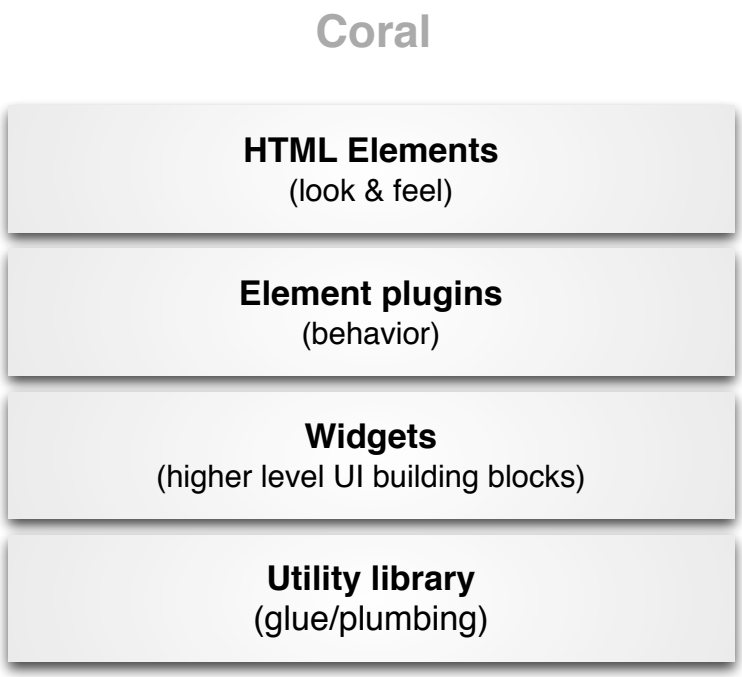
WARNING - this is a draft document!

Introduction to the Coral

The Coral is a collection of building blocks for developing web applications. Designed to be modular from the start, each module forms a distinct layer based on its primary role. Although layers are intended to support each other, they may be used independently if one desire to do so. This makes it possible to implement Coral's user experience in any HTML-capable environment.

The Coral doesn't mandate use of any particular development model and/or platform. Its primary goal is to provide unified and clean HTML5 markup, regardless of an actual method used to emit this markup. This may be client or server-side rendering, templates, JSP, PHP or even Adobe Flash RIA applications - just to name few.



WARNING - this is a draft document!

HTML Elements - the markup layer:

It's purpose is to provide common look and feel for all common UI elements such as navbar, button, menu or rail.
An element is just a HTML tag with dedicated class name. More complex elements may be composed of multiple tags, nested within each other in specific way. The CSS is used to provide the actual look and feel. In order to make it possible to customize the L&F easily (e.g. in case of branding), actual style values are declared as variables which will be expanded by the LESS pre-processor during the runtime.

purpose:
- basic UI elements with common look and feel
- default grid system

implementation:
- HTML tags with bootstrap-inspired styles
- classes are defined in LESS files
- icons as font sprites

example:

markup:

```
<button class="btn btn-large btn-primary" type="button">Large button</button>
<button class="btn btn-large" type="button">Large button</button>
```

shows as:

the look and feel is defined in a LESS, tied to an element by dedicated class name:

```
.btn {
  font-size: @baseFontSize;
  line-height: @baseLineHeight;
  .buttonBackground(@btnBackground,
                    @btnBackgroundHighlight,
                    @grayDark, 0 1px 1px rgba(255,255,255,.75));
```
(shortened for a sake of brevity)

with actual values defined in LESS variable file:

```
@btnBackgroundHighlight:           darken(@white, 10%);
@btnPrimaryBackgroundHighlight:    spin(@btnPrimaryBackground, 20%);
@baseFontSize:           17px;
@baseFontFamily:         @sansFontFamily;
```
(shortened for a sake of brevity)

WARNING - this is a draft document!

Element plugins:

Many of the HTML Elements should exhibit some kind of dynamic behavior such us popup menu opening and closing, which is not possible to accomplish without resulting to manipulate the DOM via JavaScript. This is the role of Element Plugins. A plugin either works on specific DOM element, e.g. dialog plugin expects to find DIV class=dialog or is generic in nature, e.g. a layout manger would provide layout for any list of DIV or LI elements.

Some plugins require parameters to customize their behavior. This is accomplished in two ways; by passing parameters programmatically via JS call or by using dedicated data-* attributes tied to the HTML markup. Its up to a developer to choose the best approach for a given plugin. The rule of thumb is to use data-* attributes for options related to HTML layout (e.g. specify number of columns) and API options/class for functionality related to data (e.g. list of items to display).

The same concept is used to implement form validation. An element that need to be validated would specify its desired input format as custom data-* attribute. This data attribute will then be used as an option for a validation plugin.
NOTE: HTML5-native form validation should be used whenever possible and/or expanded upon.

purpose:
- provide dynamic behavior for HTML Elements
- custom layouts not possible with pure CSS
- form validation
- advanced DOM manipulation

implementation:
- jQuery plugin tied to specific DOM element(s)
- using data-* attributes to customize its behavior

example:

markup (note the options specified as data-* attributes):

```
<ul data-column-width="220" data-layout="card" class="cards">
     <li class="item">
          <div class="thumbnail">
               <img href="/a.html" src="/a.thumb.319.319..png">
               <div class="caption">
                    <h4>Toolbar</h4>
                    <p><small>toolbar</small><br></p>
               </div>
          </div>
     </li>
     <li class="item">
          <div class="thumbnail">
               <img href="/a.html" src="/a.thumb.319.319..png">
               <div class="caption">
                    <h4>Toolbar</h4>
                    <p><small>toolbar</small><br></p>
               </div>
          </div>
```

WARNING - this is a draft document!

```
    </li>
```

call jQuery plugin:

```
$('.cards').cardlayout ();
```

shows as:

The cardLayout plugin will laid out the enclosed UL elements based on their respective heights, taking parent's width into consideration.

WARNING - this is a draft document!

HTML Element widgets:

A widget combines one or more basic level elements with a JS plugin to form 'higher level' UI elements to implement more complex behavior and/or look & feel then a single could provide. Good example could be a tag picker or rail widget. A widget can both trigger and listen to custom events, thus cooperating with other widgets on a page. Some widgets may actually be a native jQuery widgets using the Coral HTML elements.

purpose:
- implement higher level UI elements exhibiting complex behavior
- triggering and handling events

implementation:
- jQuery plugin + HTML markup
- may utilize client/server side templates

example:

markup:

```
<input type="text" name="tags" placeholder="Tags" class="tagManager"/>
```

call jQuery plugin with options:

```
$(".tagManager").tagsManager({
                        prefilled: ["Pisa", "Rome"] })
```

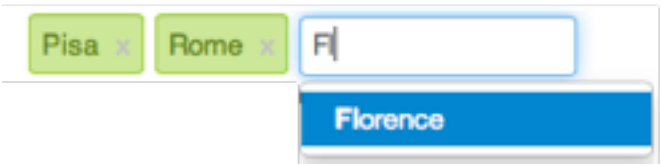
plugin emits HTML markup (this markup uses basic elements, which intern may use other plugins):

```
<span>Pisa</span>
<a title="Removing tag" tagidtoremove="0"
   id="myRemover_0" class="myTagRemover" href="#">x</a></span>

<span id="myTag_1" class="myTag"><span>Rome</span>
<a title="Removing tag" tagidtoremove="1"
   id="myRemover_1" class="myTagRemover" href="#">x</a></span>

<input type="text" data-original-title="" class="input-medium tagManager"
       placeholder="Tags" name="tags" data-provide="typeahead" data-items="6"
       autocomplete="off">
```

shows as:



WARNING - this is a draft document!

Utility library:

Collection of JS helper plugins or functions that are UI independent, yet crucial for building full featured web applications, such as XSS handling or event bus. Although the HTML element plugins and widgets may rely on functionality provided by the utility library, the utility library may not have any hard dependency on the elements nor widgets itself.

purpose:
- provide common functionality
- event bus implementation
- client-side templates
- XSS

implementation:
- jQuery plugins or AMD-compliant JavaScript modules

NEEDS some examples here??