Gtk2Hs Tutorial

Krasimir Angelov

Chalmers University of Technology

November 24, 2011

1. Introduction to GUI programming
2. Introduction to GTK+ and Gtk2Hs
3. The First Gtk2Hs Program

1 Introduction to GUI programming

2 Introduction to GTK+ and Gtk2Hs

3 The First Gtk2Hs Program

Change of control!

- The biggest difference between CUI and GUI is the control mechanism
  - in the console - the application is in control of everything
  - in the graphical interface - the environment is in control
- The environment notifies the program for all interesting **events**
- The application responds by executing some **actions**

The visual elements are objects

- In the console, everything is text which flows from and to the application
- In the graphical interface, every visual element is an **object** which is owned by the application
- The visual elements are nested in each other (i.e. some elements are **containers** for other elements)

1 Introduction to GUI programming

2 Introduction to GTK+ and Gtk2Hs

3 The First Gtk2Hs Program

The GTK+ Project

- **GTK** (Gimp ToolKit) is an open-source and cross-platform graphical library:

  http://www.gtk.org/

- The core technology behind **GNOME**

- Written in **C** with architecture that makes it easy to write bindings for other languages.

GTK+ Bindings

Language	2.8	2.10	2.12	2.14	2.16	2.18	2.20	2.22	
C++	☑	☑	☑	☑	☑	☑	☑	☑	
C#	☑	☑	☑	☐	☐	☐	☐	☐	
Java	☑	☑	☑	☑	☑	☐	☐	☐	
Python	☑	☑	☑	☑	☑	☑	☑	☑	
JavaScript	☑	☑	☑	☑	☑	☑	☑	☑	
Perl	☑	☑	☑	☑	☑	⊟	⊟	☐	
Vala	☑	☑	☑	☑	☑	☑	☑	☑	
R	☑	☑	☑	☐	☐	☐	☐	☐	
Lua	☑	☑	☑	☑	☑	☐	☐	☐	
Guile	☑	☑	☑	⊟	☐	☐	☐	☐	
Ruby	☑	☑	☑	⊟	☐	☐	☐	☐	
PHP	☑	☑	☑	☑	☑	☐	☐	☐	
Ada	☑	☑	☑	☑	☑	☑	☑	☑	
OCaml	☑	☑	☑	☑	☑	☐	☐	☐	
Haskell	☑	☑	☑	☑	☑	☑	☑	☑	
FreeBASIC	☐	☐	☐	☐	☐	☑	⊟	☑	
D	☐	☐	☑	☑	☑	☑	☐	☐	

Gtk2Hs

- Gtk2Hs is the Haskell binding to GTK+

  http://www.haskell.org/gtk2hs/

- Rather low-level library. Every function and type is just exported to Haskell.

- There are high-level abstractions for some concepts i.e.: Attributes, Events, etc.

- We will stick to the low-level API since it has better coverage

Gtk2Hs Installation

- You need:
  - Gtk2Hs 0.12.2
  - GTK+ 2.22
- On Linux GTK is probably already installed; On Windows you have to install GTK first and Gtk2Hs after that. Read more here:
  `http://www.cse.chalmers.se/edu/course/TDA451_Functional_Programming/labs/4/gtk2hs-install.html`
- Everything is already installed on the Chalmers machines.

Sources of Inspiration

GTK+ is a huge library. If you don't know how to do something:

1. Look at the demos (http://code.haskell.org/gtk2hs/gtk/demo/)
2. Search the documentation (http://hackage.haskell.org/package/gtk-0.12.0).

   Note: If you cannot find some function for a given type, then look at the super class.
3. Ask in the course's Google Group

Web Tutorial

There is a Gtk2Hs tutorial on the course web page

`http://www.cse.chalmers.se/edu/course/TDA451_Functional_Programming/labs/4/gtk2hs.html`

1 Introduction to GUI programming

2 Introduction to GTK+ and Gtk2Hs

3 The First Gtk2Hs Program

Scope

A simple graphical interface that:

- opens a window
- lets the user to draw lines
- the width of the lines is configurable

Initialization

- You need to import the library:

  ```
  import Graphics.UI.Gtk
  ```

  Note: This is the top-level module which imports several other modules, so you don't need to list them explicitly

- Initialize the library:

  ```
  initGUI :: IO ()
  ```

- Run the event loop:

  ```
  mainGUI :: IO ()
  ```

Create a Window

- Look at Graphics.UI.Gtk.Windows.Window

  windowNew :: IO Window

- You also need to show the Window, so since it is a Widget look at: *Graphics.UI.Gtk.Abstract.Widget*

  widgetShow :: WidgetClass self ⇒ self → IO ()
  widgetShowAll :: WidgetClass self ⇒ self → IO ()

Why the program doesn't terminate?

- I close the window but the program doesn't terminate, why?
- We have to tell GTK when to terminate the event loop:

```
onDestroy :: WidgetClass w ⇒ w → IO () → IO (ConnectId w)
mainQuit :: IO ()
```

Create a DrawingArea

- Look at Graphics.UI.Gtk.Misc.DrawingArea

```
drawingAreaNew :: IO DrawingArea
```

- DrawingArea have to be added as a child of Window, look at *Graphics.UI.Gtk.Abstract.Widget*

```
containerAdd :: (ContainerClass self, WidgetClass widget) ⇒ self → widget → IO ()
```

Reaction to Events

- When the user clicks on the window, then the program have to react.
- We need a listener for the 'button press' event:

```
onButtonPress :: WidgetClass w ⇒ w → (Event → IO Bool) → IO (ConnectId w)
```

(in Graphics.UI.Gtk.Abstract.Widget)

Using State

- The default way to model state in GTK is by using the IORef type from the standard Haskell library. In module *Data.IORef*:

```
newIORef :: a → IO (IORef a)
readIORef :: IORef a → IO a
writeIORef :: IORef a → a → IO ()
```

Drawing (1)

- GTK+ has two (and more) layers - GTK and GDK
- GTK is the higher level where the user interface is composed of controls
- GDK is the lower level which is closer to the "device"
- **The drawing operations are on the device level**

Drawing (2)

- Every GTK widget is associated with one GDK window. The relation is:

  widgetGetDrawWindow :: WidgetClass widget ⇒ widget → IO DrawWindow

- Every drawing session is within some Graphical Context (GC):

  gcNew :: DrawableClass d ⇒ d → IO GC

Drawing (3)

- The graphical context remembers attributes like current color, font, filling pattern, etc.

```
gcSetValues :: GC → GCValues → IO ()
gcGetValues :: GC → IO GCValues
```

- They are used by all drawing primitives

```
drawLines :: DrawableClass d ⇒ d → GC → [Point] → IO ()
```

Drawing is yet another event

- The environment tells the program when it have to redraw the window.
- This is yet another event:

```
onExpose :: WidgetClass w ⇒ w → (Event → IO Bool) → IO (ConnectId w)
```

The program also could initiate redrawing

- When the program have changed its state then it have to tell the environment that it have to refresh its windows.

  widgetQueueDraw :: WidgetClass self $\Rightarrow$ self $\rightarrow$ IO ()

- After that the enviroment activates the 'Expose' event.

Layout widgets

- If you want more than one widget in the window then you have to arrange them somehow.
- GTK provides **Layout Widgets**:

```
vBoxNew :: Bool → Int → IO VBox
hBoxNew :: Bool → Int → IO HBox
```

- We pack widgets into boxes using:

```
boxPackStart :: (BoxClass self, WidgetClass child) ⇒ self → child → Packing → Int →
```

Two other widgets

- Entry

  entryNew :: IO Entry

- Button

  buttonNew :: IO Button

  buttonNewWithLabel :: String $\rightarrow$ IO Button

The size request event

- We have to tell GTK how big drawing area we want

  onSizeRequest :: WidgetClass w $\Rightarrow$ w $\rightarrow$ IO Requisition $\rightarrow$ IO (ConnectId w)

Now we can change the width of the line

- Attach a listener to the button click:

  onClicked :: ButtonClass b ⇒ b → IO () → IO (ConnectId b)

- Get the text from the entry box

  entryGetText :: EntryClass self ⇒ self → IO String

- The text into the entry may not be a number - use message boxes! (Graphics.UI.Gtk.Windows.MessageDialog)

- Set the actual width. Remeber the GCValues structure!

Done

Now we have a complete program!