TkRibbon: Windows Ribbons for Tk

Georgios Petasis

Software and Knowledge Engineering Laboratory,
Institute of Informatics and Telecommunications,
National Centre for Scientific Research “Demokritos”,
Athens, Greece
petasis@iit.demokritos.gr
Overview

- The Windows Ribbon framework
- Creating a Ribbon
  - Writing the XAML Markup
  - Compiling the Markup
- Creating the TkRibbon widget
- Interacting with the Ribbon
- Conclusions – Future work
The Windows Ribbon Framework

A new UI paradigm, aiming to unify into a single UI element:
- Multilayered menus
- Toolbars
- Task panes

Paint for Windows 7

TkRibbon: Windows Ribbons for Tk
The Ribbon framework consists of two UI components:
- The Ribbon command bar, which contains:
  - The Application menu
  - A set of standard tabs
  - A Help button
- A rich contextual menu
Ribbons and Applications

- Two distinct but dependent development platforms:
  - A XAML-based markup language, which describes the controls, their properties and their visual layout.
  - A set of COM C++ interfaces that ensure interoperability between the Ribbon framework and the application.

- A Ribbon is actually a COM object:
  - Attaches to the top part of a window
  - Redraws window and decoration as needed
  - Interacts with the user & application
TkRibbon: Ribbons for Tk

- TkRibbon provides the needed middleware for:
  - Loading a resource DLL containing one (or more) Ribbons
  - Initialise the Ribbon framework
  - Create a “fake” Tk widget, for occupying the needed space for Tk widget managers
  - Attach a Ribbon to a Tk toplevel widget
  - Communicate user actions from the Ribbon to the application
    - Through Tk virtual events
  - Send requests to the Ribbon
    - By invoking widget subcommands
Creating a Ribbon in Tk Ribbon

**MARKUP**

- Ribbon Declaration (Markup)
- Header .h
- Resource .rc
- Markup Binary .bml

**CODE**

- Create Ribbon Tk Widget
- Load Resource DLL
- Load Ribbon from Resource DLL
- Register Command Callback
- Register Command Event Handlers

**TkRibbon: Windows Ribbons for Tk**

13 Oct 2010
Writing the XAML Markup (1)

- Two major parts:
  - Definition of commands
  - Layout of commands in tabs, groups inside a tab, and commands inside a group

- Everything is a command

- Commands have properties:
  - Label
  - Tooltip
  - Images
  - etc.
<?xml version='1.0' encoding='utf-8'?>
  <Application.Commands>
    <Command Name="cmdExit" Symbol="cmdExit" LabelTitle="Exit"
      TooltipTitle="Exit" TooltipDescription="Exit Application..." />
  </Application.Commands>

  <Application.Views>
    <Ribbon>
      <Ribbon.Tabs>
        <Tab>
          <Group>
            <Button CommandName="cmdExit" />
          </Group>
        </Tab>
      </Ribbon.Tabs>
    </Ribbon>
  </Application.Views>
</Application>
Compiling the XML into a DLL

- Ribbons are contained in DLLs
  - Thus, the XAML describing a Ribbon must be compiled

```plaintext
uicc.exe  ribbon1.xml  ribbon1.bml  /header:ribbon1.h \ 
/res:ribbon1.rc  /name:RIBBON1
rc.exe  ribbon1.rc
link.exe  /NOENTRY  /DLL  /MACHINE:X86  /OUT:ribbon1.dll \ 
ribbon1.res
```

```
//  ******************************************************************************
//  * This is an automatically generated header file for UI Element definition *
//  * resource symbols and values. Please do not modify manually.             *
//  ******************************************************************************
#pragma once
#define cmdExit 2
#define cmdExit_LabelTitle_RESID 60001
#define cmdExit_TooltipTitle_RESID 60002
#define cmdExit_TooltipDescription_RESID 60003
#define InternalCmd2_LabelTitle_RESID 60004
#define InternalCmd4_LabelTitle_RESID 60005
#define InternalCmd6_LabelTitle_RESID 60006
```
package require Tk
package require tkribbon
set ScriptDir [file dirname [file normalize [info script]]]
## The resources DLL containing the Ribbon...
set RibbonDLL $ScriptDir/ribbon1.dll
## Create a Ribbon widget:
set toolbar [tkribbon::ribbon .ribbon -command 
            onRibbonUpdatePropertyDispatch]
## Load the resources DLL: must be executed at least once
## for each DLL...
$toolbar load_resources [file nativename $RibbonDLL]
## Load the Ribbon UI from the DLL...
$toolbar load_ui [file tail $RibbonDLL] RIBBON1_RIBBON
## Pack the widget at Toplevel top: ensure expanding is false!
pack $toolbar -side top -fill x -expand false
## Important: The Ribbon will not be drawn,
## unless the window is large enough!
wm geometry . 300x250 ;# The minimum size for showing the Ribbon!
## Events:

```bash
foreach event {Execute Preview CancelPreview CreateUICommand ViewChanged DestroyUICommand UpdateProperty} {
    bind $toolbar <<on$event>> \ 
    [list onRibbonEventDispatch $event %d]
}
```

```bash
proc onRibbonUpdatePropertyDispatch {args} {
    puts "onRibbonUpdatePropertyDispatch: args: $args"
} ;# onRibbonUpdatePropertyDispatch
```

```bash
proc onRibbonEventDispatch {event args} {
    puts "onRibbonEventDispatch: event: $event, args: $args"
} ;# onRibbonEventDispatch
```
The result: A Ribbon inside Tk

TkRibbon: Windows Ribbons for Tk
Interacting with a Ribbon

- Three means of interaction:
  - Through the widget callback
    - When the Ribbon requests property values
  - Through virtual events
    - When an event occurred in the Ribbon
  - Through widget subcommands invocation
    - When the application performs a request to the Ribbon
The widget callback (1)

- The callback is invoked when a property value is needed, because:
  - A property is not defined in the XML
  - A property has been invalidated
- Three parameters (at most):
  - The command id
    ✓ An integer
  - The property type
    ✓ One from: UI_PKEY_Enabled, UI_PKEY_RepresentativeString,
      UI_PKEY_ItemsSource, UI_PKEY_Categories, UI_PKEY_SelectedItem,
      UI_PKEY_BooleanValue
  - The current value (if available)
The widget callback (2)

- The callback is expected to return a value
  - According to the property type
  - The most complex types relate to galleries

- UI_PKEY_Categories

```plaintext
[[list [list categoryName1 ... categoryNameN] {}]]
[[list [list categoryName1 ... categoryNameN] [list imageResourceId]]]
[[list [list categoryName1 ... categoryNameN] [list imageResourceId1 ... imageResourceIdN]]]
```

- UI_PKEY_ItemsSource

```plaintext
[[list [list item1 ... itemN] images-list categories-list]]
```

- Combo boxes are also galleries!
Virtual events

- `<onExecute>`
  - Delivered when the user has executed a control
  - “%d” contains the command id

- `<onPreview>`
  - Delivered when mouse hovers over a command

- `<onCancelPreview>`
  - Cancels an `<onPreview>`

- `<onCreateUICommand>`, `<onViewChanged>`, `<onDestroyUICommand>`, `<onUpdateProperty>`
  - Reserved for future use
Many available subcommands

- `pathname load_resources native-dll-path`
  ✓ Relate to loading a Ribbon

- `pathname load_ui module ribbon-name`

- `pathname get_property property-type control-id`
  ✓ Retrieve the value of a property

- `pathname invalidate_state state-property control-id`
- `pathname invalidate_value property control-id`
- `pathname invalidate_property property control-id`
  ✓ Invalidate property aspects, so as new values will be requested
Conclusions – Future work

- TkRibbon allows usage of Ribbons from Tk
  - A large percentage of Ribbon functionality is supported

- Future work will concentrate on:
  - Supporting missing features
    - Recent Files item list not available
    - Saving and restoring state of the Quick toolbar

- Contextual tabs are supported
  - But not tested yet

- Support for contextual menus missing
  - Is it important?
Thank you!