

# Elm Integration

## Functional Programming

Jens Egholm Pedersen and Anders Kalhauge



Spring 2017

Git practices

Advanced Elm

Scaling Elm

Integrating Elm

Elm in JavaScript

Exercise 2

JavaScript in Elm

Why integrating?

Elm real-life examples

A version control system (VCS)

A **version control system** (VCS) ... also known as a **source control system**.

A version control system (VCS) ... also known as a **source control system**. ... **source**.

A file to **ignore** things that does not belong in source control systems.

A file to **ignore** things that does not belong in source control systems.

□ Binaries

A file to **ignore** things that does not belong in source control systems.

- ❑ Binaries
- ❑ Compressed files



A file to **ignore** things that does not belong in source control systems.

- ❑ Binaries
- ❑ Compressed files
- ❑ Logs


A file to **ignore** things that does not belong in source control systems.

- ❑ Binaries
- ❑ Compressed files
- ❑ Logs
- ❑ Password files and secret keys

A file to **ignore** things that does not belong in source control systems.

- ❑ Binaries
- ❑ Compressed files
- ❑ Logs
- ❑ Password files and secret keys
- ❑ Weird Mac files like .DS\_Store, .Thrashes etc.

 ELM-part

 REST-part/restapi

 elm-stuff

---

 README.md

---

 elm-package.json

 .vscode

---

 bin

---

 node\_modules

---

 public/stylesheets

---

 routes

---

 views

---

 app.js

 package.json

---

# Why is this important?

Size with binaries 14000Kb

# Why is this important?

Size with binaries	14000Kb
Size without binaries	44Kb



## Removed binaries #1

[Edit](#)[Open](#) Jegp wants to merge 2 commits into `mathiasj64:master` from `Jegp:master`

Conversation 0 Commits 2 Files changed 1,330

+50 -234,470



Jegp commented just now



1330 files changes. Auch



Jegp added some commits a minute ago



Removed binaries

Verified

81da549



Added gitignore

Verified

53895fd

Add more commits by pushing to the `master` branch on `Jegp4_FP_ELM_REST`.

### Reviewers

No reviews

### Assignees

No one assigned

### Labels

None yet

### Projects

1. Do not include binaries

1. Do not include binaries
2. If you do, remove them!

1. Do not include binaries
2. If you do, remove them!

```
git rm -cached -r directory
```



- types, records, immutability

- types, records, immutability
- **Elm architecture**
  - Model - View - Update

- types, records, immutability
- **Elm architecture**  
Model - View - Update
- Asynchronous calls with HTTP



- types, records, immutability
- **Elm architecture**  
Model - View - Update
- Asynchronous calls with HTTP
- Timed events with **subscriptions**

- Scaling elm with cool functional programming

- Scaling elm with cool functional programming
- Integrating Elm in your normal JavaScript

- Scaling elm with cool functional programming
- Integrating Elm in your normal JavaScript
- Real-life Elm examples

Clone the elm-exercises from  
cphbus-functional-programming

`https://github.com/cphbus-functional-programming/  
elm-exercises`

Work on the `toggle.elm` and `radio.elm` files in the `advancedelm` folder

**Goal:** The view code in both examples can be simplified drastically. Figure out how and implement it!

... or **Reusable views**

... or **Reusable views**

- Decouple functionality

... or **Reusable views**

- Decouple functionality (not state)



... or **Reusable views**

- ❑ Decouple functionality (not state)
- ❑ Reuse components without knowing the context

Reusable views gives us independent code.

Reusable views gives us independent code.

Independent code can be neatly structured!

Reusable views gives us independent code.

Independent code can be neatly structured!

```
module MyView exposing (..)

checkbox : String -> Msg -> Html Msg
checkbox ...
```

Reusable views gives us independent code.

Independent code can be neatly structured!

```
module MyView exposing (..)  
  
checkbox : String -> Msg -> Html Msg  
checkbox ...
```

```
import MyView  
  
view : Msg -> Model -> (Model, Cmd Msg)  
view =  
    ...  
    MyView.checkbox
```



Elm compiles to JavaScript

Elm compiles to JavaScript

No problem! We can work with JavaScript like we normally do!



Elm compiles to JavaScript

No problem! We can work with JavaScript like we normally do!

- How to embed Elm in JavaScript

Elm compiles to JavaScript

No problem! We can work with JavaScript like we normally do!

- How to embed Elm in JavaScript
- How to communicate between Elm and JavaScript

```
elm-make Main.elm --output=main.html
```

```
elm-make Main.elm --output=main.html
```

```
elm-make Main.elm --output=main.js
```

```
elm-make Main.elm --output=main.html
```

```
elm-make Main.elm --output=main.js
```

```
<script src="main.js"></script>
```

```
elm-make Main.elm --output=main.html
```

```
elm-make Main.elm --output=main.js
```

```
<script src="main.js"></script>
```

Full screen:

```
<script>  
    var app = Elm.Main.fullscreen();  
</script>
```

```
elm-make Main.elm --output=main.html
```

```
elm-make Main.elm --output=main.js
```

```
<script src="main.js"></script>
```

Full screen:

```
<script>
  var app = Elm.Main.fullscreen();
</script>
```

Inside an element:

```
<div id="main"></div>
<script>
  var node = document.getElementById('main');
  var app = Elm.Main.embed(node);
</script>
```

Clone the elm-exercises from  
cphbus-functional-programming

`https://github.com/cphbus-functional-programming/  
elm-exercises`

Work on the password.html and include using  
`Elm.Password.fullscreen()`

**Goal:** When you integrate the password.opt.js file into the password.html file and tell me the content, you get a break!



How can we communicate with Elm?

How can we communicate with Elm?

How does Elm do it internally?

How can we communicate with Elm?

How does Elm do it internally?

What happens when a user clicks a button?

How can we communicate with Elm?

How does Elm do it internally?

What happens when a user clicks a button?

Conclusion: Elm is event-driven!

```
port module Main exposing (...)  
...
```



```
port portedFunction : String -> Cmd String
```

```
port portedFunction : String -> Cmd String
```

```
var node = document.getElementById('main');  
var app = Elm.Main.embed(node);  
app.ports.portedFunction...
```



```
port portedFunction : String -> Cmd String
```

```
var node = document.getElementById('main');  
var app = Elm.Main.embed(node);  
app.ports.portedFunction...
```

```
app.ports.portedFunction  
  .subscribe(function(input) { ... });
```

```
port module Main exposing (..)
...
```

```
port module Main exposing (..)
...
```

```
port portedSubscription : (String -> msg) -> Sub msg
```

```
port module Main exposing (...)  
...
```

```
port portedSubscription : (String -> msg) -> Sub msg
```

```
var node = document.getElementById('main');  
var app = Elm.Main.embed(node);  
app.ports.portedSubscription...
```

```
port module Main exposing (...)  
...
```

```
port portedSubscription : (String -> msg) -> Sub msg
```

```
var node = document.getElementById('main');  
var app = Elm.Main.embed(node);  
app.ports.portedSubscription...
```

```
app.ports.portedSubscription.send("hullubullu");
```

Clone the elm-exercises from  
cphbus-functional-programming

[https://github.com/cphbus-functional-programming/  
elm-exercises](https://github.com/cphbus-functional-programming/elm-exercises)

Work on the `Event.elm` and `Subscription.elm` files.

**Goal 1:** Read and execute instructions in `Event.elm`

**Goal 2:** Read and execute instructions in `Subscription.elm`

# Why integrating?

*The correct path is to first use Elm in a small experiment. If the experiment goes bad, stop it! If it goes great, expand the experiment a bit more. Then just repeat this process until you are using Elm or not!*



`http://debug.elm-lang.org/edit/Mario.elm`

`http://unsoundscapes.com/elm-flatris.html`