Managing CSS and JS in Ushahidi

The following is based on changes made in Ushahidi 2.7 and may not be accurate for earlier versions

CSS and JS in Ushahidi are managed through a combination of the Themes library, and the Requirements library.

Adding your own JS / CSS

In a theme

To add JS and CSS files to a theme, you need to specify the file to be added in you Themes readme.txt file. This is best explained by example

themes/mytheme/readme.txt

```
Theme Name: MyTheme
Description:
Version: 1.0
Author: Ushahidi
Author Email: team@ushahidi.com
Demo: http://www.ushahidi.com
CSS: mystyles
JS: myscripts
```

This will include 2 files

- themes/mytheme/css/mystyles.css
- themes/mytheme/js/myscripts.js

These will be included on every page on the frontend.

Special Cases

Older themes / Missing CSS file list

If you don't specify the CSS line in your readme.txt, Ushahidi will look for the following CSS files: base.css, style.css, _default.css
This is to make it easier to use older themes created before the CSS / JS lines were added to readme.txt.

Base Themes

In addition to the CSS / JS files you specify in your themes readme.txt, we will include CSS / JS files from the base theme.
For most themes this is the "default" theme, which includes base.css, accordion.css, slider.css, style.css

IE CSS

The following IE CSS files are included by default:

- themes/default/css/iehacks.css
- themes/default/css/ie7hacks.css
- themes/default/css/ie6hacks.css

These can be overridden by including a file of the same name in your themes css folder. For example "mytheme/css/iehacks.css"

RTL CSS (RTL = Right To Left)

All CSS files can provide a RTL version - this will be used when site language is RTL.

Example

- themes/default/css/base.css would be swapped for themes/default/css/base-rtl.css if it exists

RTL languages are defined based on the translation of the 'core.text_direction' string. If this is translated to 'rtl' then language is RTL.
For testing this can be overridden in config/locale.php: locale.force_text_direction = 'rtl';

In a plugin or theme hook

The easiest way to add JS or CSS is through the Theme config in readme.txt. This adds the CSS/JS to every page on the frontend.
However if you need to add CSS to only some pages, or through a plugin, or to the admin - you can do that using the Requirements library in a Theme/Plugin hook.
See How To Write A Plugin#TheHook for more info on hooks.

Examples of Requirements methods

```
// Include fish.css
Requirements::css('plugins/awesomeplugin/css/fish.css');

// Include fish.js
Requirements::js('plugins/awesomeplugin/js/fish.js');

// Include some custom, dynamically generated CSS
$color = 'red';
Requirements::customCSS("p {color: $color;}", 'red-para');
// The 2nd parameter provides a unique ID for this CSS that can be used with Requirements::block();

// Include some custom, dynamically generated JS
$alert = 'Something awesome just happened!';
Requirements::customJS("alert('$alert')", 'awesome-alert');
// The 2nd parameter provides a unique ID for this JS that can be used with Requirements::block();

// Add a custom head tags, this can be any HTML to go in the page <head>
Requirements::customHeadTags('<meta http-equiv="keywords" content="some, key, words" />', 'keywords');

// Include a themed CSS files
// This will look for boxes.css in css dir of current theme, the theme parents, and then "media/css"
Requirements::themedCSS('boxes.css');
// This will look for boxes.css in css dir of current theme, the theme parents, THEN "plugins/myplugin" and finally
media/css
Requirements::themedCSS('boxes.css', 'plugins/myplugin');

// Add a CSS file inside an IE conditional comment
Requirements::ieCSS("lt IE 7", 'media/css/ie6.css');
// Output: <!--[if lt IE 7]><link rel="stylesheet" type="text/css" href="http://mydomain.com/media/css/ie6.css" /><![endif]-->

// The same as themedCSS by the output will be wrapped in a IE conditional comment
Requirements::ieThemedCSS('boxesIEhacks.css');

// This will combine the following CSS files into 1 CSS file: media/uploads/boxes_combined.css
Requirements::combine_files('boxes_combined.css', array(
  'plugins/boxes/css/boxes.css',
  'plugins/boxes/css/boxes2.css'
));
```

Configuring the Requirements library

The Requirements library handles actually generating the style / script tags to include CSS/JS in the page.
There are a few settings available in the requirements.php config file

Suffix requirements:
when enabled this adds a modification time the end of URLs like: http://mydeployment.com/media/js/OpenLayers.js?m=1353545029
This means the URL changes when the file is updated and forces the browser to fetch the file again, rather than use a cached version.

This is useful if you are using future Expires headers: http://developer.yahoo.com/performance/rules.html#expires

application/config/requirements.php

```
/**
 * Do we want requirements to suffix modification time onto file names
 * */
$config['suffix_requirements'] = TRUE;
```

Combine files:
This will combine some main groups of CSS and JS files into a single file. This reduces the number of HTTP requests and can improve page load speed. http://developer.yahoo.com/performance/rules.html#num_http

- The combine with JSMin/CSSMin option will also compress the combined JS/CSS files.
- If you're are using a CDN you need to enable the CDN store combined files option OR manually upload combined files to the CDN.

application/config/requirements.php

```
/**
 * Enable combining of css/javascript files.
 **/
$config['combined_files_enabled'] = TRUE;

/**
 * Using the JSMin library to minify any
 * javascript file passed to {@link combine_files()}.
 **/
$config['combine_js_with_jsmin'] = TRUE;

/**
 * Using the CSSMin library to minify any
 * css file passed to {@link combine_files()}.
 **/
$config['combine_css_with_cssmin'] = TRUE;

/**
 * Enable auto uploading combined css/js to CDN
 **/
$config['cdn_store_combined_files'] = TRUE;
```

Write JS to body
This option will output <script> tags at the end of the page body, instead of in the head. However this option is not yet supported and will break some of the core javascript

application/config/requirements.php

```
/**
 * Put all javascript includes at the bottom of the template
 * before the closing <body> tag instead of the <head> tag.
 * This means script downloads won't block other HTTP-requests,
 * which can be a performance improvement.
 * @see Requirements_Backend::$write_js_to_body for details
 **/
$config['write_js_to_body'] = FALSE;
```

Themes library

The Themes library is responsible for several things

1. Loading themes into the module path
2. Loading themes JS and CSS
3. Managing bundled JS libraries and including them as needed
4. Generating various bits of HTML, such as the header and footer blocks.

The themes object is created in the Main and Admin controllers, then called in other controllers later.

Creating Themes Object in Main Controller

```
// Themes Helper
$this->themes = new Themes();
$this->themes->requirements();
$this->themes->frontend = TRUE;
```

The following code enables JS mapping and slider libraries, and the timeline if its enabled.

Enabling JS libraries later in controller execution

```
// Javascript Header
$this->themes->map_enabled = TRUE;
$this->themes->slider_enabled = TRUE;

if (Kohana::config('settings.enable_timeline'))
{
   $this->themes->timeline_enabled = TRUE;
}
```

The actually execution and inclusion of CSS / JS happens just before the "layout" view is rendered using the view_pre_render-layout hook:

Extract from Main Controller

```
class Main_Controller extends Template_Controller {
  ...

  public function __construct()
  {
    ...

    Event::add('ushahidi_filter.view_pre_render-layout', array($this, '_pre_render'));
  }


  ...

  /**
   * Trigger themes->requirements() at the last minute
   *
   * This is in case features are enabled/disabled
   */
  public function _pre_render()
  {
    $this->themes->requirements();
    $this->template->header->header_block = $this->themes->header_block();
    $this->template->footer->footer_block = $this->themes->footer_block();
  }
}
```

More details:

- Themes library
- Main controller