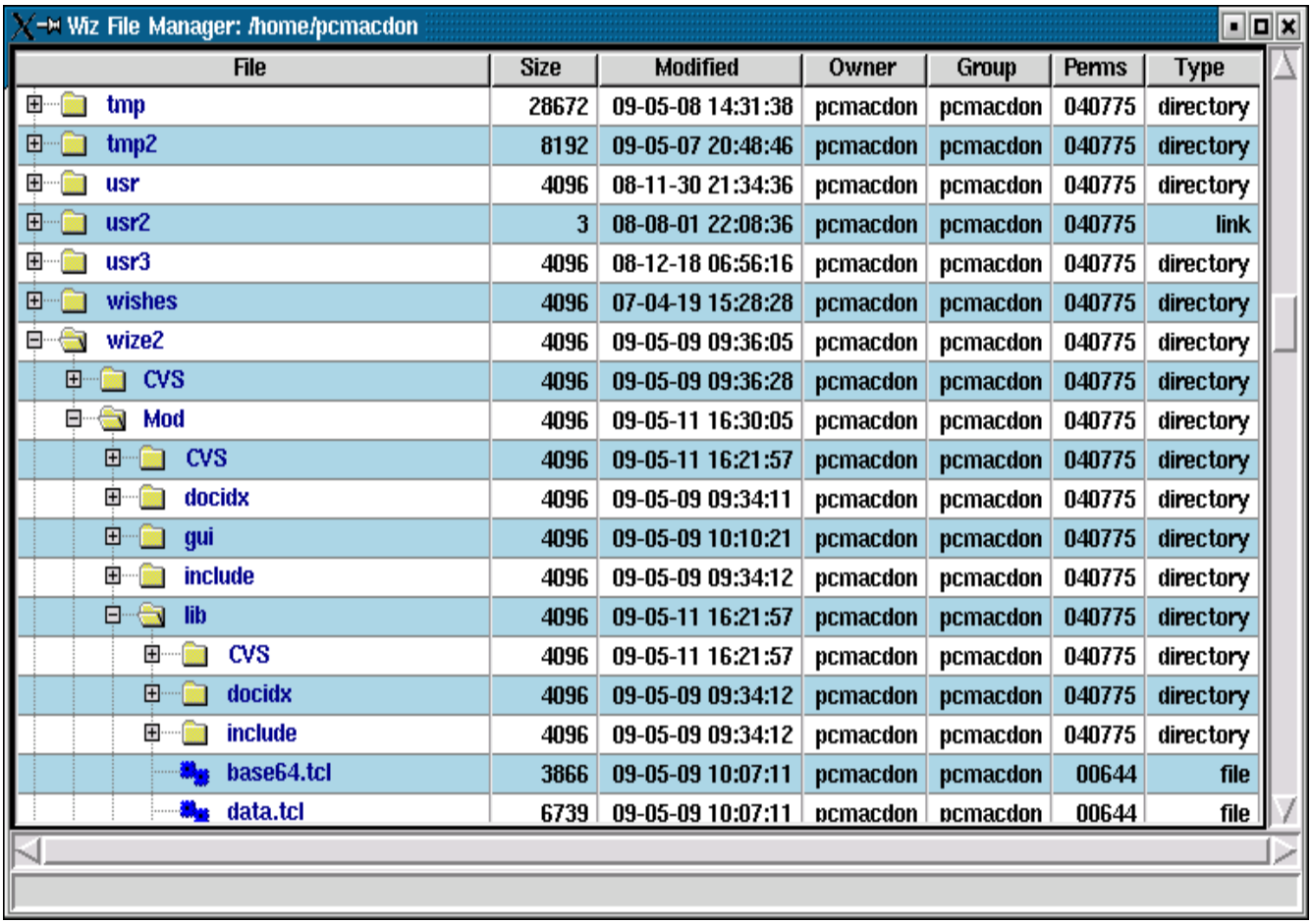


Icons is an icon/image browser. eg.

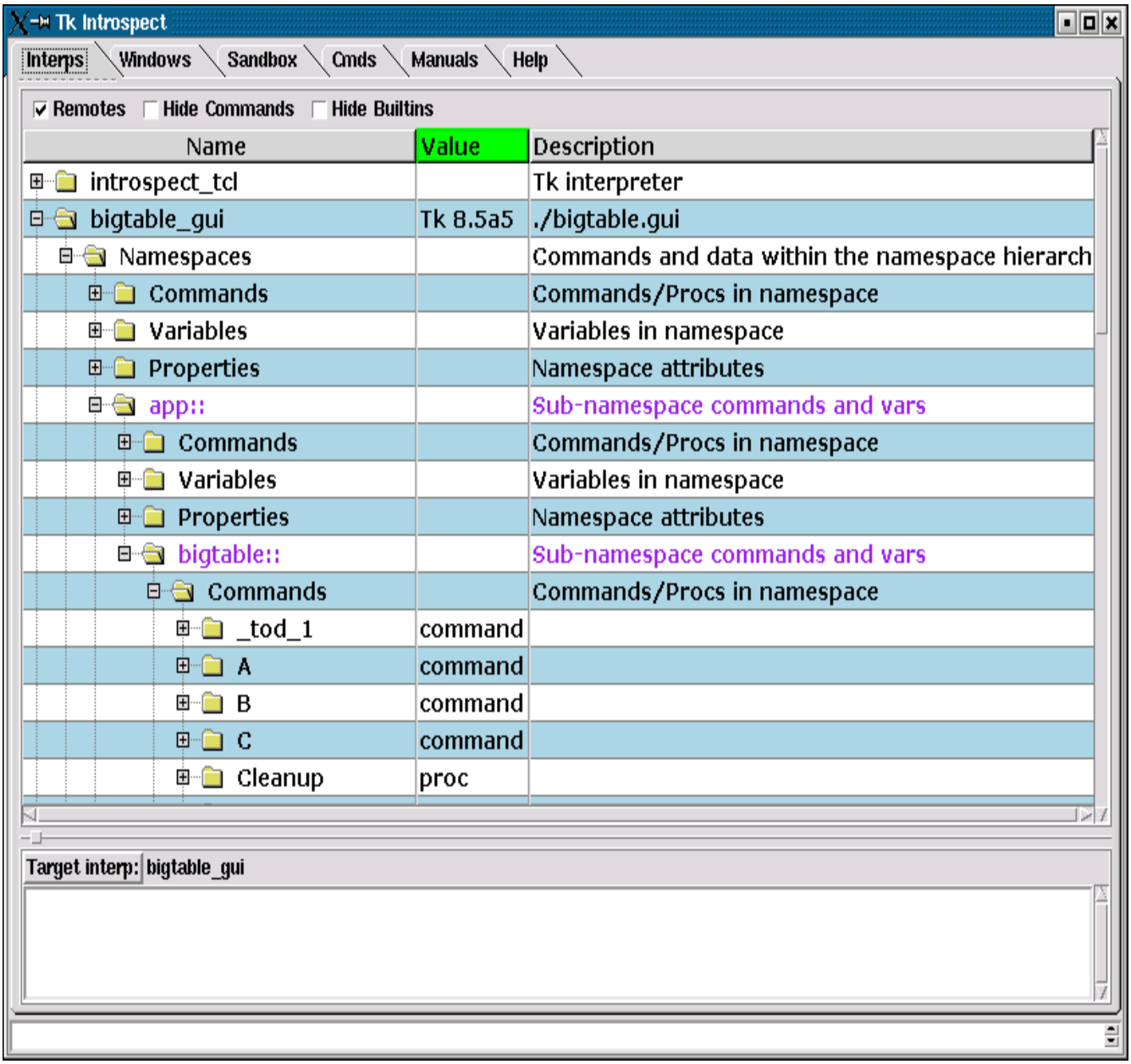
Fileman is simple file manager. eg.


Wiz File Manager: /home
File
Desktop
Desktop1
Doc
dosemu
Eddi_Project
enemy-territory
Games
ePSX
epsxe160lin.zip
ePSXe_FAQ.txt
ePSXeCutor1060.zip
quake.x11-1.0-i386-unknown-linux2.0.tar.gz
QUAKE106.ZIP
simgear-0.3.5-1grk.i386.rpm
games
tksudoko
GPS
HTML
size=4096 /home/pcmacdon/games


Wiz File Manager: /home/pcmacdon
File
Size
Modified
Owner
Group
Perms
Type
tmp 28672 09-05-08 14:31:38 pcmacdon pcmacdon 040775 directory
tmp2 8192 09-05-07 20:48:46 pcmacdon pcmacdon 040775 directory
usr 4096 08-11-30 21:34:36 pcmacdon pcmacdon 040775 directory
usr2 3 08-08-01 22:08:36 pcmacdon pcmacdon 040775 link
usr3 4096 08-12-18 06:56:16 pcmacdon pcmacdon 040775 directory
wishes 4096 07-04-19 15:28:28 pcmacdon pcmacdon 040775 directory
wize2 4096 09-05-09 09:36:05 pcmacdon pcmacdon 040775 directory
CVS 4096 09-05-09 09:36:28 pcmacdon pcmacdon 040775 directory
Mod 4096 09-05-11 16:30:05 pcmacdon pcmacdon 040775 directory
CVS 4096 09-05-11 16:21:57 pcmacdon pcmacdon 040775 directory
docidx 4096 09-05-09 09:34:11 pcmacdon pcmacdon 040775 directory
gui 4096 09-05-09 10:10:21 pcmacdon pcmacdon 040775 directory
include 4096 09-05-09 09:34:12 pcmacdon pcmacdon 040775 directory
lib 4096 09-05-11 16:21:57 pcmacdon pcmacdon 040775 directory
CVS 4096 09-05-11 16:21:57 pcmacdon pcmacdon 040775 directory
docidx 4096 09-05-09 09:34:12 pcmacdon pcmacdon 040775 directory
include 4096 09-05-09 09:34:12 pcmacdon pcmacdon 040775 directory
base64.tcl 3866 09-05-09 10:07:11 pcmacdon pcmacdon 00644 file
data.tcl 6739 09-05-09 10:07:11 pcmacdon pcmacdon 00644 file




**Introspect** is a widget manager and command browser. eg.




Console invokes the Tk console. Works even on Unix.

## 17.6 Gui (or Mod)

Mod is a package used when writing complex and sophisticated Tk applications.
The Admin entry Gui contains a number of example applications that use Gui from the Mod package (included in wizmod.zip), these include:

**Gsqlite** is an Sqlite client written using **Gui**

**Geditor** is an editor written using **Gui**

**Ted** is a programming editor designed to simplify Tcl development in Wize. In particular, the command completion feature can greatly simplify writing Tcl (and Tk), particularly for those who might not know the language that well.
(Note: ted is in wizapp.zip and is not part of wizmod.zip)

---

# 18. Exploring Wize Applications

Examining or exploring a Wize application is easy. Just use the key/mouse sequence:

```
<Control-Alt-Shift-2>
```

This opens the window config-editor, allowing you to explore various aspects of the running program.

## 18.1 A Sample Session

What follows is a sample session that explores the **bigtable.gui** demo.

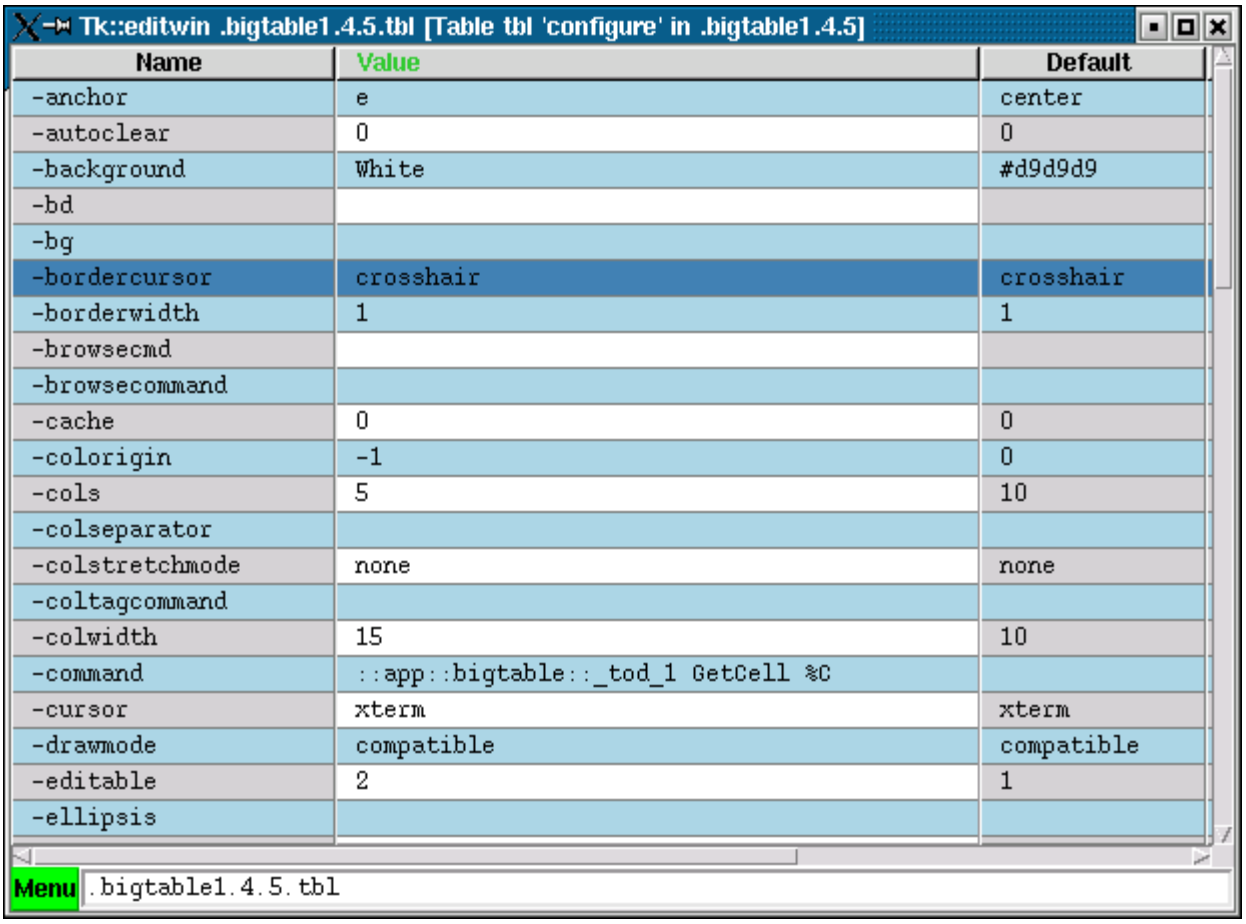
```
% wize -Wall bigtable.gui
```

or use:

```
% wize -Wall / Gui/Bigtable
```

## 18.2 Widget Configuration

When you bring up the window editor using mouse/key sequence <Control-Alt-Shift-2>, [Tk::editwin] is invoked:




You can then double-click on the Value column to edit widget values.
You may also try the following:

- open the **console** from the Menu.
- in the console, type set t, space, then paste with <Control-v>.

Now try some commands in the console, eg: $t conf -bd 4. (Note, this works because the widget is selected upon open)

## 18.3 The Menu

Hit <F10> or click on Menu in [Tk::editwin] to bring up the menu:


This menu has numerous facilities which are discussed in detail below.

## 18.4 Edit Source

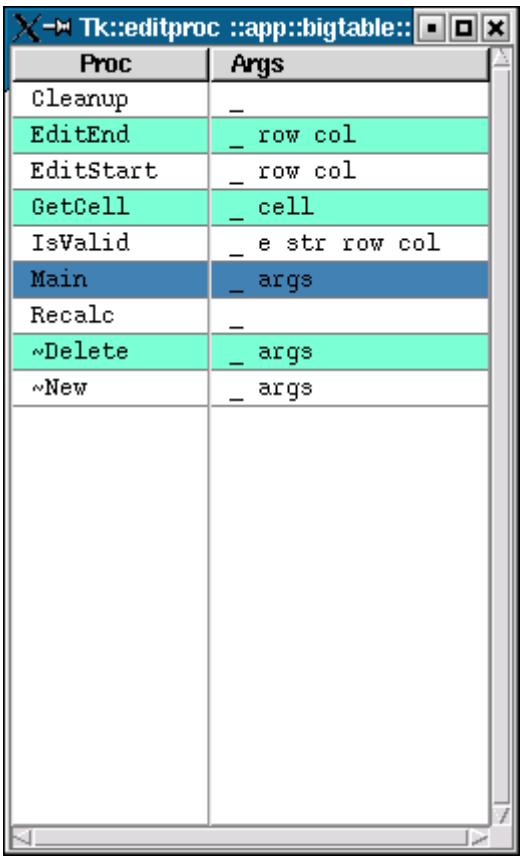
Select the menu entry Edit Source: bigtable.gui.

This will bring up the source in **Geditor**.

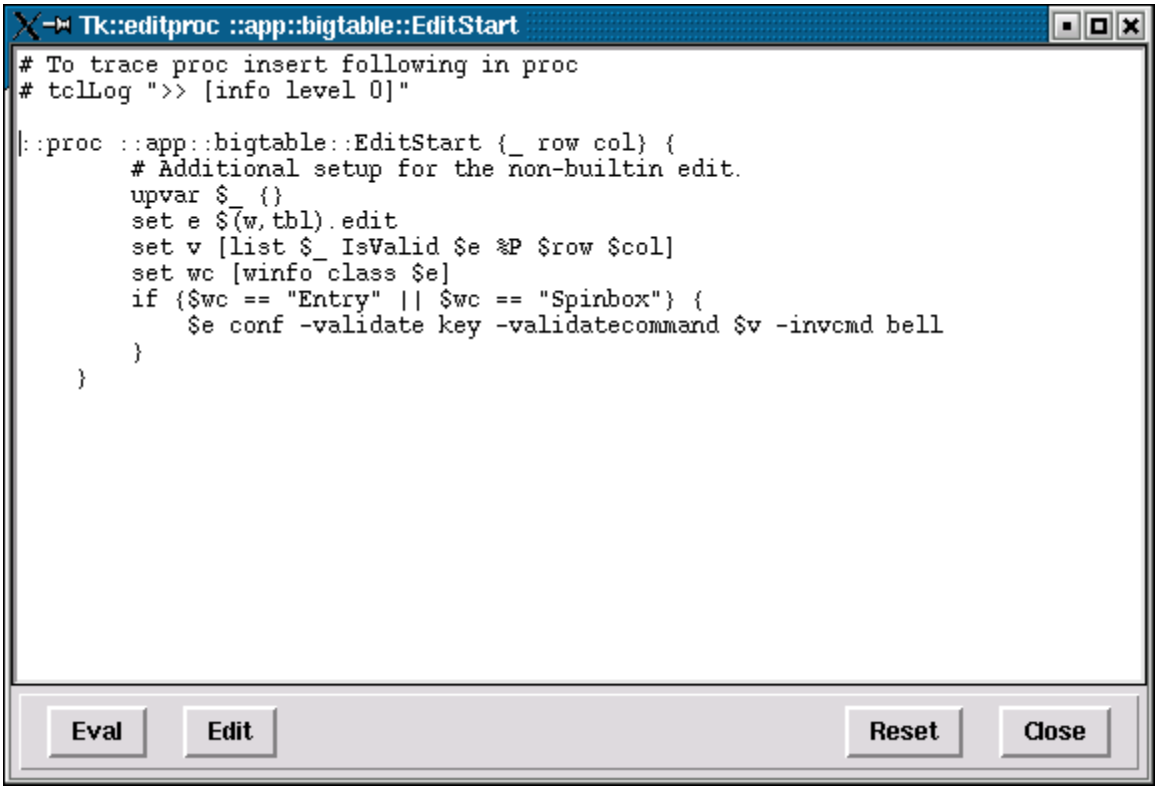
## 18.5 Procs

The Procs menu entry will invoke [Tk::editproc] to allow you to edit code dynamically, right in the running program.

From the menu, select 'Procs in ::app::bigtable:




Then select EditStart and hit Enter (or double click):

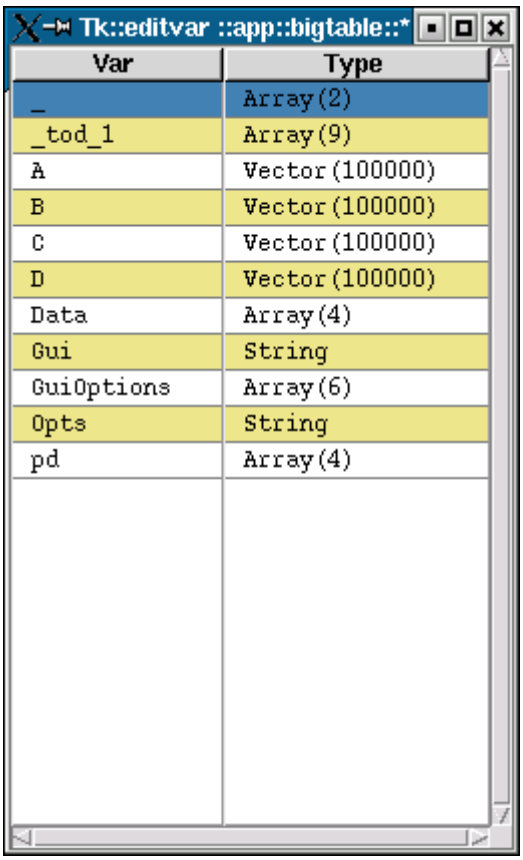



Insert a line of code in the proc, eg. puts "Editing: $row" and click Eval.
Now go back to bigtable and edit a cell by double clicking. Note how your output appears in the xterm.
Alternatively, you can open the console window and use putc.

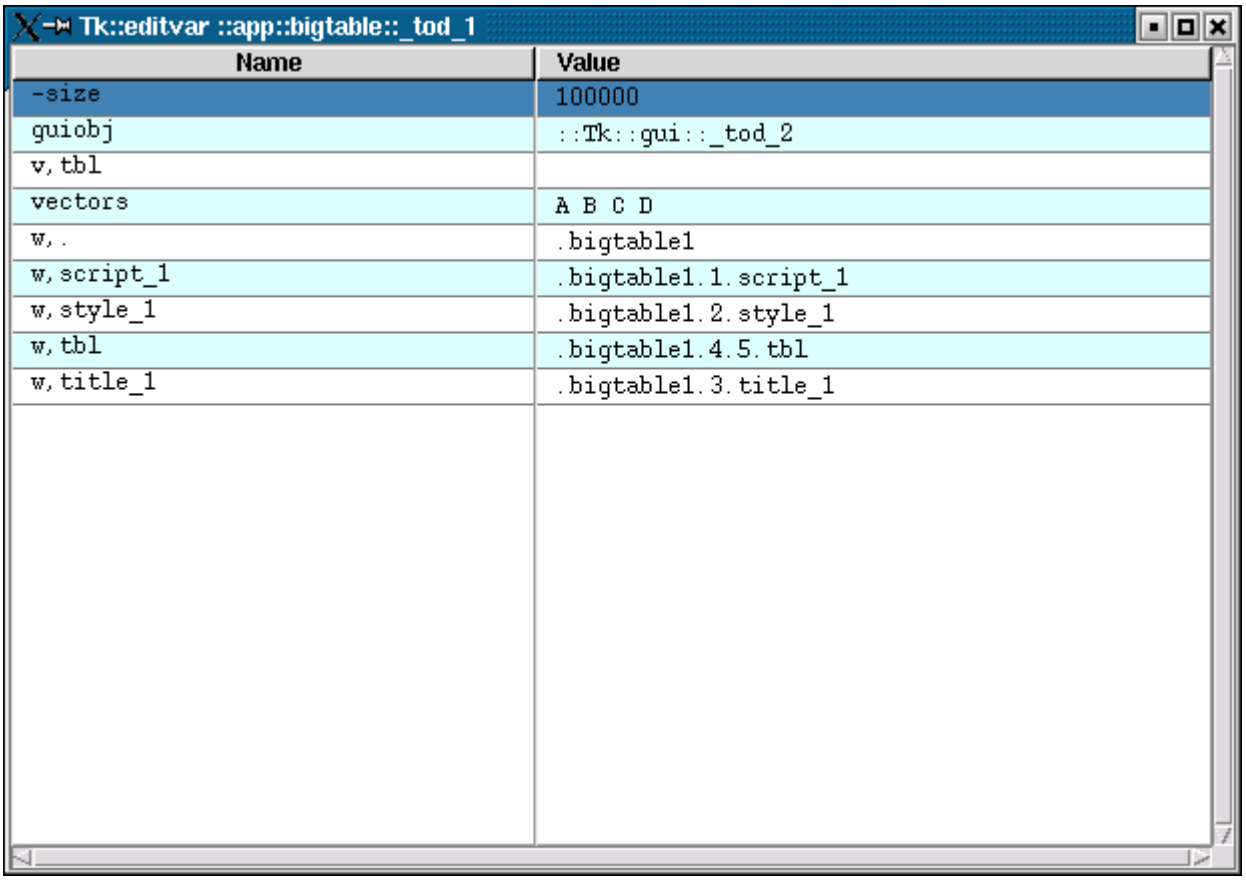
## 18.6 Variables

The Vars menu entry invokes [Tk::editvar] letting you examine and change variables in the running program.

When you select 'Vars in ::app::bigtable you will see:




Then select _tod_1 and hit Enter:

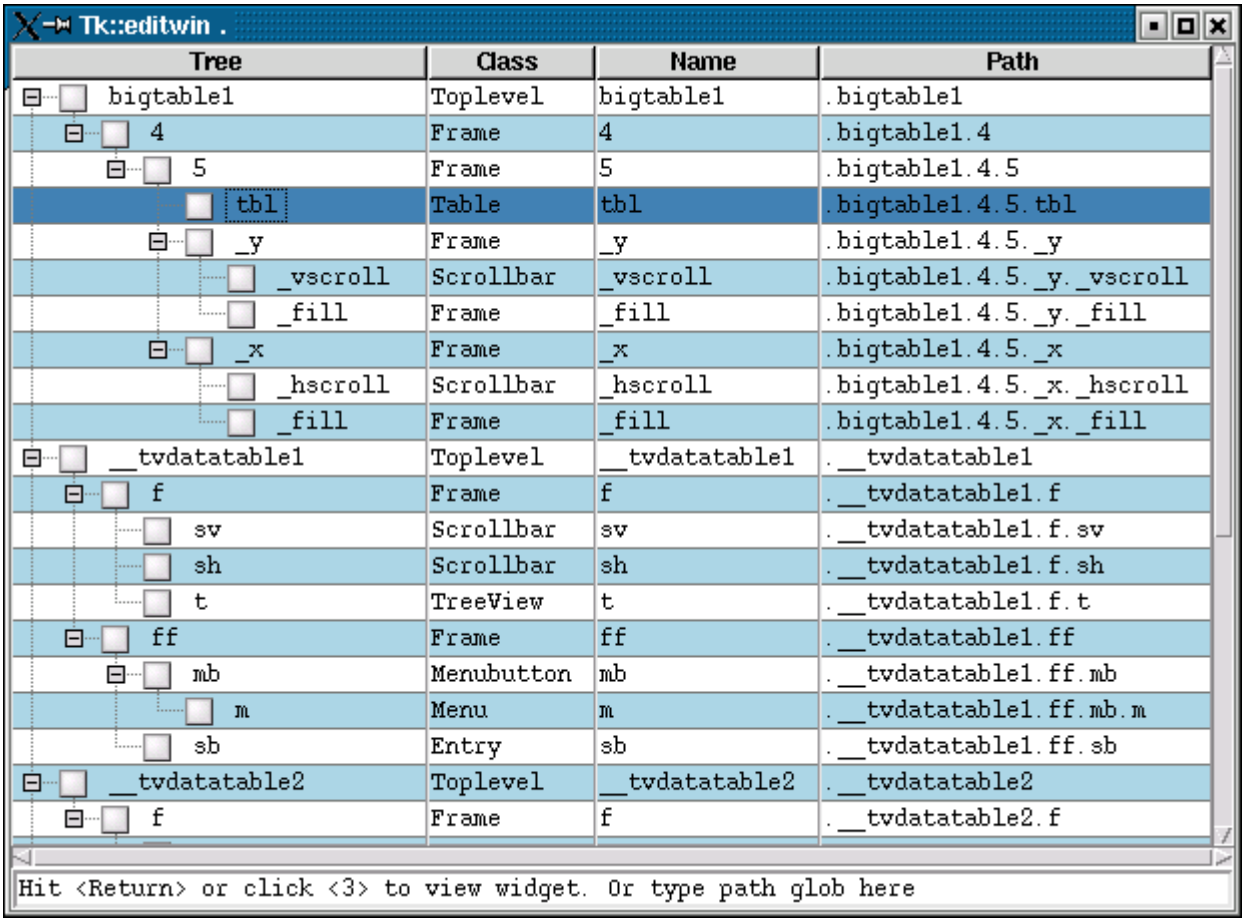



Double click or hit enter to change any variable.

## 18.7 Window Tree

Selecting Window Tree from the Menu will invoke:

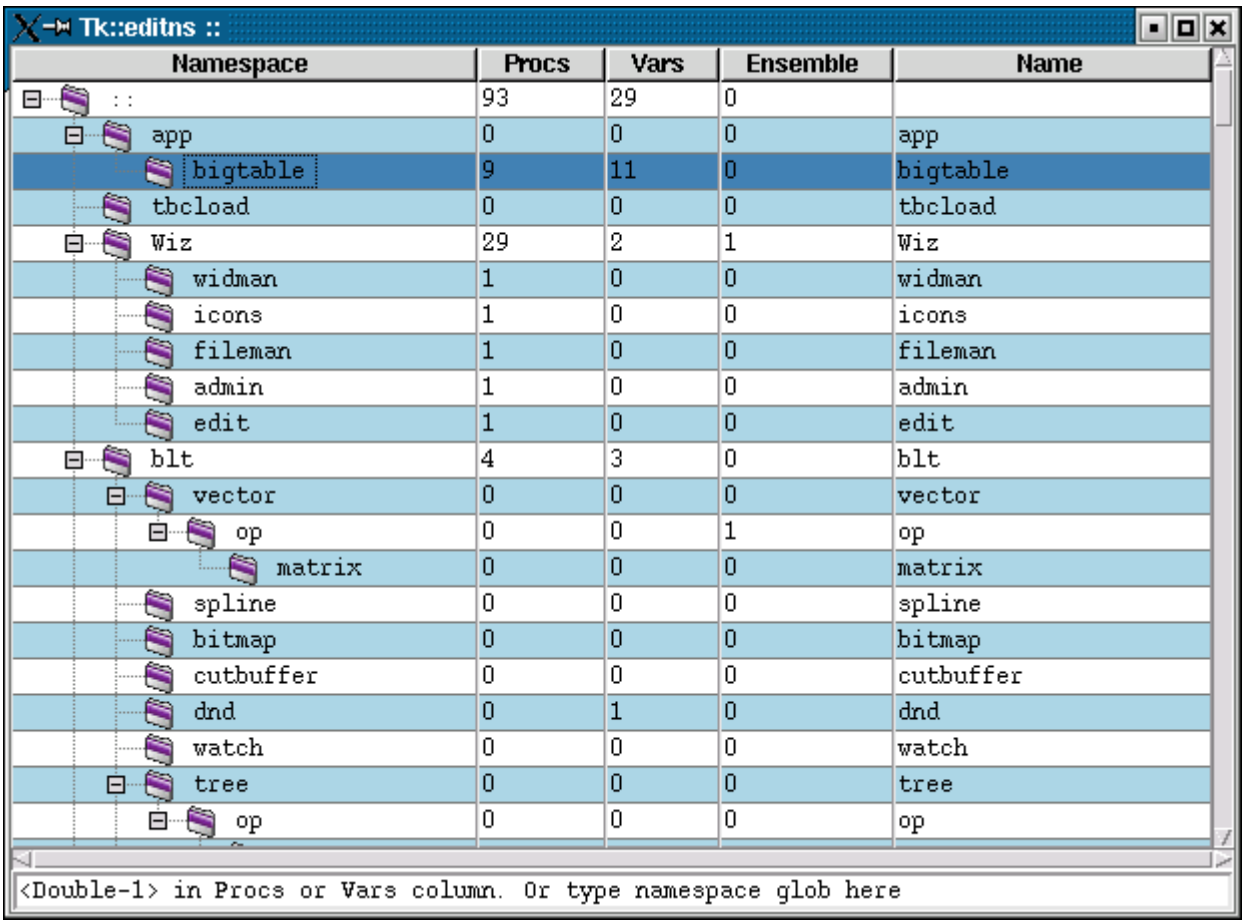
```
Tk::editwin .*
```




The current window will be selected. You can then select a different window and hit enter to edit it.

## 18.8 Namespace Tree

Selecting Namespace Tree from the Menu will invoke [Tk::editns ::*] letting you browse the namespace tree, eg:




By default the current namespace is selected. You can then double-click on the Procs column to edit procs, or the Varscolumn to edit variables (in that namespace).

## 18.9 Introspect

**Introspect** is a graphical application for examining and modifying the program state of Tk applications.

The default is ot start introspect inside the application process you are debugging. The exec option however starts introspect as an external process.

## 18.10 Edit

Edit invokes the editor on the runtime file. See Start Ted.

## 18.11 Start Ted

**Ted** is a programming editor with builtin command completion for Tcl/Tk.
If ted is running when you invoke Edit, then the file will be edited in Ted. Otherwise, the builtin editor will be used.

## 18.12 Admin

**Admin** is the administrative interface into Wize.

## 18.13 The Status Line

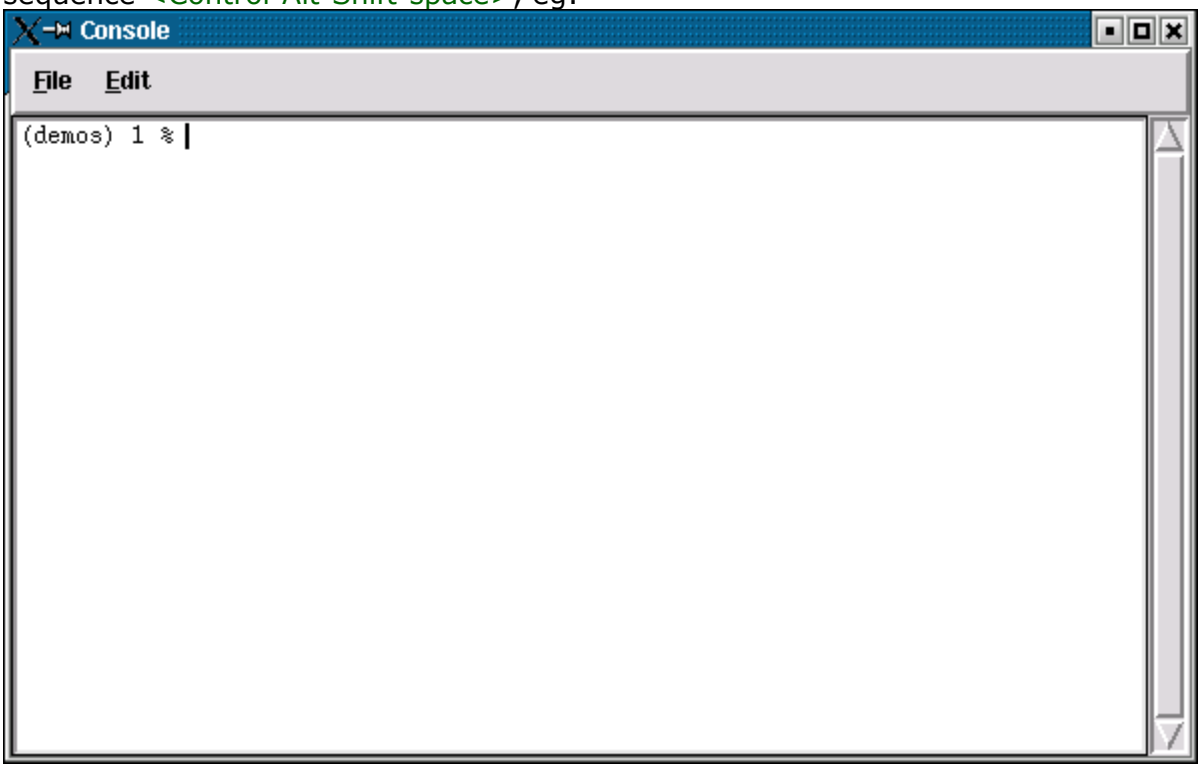
Arguments can be appended to the path in the **editwin** status line to be evaluated, eg:

- add space, style names and hit enter.
- change above to style conf alt and hit enter.

Note, if the last or second last argument matches pattern *conf then 'Tk::editwin'' is invoked to edit widget items.
Otherwise, the results are displayed in a popup.

## 18.14 Console

A console can be opened from the editwin menu or by typing the key sequence <Control-Alt-Shift-space>, eg:




Any Tcl command can be typed into the console. However, the following are most useful:

- **Tk::find** - search for window by name or class.
- **Tk::editproc** - edit running procs
- **Tk::editvar** - visual variable editor
- **Tk::editwin** - visual widget editor/browser

- **Tk::editns** - visual namespace browser.

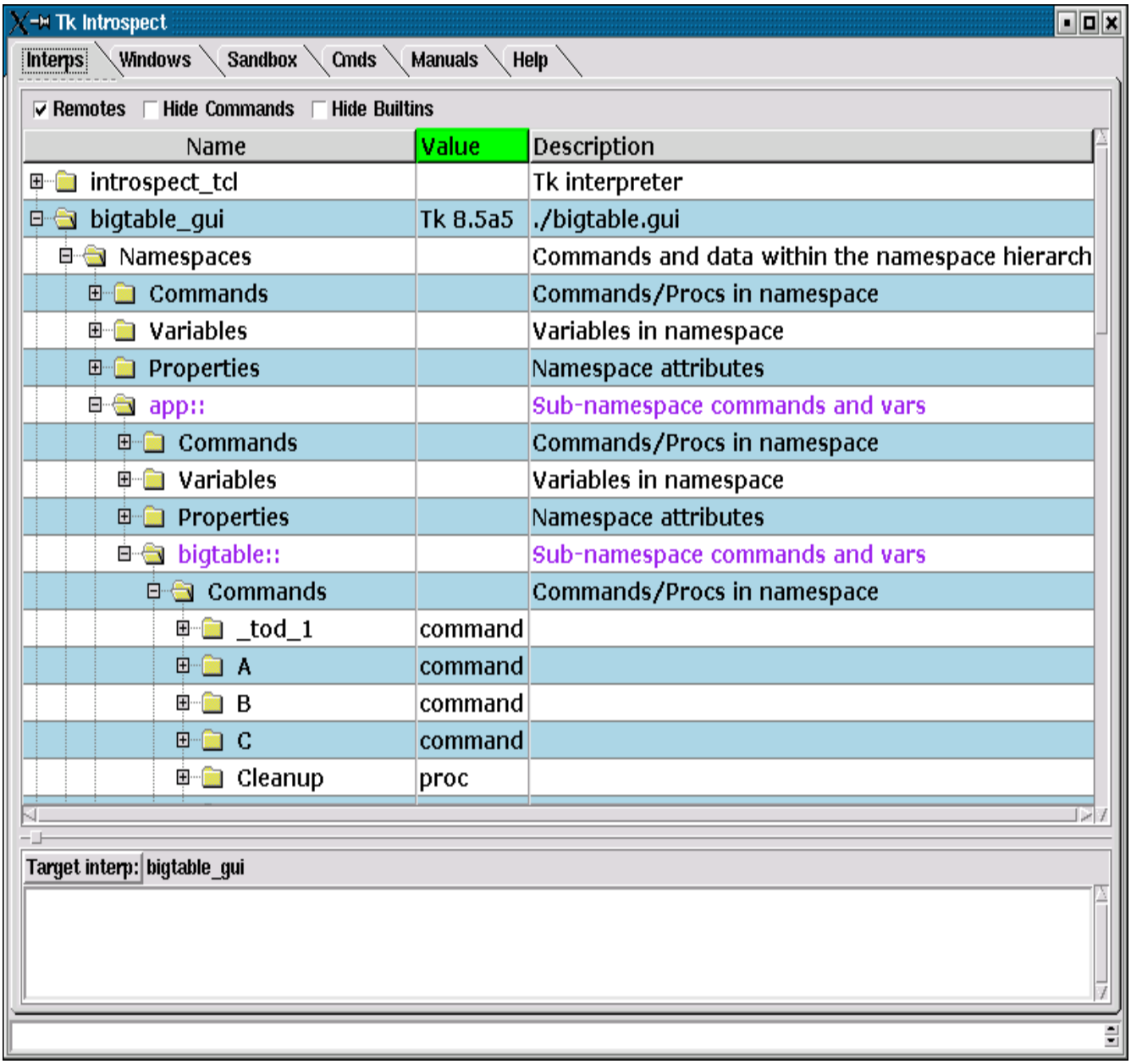
---

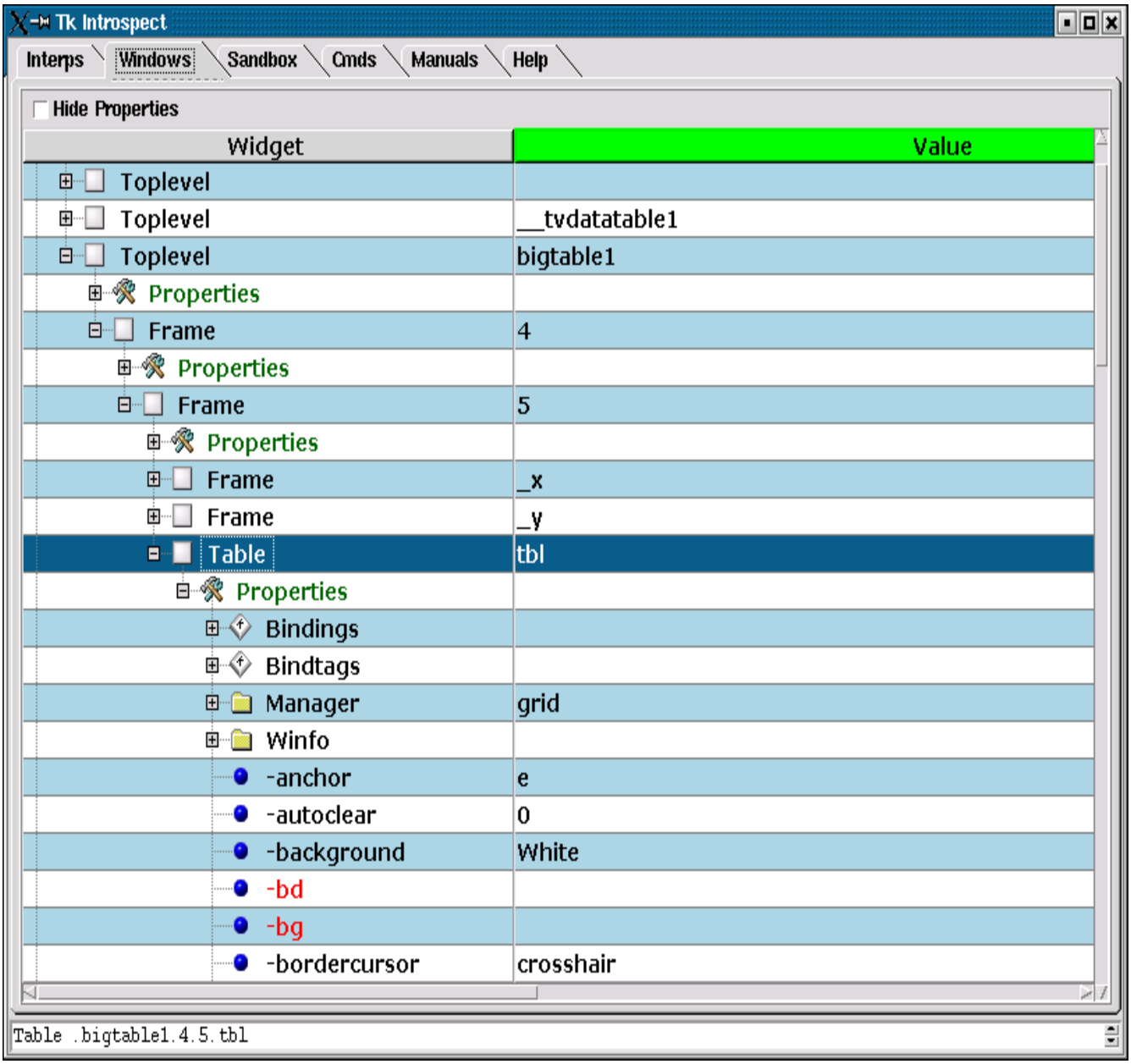
## 19. Introspect

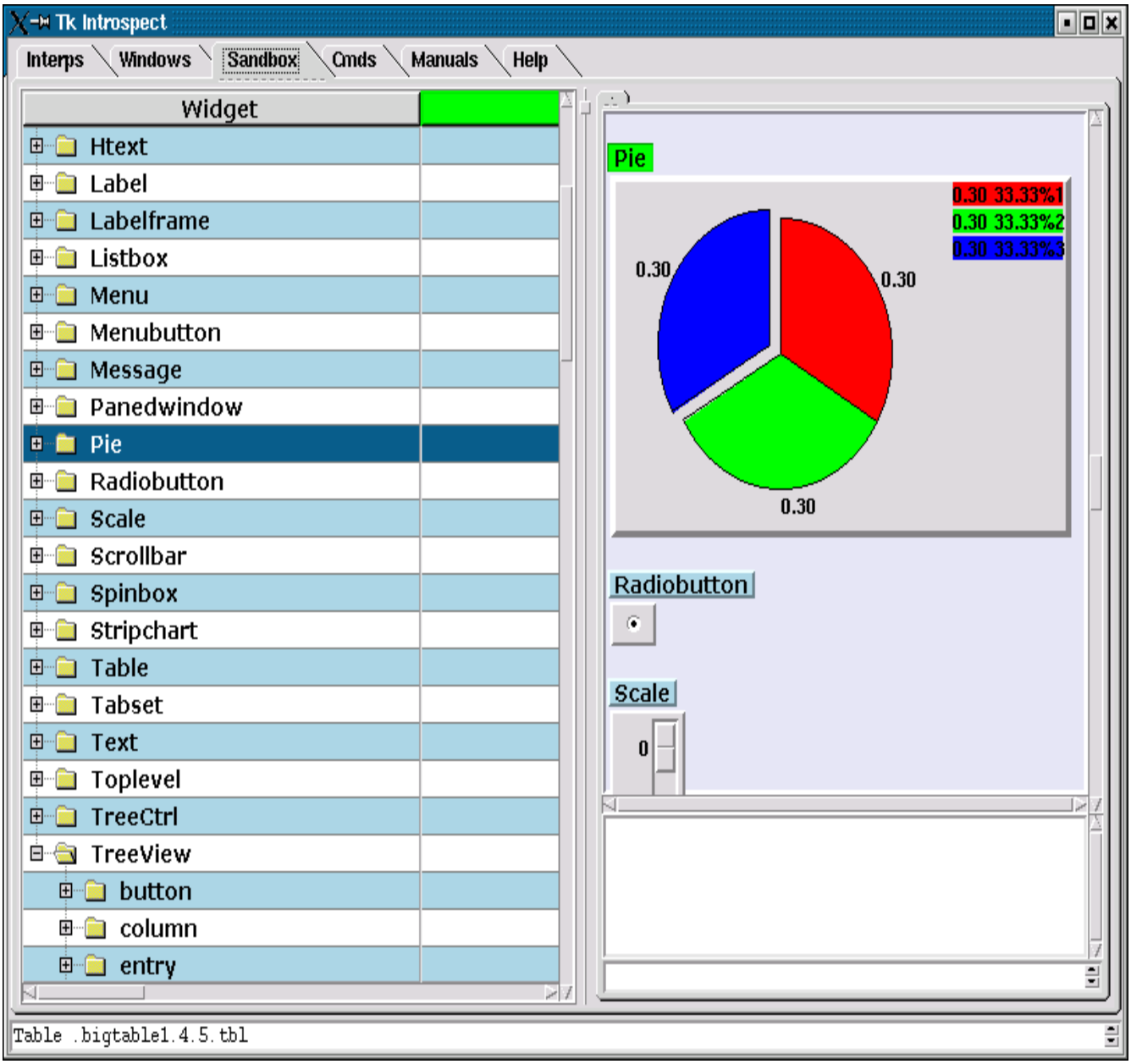
Introspect is a graphical application for examining and modifying the application state of other programs viasend/dde. It can be invoked in an application via <Control-Alt-Shift-2> or run directly using:

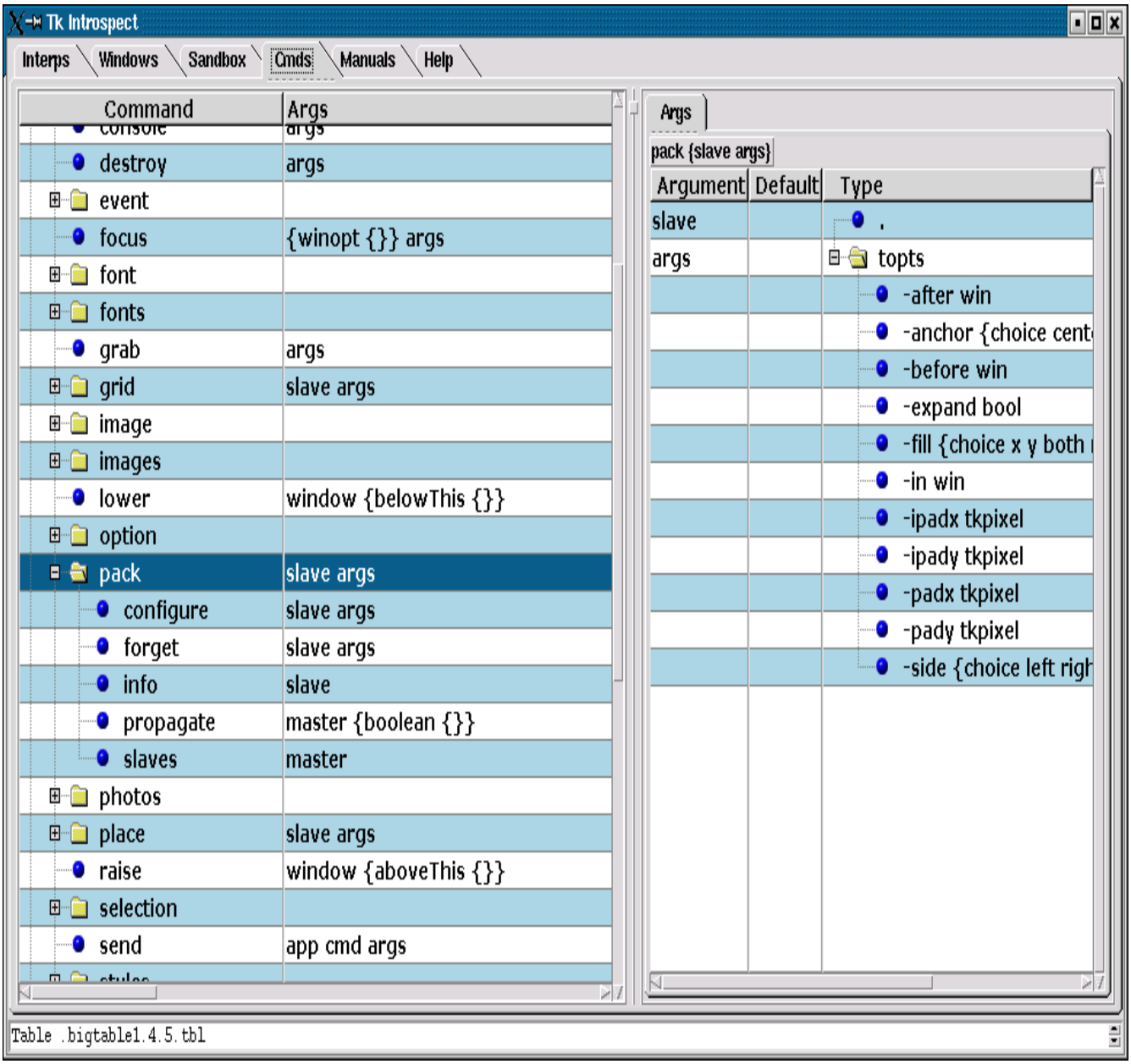
```
wize /zvfs/wiz/introspect.tcl
```

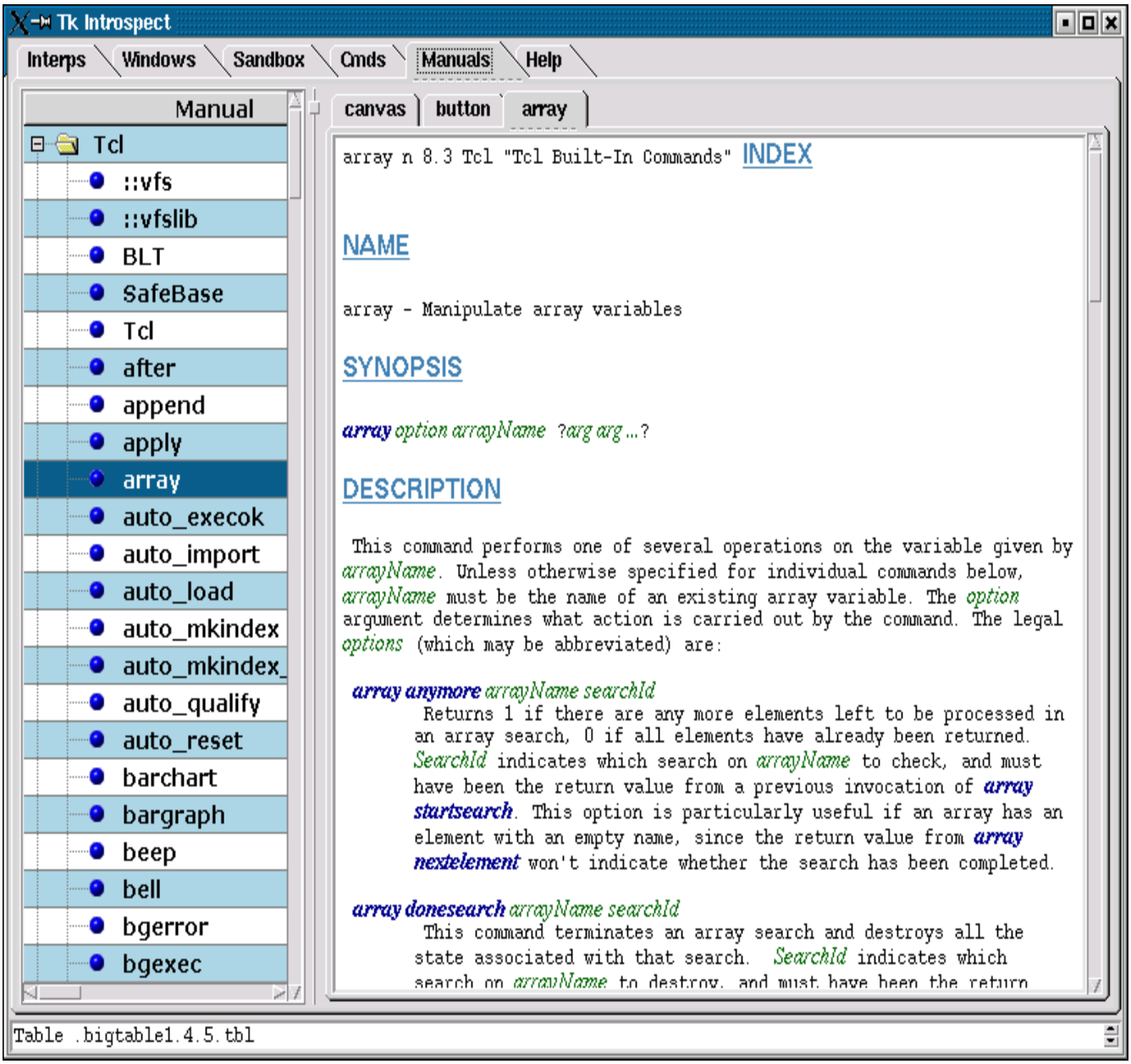
Introspect uses a TreeView to display resources such as Procs, Vars, Widgets, Fonts, etc. It contains a Sandbox environment to let you experiment with widgets/elements, the command/option hierarcy of all builtin commands, and access to all online documention.
Here are screenshots of the Introspect tabs:


Tk Introspect
Interps
Windows
Sandbox
Cmds
Manuals
Help
Remotes
Hide Commands
Hide Builtins
Name
Value
Description
introspect_tcl
Tk interpreter
bigtable_gui
Tk 8.5a5
./bigtable.gui
Namespaces
Commands and data within the namespace hierarch
Commands
Commands/Procs in namespace
Variables
Variables in namespace
Properties
Namespace attributes
app::
Sub-namespace commands and vars
Commands
Commands/Procs in namespace
Variables
Variables in namespace
Properties
Namespace attributes
bigtable::
Sub-namespace commands and vars
Commands
Commands/Procs in namespace
_tod_1
command
A
command
B
command
C
command
Cleanup
proc
Target interp:
bigtable_gui


Tk Introspect
Interps
Windows
Sandbox
Cmds
Manuals
Help
Hide Properties
Widget
Value
Toplevel
Toplevel
__tvdatatable1
Toplevel
bigtable1
Properties
Frame
4
Properties
Frame
5
Properties
Frame
_x
Frame
_y
Table
tbl
Properties
Bindings
Bindtags
Manager
grid
Winfo
-anchor
e
-autoclear
0
-background
White
-bd
-bg
-bordercursor
crosshair
Table .bigtable1.4.5.tbl


Tk Introspect
Interps
Windows
Sandbox
Cmds
Manuals
Help
Widget
Htext
Label
Labelframe
Listbox
Menu
Menubutton
Message
Panedwindow
Pie
Radiobutton
Scale
Scrollbar
Spinbox
Stripchart
Table
Tabset
Text
Toplevel
TreeCtrl
TreeView
button
column
entry
Pie
0.30 33.33%1
0.30 33.33%2
0.30 33.33%3
0.30
0.30
0.30
Radiobutton
Scale
0
Table .bigtable1.4.5.tbl


Tk Introspect
Interps
Windows
Sandbox
Cmds
Manuals
Help
Command
Args
destroy
args
event
focus
{winopt {}} args
font
fonts
grab
args
grid
slave args
image
images
lower
window {belowThis {}}
option
pack
slave args
configure
slave args
forget
slave args
info
slave
propagate
master {boolean {}}
slaves
master
photos
place
slave args
raise
window {aboveThis {}}
selection
send
app cmd args
Args
pack {slave args}
Argument
Default
Type
slave
.
args
topts
-after win
-anchor {choice cent
-before win
-expand bool
-fill {choice x y both
-in win
-ipadx tkpixel
-ipady tkpixel
-padx tkpixel
-pady tkpixel
-side {choice left righ
Table .bigtable1.4.5.tbl




## 19.1 Interps

Interps uses a TreeView to display all non-windows resources. This includes namespaces (both commands and variables), fonts, images and events. These are all indexed by interp name, one for each Tk program running under the window managers display.

Variables can have their value changed by double clicking on the Value column.

Command procs can be dynamically edited in the running program by double clicking on the proc value column. The file containing a proc can be edited by double clicking on the file value column.

Usually, a sub-tree can be refreshed just by closing and reopening it.
There are several checkboxes that control viewing:

### Remotes

Check to show all remote interps.

### Hide Commands

Hide all non-proc commands.

### Hide Builtins

Hide all commands, procs or vars that are considered builtin. These mostly affect only the :: namespace.

## 19.2 Windows

Windows uses a TreeView to display all widget window resources. These are all indexed by interp name, one for each Tk program running under the window managers display. Option values can be changed by double clicking on the Value column.
The following checkbox option is available.

### Hide Properties

Check to display only the window tree hierarchy, without the properties (bindings, winfo and options).

## 19.3 Sandbox

The Sandbox environment contains one of every Tk widgets available in Wize as well as one of each type of item (for widgets supporting items).

The widgets/items may be examined and changed dynamically. This provides instantaneous access to real working widgets and items and their options.

## 19.4 Cmds

The command/option hierarcy for all builtin commands in Wize. Many commands in Tcl take sub-commands and even sub-sub-commands each of which may take various arguments and options. This allows you to view the signatures for each command.

Double clicking on any command gives a detailed breakdown of that commands arguments in the right hand pane. Some commands have detailed type information included For and example, checkout Tcl/fconfigure.

These are broken down into 5 groups:

- Widgets - The widget commands.
- Tcl - The Tcl commands.
- Tk - The Tk commands
- Blt - The BLT specific extension commands.
- Misc - Reserved for future use.

## 19.5 Manuals

Finally, it provides access to all online documention, both for the Tcl/Tk commands and for Tcl's C-programming API.

Double clicking on any man page will display the manual in a new tab. Right click on any tab to close it. Or use <Control-s> to search the page.

Click on the INDEX link at top to go to the table of contents, where you can click on more links. Use <Alt-left> to return from a link.

---

# 20. Development

The devel macro commands are used to simplify debugging when **warnings** are enabled. If warnings are disabled, these all return the empty string and do nothing. Moreover, the commands can become Tcl noops by calling **Mod ndebug**: A noop has zero runtime overhead.

Here's an example:

```
proc Foo {n m} {
    .Trace
    .Assert {$n>0 && $n<1000} 1
    if {[.Debug] != {}} {
        CheckRange $m $n
    }
    .Debug {
        if {$n < $m} { .Break BadN1 }
    }
    .Warn "Begin processing"
    return $n.0
}
```

Note that all commands start with period + capital letter.

Below are the supported commands.

## 20.1 .Assert expr ?warnonly?

Evaluate the expression expr. The expression should use curley braces to avoid a double eval. If warnonly==1 then calls .Warn instead of causing an error. If warnonly>1 the output contains detailed stack info (ie. to help debugging).

```
.Assert {$n>0}
.Assert {$n>1} 1
.Assert {$n<-1} 2
```

## 20.2 .Break ?str?

Invoke **Tcl inspect**, eg.

```
.Break stop1
```

## 20.3 .Debug ?script?

If called with no argument it returns the current debug level. Otherwise evaluate the script and issue a warning only if an error occurs.

Usage:

```
if {[.Debug]!=""} {
    if {$m==$n} { error "equal error" }
}
.Debug {
    if {$n<$m} { error "range error" }
}
```

WARNING: Do not do the following as it can result in a runtime error:

```
if {[.Debug]} { #... }
```

## 20.4 .Error str ?subst?

Kick an error. If subst is true, evaluate str first.

The following are roughly equivalent:

```
.Error {bad call: $n} 1
if {[.Debug] != {}} { error "bad call: $n" }
```

## 20.5 .Trace ?-num cnt? ?-fmt bool? ?-prefix str?

Dump the call-stack info from the current proc. The default is to dump only the current proc, with no formatting or prefix. Using a cnt of -1 will dump the whole call-stack.

If -fmt is true, show a in name=value form

```
proc Foo {n} {
    .Trace -num -1 -fmt 1
}
```

## 20.6 .Warn str ?subst?

Log a warning message using tclLog. If subst is true, evaluate str first.

```
.Warn "Something bad happened"
.Warn {Range error: $m>$n} 1
```

---

# 21. Debugging Programs

In Tcl, debugging has traditionally been limited to using puts or tclLog statements in the code. Herein we discuss some other alternatives.

## 21.1 Validating Programs

Wize provides static code checking with:

```
wize -Wall prog.tcl
```

This statically checks Tcl procs for **validation**
Even if a program passes validation, there can still be errors. Here are a few debugging utilities.

## 21.2 .Break

You can inspect variables within a running proc vy inserting a **.Break XXX** statement. When this gets executed, the**TclInspect** console is invoked allowing the user to view/modify variables, procs or edit the file. The XXX label is optional and is only used in locating code with multiple .Breaks.

For example:

```
# File "foo.tcl"
package require Mod

proc Foo {n} {
    incr n
    .Break 1
    set n [expr {$n*2.3}]
    .Break 2
    return $n
}
```

```
puts [Foo 1]
exit 0
```

Run this with:

```
wize -Wall foo.tcl
```

This will invoke **TclInspect** where you can examine and change variables.

## 21.3 Error Trap

Sometimes it's desirable to debug a proc that is causing a traceback. Tracebacks are useful for showing that an error occurred, but unfortunately the current state information is lost by the time the stack unwinds.

With Mod an application can trap errors using ::env(TCL_TRAP). This invokes **TclInspect** right at the error, much like .Break, eg.

```
wize  -Wlevel=all,trap=1 bad.tcl
```

## 21.4 Tracing Proc Calls

You can trace all commands by calling **bltdebug**.

Wize supports tracing of all proc calls using:

```
wize -Wproccalls=3 prog.tcl
```

---

# 22. Backtrace

Decoding a Tcl error traceback can be very tedious. This is particularly true in larger applications involving hundreds of lines of backtrace and dozens of stack levels. Therefore **Mod** provides a facility that automatically decodes stack tracebacks, presenting them in one-level-per-line format. Th can also optionally stop the program right at an error, before the stack unwinds in a traceback. (note: this facility is for handling runtime errors, and presumes program files have already sourced without error.)

Mod handles background errors by unwinding the stack backtrace into a one per line listing which can then be used to navigate through source code involved in an error. Here a couple of screenshots. The first is a **real-error screenshot**and the other the **install demo-error screenshot**. Clicking on any given level, a Mini-EDitor (Med) will popup displaying the file/line of error. Med provides only rudimentary

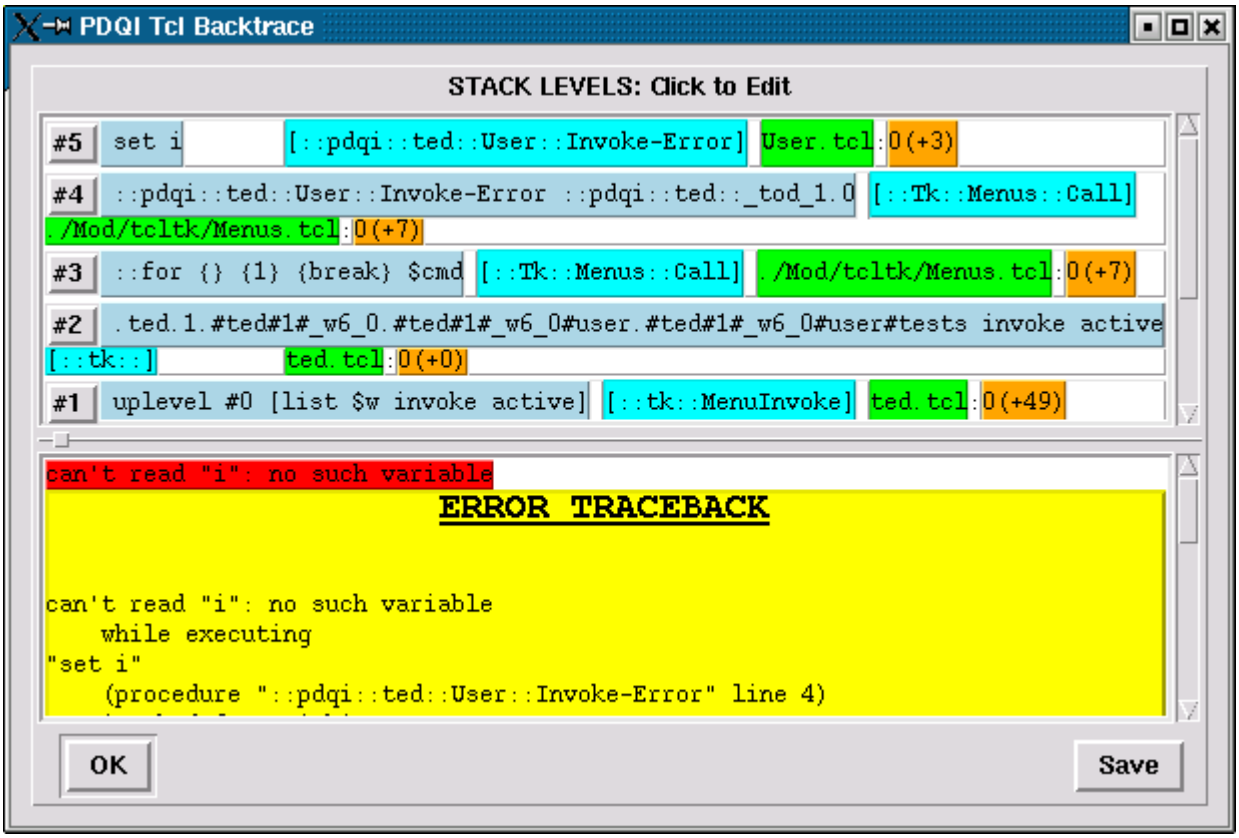
capabilities, however, it does support save and so allows immediate editing and fixing of problems.

One issue with debugging Tcl is that it normally does not collect file or line information associated with procs. Mod allows forcing this collection by adding the following to the top of your program (or setting it from command-line).

```
set ::env(TCL_WARN) all
```

When not using TCL_WARN or Wize -Wall, Mod instead falls back to show just the proc definition.

The backtrace window should look like:




Clicking on any line should open a Mini-Editor window (see below).

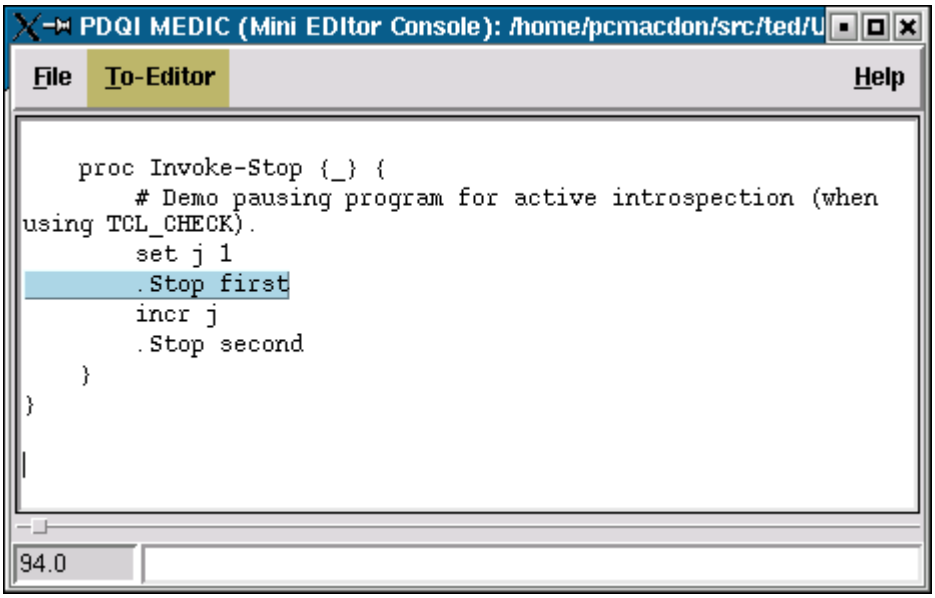
## 22.1 Pausing A Program

It is sometimes desirable to pause a running program right inside a proc, to allow inspection of the runtime variables. This can be achieved using the .Break directive while running a program while running with -Wall.

Here is an example:.

```
    proc Invoke-Stop {_} {
        # Demo of pausing a program for inspection.
        set j 1
        .Break first
        incr j
        .Break second
    }
```

If run with checking on, this should open a window something like:




As shown, Tcl commands can be executed in the command input at bottom. Closing the window will resume execution, pausing again at the next .Break.

## 22.2 Trap

Trap deals with uncaught errors by stopping the program right at the error to enable the user to inspect variables.

To enable it run the program like so:

```
  wize -Wlevel=all,trap=1 script.tcl
```

or put the following at the top of the main script

```
    set ::env(TCL_WARN) "level=all,trap=1"
    package require Mod
```

Trap stops a program-event right at the point of error, to allow introspection of the running program. Commands can then be run within a procs error context, prior to the unwinding of the stack.

Another way to use trap is selecting the trap option from Teds Run-Tcl.

WARNING: Do not always use the trap option as it exercises obscure areas of Tcl and can intermittently crash.

---

# 23. Util Macros

The **Util** macros are a collection of frequently used code. These all start with a star character *. Following are some of the more commonly used ones.

## 23.1 *catch

Eval with catch, displaying any errors as a warning. The warning message also contains the namespace (and proc if possible) of the offending call. When not running with wize -Wall, errors are silently ignored.

```
*catch { CallFunc 1 "X" }

# Equivalent to ...

if {[catch { CallFunc 1 "X" } erc]} {
   .Warn "Catch: $erc"
}
```

## 23.2 *value

Returns the value of a variable if it exists, otherwise returns the default, or if no default is given, an empty string, eg.

```
set n [*value ::MyNs::Arr(Really_Long_Value) 0]

# Equivalent to ...

if {[info exists ::MyNs::Arr(Really_Long_Value)]} {
    set n $::MyNs::Arr(Really_Long_Value)
} else {
    set n 0
}
```

## 23.3 *bvalue

Return the value for an element from a binary (name/value pair) list. If available, the dict command is used, otherwise falls back to a list search.

```
set LookupTable {able 1 baker 2 charlie 3}

set val [*bvalue $LookupTable baker 0]

# Equivalent to ...

if {[dict exists $LookupTable baker]} {
   set val [dict get $LookupTable baker]
} else {
   set val 0
}
```

## 23.4 *fread/*fwrite

Read or write a file. Additional options are passed to fconfigure:

```
set dat [*fread file1.dat]
*fwrite file2.dat $dat -translation binary

# Equivalent to ...

set fp [open file1.dat]
set dat [read $fp]
close $fp

set fp [open file2.dat w+]
fconfigure $fp -translation binary
if {[catch { puts -nonewline $fp $dat } erc]} {
   close $fp
   error $erc
}
close $fp
```