GUI Programming with wxHaskell

Wei Tan
<wlta543@cse.unsw.edu.au>

Countdown Alarm Clock

Widgets used

Frame
Menu
StaticText
Panel
SpinCtrl
Button

UI Creation Code

```haskell
-- create main window & container for widgets
f <- frame [text := "Countdown Watch", clientSize := sz 300 200]
panel <- panel f []

-- create menu
timerMenu <- menuPane [text := "&Timer"]
tstart <- menuItem timerMenu [text := "&Start"]
quit      <- menuQuit timerMenu [help := "Quit"]

-- labels
timeLabel <- staticText panel [text := "Time:", fontWeight := WeightBold]

-- spin control
sec <- spinCtrl panel 0 59 [outerSize := sz 35 20]
```
UI Creation Code (cont’d)

```haskell
-- start/cancel button
startBtn <- button panel [text := “Start”]
set startBtn <- [on command := setAlarm f startBtn hr min sec]

-- layout
set f [layout := column 1 $
 [hfill $ hrule 1 , -- ruler to separate menu from panel
  fill $ container panel $
  margin 10 $ column 10
  [hfill $ row 1 , -- current time
    [widget timeLabel, glue, widget timeStatic],
  ,hfill $ row 1 , -- set alarm interval spinCtrl
    [widget intvLabel, glue, widget hr, label “:”, ...
    ,floatBottomRight $ widget startBtn]]] -- start button
```

What is wxHaskell and what’s so good about it?

- Haskell binding for wxWidgets
- wxWidgets is a cross-platform GUI library written in C++. Mature, extensive, actively being developed.
- supports 75% of wxWidgets’ functionality
- wxHaskell is a medium-level library – it offers simple functional bindings + higher level abstraction (really neat)

Why use wxHaskell?

- Rapid prototyping
- Commercial applications
- Multi-platform support, native look-and-feel
- Integrate with existing Haskell code
- Because we can ☺

Tour of wxHaskell

- Packages
- Controls
- Types & Inheritance
- Events
- Attributes and Properties
- Layout
- Miscellaneous – Db, Timer, Var, OpenGL
**wxHaskell Packages**

- **Graphics.UI.WXCore**
  - Lower level interface to wxWidgets library
  - Almost one-to-one mapping between C++ and Haskell
- **Graphics.UI.WX**
  - Built on top of WXCore
  - Provides nice functional abstraction (attributes, layout combinators, etc.)

**Controls**

```haskell
p <- panel []
txt <- textEntry p AlignLeft [text := “your name here”]
cb <- comboBox p true [“NSW”, “ACT”, “VIC”, “WA”] []
rd <- radioBox p Horizontal [“one”, “two”] [on select := logSelection]
```

Other widgets: Gauge, Choice, ListBox, Slider, TreeCtrl, SplitterWindow, Toolbar

**Types & Inheritance**

- Encodes inheritance relationship between different widget types using ADT

```haskell
Object (Ptr)
| ..
  | Window
  |  Frame
  |  Control
  |  Button
  |  RadioBox
```

```haskell
Button a === Ptr (... (CWindow (CControl (CButton a))) ...)
```

**Attributes I**

We can control various attributes of widgets, e.g. caption, colour, font, size, etc.

But what attributes can I use on which widget?

- Attributes are organized into Haskell classes
- Types of widgets instantiate appropriate classes
- Inherit instance definitions from “parent types”
Attributes II

Type Frame a = Window (CFrame a)
Frame a instantiates HasImage, Form, Closable, and everything that Window instantiates
Window a instantiates Textual, Literate, Dimensions, ...

The HasImage class defines the 'image' attribute, Textual class defines the 'text' attribute.

So, we can:

```haskell
f <- frame []
set f [text := "Window Title", image := "/some/image.ico"]
```

Events

- Organized into Haskell classes (like Attr)
- A widget that instantiates an event class means it can receive events of that class.
- Event handlers can be defined by turning it into an attribute using the 'on' function:

```haskell
paint :: (Paint w) => Event w (DC () -> Rect -> IO ())
Window is an instance of Paint, so we can define our own paint routine for all window types (including buttons and text boxes).
set f [on paint := drawObjects]
```

Attributes and Properties

Attributes are turned in Properties with ( := )

```haskell
Prop (Button a)     Prop (Button a)
set btn [text := "Stop", on command := doSomething] |
| Attr String |
| Event IO |
| (Button a) String (Button a) |
| (IO ()) |

set :: forall w. w -> Prop w -> IO ()
set :: forall w a. Attr w a -> a -> Prop w
on :: forall a w. Event w a -> Attr w a
```

Layout

- Manages the positioning and sizing of widgets within a container widget
- wxHaskell uses layout combinators which allows a more declarative style of specifying layout
- The return type of a layout combinator is always Layout
- It may take other arguments, often another Layout
- Allows precise control of behavior when window is resized (or to prevent resizing)
- Types of layout combinators: layouts (widgets, containers, glue, spacers) and transformers (stretch, expand, margin, alignment)
Layout examples

```haskell
set f [layout := column 1 $
   [hfill $ hrule 1 -- ruler to separate menu from panel
   ,fill $ container panel $
   margin 10 $ column 10
   [hfill $ row 1 -- current time
      [widget timeLabel, glue, widget timeStatic],
   ,hfill $ row 1 -- set alarm interval spinCtrl's
      [widget intvLabel, glue, widget hr, label ":", ...
   ,floatBottomRight $ widget startBtn]]] -- start button
```

HPView

- Assignment: Heap Profile Viewer for GHC profiling output.
- Similar to hp2ps utility, but interactive.
- Draws a lot of lines so the mathematical model of Haskell helps.

Screenshot

Round up

- wxHaskell is great!
  - script-like GUI creation, speeds up development
  - no need to declare variables in IO monad, types deduced automatically (no need to keep track of intermediate objects)
  - uniform interface for getting/setting properties
  - closure for passing vars to event handlers, no special handling of `void * data`!