# Elm 0.17

Michael Twomey @ Elm Dublin May 2016

https://github.com/micktwomey/elm-0.17-dublin-elm-may-2016

# What is Elm?

# Really Short Introduction

Elm is a functional language that compiles to JavaScript.

Noted for helpful compiler errors.

ML style language.

# Basic Example

## (Pardon the formatting to fit)

```
import Html
import Html.App

main = Html.App.beginnerProgram
    { model = model
    , update = update
    , view = view }

type alias Model = { message : String}
model = { message = "Hello World!" }

type Msg = Nothing
update msg model = model

view model = Html.h1 [] [ Html.text model.message ]
```

# The Pattern

1. Model
2. Update
3. View

(AKA Redux in ReactJS land)

# What's New in 0.17?

# No More Signals 😄

## foldp and mailboxes are gone too

## Subscriptions are the new hotness

# Subscription Example

```
import Html
import Html.App
import Time

main = Html.App.program
    { init = init, update = update , view = view
    , subscriptions = subscriptions
    }

type alias Model = { time : Time.Time}
init = ( { time = 0 }, Cmd.none)

type Msg = Tick Time.Time
update msg model =
    case msg of
        Tick newTime -> ( { model | time = newTime } , Cmd.none)

subscriptions : Model -> Sub Msg
subscriptions model = Time.every Time.second Tick

view model = Html.h1 [] [ Html.text ("The time is " ++ toString model.time) ]
```

# Just the Subscription Bits

```
main = Html.App.program
    { ...
    , subscriptions = subscriptions }

subscriptions : Model -> Sub Msg
subscriptions model =
    Time.every Time.second Tick

update : Msg -> Model -> Model
update msg model =
    case msg of
        Tick newTime -> ...
        SomeActionFromUser -> ...
```

# SVG Version

## (from Elm docs)

```
main = Html.App.program
    { init = init, update = update , view = view
    , subscriptions = subscriptions }

type alias Model = Time.Time
init = ( 0 , Cmd.none)
update msg model =
    case msg of
        Tick newTime -> ( newTime , Cmd.none)
type Msg = Tick Time.Time
subscriptions model = Time.every Time.second Tick

view model =
  let
    angle = turns (Time.inMinutes model)
    handX = toString (50 + 40 * cos angle)
    handY = toString (50 + 40 * sin angle)
  in
    svg [ viewBox "0 0 100 100", width "300px" ]
      [ circle [ cx "50", cy "50", r "45", fill "#0B79CE" ] []
      , line [ x1 "50", y1 "50", x2 handX, y2 handY, stroke "#023963" ] []
      ]
```

# The only difference is the view

```elm
-- HTML
view model =
    Html.h1 [] [ Html.text ("The time is " ++ toString model.time) ]

-- SVG
view model =
  let
    angle = turns (Time.inMinutes model)

    handX = toString (50 + 40 * cos angle)

    handY = toString (50 + 40 * sin angle)
  in
    svg [ viewBox "0 0 100 100", width "300px" ]
      [ circle [ cx "50", cy "50", r "45", fill "#0B79CE" ] []
      , line [ x1 "50", y1 "50", x2 handX, y2 handY, stroke "#023963" ] []
      ]
```

# More 0.17 Changes

# No more Signal.Address (simpler code)

```
-- 0.16
view : Signal.Address Action -> Model -> Html
view address model =
  div []
    [ button [ onClick address Decrement ] [ text "-" ]
    , div [ countStyle ] [ text (toString model) ]
    , button [ onClick address Increment ] [ text "+" ]
    ]

-- 0.17
view : Model -> Html Msg
view model =
  div []
    [ button [ onClick Decrement ] [ text "-" ]
    , div [ countStyle ] [ text (toString model) ]
    , button [ onClick Increment ] [ text "+" ]
    ]
```

→ Effects are now Cmd

→ Action is now Msg

→ StartApp is now Html.App

→ Many packages moved into core

→ Graphics.* now lives in evancz/elm-graphics

# Msg Example (clicking buttons)

```elm
type Msg
    = SayHello
    | SayGoodbye

update msg model =
    case msg of
        SayHello ->
            { model | message = "Hello there!" }

        SayGoodbye ->
            { model | message = "Goodbye!" }


view model =
    Html.div []
        [ Html.button [ Html.Events.onClick SayHello ] [ Html.text "Say Hello!" ]
        , Html.button [ Html.Events.onClick SayGoodbye ] [ Html.text "Say Goodbye!" ]
        , Html.h1 [] [ Html.text model.message ]
        ]
```

# Effect Managers Added

## Moves the hard work to the library author

# Example: Web Sockets

# Simple API

```
WebSocket.send "ws://echo.websocket.org" input
WebSocket.listen "ws://echo.websocket.org" NewMessage
```

(Effect manager handles connection management, errors, re-connecting for you.)

# Websocket Example

```
import Html exposing (Html, div, text)
import Html.App as Html
import WebSocket

main = Html.program
    { init = init, view = view
    , update = update, subscriptions = subscriptions }

echoServer = "ws://echo.websocket.org"

type alias Model =
  { input : String , messages : List String }

init = ( Model "" [], WebSocket.send echoServer "Hello" )

type Msg = NewMessage String

update msg {input, messages} =
  case msg of
    NewMessage str -> (Model input (str :: messages), Cmd.none)

subscriptions model =
  WebSocket.listen echoServer NewMessage

view model =
  div [] (List.map viewMessage (List.reverse model.messages))

viewMessage msg = Html.p [] [ text msg ]
```

# Graphics Demoted to External library

HTML app first, with possible embedded Graphics.

Html.toElement is now Element.toHtml.

No more Graphics.Input (use HTML forms).

```
import Color
import Html
import Html.App
import Element
import Collage exposing (..)

main =
    Html.App.beginnerProgram { model = model, update = update, view = view }

type alias Model = String
model = "Hello"

type Msg = Nothing
update msg model =
    model

view model =
    Html.div []
        [ collage 320 200
            [ group
                [ circle 50 |> filled Color.charcoal
                , circle 40 |> filled Color.yellow
                ]
            ] |> Element.toHtml
        ]
```

```elm
-- 0.16
view : Model -> Element
view reading =
    collage 320 200
        [ group
            [ circle 50 |> filled Color.charcoal
            , circle 40 |> filled Color.yellow
            ]
        ]

-- 0.17
view : Model -> Html Msg
view model =
    Html.div []
        [ collage 320 200
            [ group
                [ circle 50 |> filled Color.charcoal
                , circle 40 |> filled Color.yellow
                ]
            ] |> Element.toHtml
        ]
```

# Graphics Clock Example

```
Html.div []
    [ collage 640
        480
        [ group
            [ circle 150 |> filled Color.darkBlue
            , path
                [ ( 0, 0 )
                , ( handX, handY )
                ]
                |> traced
                    { defaultLine
                        | width = 5
                        , color = Color.darkCharcoal
                    }
            ]
        ]
        |> toHtml
    , Html.h1 [] [ Html.text ("The time is " ++ time) ]
    ]
```

http://elm-lang.org/blog/farewell-to-frp

http://guide.elm-lang.org

http://www.lambdacat.com/migrating-from-elm-0-16-to-0-17-from-startapp/

http://noredink.github.io/posts/signalsmigration.html

https://github.com/micktwomey/elm-0.17-dublin-elm-may-2016