a Small Showcase of blucontrol

NOTE This article is written for blucontrol v0.7.0.0.

A customizable blue light filter

The main structure of blucontrol is the split into 3 individual environments. These environments `m` are implemented using type classes with `Monad m` constraints.

- `MonadControl`: Idle environment. Here you can handle exceptions from the other environments and set up stuff like the time interval.
- `MonadPrepareValue`: Prepare a value (RGB value for example) that can be applied.
- `MonadApplyValue`: Apply a value (RGB value most likely). This is the backend of the blue light filter.

The idea behind these different type classes is, that the user has complete control over which features end up in the configured application. It also allows the user to rewrite every small part in an xmonad-like configuration at `$XDG_CONFIG_HOME/blucontrol/blucontrol.hs`:

```haskell
{-# LANGUAGE TypeApplications #-}

module Main where

import Control.Monad (void)

import Blucontrol
import Blucontrol.Monad.ApplyValue.X
import Blucontrol.Monad.Control.Count
import Blucontrol.Monad.Control.Print
import Blucontrol.Monad.Control.Wait
import Blucontrol.Monad.PrepareValue.Linear
import Blucontrol.Value.RGB.Temperature

main :: IO ()
main = void $ blucontrol configControl
  where
    configControl = ConfigControl
      { runControl =
          runControlPrintT !>
          runControlCountT def !>
          runControlWaitT def
      , runPrepareValue =
          runPrepareValueLinearT @Temperature temperatureMap
      , runApplyValue =
          runApplyValueXTIO def
      }
    temperatureMap = 00:.00 ==> 4000
```

```
                    :| [ 08:.00 ==> 4600
                       , 12:.00 ==> 6600
                       , 18:.00 ==> 6000
                       ]
```

So what is actually going on here?

All the user has to do is set the 3 environments and the rest will be handled by the type class instances. The constrained type signature of the `blucontrol` function will make sure that everyone works in conjunction with one another.

The `ConfigControl` record takes the monad runners with type signature `m a → IO a`. Some of them take arguments and `def` can be used as a default most of the time.

If you are looking for alternatives you should look through the type class instances here for example.

Conclusion

In general you should look through the documentation of the Blucontrol module.

To get started I recommend sticking to the example configuration for the `runControl` and `runApplyValue` fields. `runPrepareValueLinearT` allows a time dependent gradient, which can be declared for different value-types such as `Temperature` or `RGB Word8` (8-bit red, green and blue). Another possibility for `runPrepareValue` would be a geolocation-based day/night or sunrise/sunset cycle. The implementation for this particular idea is left as an exercise to the ambitious reader. ;)

Documentation is lacking for some modules but the source code is quite readable and well structured. I tried to stick to the xmonad philosophy, but also used some newer features of GHC. So you should either be confident in writing Haskell or interested in learning new language extensions if you want to understand the source.

The preferred way of installation is by using the nix package manager. Make sure to install the wrapper `pkgs.blucontrol-with-packages` from the nixpkgs-unstable channel.

More detailed instructions can be found on GitHub.