

Perl/Tk

Francesco “Nids” Nidito

IPW 2008

Outline

Introduction

Anatomy of a Perl/Tk Application

Advanced Features

Perl/Tk Best Practices

Conclusions

Outline

Introduction

Anatomy of a Perl/Tk Application

Advanced Features

Perl/Tk Best Practices

Conclusions

Once upon a time...

- ▶ ...computer interfaces were very boring
 - ▶ few colors
 - ▶ text based interfaces
 - ▶ or just the command line!
 - ▶ well... I still love command line but this is another story...
- ▶ In the 70s some funny programmers invented the GUI
 - ▶ better human-computer interaction
 - ▶ bells & whistles
 - ▶ scripting languages wanted GUIs too!
 - ▶ Tk was born has an extension of TCL
 - ▶ Perl adopted Tk in 1994

Why Use a Graphical Interface?

- ▶ Interesting question! Let's try to reply
 - ▶ when your boss asks you to make something simpler to use
 - ▶ when you need some interaction in your programs
 - ▶ an application used to apply filters to pictures (e.g. using ImageMagick)
 - ▶ a videogame!
 - ▶ for fun... (that's how I begun using Tk)
 - ▶ for profit! (selling twice the same script adding just the interface!)

Reasons to Use Perl/Tk

- ▶ There are **good** reasons to use Perl/Tk
 - ▶ easy to install and to use
 - ▶ from the CPAN shell `install Tk`
 - ▶ to use Tk in your programs just `use TK`
 - ▶ good library design
 - ▶ object oriented design
 - ▶ things are called with their name!
 - ▶ a lot of modules around (searching on CPAN, 959 results for "Tk")
 - ▶ old enough to be very stable
 - ▶ portable (I personally used Perl/Tk in Linux, Solaris, Win32 and MacOSX)
 - ▶ good look and feel (if you like WWII tanks and/or CDE... like me)

Reasons Not to Use Perl/Tk

- ▶ There are good reasons to use **something else** too
 - ▶ you already program GUIs in language X using library Y and you want to use it in Perl too
 - ▶ you prefer to use a new cutting edge library still in pre-alpha
 - ▶ you do not want to use GUIs at all (why are you following this talk?)
 - ▶ you **still** think that web based applications simplified the GUI programming
 - ▶ bad look and feel (if you like MacOSX)

Why Use Perl/Tk in a Web-Based-World?

- ▶ Probably, you think that web based applications simplified the GUI programming
 - ▶ easy to make GUIs (HTML forms, images just using `img` tag...)
 - ▶ portability (your application will run in every browser on every platform)
- ▶ Probably, you are kidding me!
 - ▶ GUIs are (sometimes) simpler but application state is much more difficult to handle
 - ▶ portability (just three words: IE and CSS)

About This Presentation

- ▶ I suppose that you know Perl syntax and main features
- ▶ Too little time for a Perl/Tk tutorial
- ▶ I want to focus on
 - ▶ what Perl/Tk can do for you
 - ▶ what you must pay attention to using Perl/Tk
 - ▶ how to write reasonable things in Perl/Tk
- ▶ Feel free to interrupt me in any moment for questions

Outline

Introduction

Anatomy of a Perl/Tk Application

Advanced Features

Perl/Tk Best Practices

Conclusions

A Stupid... ehm... Simple Example

```
use Tk;

my $mw = MainWindow->new( -title => "Simple Example" );
my $b  = $mw->Button( -text => "push here!",
                    -command => sub{ exit(0); } );
$b->pack( );

MainLoop;
```

A Note About Perl/Tk Syntax

- ▶ Object oriented (`MainWindow->new`)
- ▶ “Switches” to pass arguments to widgets (`-text => "push here!"`)
- ▶ Closures to bind actions to widgets (`-command => sub{ exit(0); }`)
 - ▶ plain closures: `-command => sub{ ... }`
 - ▶ function: `-command => \&foo`
 - ▶ function names: `-command => 'foo'`
- ▶ The closures can be used with arguments too (using array references)
 - ▶ `-command => [sub{ ... }, $x, $y]`
 - ▶ `-command => [\&foo, $x, $y]`
 - ▶ `-command => ['foo', $x, $y]`

Where everything starts...

```
my $mw = MainWindow->new( -title => "Simple Example" );
```

- ▶ The `MainWindow` is the principal window of your application
- ▶ All the other widgets are generated starting from it
- ▶ In Perl/Tk parlance, the `MainWindow` is a `TopLevel` widget

Populating the Window...

```
my $b = $mw->Button( -text => "push here!",  
                    -command => sub{ exit(0); } );
```

- ▶ The standard way to create widgets is starting from a parent widget using the `type` as a method name
 - ▶ `$parent->widget_type(parameters);`
- ▶ Base widgets supported by Perl/Tk:
 - ▶ Button
 - ▶ Label & Entry
 - ▶ Checkbutton & Radiobutton
 - ▶ Scrolbar
 - ▶ Listbox
 - ▶ Text (read/write and read only)
 - ▶ Scale
 - ▶ Menu
 - ▶ Frame (sets of other widgets)
 - ▶ Toplevel (new windows)

The Entry Widget (Just an Example)

- ▶ Widgets are used to communicate with the user
- ▶ ...at least it must be able to read (just the Entry example)

```
my $x;  
my $entry = $mw->Entry( -textvariable => \$x );
```

- ▶ That's all!

Scrollable Widgets...

- ▶ **Scrolbar** is not very useful by itself...
- ▶ ... it can be combined with other widgets

```
my $s = $mw->Scrolled('widget_type', -scrolbars => "string",  
                        -other_options );
```

- ▶ `widget_type` as the same meaning of the previous slide
- ▶ The `scrolbars` says where the scrolbars must be placed
 - ▶ base places: `n`, `s`, `w` and `e`
 - ▶ optional places: `on`, `os`, `ow` and `oe`
 - ▶ combined places: `ne`, `nw`, `se` and `sw` (`ns` and `ew` are not valid)

Geometry Management...

```
$b->pack( );
```

- ▶ Used to physically insert a widget inside the window
- ▶ Without the geometry management call the widget is **never** placed
- ▶ Three different geometry management calls
 - ▶ pack
 - ▶ place
 - ▶ grid
 - ▶ form

Geometry Management: pack

```
$b->pack( ... );
```

- ▶ Literally packs the widgets together without overlapping
- ▶ When you do not need to place your widgets in more complex ways, pack is the right choice
- ▶ Typical options for pack
 - ▶ `-side` can be 'left', 'right', 'top' or 'bottom'
 - ▶ `-fill` can be 'x', 'y' or 'both'
 - ▶ `-anchor` can be 'n', 's', 'e', 'w' (and their combinations) or 'center'

Geometry Management: place

```
$b->place( ... );
```

- ▶ Uses coordinates inside the widget to place the widgets
 - ▶ using the `-x` and `-y` options
- ▶ By default the coordinates are relative to the upper-left corner of the window
 - ▶ settable using the `-anchor` option (values are `'e'`, `'w'`, `'n'`, `'s'`...)
- ▶ The widgets can overlap!

Geometry Management: grid

```
$b->grid( ... );
```

- ▶ Divides the window into a grid made up of **columns** and **rows**
- ▶ The (0,0) cell is in the upper-left corner of the window
- ▶ The usage is a little bit **strange**

```
$mw->Button( ... )->grid( $mw->Button( ... ),  
                           'x',  
                           $mw->Button( ... ) );
```

Defines a row made up of 4 columns (the first, the second and the fourth with a button inside and the third one with nothing inside)

- ▶ You can still set explicitly the row and the column of a widget using `-row` and `-column`

Geometry Management: form

```
$b->form( ... );
```

- ▶ Something like `pack` and `place...` together!
 - ▶ widgets can be *placed* using `-x` and `-y`
 - ▶ but you `-x` and `-y` can be relative to another widget
 - ▶ and you can tell the widgets that it can expand as when you *pack* widgets
- ▶ You can force other widgets away from each other using **springs**
 - ▶ `-(left|right|top|bottom)spring` using a weight
 - ▶ the weight is the strength used to push other widgets away

Where the Interface Runs...

```
MainLoop();
```

- ▶ Draws the main window
- ▶ Waits for events and call the handles in response to events
 - ▶ in GUI parlance, the `MainLoop` implements the **event loop**

Anatomy of an Event Loop

The `event loop`, in a GUI application, can be written like this:

```
while(1) {
    event = get_event();

    if event is mouse_click, call process_mouse_click
    elsif event is keyboard_click, call process_keyboard_click
    elsif event is window_event, call process_window_event
    ...
    elsif event is exit, call process_exit and break
}
```

exit VS destroy

- ▶ Perl/Tk provides a `destroy` method to exit applications
- ▶ What is the difference between `exit` and `destroy`?
 - ▶ `exit` just exits the application
 - ▶ `destroy` just exits the `MainLoop`

exit VS destroy

```
use Tk;

my $mw = MainWindow->new( -title => "Simple Example" );
my $be = $mw->Button( -text => "exit",
                    -command => sub{ exit(0); } );
my $bd = $mw->Button( -text => "destroy",
                    -command => sub{ $mw->destroy(); } );

$be->pack( );
$bd->pack( );

MainLoop;

print "I appear only if you destroy the main window\n";
```

Outline

Introduction

Anatomy of a Perl/Tk Application

Advanced Features

Perl/Tk Best Practices

Conclusions

What Are Advanced Features After All?

- ▶ Indeed advanced features are not so advanced
- ▶ Advanced features are all the features that
 - ▶ can be used in developing more complex applications
 - ▶ you must know to do richer applications and/or new modules
- ▶ Yes, the things seen until now are the base one...

The Menu Widget

- ▶ Large **document-centric** applications are structured in the following way
 - ▶ a large part of the window is given to the document (text, image...) to edit
 - ▶ a bar providing all the actions that can be performed on such document
 - ▶ the bar is organized in menus to group actions
- ▶ The menu is **essential** to structure the actions in a **tree-like** way
 - ▶ in a menu/sub-menu way actions are grouped in a logical way

The Menu Widget Syntax

```
my $menubar = $mw->Frame( )->pack( -fill => 'x' );
my $f = $menubar->Menubutton(
    -text => 'File',
    -menuitems => [[ 'command' => 'Save',
                    -command => \&doSave ],
                  [ 'command' => 'Load',
                    -command => \&doLoad ],
                  '- ', # inserts a divisor
                  [ 'command' => 'Exit',
                    -command => sub{ $mw->destroy } ]]
);
```

- ▶ A menu bar is just a collection of Menubuttons
- ▶ A Menubutton is a collection of menuitems
- ▶ menuitems is an array reference of actions

A lot of menuitems

```
$mb = $menubar->Menubutton( ...,  
    [ [ 'command' => 'Save', -command => \&doSave ],... ] );
```

```
$mb = $menubar->Menubutton( ... );
```

```
$mb->AddItems([ 'command' => 'Save', -command => \&doSave ]);
```

```
$mb->radiobutton( ... );
```

```
$mb->checkboxbutton( ... );
```

```
$sm = $mb->cascade( -label => 'New' );
```

```
$sm->AddItems([ 'command' => 'Image', -command => \&newImg ]);
```

Dialog Windows

- ▶ When you want to warn the user
- ▶ When you want the user to interact briefly with the application (Continue/Abort/Retry)
- ▶ When you just want to say "Hello!" to the user

```
use Tk::Dialog;
...
$d = $mw->Dialog( -title => 'Hi!',
                  -text => 'I just wanted to say "hi!"',
                  -default_button => 'Hi!',
                  -buttons => ['Hi!', 'Don\'t piss me off'] );
$r = $d->Show();
print "You are not my friend anymore!\n" if( $r !~ m/Hi!/ );
```

- ▶ A lot of dialogs ready to be used (with Tk and in CPAN):
 - ▶ messageBox, DialogBox, ErrorDialog, chooseColor, FileDialog...

Custom Events Binding

- ▶ Sometimes it is useful to redefine the binding for keys, buttons etc.
- ▶ Is it easy?
- ▶ Yes!
- ▶ `$widget->Bind('<Button-1>', sub{ exit; });`

Dealing with Input Streams

- ▶ Perl can open other application as stream
 - ▶ `open FH, "some_app |" or die ...;`
- ▶ How can I read without blocking on a *read*?
- ▶ If a stream is ready to be read, it rise an event... just as a button
- ▶ `$mw->fileevent(*FH, 'readable', sub{ ... });`
- ▶ **ACHTUNG!!!!!!!!!!!!!!!** Inside the function that reads from FH do not use `$x = <FH>!!`
- ▶ `$x = <FH>` will block waiting for whole line... use `sysread` instead

Perl/Tk and Threads

- ▶ Sorry... Perl/Tk is not reentrant
- ▶ Events and Threads are not a good idea!
- ▶ (At this point someone should ask if it is *really* not possible)
- ▶ (Don't be shy!)
- ▶ Technically... no
- ▶ But if you coordinate (and well isolate) the threads and the event loop
yes

Outline

Introduction

Anatomy of a Perl/Tk Application

Advanced Features

Perl/Tk Best Practices

Conclusions

Something That I learned

- ▶ Designing a good interface is difficult
- ▶ Some things that I learned
 - ▶ keep the interface simple
 - ▶ you know the interface... the user doesn't
 - ▶ organize things in a tree-like way
 - ▶ make things as configurable as possible
 - ▶ before starting the implementation take paper, scissors and a pencil to draw and to try to compose your interface

Something Personal

- ▶ The story so far
 - ▶ I started programming Perl/Tk for fun
 - ▶ I used Perl/Tk to make applications to show XML files as trees, do visualize simulations results...
- ▶ The future
 - ▶ I am working on a Desklets toolkit (*a la* gDesklets but without Python)
 - ▶ I would like to create an image processing application (using ImageMagick)

Outline

Introduction

Anatomy of a Perl/Tk Application

Advanced Features

Perl/Tk Best Practices

Conclusions

Conclusions

- ▶ We have seen (briefly) Perl/Tk
- ▶ I hope that you learned something
- ▶ I learned a lot!!!!
- ▶ I hope you had fun

References

- ▶ Walsh N. “Learning Perl/Tk”, O’Reilly
- ▶ Lidie S., Walsh N., “Mastering Perl/Tk”, O’Reilly

Questions?

(Answers are better... but questions are good too)