Working with Perl modules

Lesson 3

Graphical Interfaces with Perl/Tk

Perl/Tk

- Collection of modules that make the Tk toolkit available for perl programmers
  - originally designed by John K. Ousterhout for Tcl only
  - adaptation to perl by Nick Ing-Simmons
- Tk version 8 is implemented, also support for Tk 4
- Available at DESY both for NT and UNIX
- GUI builder available(specTcl, specPerl, specJava)
  - generates Tk4 code, not very useful
  - installed on /afs/ifh.de/products/SpecTcl/bin
  - sources at http://keck.ucsf.edu/~kvale/specperl.html

Outline of a Perl/Tk Application

- A typical Perl/Tk program does the following steps
  - create a main window
  - create widgets in the main window
  - define additional callbacks and key bindings
  - pack / place the widgets within the main window
  - display the windows and widgets and wait for events
- Perl/Tk uses the object oriented features of Perl
- Widgets are defined as subclasses of Tk
  - get loaded on demand

Elements of Tk

- To write GUI's the following elements are available
  - Standard widgets (15), derived (combined) widgets
  - Widgets that come in additional Tk modules (CPAN)
  - geometry managers to arrange the widgets
  - Widget attributes (color, size, …)
  - Callbacks (routines that are triggered by events)
  - Bindings (association between events and callbacks)

Standard widgets

- Very impressive demonstration of available widgets by calling **`widget`** from the command line
  - program also displays its source code
- normal Widgets : **`Label`**, **`Button`**, **`Menu`**, **`Text`**, **`Scrollbar`**, **`Canvas`**, **`Entry`** and others
- Container widgets
  - **`Toplevel`** (independent window like MainWindow)
  - **`Frame`** (grouping or separation of widgets)

Widget Options

- Widget configuration done by options
- Options get specified in anonymous hash
- ***Widget*** specific options described in

  perldoc Tk::*Widget*
- General options for all widgets (color, border,..) in

  perldoc Tk::options
- Manipulation of options after widget creation by

  $widget->configure(-*option => 'value');*

Arranging Widgets

- Done by geometry managers
  - pack - placing according to available space for widget
  - grid - placement of widgets in a rectangular grid
  - place - placement by giving absolute or relative positions
- Takes place within MainWindow or Frame
  - use only one geometry manager within one container
  - but in different frames different managers can be used
- Geometry manager algorithm changeable by options
  - e.g. orientations **`n,e,s,w,ne,se,nw,sw,center`**

The pack geometry manager

- Calling sequence of widgets defines the layout
  - widgets get aligned at a side:

  (**`-side => 'left,right,top,bottom'`**)
  - selected side determines size of "allocation rectangle"
  - placement within rectangle using the **`-anchor`** option

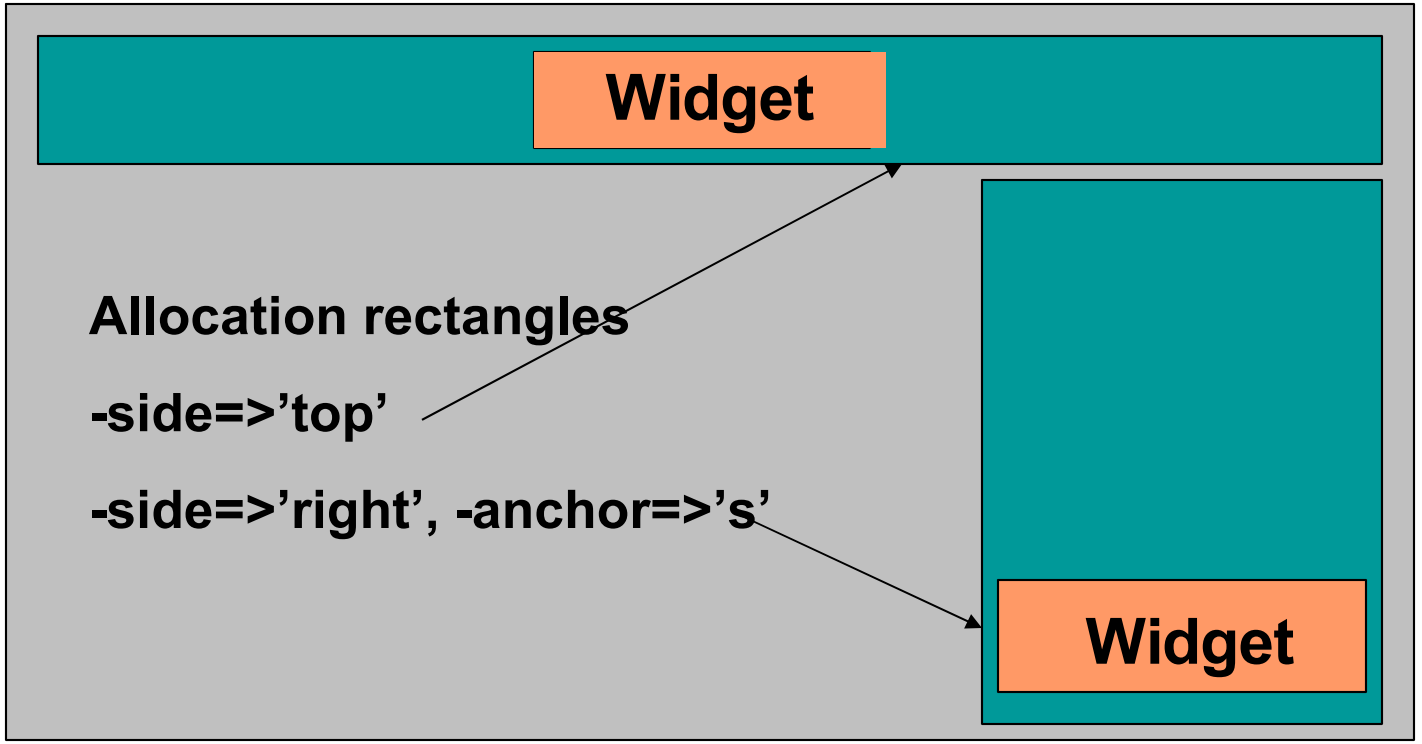
  (**`-anchor => 'n,e,s,w,ne,se,sw,nw,center'`**)
  - expansion of widget to borders of the rectangle using

  (**`-fill => 'none,x,y,both')`**
  - for further information see **`perldoc Tk::pack`**

Pack Demonstration

```
use Tk;
sub pack_demo {
my $main = MainWindow->new;
my $quit = $main->Button(
    -text => 'Welcome to Perl/Tk',
    -command => sub {exit;} );
$quit->pack;
}
pack_demo;
MainLoop;
```

Widget arrangement using pack



size of rectangle determined by full available length at side '`-side`' and widget size (other side)

Pack Demonstration (2)

```
use Tk;
sub pack_demo {
  my $main = MainWindow->new;
  my $quit = $main->Button(
     -text => 'Welcome to Perl/Tk',
     -command => sub {exit;} );
  $quit->pack;
  my $quit2 = $main->Button(
     -text => 'another button',
     -command => sub {exit;} );
  $quit2->pack(-side=>'right',-anchor=>'s');
}
pack_demo;
MainLoop;
```

The grid geometry manager

- grid subdivides the area into rectangles
- every **`grid`** call creates a new row
- number of widgets in call defines number of columns

```
$w1->grid($w2, $w3, .., -opt1=>'val1', ..);
```

- explicit **`-row`** or **`-column`** location possible
- widgets can span several rows or columns

  `-rowspan` or **`-columnspan`**
- widget span can be coded into calling parameters

  `x` means leave cell empty, **`-`** is rowspan, **`^`** is columnspan

Grid usage

- Widgets occupy minimum required space
- can be changed using option `-sticky=>'n,s,e,w'`
  - widget sticks to side `n, s, e` or `w`
  - this changes size or placement of widget
- Options to influence space between widgets
- Further configuration parameters available
- For a detailed description see `perldoc Tk::grid`

A grid demo

```
use Tk;
sub grid_demo {
  my $main = MainWindow->new;
  $main->Button(-text=>'1')
               ->grid($main->Button(-text=>'2'),
                      $main->Button(-text=>'3'),
                      -sticky => 'ew');
  $main->Button(-text=>'4')->grid('-', 'x', -sticky => 'ew');
  $main->Button(-text=>'quit', -command => sub {exit;})
               ->grid(-columnspan => 3);
}
grid_demo;
MainLoop;
```

The place geometry manager

- Overlapping widgets can be created
  - not possible with the other two managers
- Placement of widgets by
  - coordinates **`x`** and **`y`** (units: pixels on screen)
  - relative position **`relx`** and **`rely`** with respect to parent
- Size of widgets is determined by options
  - **`height`** and **`width`** (pixels)
  - **`relheight`** and **`relwidth`** w.r.t. parent
- Anchor position within widget to place by option
  - **`-anchor => 'n,e,s,w,ne,se,nw,sw,center'`**
- For further information see **`perldoc Tk::place`**

Common methods

- All geometry managers offer common methods
  - to extract geometry information

    `packInfo(), gridInfo (), placeInfo`
  - to undo the placement, this has the effect that widgets become invisible but do not get destroyed

    `packForget(), gridForget(), placeForget`

A simple database GUI

```
use DBI;
use Tk;

my $top = MainWindow->new;                    ### create top level widget
$top->configure(-width => 350, -height => 250);
$top->minsize(350, 250);
$top->title ("Accounts in Zeuthen");

my $t_apps = $top->NoteBook(-ipadx => 6, -ipady => 6)->pack;
my $t_results = $top->Frame->pack;            ### frame for results messages
my $t_common = $top->Frame->pack;             ### frame for common buttons and messages

my $page1 = $t_apps->add ("Accounts",  -label => "Accounts"); ### Subpages for Notebook
my $page2 = $t_apps->add ("Persons",  -label => "Persons");
my $page3 = $t_apps->add ("Phonebook",  -label => "Phonebook");
my $page4 = $t_apps->add ("Misc",  -label => "Options");
```

A simple database GUI (2)

```
%comp =  ( accountname => '%',    uid => '',      homedir  => '%%'
             , expiredate => '%',         shell => '%%‘ );
%table = ( accountname  => 'Account',              uid => 'Unix User Id‘
           , homedir      => 'Home Directory',       expiredate   => 'Expire Date'
           , shell        => 'Login shell‘ );
@entries = qw( accountname uid homedir expiredate shell );
for ( @entries ) {
  my $fram = $page1->Frame->pack();
  $fram->Menubutton(-text => $table{$_}, -width =>20, -menuitems =>
       [
         [Radiobutton => 'starts with', -variable => \$comp{$_}, -value => '%'],
         [Radiobutton => 'equals',      -variable => \$comp{$_}, -value => ''],
         [Radiobutton => 'contains',    -variable => \$comp{$_}, -value => '%%'],
       ])->pack(-side => 'left', -fill => 'x');
  my $entry = $fram->Entry(-textvariable => \$shown{$_})->pack();
  $entry->bind('<Key-Return>', \&main::search);
}
```

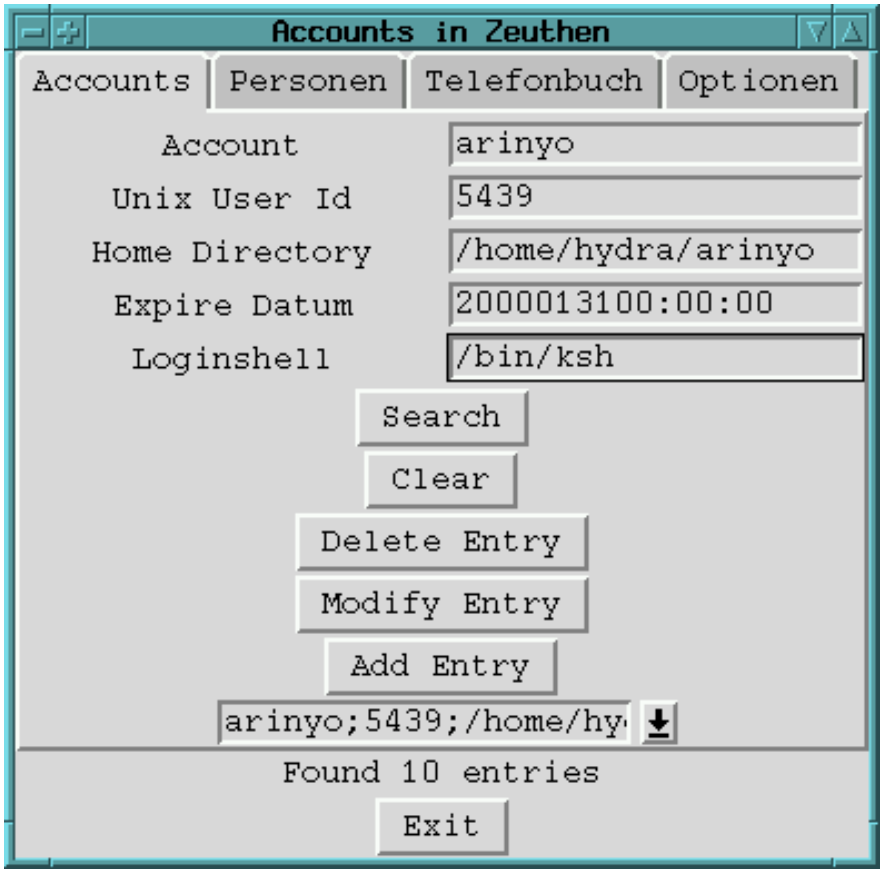
A simple database GUI (3)

```
$page1->Button(-text => "Search",       -command => \&main::search)->pack;
$page1->Button(-text => "Clear",        -command => \&main::clear)->pack;
$page1->Button(-text => "Delete Entry", -command => \&main::delete)->pack;
$page1->Button(-text => "Modify Entry", -command => \&main::modify)->pack;
$page1->Button(-text => "Add Entry",    -command => \&main::add)->pack;

my $foundEntries = $page1->BrowseEntry (-browsecmd=>\&main::select,
                                -textvariable => \$result,
                                -variable => \$selection)->pack;

$t_results->Label(-textvariable => \$result)->pack;             ### common buttons and messages
$t_common->Button(-text => "Exit", -command => \&main::cleanup)->pack;

MainLoop();                                                     ### Now do the real work
```

 # Routines which do the database specific work(add, modify, delete,…) are not reproduced here

 # complete example with all subroutines in /afs/ifh.de/user/f/friebel/public/demo11.pl

The resulting GUI



Definition of widget attributes

- get all widget attributes with `configure()`
- get one widget attribute with `cget(-option)`
- set widget attributes with `configure(-opt=>'val')`
- More detailed discussion of some attributes:
  - variable text
  - color
  - mouse cursor
  - fonts
  - callbacks
  - tags
- Documentation in `perldoc Tk::options`

configure and cget

- To query options use
  - `$curopt = $widget->cget(-option); #one value`
  - `@curopts = $widget->configure(-option);`

    returns 2 values (alias_name, option_name) for aliases

    else 5 values (optionname, dbname, class, defaultval, actval)
- To set options use
  - `$widget->configure(-opt1=>'val1',-opt2=>…);`

Variable Text in Attributes

- Reference to a variable is used in widget
- option **`-textvariable`** (Scale widget: **`-variable`**)

```
my $qbutton = $main->Button    (
    -textvariable => \$a,
    -command => sub {$a .= "!";} );
```

- Initial value gets set at widget creation time
- Changes of the variable value are instantly visible
  - be careful to stay in scope of variable
  - variable has to exist at runtime (MainLoop)

Color attributes

- used in many options, e.g.
  - **`-background,-foreground,-activebackground -highlightcolor,-selectcolor`**
- Usage of hex notation (rgb values like in HTML)
  - **`$color = "#d9d9d9";`**
- Usage of color names as defined in
  - rgb.txt (UNIX, e.g. for SuSE in /usr/X11R6/lib/X11)
  - Tk source code (pTk/mTk/xlib/xcolors.c)
  - definitions are identical, currently 752 names

Mouse cursor

- Default Cursor is arrow
- Shape of cursors may be changed
  - Cursor shape defined by widget option **`-cursor`**

```
$qbutton->configure(-cursor=>'hand1');
```

  - List of cursor names in cursorfont.h in directory \Perl\site\lib\Tk\X11 (Wxx, NT) or /usr/X11R6/include/X11/ (or similar, UNIX)
- the new cursor shape gets displayed only in the widget, for which the **`-cursor`** option was given

Font selection

- with widget option **`-font`** (see **`perldoc Tk::Font`**)
  - OS independent notation [***`family, size, type`***]

    `(family = Courier,Times,Helvetica, +os specific)`

    `(type = normal,bold,italic,underline,overstrike)`

    `-font=>[Helvetica,14,italic]`
  - OS specific notation: UNIX (see xlsfonts)

    `-font=>'-*-helvetica-*-normal-*-180-*'`
  - OS specific notation: NT (see \WINDOWS\FONTS)

    `-font => 'Times New Roman 12 normal'`
- Text width can be determined if font given
  - **`$widget->fontMeasure(font, text)`**

Callbacks

- Option **`-command`** see **`perldoc Tk::callbacks`**
- Option requires reference to subroutine or name
  - anonymous subroutine: **`-command=>sub {exit}`**
  - subroutine reference **`-command=>\&subroutine`**
  - subroutine name **`-command=>'subroutine'`**
- or anon array, first element as above, further elements are subroutine arguments
  - **`-command=>[\&sub, $arg1, \@arg2, $arg3 ...]`**
- respect the scope of variables (as already said above)

Tags in the Text widget

- used to
  - change the display options of text (font)
  - give a behavior to ranges of characters (binding)
  - deal with selected text
- Usage in two steps (**`perldoc Tk::Text`**)
  - creation of a tag with tagConfigure
  - usage of tags in methods, tag gets passed in parameter list
- Binding of (Tag, Event, Callback) see bind

Usage of tags

- Definition of tags

```
$t->tagConfigure('blue', -foreground=>'blue');
$t->tagConfigure('bold', -font=>['Courier',14,'bold']);
```

- Insertion of text without/with tags into Text widget

```
$t->insert('end', "normal text\n");
$t->insert('end', "blue text\n", 'blue');
```

- Adding tags to existing text (specify start and end)

```
$t->tagAdd('bold', '2.0', 'end');
```

Events

- Events follow the X11 scheme for events
- more than 20 event types, e.g.:
  - Key, Button, Motion, Enter, Leave
- Event described as three part string

  "<Modifier-Event-Detail>"
  - Modifier: **`Control,Shift,Alt,Button#,Double,Triple`**
  - for Unix also: **`Meta, Mod1,..Mod5`**
  - Detail: further describes event: *key symbol, button#*
  - Examples: **`<Alt-Key-a>,<Double-Button-1>,<Return>`**

Further Event Types

- Timer Events can be defined
  - `$widget->after($delay_ms, $callback);`
  - `$widget->repeat($delay_ms, $callback);`
- I/O Events for asynchronous reading of files
  - `$main->fileevent(FH,'readable',$callback);`
  - gets called as soon as new input in FH available
  - e.g. to program the tail(UNIX) functionality using Tk
- Idle Events for own tasks with low priority
  - `$main->afterIdle(\&callback);`

Bindings

- Event triggers an action, if bound to callback
- Many standard bindings
- Own bindings using method **`bind`** possible
- Bind a callback to an event:
  - **`$widget->bind("<Event>", \&subroutine);`**
- Parameter passing like for callbacks:
  - **`$widget->bind("<Ev>", [\&sub, $p1, $p2,..]);`**
- Evaluation of parameters done when event occurs
  - difference to callback definition with **`-command`** option

Bindings (2)

- Bind a callback to tagged text (tag defined previously with **`tagConfigure`**)
  - **`$t->tagBind("tag", "<Event>", \&subroutine);`**
- Parameter passing using anon array as before
- Well suited to program hypertext documents
  - when clicking on tagged text the callback can display new text (hyperlink)
- Information on defined event types with
  - **`@events = $widget->bind(ref $widget);`**

Event Information

- Fetched using function **`Ev()`**
- **`Ev`** passed to callback as argument
  - **`$b->bind("<Key>",[sub{$a="@_";},Ev('k')]);`**
  - first argument for callback is widget object
- function **`Ev()`** retrieves the following info:
  - coordinates, where event occurred : e.g. **`Ev('x')`**
  - which mouse button was pressed : **`Ev('b')`**
  - which key was hit : **`Ev('K')`**

Literature

- Learning Perl/Tk, Nancy Walsh, O'Reilly (1999)
- Perl and the Tk Extension, Steve Lidie, Perl Journal, Issues 1-9
- http://www.perlmonth.com (Issues 2,3,6)
- FAQ at ftp://ftp.uni-hamburg.de:
  /pub/soft/lang/perl/CPAN/doc/FAQs/tk/
- **`perldoc Tk`**