Emy Documentation

Getting Started

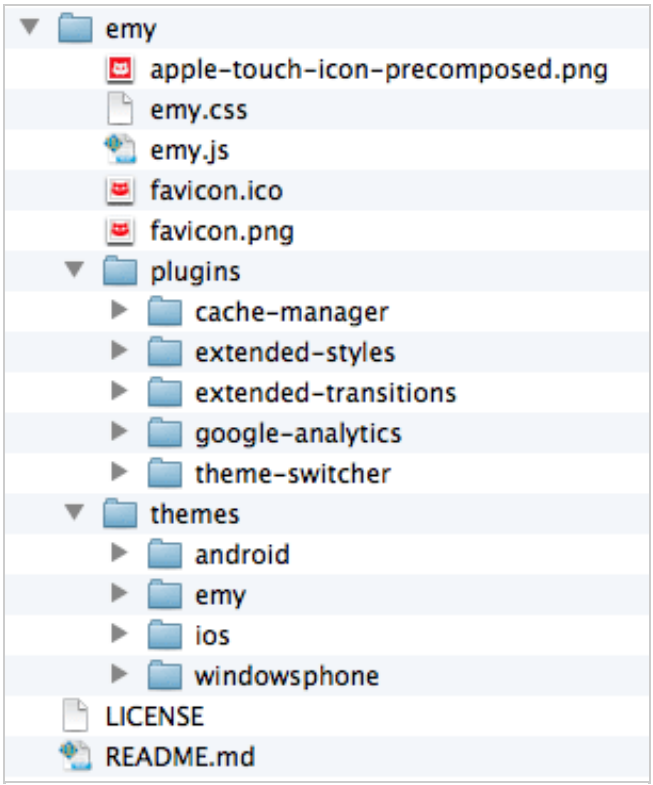
You probably never heard about it, but Emy is initially a fork of a library called "iUI", created by Joe Hewitt at the first iOSDevCamp. As he notes in the iUI introductory blog post, it's intended to convert "ordinary standards-based HTML" into a polished UI that looks like a native iPhone app (this is not 2007 anymore so let's rather talk about "Mobile webapps"). if you are already using iUI, we created Switch to Emy from iUI for you.

Introduction

First, thanks for your interest in Emy Mobile web librarY!

There is no installation for Emy, it consists of a combination of html, css, javascript & image files. All you will need to do is download the latest archive and unpack/extract it in the location you want.

Once downloaded, "emy" folder tree should look like the following screen:



- *emy.css* is the main CSS file, containing only its "views“ controlling mechanism (all the rest belongs in the theme).
- *emy.js* is the core Javascript file.
- *theme* folder contains themes ready to use. They all consists of a "main.css" file & other files needed (images, icons, webfont,...) which deals with colors, fonts, margins, padding, ...

- *apple-touch-icon-precomposed.png* is only used by Apple devices when user taps "add to homescreen" (optional)
- *favicon.png* & *favicon.ico* are used by browsers for different purpose (icon next the the url, bookmark, ... optional)
- *plugins* folder contains add-ons for Emy, for features not needed for basic scenario (optional)

Let's do some code

In order to start playing with Emy, you first need to create a basic HTML file that pulls in Emy's CSS and Javascript, plus a couple of meta tags. Here's what it looks like:

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="utf-8">
      <title>My app</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">

      <link rel="stylesheet" href="emy/emy.css" type="text/css">
      <link rel="stylesheet" href="emy/themes/ios/main.css" type="text/css">
      <script type="application/x-javascript" src="emy/emy.js"></script>

    </head>
    <body>
      ...
    </body>
  </html>
```

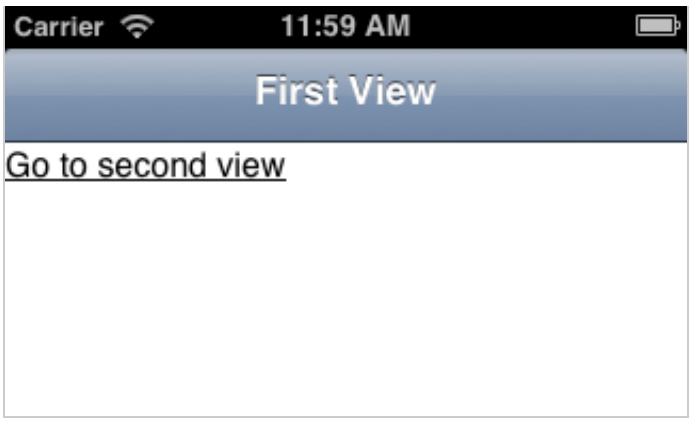
Everything else will be done inside the body tag. Emy is a single-page application library, which means all "views" and the "toolbar" are part of the HTML document on load. The main difference between Emy and other mobile web libraries is the way the toolbar is managed. The toolbar sits at the top of your app's screen and serves as the main menu for navigation and content titles.

Where other frameworks require you to redefine a toolbar for every individual page/screen, Emy keeps the toolbar fixed and changes its title automatically (based on the value of the data-title attribute of the active screen).

The basic type of view in Emy is the panel-type view, which must be a top-level section element (i.e. directly under the body element) with a "panel" class. Let's start with a couple of views and a toolbar.

```
<body>

    <header class="toolbar">
      <div><a id="backButton" class="button" href="#"></a></div>
      <h1 id="viewTitle"></h1>
    </header>

    <section id="first" data-title="First View" selected="true">
        <a href="#second">Go to second view</a>
    </section>

    <section id="second" data-title="Second View" class="panel">
      My second view.
    </section>

</body>
```

This should ends up looking something like this:



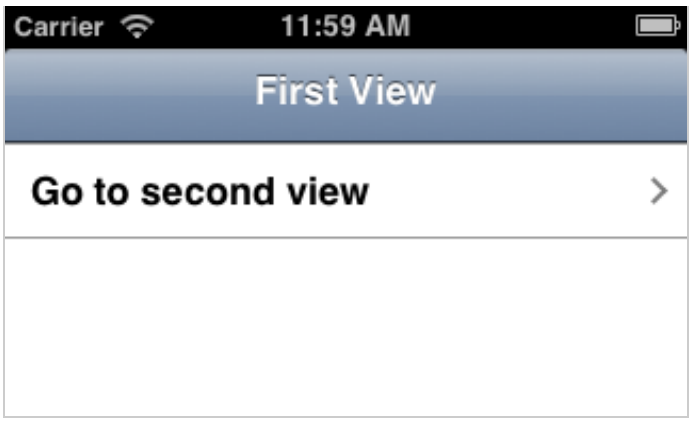
Typical rookie mistake is to forget *selected="true"* on the first view to display by default (you won't see anything if not set).

As you can, the toolbar renders "First View". Emy gets the *data-title* attribute of the current view to populate this *h1* element. When user navigates/slides to a new view, it automatically update this *h1* element text by this new view *data-title* value. (advice: keep it short !)

While this technically works, the link doesn't exactly look like a native app, nor does it comply with the Apple Human Interface Guidelines. For that, we should put the link in a table view. In Emy, a table view is an unordered list (ul), like this:

```
<body>

    <header class="toolbar">
      <div><a id="backButton" class="button" href="#"></a></div>
      <h1 id="viewTitle"></h1>
    </header>

    <section id="first" data-title="First View" selected="true">
       <ul>
          <li><a href="#second">Go to second view</a></li>
       </ul>
    </section>

    <section id="second" data-title="Second View" class="panel">
      My second view.
    </section>

</body>
```

This should ends up looking something like this:



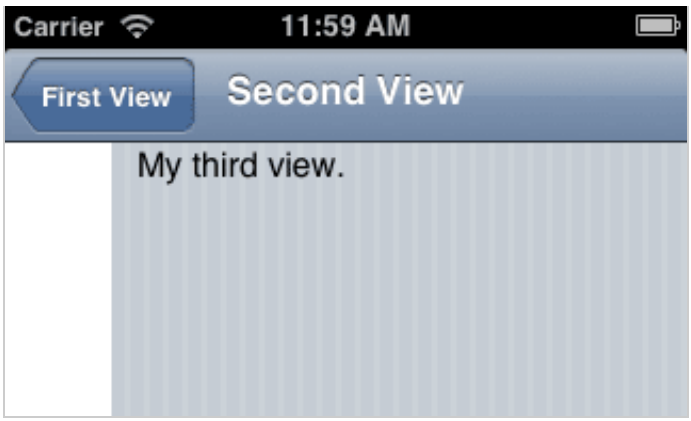
Of course, it doesn't do you a whole lot of good to have two screens without a way to navigate between them.

Navigate between views is easy, since you just need a link to its ID prefixed with a number sign (aka anchor or hash). To go to the second view, which has an ID equals to "second", i just need a link to "#second".

Let's add a link, a third screen and a couple of IDs:

```
<body>

    <header class="toolbar">
      <div><a id="backButton" class="button" href="#"></a></div>
      <h1 id="viewTitle"></h1>
    </header>

    <section id="first" data-title="First View" selected="true">
        <ul>
            <li><a href="#second">Go to second view</a></li>
        </ul>
    </section>

    <section id="second" data-title="Second View">
      My second view.<br>
      Go to <a href="#third">third view</a>
    </section>

    <section id="third" data-title="Third View" class="panel">
      My third view.
    </section>

</body>
```

Now you should have a link on your first screen that does a smooth sideways animation to the second screen, which should look like this:



Congratulations! You've just created your first, working Emy application!
This sample app code is available here. You can also test & edit it live on codio.

You may have noticed there is a piece of code in there for a "backButton"but no back button appears in this page. Just like the toolbar itself, Emy takes care of the back button and shows it only when needed. If there is no previous screen in your navigation history, the back button is hidden. It shows up automatically and is titled using the previous screen's *data-title* attribute value.

Advanced navigation

On-demand / External file loading

Navigating from a view to another is very simple using IDs. But if your app is made of dozens (or hundreds) of views containing images or a lot of texts, the initial loading time might quickly be a problem. Some of your views might also need to be generated on-demand when the user navigates to it.

To load an external view or a group of views from an external file, just put its URL in the href attribute, just like for regular web link. The two following files are 1) the main view 2) the external view.

```
<body>

    <header class="toolbar">
      <div><a id="backButton" class="button" href="#"></a></div>
      <h1 id="viewTitle"></h1>
    </header>

    <section id="first" data-title="First View" selected="true">
       <ul>
          <li><a href="view.frag">Load and go to an external view</a></li>
       </ul>
    </section>
```

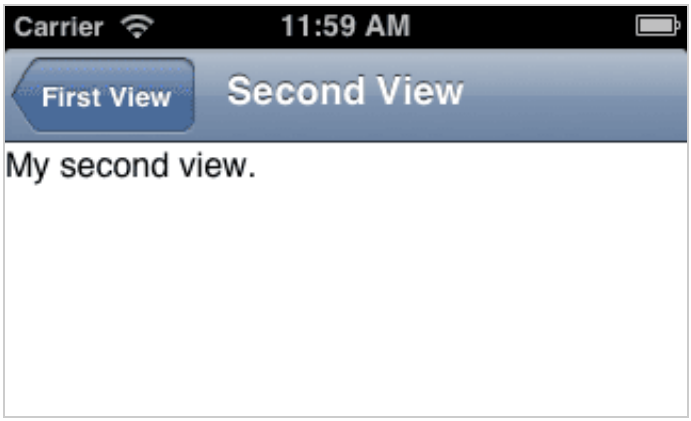
```
</body>
```

index.html

```
<section id="second" data-title="Second View">
    My second view.
</section>
```

view.frag

Emy loads asynchronously the content of the external file, adds it to the DOM and do a transition from the current view to this newly loaded view. While loading, Emy does set the selected attribute on the tapped link to "progress" rather than "selected". It changes the arrow by a progress spinning loader (once loaded, "progress" is replaced by "selected").



This sample navigation code is available here (and view.frag)

This is very nice for a database-driven webapp since this link can be view.php, view.aspx, including get parameters (ex: product.php?id=1234).

What really is important is that this DOM fragment is not limited to one single view, but can be how many views you want. Emy will always show the first one by default, using its ID attribute value. But this means all others would already be in the DOM, ready to be used.

```
<section id="second" data-title="Second View">
    <a href="#third">Go to third view</a>
  </section>

  <section id="third" data-title="Third View">
    <a href="#third">Go to fourth view</a>
  </section>

  <section id="fourth" data-title="Fourth View">
    My fourth view.
</section>
```

views.frag

A fragment HTML document can be a static file, but of course can also be a server-side script. On the following example, the fragment contains a first loop to fill the first view with a list of links to products, and a second loop to generate a complete view for each product. First "#products" view will be shown, while other views will be already loaded in the DOM too.

```
<?php $products = Products->getAll(); ?>
   <section id="products" data-title="Products list">
      <ul>
      <?php foreach($products as $item) {
            echo '<li><a href="#product'.$item['id'].'">'.$item['title'].'</a></li>';
      } ?>
      </ul>
   </section>

<?php foreach($products as $item) { ?>
   <section id="product<?= $item['id'] ?>" data-title="<?= $item['title'] ?>">
      <?= $item['title'] ?>
      ...
   </section>
<?php } ?>
```

product.php

Generate new views locally in Javascript

At some point, a server-side approach is not the right solution. Offline-capable apps for example, or when you store datas in local storages (LocalStorage, WebSQL, IndexedDb, ...). Anyway, feel free to generate your views in Javascript via `document.createElement` or as a string variable, and insert them inside the DOM via the "emy.insertViews()" method.

```
var myView = document.createElement('section');
myView.setAttribute('id','mynewview');
myView.setAttribute('data-title','My New View');
myView.innerHTML = 'Hello View !';
emy.insertViews(myView);
```

For more information on this method, take a look at emy.insertViews() in the Core documentation.

If you have a lot of elements to create/insert for you app, you might want to give our "element" plugin a look, and specially its "createElement" function which can resume the code above to 2 lines as the following:

```
var myView = emy.createElement({'section':{'id':'mynewview','data-title':'My New View','_text':'Hello View !'}});
emy.insertViews(myView);
```

Please note that in both cases, this method has a second parameter as `emy.insertViews(fragment, go);`, where "go" is `true` by default but can be set to `false` if you just want to insert the view(s) inside the DOM without navigating to its first node.

Load more items to a current list

You might also want to add some list items to a current view's list. This is mostly seen as a "Load more" link (but might be used anywhere else actually). By doing so, it loads its content from an external file just like in the previous example, but replaces the link element by what's in this external file.

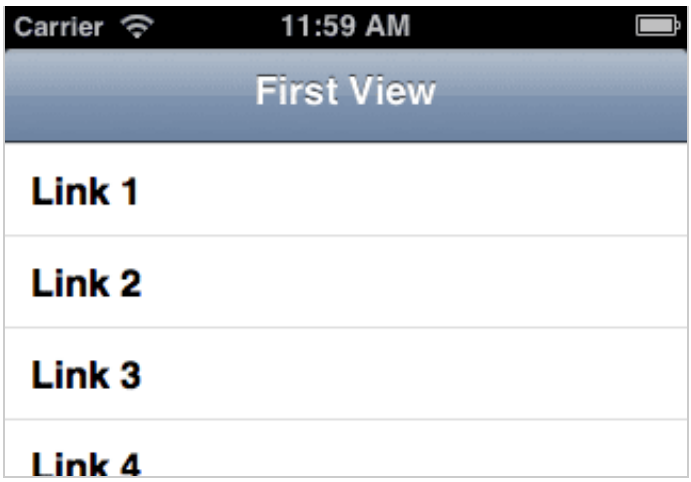
```
<body>

    <header class="toolbar">
      <div><a id="backButton" class="button" href="#"></a></div>
      <h1 id="viewTitle"></h1>
    </header>

    <section id="first" data-title="First View" selected="true">
       <ul>
          <li><a href="#">Link 1</a></li>
          <li><a href="more-links.frag" target="_replace">Load more links</a></li>
       </ul>
    </section>

</body>
```

index.html

```
<li><a href="#">Link 2</a></li>
<li><a href="#">Link 3</a></li>
<li><a href="#">Link 4</a></li>
```

more-links.frag

It should works like the following:



This sample navigation code is available here (and more-links.frag)

Congratulations again, you now know a lot about how simple it is to create a web application using Emy! See additionnal notes & links if you need more informations.

Additional notes

About .frag files, you can use .txt or .html as those file fragment extension. We use .frag since it contains a fragment of an HTML document so it does not sounds right to use .html without a proper header & body in this file. Feel free to use another extension if your server can't handle .frag, it just has to return a plain/text file stream like a .txt would do (and no, .frag has nothing to do with the Microsoft XPS file format).

More ressources

Core javascript object documentation
If you want to learn all nitty-gritty details about Emy's javascript code.
Emy's CSS documentation
For designers, here is the complete guide of default elements and how we style them.
Emy community on Google+
Official community's talks, links, events & discussions about Emy
iPhoneWebDev Google Group / mailing-list
Last but not least, you may find some very useful support from this worldwide community
Follow Emy on Facebook
Like the Emy Facebook page to get latest images, links & news on the biggest network worldwide