FLTKHS - Easy Native GUIs in Haskell, Today!

Aditya Siram (@deech)

February 5, 2016

Outline

What

- FLTK (Fast Light Toolkit)
- C++
  - No multiple inheritance
  - No exceptions
  - No templates
- Stable, 20 yrs old
- Clean install on Win/Linux/Mac
- Embedded systems
  - Almost no dependencies
  - Not even glibc!
  - Static zero-dependency binaries
- Fluid Interface Builder

What

- Native GUI's in pure, impure Haskell
  - All in IO.
- Near Complete Integration With Fluid
  - Fluid interfaces compile to straight Haskell!

Why

- Installs easily
  - Yes, Windows too
- Very few dependencies
  - base, bytestring, directory, filepath,mtl, parsec, c2hs
- Zero-dependency binaries
  - Yes, Windows too

Why

- Easy To Learn™
- Add a 3rd party widget
  - Without recompiling!
  - Type-safe, no `unsafeCoerce`!

Easy

- What is "easy"?
  - Re-use existing FLTK documentation
  - Find function & datatypes
  - Straightforward translation of C++ to Haskell
  - Ape the API
- What is *not* "easy"?
  - Pure, functional Haskell

Easy

- Widget names
  - `Fl_Widget` is `Ref Widget`
  - `Fl_Light_Button` is `Ref LightButton`
- Widget Construction
  - `new Fl_Widget(..)` is `newWidget ...`
  - `new Fl_Light_Button(..)` is `newLightButton ...`

Easy

- Function names are the same as much as possible
- In FLTK
  - Type Sig

    ```
    void Fl_Valuator::bounds(double a, double b)
    ```

  - Call

    ```
    valuatorRef->bounds(1.0, 10.0)
    ```

- In FLTKHS

```
bounds valuatorRef 1.0 10.0
```

Easy

- Getters/Setters prefixed with get/set
- In C++
  - Type sig

    ```
    double Fl_Valuator::value()
    void Fl_Valuator::value(double v)
    ```

  - Call

    ```
    double v = valuatorRef->value()
    valuatorRef->value(1.0)
    ```

- In Haskell

```
v <- getValue valuatorRef
setValue valuatorRef 1.0
```

Easy

- All FLTKHS methods have multiple dispatch!
- In C++

```
int Fl_Input_::value(const char* str, int len)
```

- In Haskell

```
setValue :: Ref Input -> String -> Maybe Int -> IO (Int)
```

  - Not the real signature!

- Previously it was:

```
setValue :: Ref Valuator -> Double -> IO ()
```

- Rest of the arguments depend on the first!

Easy

- Error messages are decent too!
- Missing arguments

```
setValue inputRef
```

- Error message

```
Couldn't match type 'IO t0' with \
    'String -> Maybe Int -> IO Int'
  In a stmt of a 'do' block: setValue inputRef
```

Easy

- Missing everything

  `setValue`

- Less nice, but not horrible

```
Couldn't match expected type 'IO t0'
            with actual type 'Ref a0 -> impl0'
Probable cause: 'setValue' is applied to too \
  few arguments
```

Easy

- Wrong widget (a table does not have a `setValue`)

```
setValue tableRef
```

- Ugly, the info is there but ...

```
Couldn't match type \
  'NoFunction
     (SetValue ())
     (Graphics.UI.FLTK.LowLevel.Hierarchy.CTable Group)'
  with 'Match r0'
```

- GHC 8's custom type errors will help here

Easy

- Real type sigs. are ugly
- All widget docs show methods with friendly sigs!

Functions

```
destroy :: Ref ValueInput -> IO ()

getShortcut :: Ref ValueInput -> IO (Maybe ShortcutKeySequence)

getSoft :: Ref ValueInput -> IO (Bool)

getTextcolor :: Ref ValueInput -> IO (Color)

getTextfont :: Ref ValueInput -> IO (Font)
```

- It's all clickable.

Easy

- And also the widget hierarchy

Hierarchy

```
Graphics.UI.FLTK.LowLevel.Widget
 |
 v
Graphics.UI.FLTK.LowLevel.Valuator
 |
 v
Graphics.UI.FLTK.LowLevel.ValueInput
```

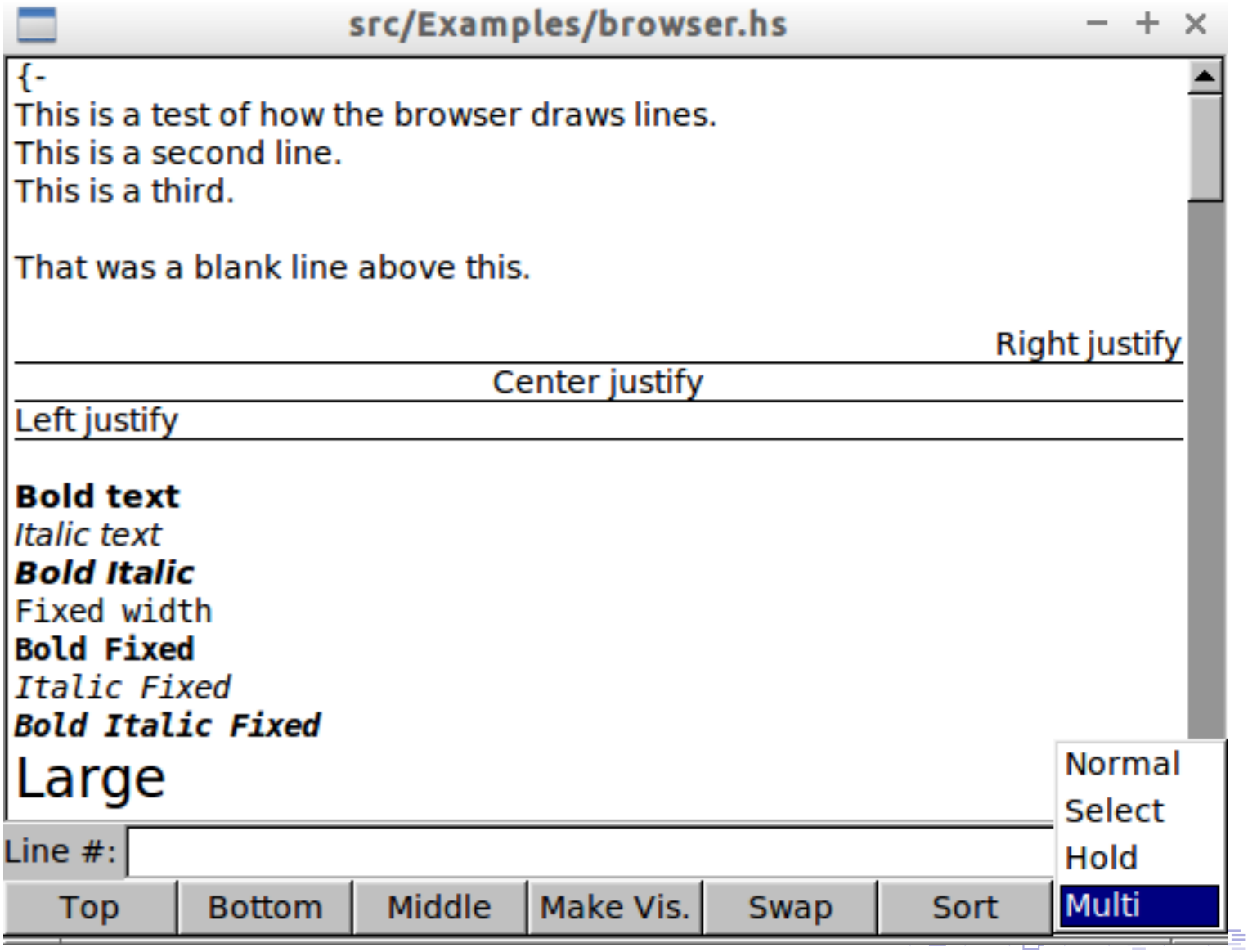
- Parent's functions transparently available!

Easy

- `fltkhs-demos` comes with 18 end-to-end demos
- 16 are exact copies of demos that ship with FLTK
- Learn by side-by-side comparison

Easy

- Browser demo (see select menu on bottom left)



Easy

- C++ and Haskell code that handles select menu callback.
- Don't worry about details, note the correspondence.

Easy

- C++ code

```
void btype_cb(Fl_Widget *, void *) {
  for ( int t=1; t<=browser->size(); t++ )
    browser->select(t,0);
  browser->select(1,0);   // leave focus box on first line
       if ( strcmp(btype->text(),"Normal")==0)
          browser->type(FL_NORMAL_BROWSER);
  else if ( strcmp(btype->text(),"Select")==0)
          browser->type(FL_SELECT_BROWSER);
  else if ( strcmp(btype->text(),"Hold"  )==0)
          browser->type(FL_HOLD_BROWSER);
  else if ( strcmp(btype->text(),"Multi" )==0)
          browser->type(FL_MULTI_BROWSER);
  browser->redraw();
}
```

Easy

- Equivalent Haskell code

```
btypeCb :: Ref SelectBrowser -> Ref Choice -> IO ()
btypeCb browser' btype' = do
 numLines' <- getSize browser'
 forM_ [1..(numLines' - 1)]
   (\l -> select browser' l False)
 _ <- select browser' 1 False -- leave focus box on first l
 choice' <- getText btype'
 case choice' of
  "Normal" -> setType browser' NormalBrowserType
  "Select" -> setType browser' SelectBrowserType
  "Hold" -> setType browser' HoldBrowserType
  "Multi" -> setType browser' MultiBrowserType
  _ -> return ()
 redraw browser'
```

Easy

- C++

```
for ( int t=1; t<=browser->size(); t++ )
  browser->select(t,0);
browser->select(1,0);   // leave focus box on first line
```

- Haskell

```
numLines' <- getSize browser'
forM_ [1..(numLines' - 1)]
  (\l -> select browser' l False)
_ <- select browser' 1 False -- leave focus box on first li
```

- Comments are preserved!

Easy

- C++

```
      if ( strcmp(btype->text(),"Normal")==0)
         browser->type(FL_NORMAL_BROWSER);
else if ( strcmp(btype->text(),"Select")==0)
         browser->type(FL_SELECT_BROWSER);
...
```

- Haskell

```
choice' <- getText btype'
case choice' of
 "Normal" -> setType browser' NormalBrowserType
 "Select" -> setType browser' SelectBrowserType
 ...
 _ -> return ()
```

Easy

- Callstacks!
  - Out-of-the-box in 7.10.x
- All FLTKHS "instance" methods check for a null `Ref`.

Easy

- Deletes itself in callback . . .

```
buttonCb b' = do
  FL.deleteWidget b'
  l' <- getLabel b'
  ...
main = do
 ...
 b' <- buttonNew ...
 setCallback b' buttonCb
 ...
```

Easy

- Callstack . . .

```
Ref does not exist. \
  ?loc, called at <full-path>/Fl_Types.chs:395:58 in ...
  toRefPtr, called at <full-path>/Fl_Types.chs:403:22 in ...
  withRef, called at <full-path>/Hierarchy.hs:1652:166 in ...
  getLabel, called at src/Main.hs:11:10 in main:Main
```

Easy

- Project skeleton available:
  `http://github.com/deech/fltkhs-hello-word`

Fluid

- A full fledged, mature GUI builder
- Ships with FLTK
- Out-of-the-box integration with FLTKHS

Fluid

- Designed to generate C++
- Now generates Haskell!
  - `fltkhs-fluidtohs` ships with FLTKHS
- Migrate existing C++ projects easily
  - `fltkhs-fluid-examples`
- Skeleton project available.
  - `https://github.com/deech/fltkhs-fluid-hello-world`

Fluid

- Project structure

```
+ fltkhs-fluid-hello-world
  - ...
  + src
    - Callbacks.hs
    - fluid-hello-world.hs
    - HelloWorld.fl
```

- Installing

```
> cabal install
```

- Running

```
> fltkhs-fluid-hello-world
```

Fluid

Unclicked

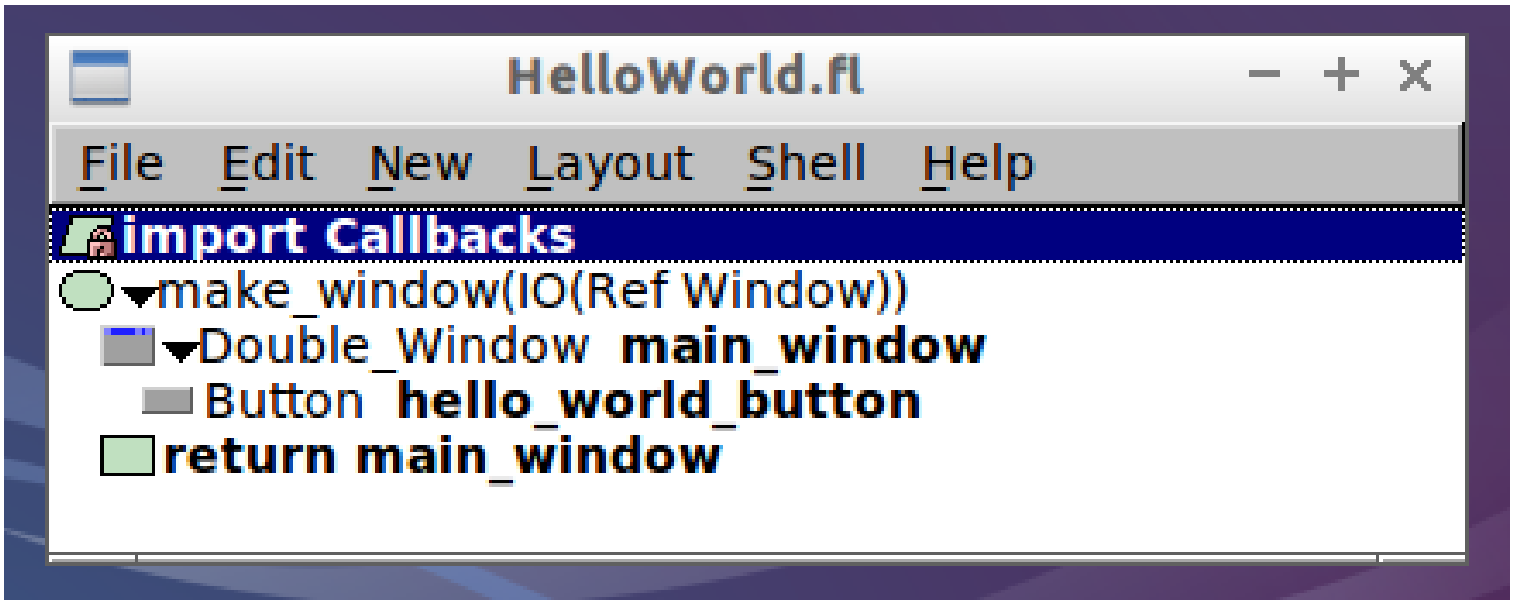
Clicking

Fluid

Clicked

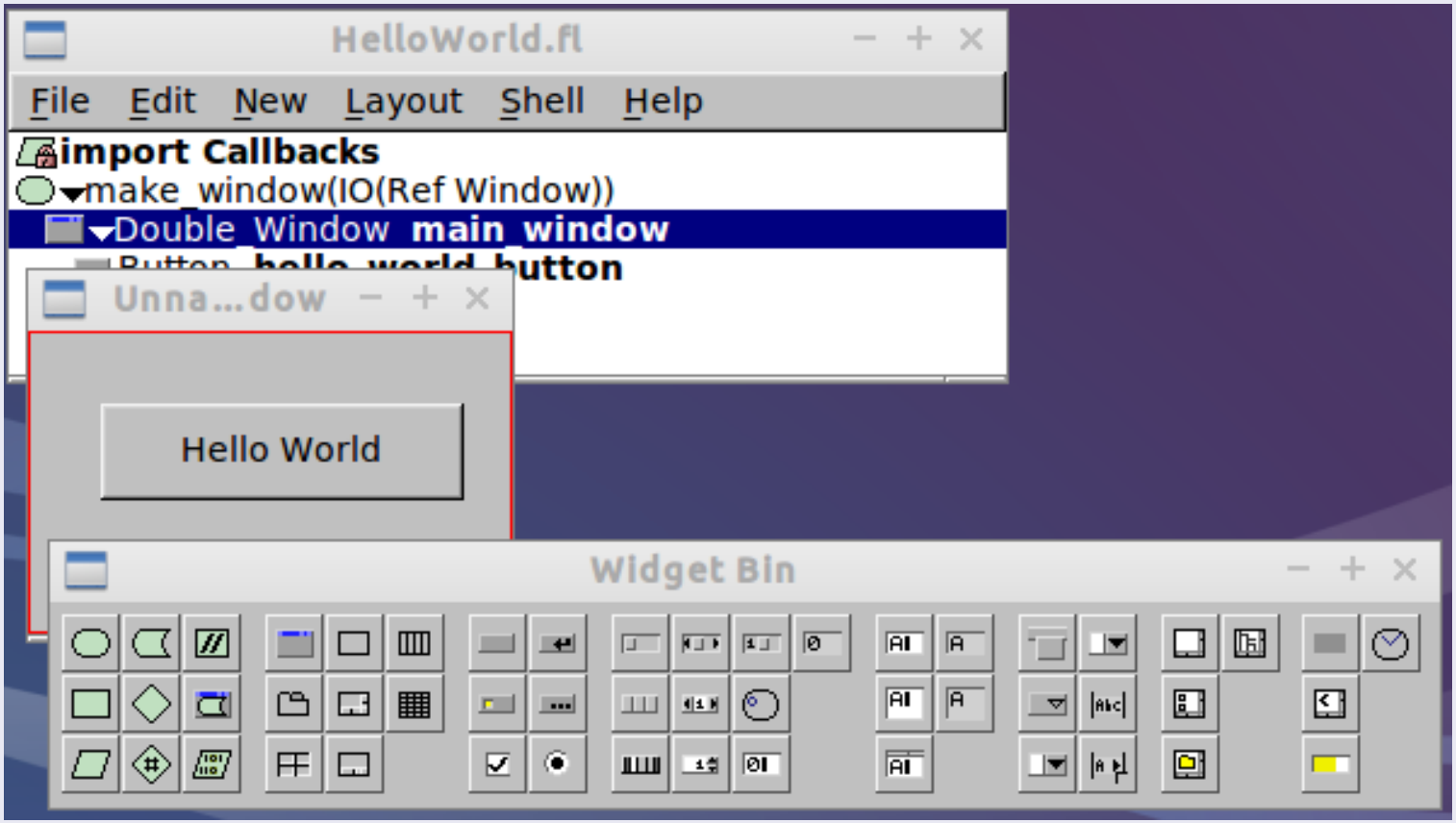
Fluid

```
> fluid HelloWorld.fl
```



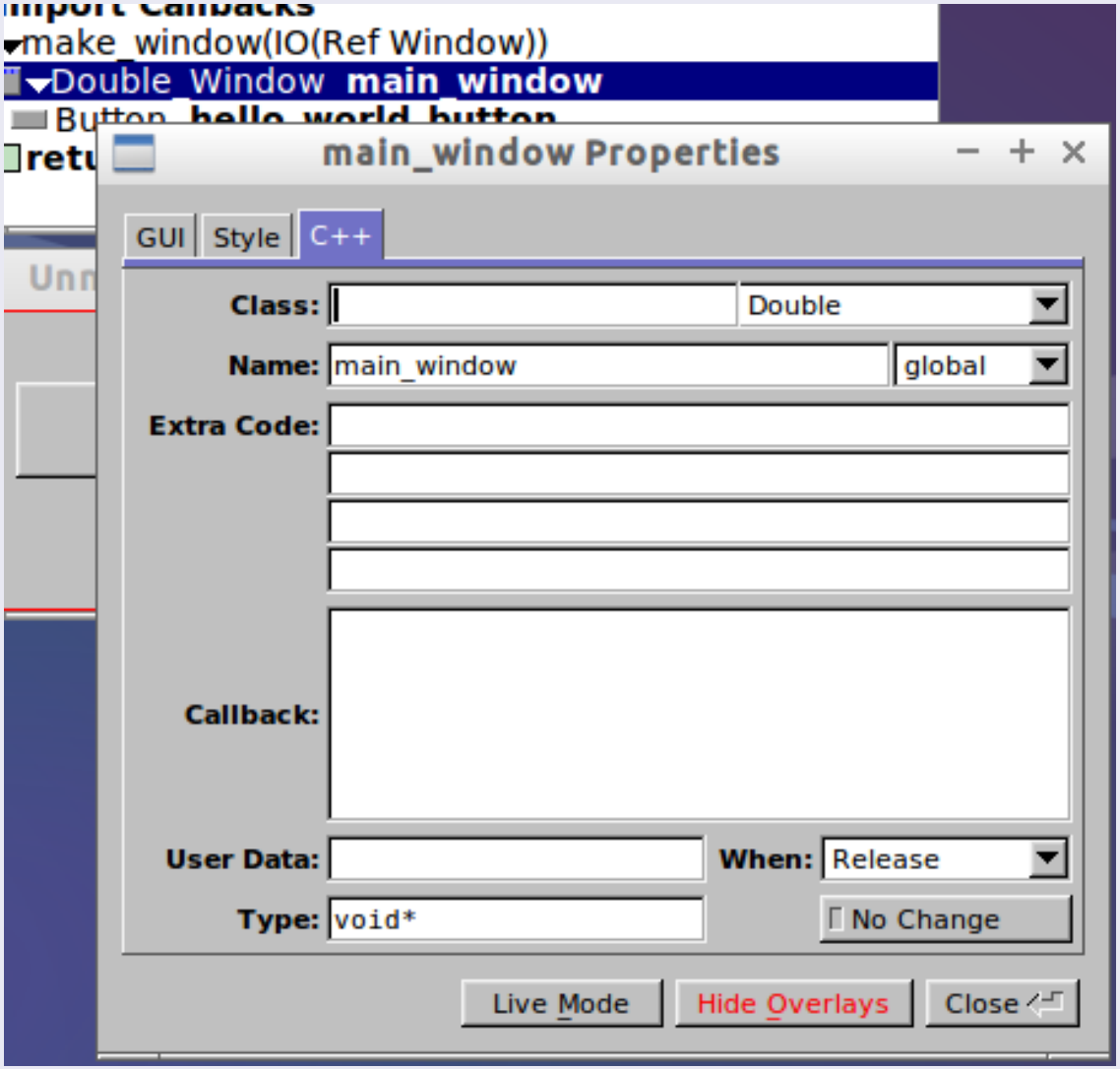
Fluid

Widget Bin



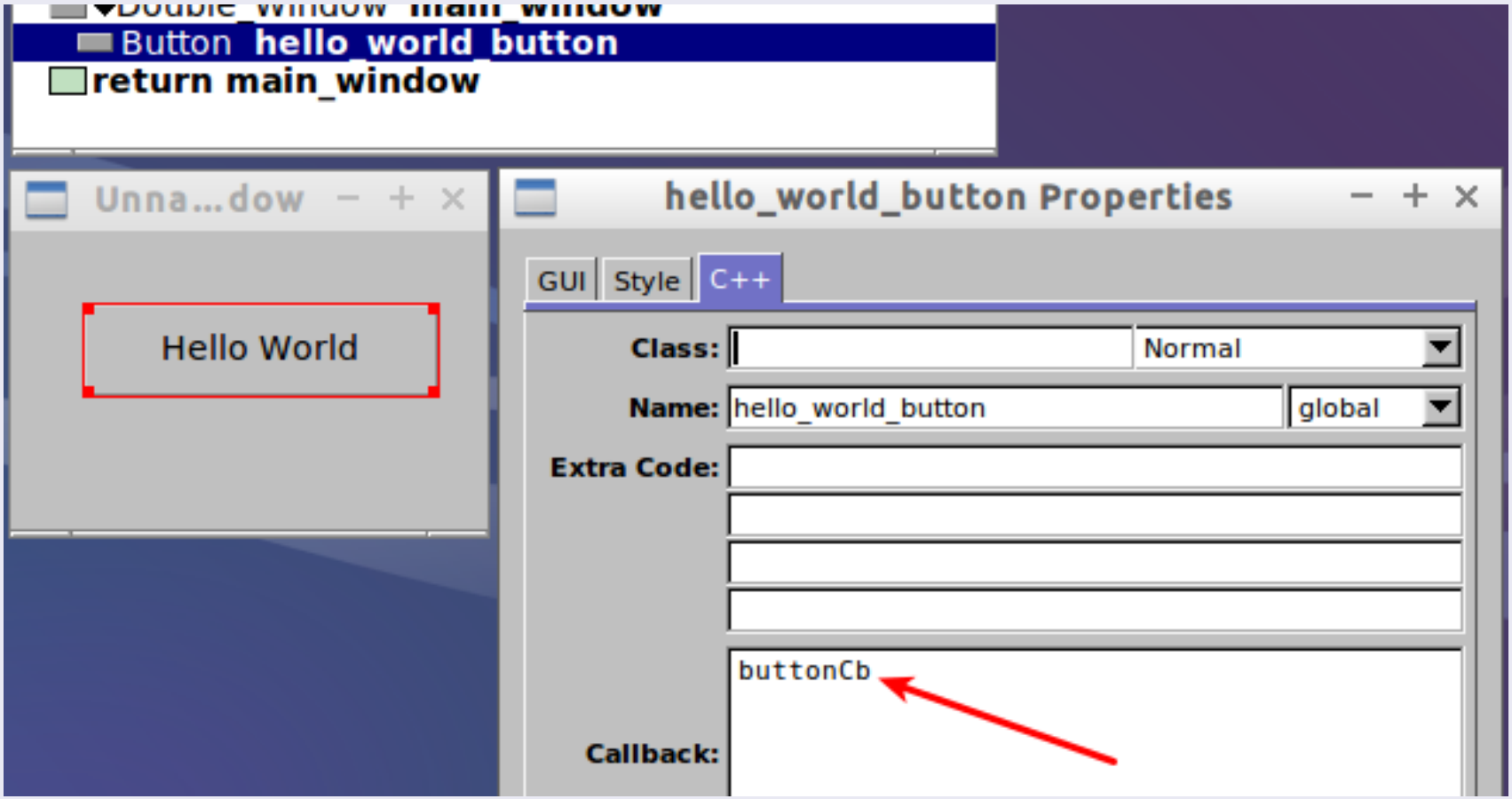
Fluid

Window properties



Fluid

Set the button callback.



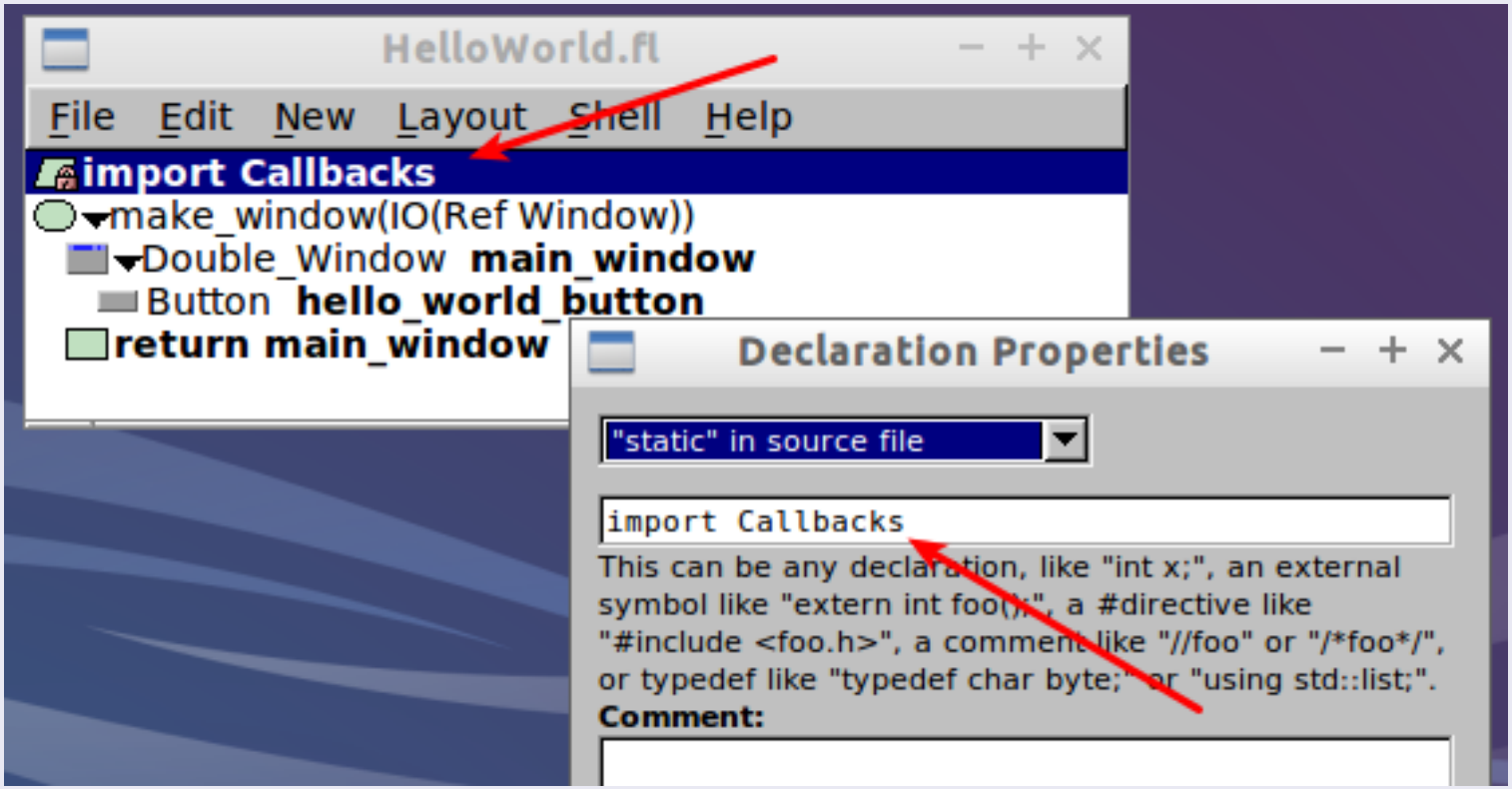
Fluid

- Callback logic

```
module Callbacks where
...
buttonCb :: Ref Button -> IO ()
buttonCb helloWorld = do
  l' <- getLabel helloWorld
  if (l' == "Hello World")
    then setLabel helloWorld "Goodbye World"
    else setLabel helloWorld "Hello World"
```

Fluid

Imports



Fluid

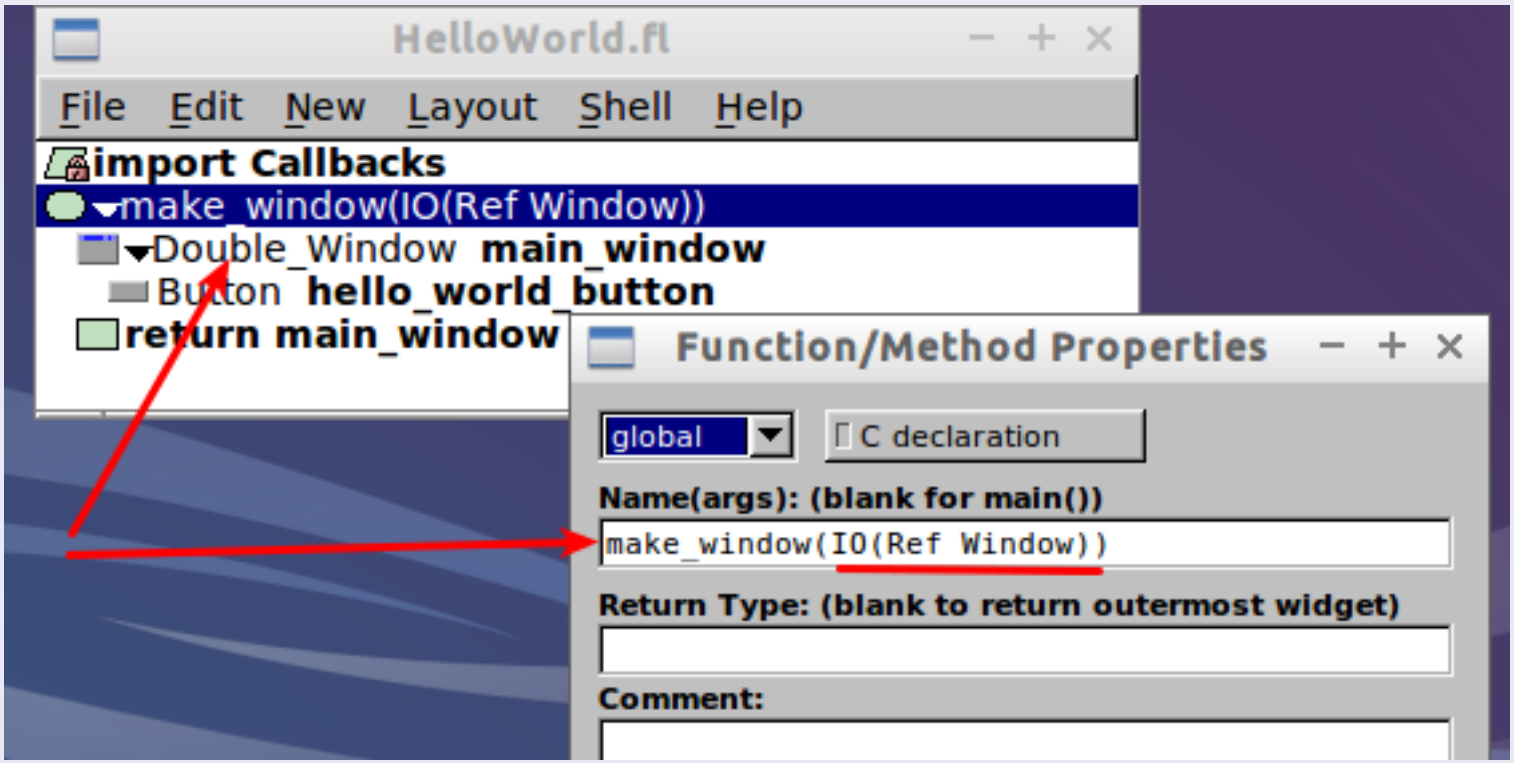
- Main Module

```
main = do
  window <- make_window
  _ <- showWidget window
  _ <- FL.run
  return ()
```

Fluid

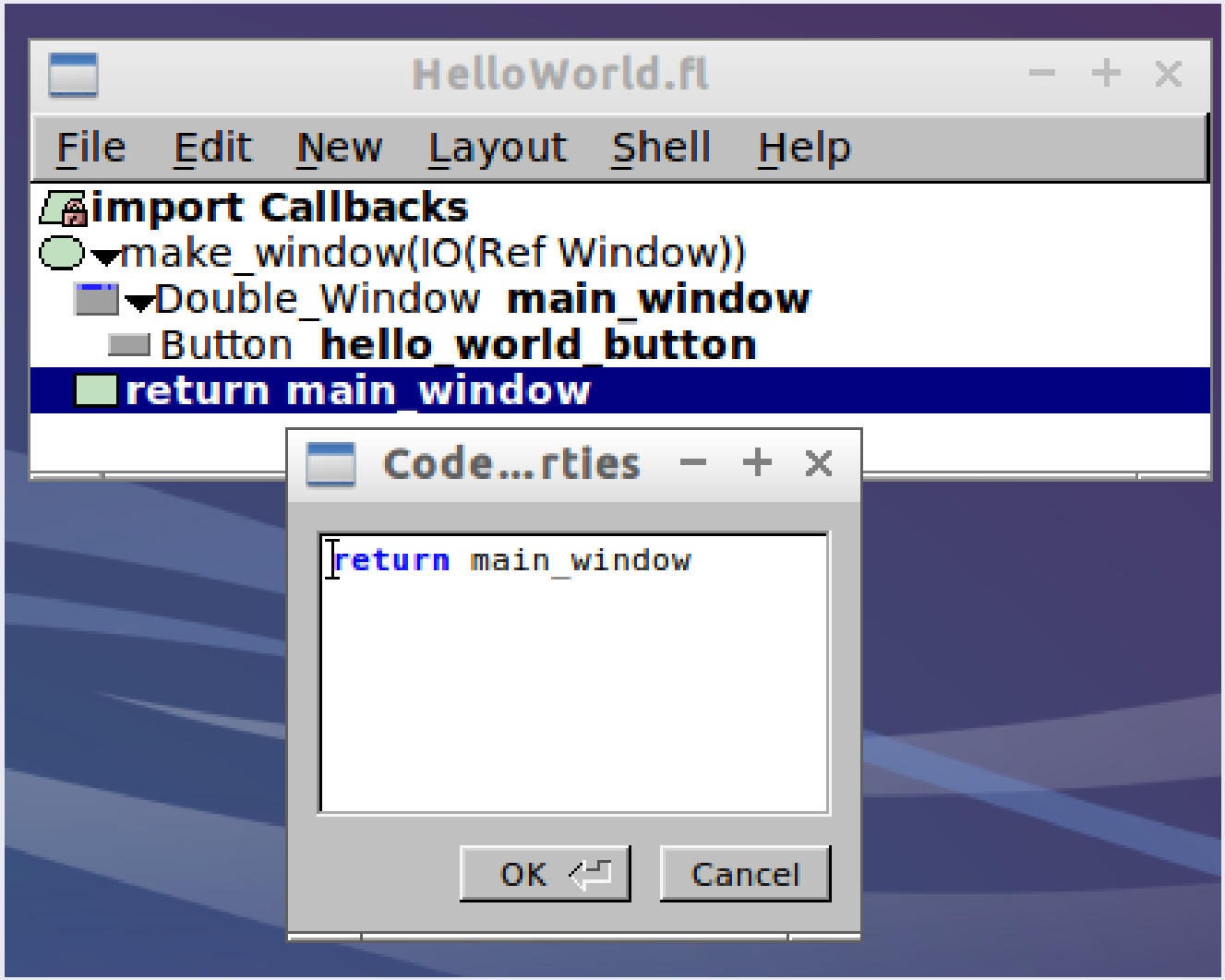
The type signature is inside the parens.

Window Creating Function



Fluid

Return Type

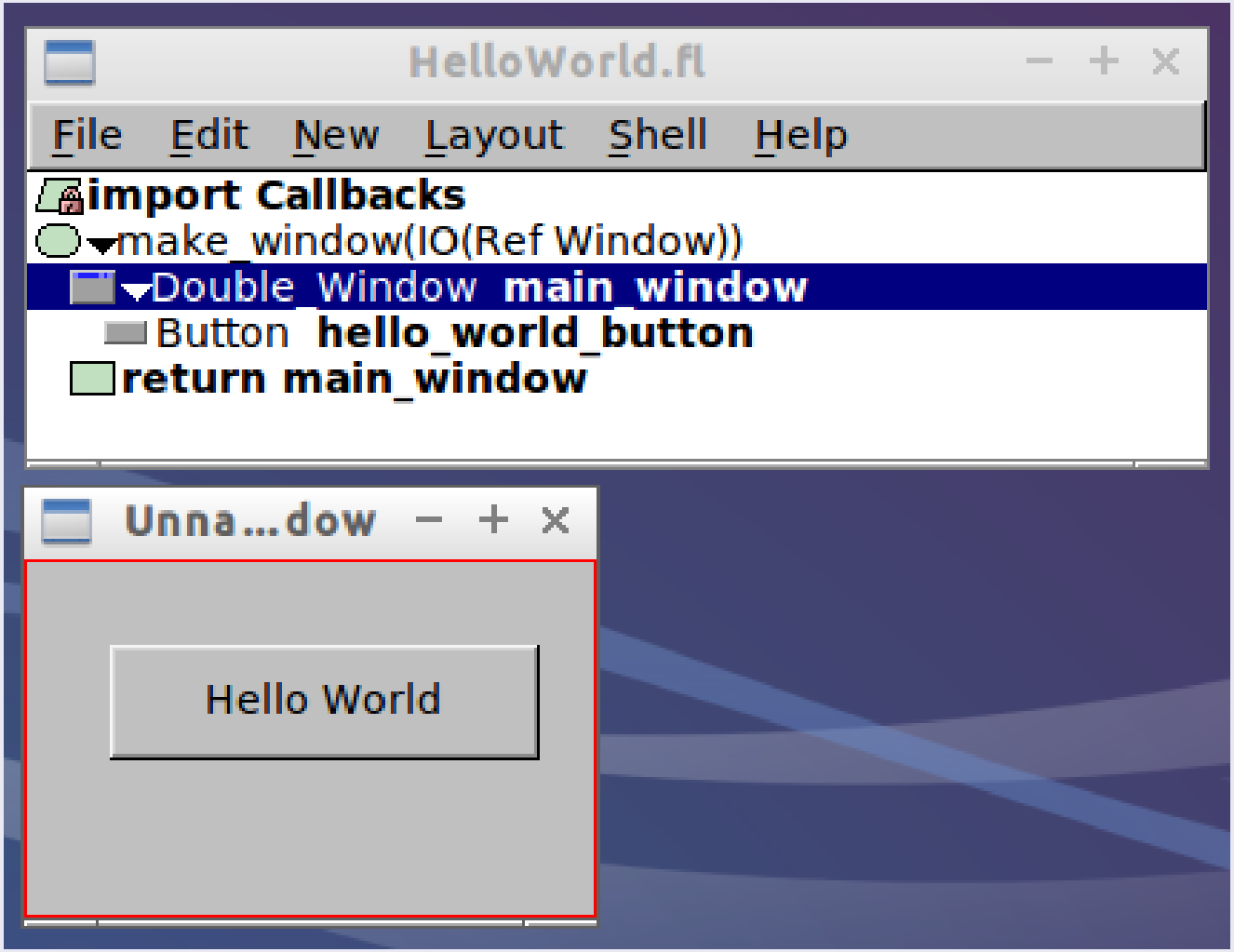


Fluid

- Preprocess Fluid Files in ‘Setup.hs‘

```
main :: IO ()
main = defaultMainWithHooks
         (simpleUserHooks {
             hookedPreProcessors =
              [("fl", ppFluidToHaskell)]})
ppFluidToHaskell = ...
```

The Main UI


HelloWorld.fl
File Edit New Layout Shell Help
import Callbacks
make_window(IO(Ref Window))
Double_Window main_window
Button hello_world_button
return main_window
Unna...dow
Hello World

Fluid

- Fluid Intermediate Format

```
...
decl {import Callbacks} {private local}
Function {make_window(IO(Ref Window))} {open
} {
  Fl_Window main_window {open
    xywh {815 469 200 125} type Double hide
  } {
    Fl_Button hello_world_button {
      label {Hello World}
      callback buttonCb selected
      xywh {30 30 150 40}
    }
  }
  code {return main_window} {}
}
```

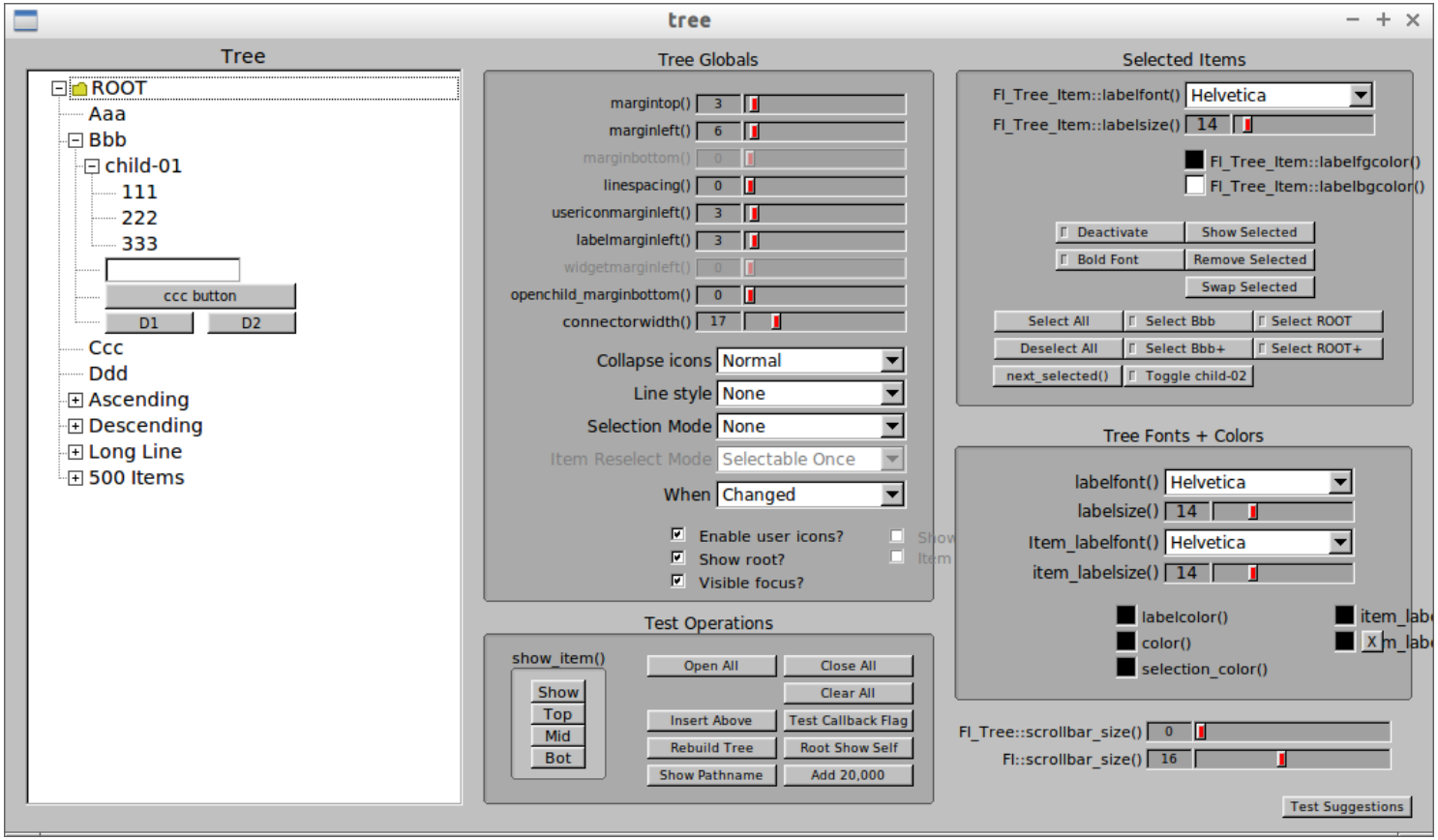
Fluid

- Haskell output

```
module HelloWorld where
...
import Callbacks
make_window :: IO(Ref Window)
make_window = do {
  main_window <-
      windowNew (Size (Width 200) (Height 125)) ..;
  begin main_window;
  hello_world_button <-
      buttonNew (toRectangle (30,30,150,40)) ...;
  setLabel hello_world_button "Hello World";
  setCallback hello_world_button buttonCb;
  end main_window;
  return main_window;
 }
```

Fluid Extras

- Capable of complicated UI's.
  - `fltkhs-fluid-tree` in `fltkhs-fluid-examples`

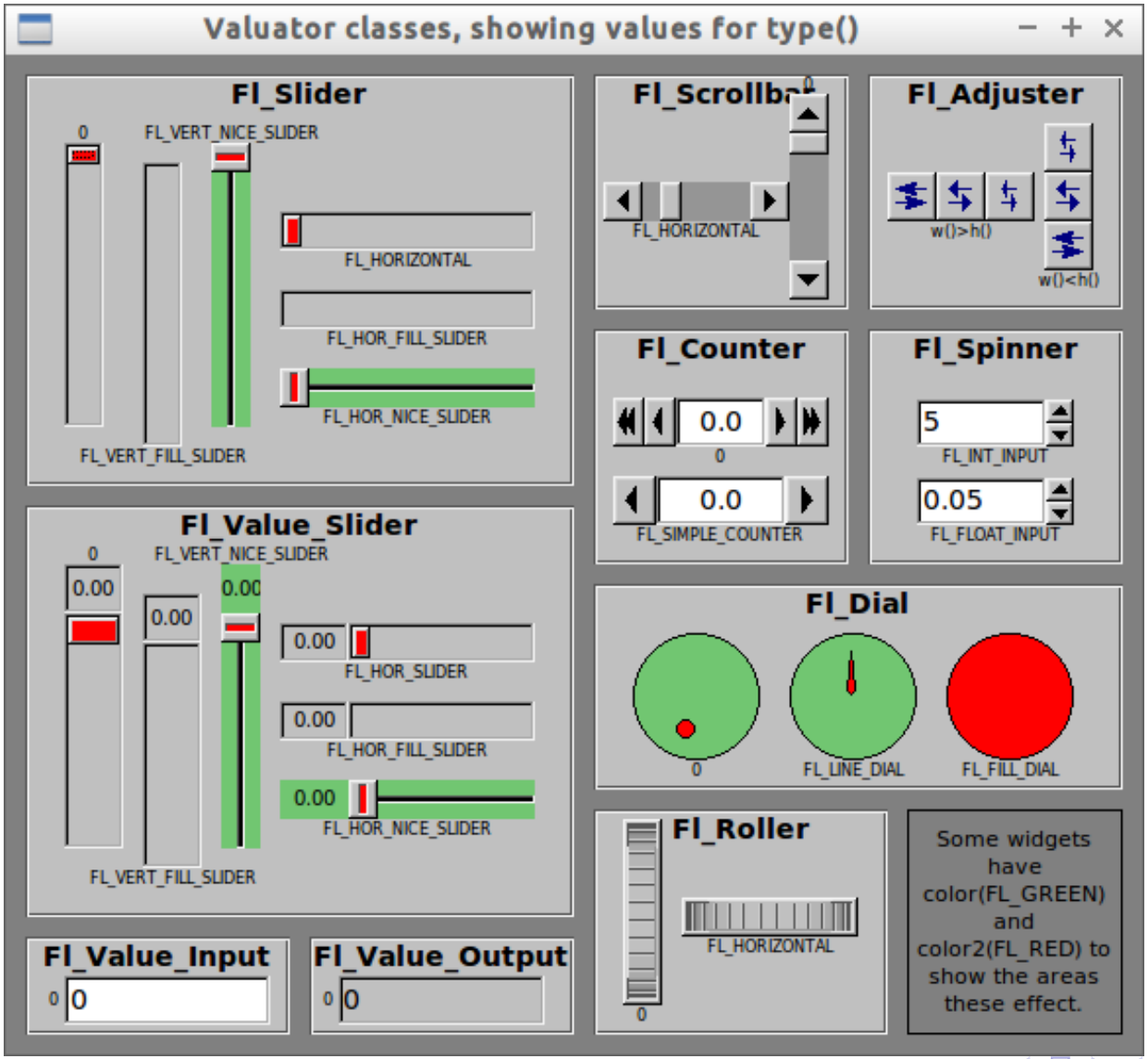


Fluid Extras

- Can add functions directly in Fluid!

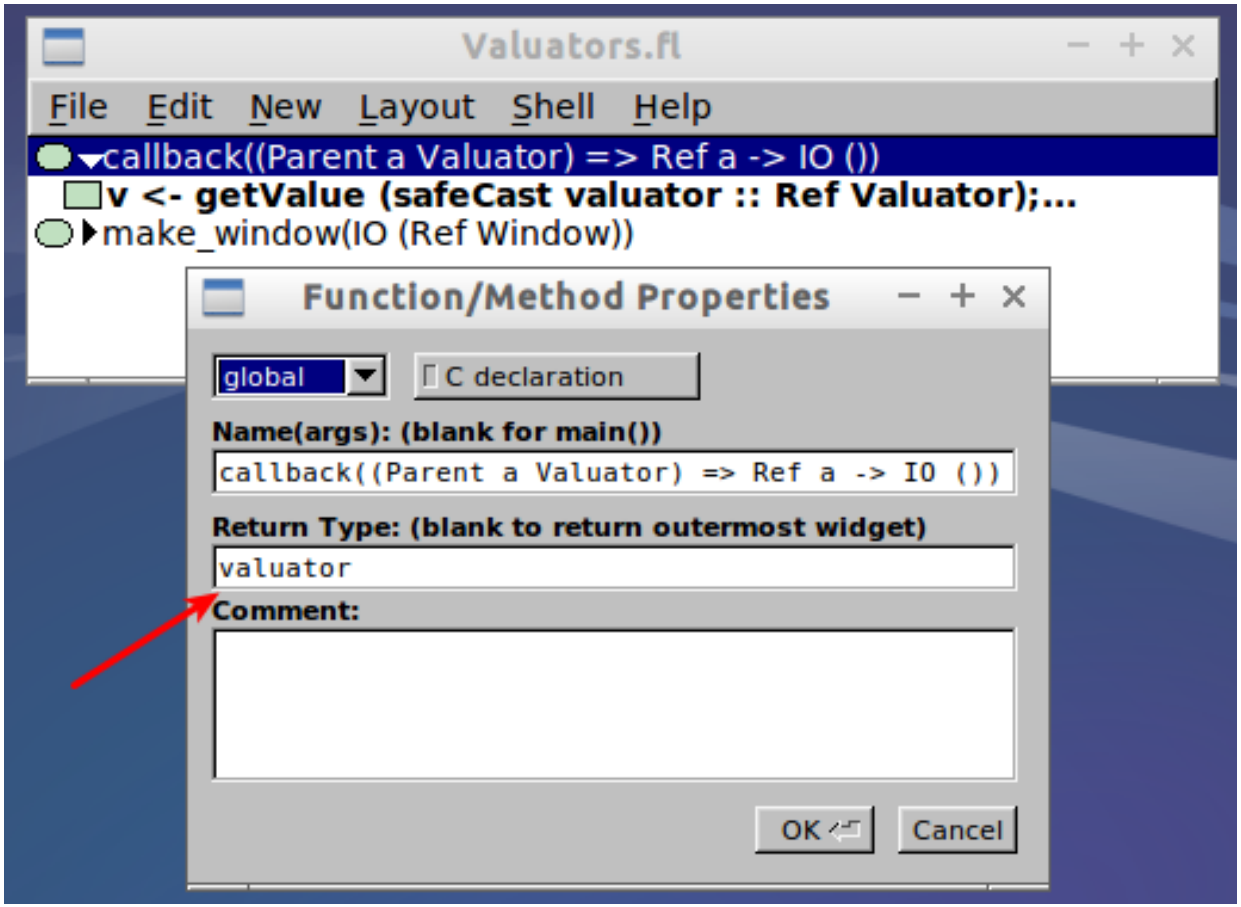
Fluid Extras

- The `fltkhs-fluid-valuators` demo, for example



Fluid Extras

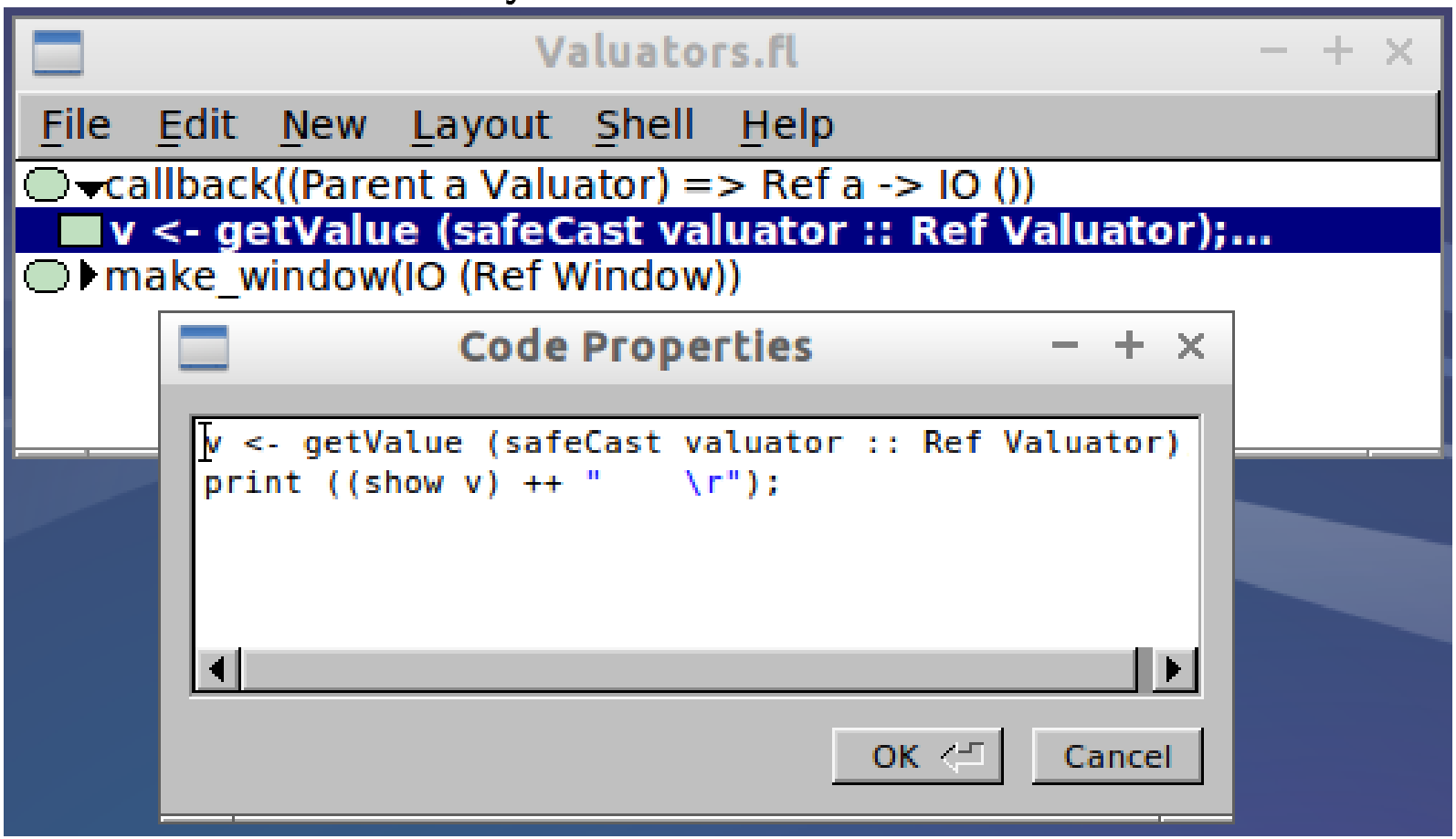
- Callback function in Fluid



  - `valuator` is the function argument
  - Super hacky, I know.

Fluid Extras

- Callback function body



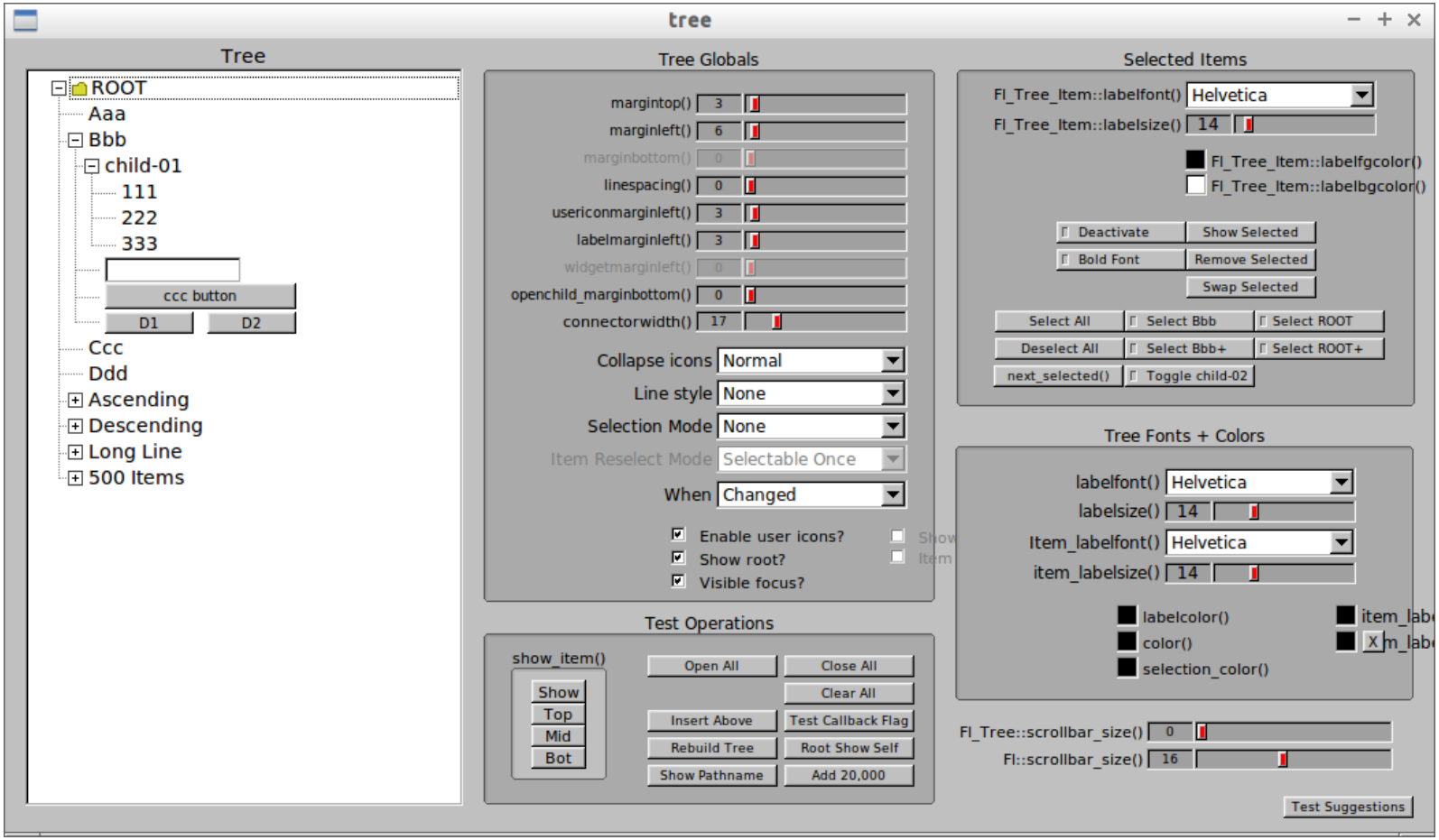
Fluid Extras

- Final output

```
module Valuators where
...
callback :: (Parent a Valuator) => Ref a -> IO ()
callback valuator =
   do {
    v <- getValue (safeCast valuator :: Ref Valuator);
    print ((show v) ++ "    \\r");;
  }
...
make_window :: IO (Ref Window)
make_window  =  ...
```

The bad . . .

- Compile times aren't great
- compile + link: 11-15 secs, REPL: 9 secs



wxHaskell

- Very elegant model for multiple dispatch!
- At the base:

```
data Object a = Object (Ptr a)
```

- A child of `Object`:

```
data CWidget a = CWidget
type Widget a = Object (CWidget a)
```

- `"type Widget a" -> "Object (Ptr (CWidget a))"`
- A child of `Widget`:

```
data CWindow a = CWindow
type Window a = Widget (CWindow a)
```

- `"type Window a" -> "Object (Ptr (CWidget (CWindow a)))"`

wxHaskell

- A function that takes a "Widget a"

```
print :: Widget a                                -> IO ()
print :: Object (Ptr (CWidget a                  )) -> IO ()
```

- Also accepts a "Window a"!

```
print :: Object (Ptr (CWidget (CWindow a))) -> IO ()
```

wxHaskell

- Extensible without recompiling!

```
data CMyWindow a = CMyWindow
type MyWindow a = Window (CMyWindow a)
```

- No typeclasses!
- Haskell 98?
- Awesome!

wxHaskell

- Can't handle changing arities.
  - eg. `setValue` -> `inputSetValue` / `valuatorSetValue`
- I went all in . . .
- Compile times were great.
- More and more inconsistent method calls

Depression

- Couldn't stand my own API.
- Depression.
- Walked away for months ...

HList

- Turned to HLists

```
data CObject parent; type Object = CObject ()
data CWidget parent; type Widget = CWidget Object
data CWindow parent; type Window = CWindow Widget
```

- `"type Window" -> "CWindow (CWidget (CObject ()))"`

HList

- A Hlist for functions too . . .

```
data SetValue a
data Print a
class Functions object functions
instance Functions Widget (SetValue (Print ()))
instance Functions Window (SetValue (Print ()))
```

- An instance for each implementation . . .

```
class Impl function object impl where
   run :: function -> (Ref object) -> impl
instance Impl (SetValue ()) (Widget ()) \
  (String -> IO ()) ...
instance Impl (SetValue ()) (Window ()) \
  (Int -> Int -> IO ())
```

HList

- a function to delegate . . .

```
setValue :: (FindFunction (SetValue ()) a impl) =>
            Ref a -> impl
```

- `FindFunction` also searches down the hierarchy.
- Essentially what's there now . . .
  - Slightly different in the codebase

HList

- Gives me multiple dispatch
- Litters the codebase with orphan instances
- More complicated

Transition

- Huge transition

Showing **28 changed files** with **1,387 additions** and **6,602 deletions**.

Showing **40 changed files** with **1,545 additions** and **6,481 deletions**.

Showing **9 changed files** with **979 additions** and **2,222 deletions**.

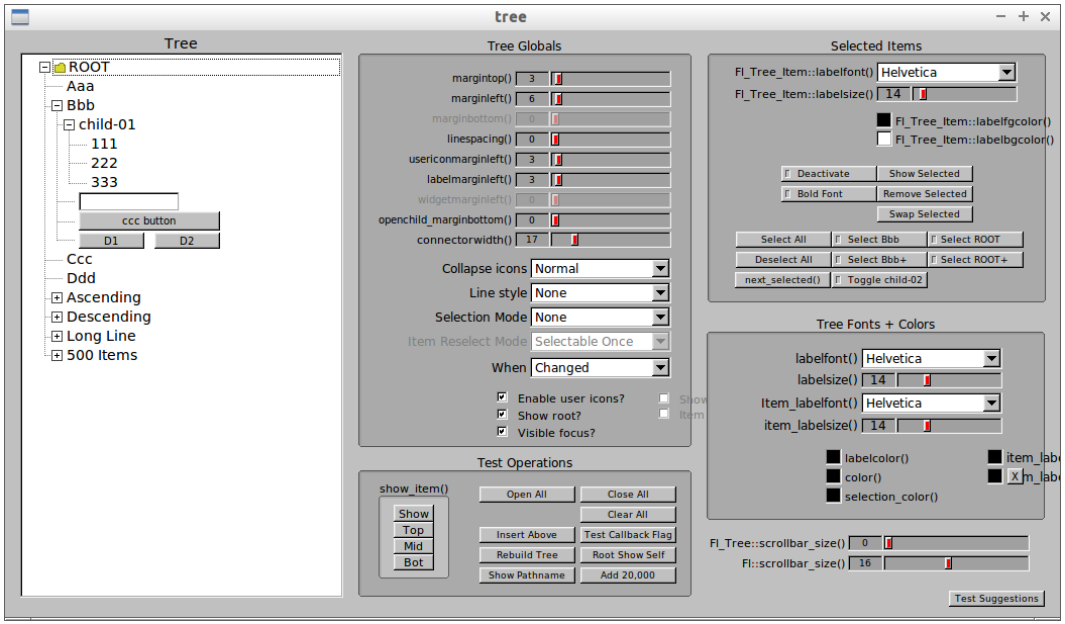
Showing **56 changed files** with **1,796 additions** and **1,811 deletions**.

HList

- Up the `context-stack`
- But smooth sailing . . .

HList

- And then few months later . . .



- compile + link: 12-15 minutes!
- Hello Darkness, my old friend . . .

HList

- `cabal build -v3` with a stop watch.
  - Can hez type-level profiler?
  - Half the time was spent in the `simplifier` phase
- Set some flags

```
ghc-Options: -fno-specialise \
             -fmax-simplifier-iterations=0 \
             -fsimplifier-phases=0
```

- 5 minutes!

HList

- Upgrade to closed type families
- No more upping `context-stack`!
- 15 secs, 9-10 in REPL!
- Still not great.
- `OverloadedRecordFields`, plz?

Why?

- Why?
  - No fuss native executables
    - Just throw something together
  - Re-use intuition from OO toolkits
  - Re-use documentation from FLTK
- Why not?
  - Definitely *retro*-looking.
    - Unlikely to change.
  - Compile times are not good
    - Very likely to change

Thanks!

- Thanks!
- Questions?