Lapstone Framework

Rich Internet Application Framework

©Martin Kattner

# Content

Table of Contents

[Content 1](#_Toc519506981)

[Lapstone CLI 3](#_Toc519506982)

[Coding Conventions 4](#_Toc519506983)

[Pages 4](#_Toc519506984)

[Create a page 4](#_Toc519506985)

[Templates 4](#_Toc519506986)

[Configuring a page 4](#_Toc519506987)

[Minimal structur of a page / Create the DOM 5](#_Toc519506988)

[Plugins 8](#_Toc519506989)

[Create a plugin 8](#_Toc519506990)

[Configuring a plugin 8](#_Toc519506991)

[Functions and fields 8](#_Toc519506992)

[Lapstone startup 10](#_Toc519506993)

[DOM manipulation 11](#_Toc519506994)

[Examples 11](#_Toc519506995)

[User Session 11](#_Toc519506996)

[login 11](#_Toc519506997)

[Styling 12](#_Toc519506998)

[Skinning 12](#_Toc519506999)

[Styling 12](#_Toc519507000)

[Style file conventions 12](#_Toc519507001)

[Plugins 13](#_Toc519507002)

[Actions 13](#_Toc519507003)

[Public functions 13](#_Toc519507004)

[Debug 13](#_Toc519507005)

[Public functions 13](#_Toc519507006)

[Detector 13](#_Toc519507007)

[Public functions 13](#_Toc519507008)

[DeviceManager 13](#_Toc519507009)

[Public functions 13](#_Toc519507010)

[dep. FormInputDesigner 13](#_Toc519507011)

[Public functions 13](#_Toc519507012)

[HelperFunctions 13](#_Toc519507013)

[Public functions 13](#_Toc519507014)

[HTML5Storage – store 13](#_Toc519507015)

[Public functions 13](#_Toc519507016)

[HtmlTemplates – template 14](#_Toc519507017)

[Public functions 14](#_Toc519507018)

[HtmlView – view 14](#_Toc519507019)

[Public functions 14](#_Toc519507020)

[ImageProvider – img 14](#_Toc519507021)

[Public functions 15](#_Toc519507022)

[Informator – info 15](#_Toc519507023)

[Public functions 15](#_Toc519507024)

[dep. jQueryExtend 15](#_Toc519507025)

[Public functions 15](#_Toc519507026)

[KeepAlive 15](#_Toc519507027)

[Public functions 15](#_Toc519507028)

[LoadExternalScripts 15](#_Toc519507029)

[Public functions 15](#_Toc519507030)

[MultilanguageIso639\_3 15](#_Toc519507031)

[Public functions 15](#_Toc519507032)

[Navigation – nav 15](#_Toc519507033)

[Public functions 15](#_Toc519507034)

[OAuth – oa 15](#_Toc519507035)

[Public functions 15](#_Toc519507036)

[RestClient – rc 16](#_Toc519507037)

[Public functions 16](#_Toc519507038)

[Session – sess 16](#_Toc519507039)

[Public functions 16](#_Toc519507040)

[Skin – skin 16](#_Toc519507041)

[Public functions 16](#_Toc519507042)

[WebServiceClient – wsc 16](#_Toc519507043)

[Public functions 16](#_Toc519507044)

[WebServiceError – wse 16](#_Toc519507045)

[Public functions 16](#_Toc519507046)

[Workflow 16](#_Toc519507047)

# Lapstone CLI

The lapstone CLI provides useful functions:

* lapstone page operations
* lapstone plugin operations
* deployment and updating of lapstone
* release lapstone app with the lapstone compiler

You need the CLI at least to create a release version of your app.

To use the CLI you have to go to the root directory of the lapstone framework and type:

java –jar lapstone.jar

The CLI will display a description how to use it.  
You can find the description in the root directory of lapstone too. Have a look to lapstone.txt

# Coding Conventions

## Pages

### Create a page

To create a page, you must do the following steps in any order:

* Create a HTML file /pages/page.<page name>.html.
  1. Copy the template HTML in the file.   
     /tools/template/page.template.html
  2. Search for ##page and replace it with <page name>.
* Create a javascript file under /js/page/page.<page name>.js.
  1. Copy the template javascript in the file.   
     /tools/template/page.template.js
  2. Search for ##page and replace it with <page name>.
* Create a json file /js/page/page.<page name>.json.
  1. Copry the template json in the file.  
     /tools/template/page.template.json
  2. Search for ##page and replace it with <page name>.
* Register the page in the /js/page/pages.json file.  
  "<page name>": true

Now the page is created and can be used in your app.

### Templates

Each page needs a HTML template. Templates can be used for multiple pages.

Lapstone uses the [html template](#_HtmlTemplates_–_template) plugin to get the templates.

A template is the basic html structure of a lapstone page.

### Configuring a page

The page.<page name>.json file contains a set of obligate parameters. You can extend them if you need your special parameters.

#### Obligate fields

##### name

Your <page name>.

##### shortname

Reserved for the future. Please fill it with your <page name>

##### template

Define your page template. An empty string for no page template.

##### asyncLoading

true or false

If you want to use asynchronous page loading or not.

##### useKeepAlive

true or false

More details at the plugin.KeepAlive section.

##### loginObligate

true or false

More details at the plugin.Session section.

##### isGlobalPage

false

Deprecated mechanic. Will be updated in future versions.

##### contentRefresh

true or false

Automatically reload the page after [contentRefreshInterval](#_contentRefreshInterval) seconds.

##### contentRefreshInterval

int (>0)

Interval in seconds when the page should be reloaded. Necessary when [contentRefresh](#_contentRefresh) is true.

#### Sample configuration

{

"name": "competenceProfile",

"shortname": "competenceProfile",

"template": "DefaultPage",

"asyncLoading": true,

"useKeepAlive": true,

"loginObligate": true,

"isGlobalPage": false,

"contentRefresh": false,

"contentRefreshInterval": 0

}

### Minimal structur of a page / Create the DOM

Depending on your configuration in the json file lapstone calls the [creator()](#_creator()_1) or the [async.creator()](#_async.creator()_2) function in your page.<page name>.js.

#### config

The config object contains the object which is defined in the page.<page name>.json file.

Have a look [how to configure](#_Configuring_a_page) a page.

#### include

Files which are defined in the include array are loaded every time before the [creator()](#_creator()) or the [async.creator()](#_async.creator()_1) function is called.

#### include\_once

#### elements

If you using page templates the the async.elements object is containing the jQuery objects.

#### constructor()

On startup lapstone calls the constructor() function of every page. The plugins are already loaded at this time.

You have to return a jQuery.Deferred().promise() object.

#### creator()

Create your HTML page in the function body. Please use the [DOM manipulation conventions](#_DOM_manipulation). If you are using a [page template](#_Templates) the elements object contains the jQuery objects.

Now DOM manipulation is over and lapstone will run a jquery enchantment and shows the page.

#### async.promise

Todo – should contain the jQuery promise.

#### async.result

After the jQuery.Deferred() object returned by [async.creator()](#_async.creator()) is either rejected or resolved the async.result object contains the result.

#### async.elements

If you using page templates the the async.elements object is containing the jQuery objects.

#### async.creator()

Create your HTML page in the function body. Please use the [DOM manipulation conventions](#_DOM_manipulation). If you are using a [page template](#_Templates) the async.elements object contains the jQuery objects.

You have to return a jQuery.Deferred().promise() object. E.g.: a asynchronous webservice call: app.rc.getJson()

#### async.done()

After resolving the deferred object the async.done() function of your page is called.

The async.result object of your page contains the <parameter> of the jQuerry.Deferred.resolve(<parameter>) method.

Create your HTML page in the function body. Please use the [DOM manipulation conventions](#_DOM_manipulation). If you are using a [page template](#_Templates) the async.elements object contains the jQuery objects.

Now DOM manipulation is over and lapstone will run a jquery enchantment and shows the page.

#### async.fail()

After rejecting the deferred object the async.done() function of your page is called.

The async.result object of your page contains the <parameter> of the jQuerry.Deferred.reject(<parameter>) method.

Create your HTML page in the function body. Please use the [DOM manipulation conventions](#_DOM_manipulation). If you are using a [page template](#_Templates) the async.elements object contains the jQuery objects.

Now DOM manipulation is over and lapstone will run a jquery enchantment and shows the page.

#### async.always()

After resolving or rejecting the deferred object the async.done() function of your page is called.

The async.result object of your page contains the <parameter> of the jQuerry.Deferred.resolve/reject(<parameter>) method.

Create your HTML page in the function body. Please use the [DOM manipulation conventions](#_DOM_manipulation). If you are using a [page template](#_Templates) the async.elements object contains the jQuery objects.

Now DOM manipulation is over and lapstone will run a jquery enchantment and shows the page.

#### async.abort()

Todo – cancle the deferred object

#### setEvents()

Register events in the setEvents() function of the page. Nowhere else!

To avoid a huge amount of events within you app lapstone has a mechanism for unbinding and rebinding events.

Declare your events in the following way:

$(“#<page name>”).on(“<event name>”, “selector”, function(event){});

or use the page id form your configuration:

$(this.config.pageId).on(“...

Lapstone will unbind the events after leaving the page and rebinds it when you come back again.

#### functions

Write your private page functions in the functions object.

#### events

The events object contains the events triggered by jQuery mobile. You can use them, but they will be updated when jQuery mobile v.2.0 is released.

## Plugins

### Create a plugin

To create a plugin you have to do the following steps in any order:

* Create a javascript file /js/plugin/plugin.<plugin name>.js
  1. Copy the template HTML in the file.  
     /tools/template/template.plugin.js
  2. Search for ##plugin and replace it with <plugin name>.
* Create a json file /js/plugin/plugin.<plugin name>.json.

1. Copy the template json in the file.  
   /tools/template/template.plugin.json
2. Search for ##plugin and replace it with <plugin name>.

* Register the page in the /js/plugin/plugins.json file.  
  "<plugin name>": true

Now the plugin is created and can be used in your app.

### Configuring a plugin

#### Obligate fields

##### name

string

Your <plugin name>

##### shortname

string

A short name to acces the plugins [public functions](#_functions_1).

e.g.: app.<shortname>.<public function>();

#### Minimum sample configuration

{

"name": "Exa",

"shortname": "exa"

}

### Functions and fields

#### config

The config object contains the object which is defined in the plugin.<plugin name>.json file.

Have a look [how to configure](#_Configuring_a_plugin) a plugin.

#### constructor()

The constructor() function is called once when the plugin is loaded on startup.

You have to return a jQuery.Deferred().promise() object.

#### pluginsLoaded()

The pluginsLoaded() function is called when all plugins constructor() functions have been run. At this point you have access to the the plugins [public functions](#_functions).

You have to return a jQuery.Deferred().promise() object.

#### pagesLoaded()

You have to return a jQuery.Deferred().promise() object.

#### definePluginEvents()

#### afterHtmlInjectedBeforePageComputing()

#### pageSpecificEvents()

#### functions

## Lapstone startup

Just for information. Do not care if you have no idea.

1. Lapstone initialisation
2. Load the configuration for lapstone. /js/lapstone.json
3. Load plugins.
   1. Load plugins configuration
   2. Verify the plugins configuration.
   3. Load the plugins and call the constructor.
   4. Verify the plugins.
   5. Calling the plugins loaded event.
   6. Define the plugin events.
4. Load pages.
5. Update framework.
6. Enchant pages.
7. Wait for jQuery mobile mobileinit event.
8. Wait for apache cordovas deviceready event.
9. Trigger lapstone event.

## DOM manipulation

Use jQuery and nothing else!

1. Always create the HTML element without attributes:   
   e.g.: $(“<div>”)
2. Add classes to the element with the jQuery addClass function   
   e.g.: .addClass(“active”);
3. Add attributes to the element by using an attribute object:   
   e.g.: .attr({“id”:”divPageContainer”})
4. Append other HTML elements:   
   e.g.: .append();
   1. Always append using a function as parameter  
      e.g.: .append(function(){ return …; });
   2. Nested appending of elements is okay.
5. Do not use the .css() function. For styling use LESS. [Documented here](#_Styling).
6. Use .show() and .hide() or .toggle() to change visibility of html elements.

### Examples

#### Add a nested HTML element

$("<div>").addClass("myClass")

.append(**function**() {

**return** $("<a>").attr({

"href" : "#"

}).addClass("click")

.append(function() {

**return** $("<img>")

.attr({

"src" : "../images/content/myImage.svg"

});

})

.append(function() {

**return** $("<p>").text(name);

})

.on("storagefilled", **function**(event) {

app.debug.event(event); $(this).parent().parent().next().toggle();

});

});

#### Add a list of HTML elements

$(“<div>”).addClass(“container”).append(function(){

var $container;

$container = $(“this);

});

## User Session

### login

Login with the app.action.login function.

Set

# Styling

## Skinning

## Styling

In debug mode Lapstone uses LESS to style the apps. Get further information at lesscss.org.

The style files are locadted under the /css/ folder. Depending on using the skin plugin the files are in the /css/ folder or in the /css/skin/<skin name>/ folder.

Lapstone comes out of the box with three skin files:

* colors.less

Define all of your colors here.

* fonts.less  
  Define all your fonts here.  
  The folder for fonts is ./fonts/
* global.css.less  
  @import all other less files in this file.

In release mode every <style name>.css.less file is mapped an compressed to a <style name>.css file. The LoadExternalScripts or the Skin plugins will automatically loads CSS version of your style.

### Style file conventions

1. Create a page.<page name>.less file for each page in your app.
2. Create a layout.<layout name>.less file for each style overlap in your pages.
3. If you need more style files in your app add <style name>.css.less files. Add it to the plugin\_LoadExternalScripts.config or plugin\_Skin.json file.

# Plugins

## Actions

Helps you to define global functions and actions which can be used in pages or plugins.

### Public functions

## Debug

Helps you to create debug output which will be removed in the release version.

### Public functions

## Detector

Detects different devices and operating systems.

### Public functions

## DeviceManager

Manages different devices and provides different code depending which device you are using.

### Public functions

## dep. FormInputDesigner

### Public functions

## HelperFunctions

Functions that

### Public functions

## HTML5Storage – store

Manages, hanles and extends the HTML5 local storage.

### Public functions

## HtmlTemplates – template

Motivation: In your app you need the same HTML element multiple times.

e.g.: the layout of a pupup; the layout of a page; the layout of tables;

The HtmlTemplates plugin provides predefined HTML elements with less styling as jQuery elements. And it helps you to get more structure in your app.  
Each template consists of two files. A html file with a html structure and a less file with the same name. Both files are in the /files/template/ folder. The files have the same name bit a different extension (\*.css.less, \*.html).

You must register a template in the /js/plugin/plugin.HtmlTemplates.json file.

"PageTemplate": {

"content": "../files/template/PageTemplate.html",

"style": "../files/template/PageTemplate.css.less",

"elements": {

"left": "div[data-role=left]",

"menuHeadline": "div[data-role=menu] > .headline",

"menu": "div[data-role=menu] > #slideMenuItemContainer",

"content": "div[data-role=content]",

"statusBar": "#statusBar"

}

}

It has 3 fields:

* “content” – The url of the templates html file;
* “style” – The url of the templates less file;
* “elements” – A dictionary with css selectors to get jQuery elements of the templates html element quickly;

### Public functions

#### get(templateId, $templateObject)

##### get(“PageTemplate”);

Returns the template with the name “PageTemplate” as a jQuery object.

The jQuery object will be extended with a getter function for each field in the “elements” dictionary. e.g: \_left()

The getter returns the element which is defined by the selector.

##### get(“PageTemplate”, $(“#pageTemplate”);

Working like [get(“PageTemplate”)](PageTemplate#_get() but you can pass a jQuery object as a second parameter. It returns the extended jQuery object with the getters.

e.g.: You have a template which is already in your DOM. After an event (“click”) you want to change something on that template in the DOM. So you use get(“PageTemplate”, $(“<unique selector>”)). And it will return the passed jQuery object with the getters.

## HtmlView – view

### Public functions

## ImageProvider – img

### Public functions

## Informator – info

### Public functions

## dep. jQueryExtend

### Public functions

## KeepAlive

### Public functions

## LoadExternalScripts

### Public functions

## MultilanguageIso639\_3

### Public functions

## Navigation – nav

Helps you to navigate between pages.

### Public functions

## OAuth – oa

Handles oAuth.

### Public functions

## RestClient – rc

Takes care of user defined webservices and handles the webservice cache.

### Public functions

## Session – sess

Handles persistent sessions by using the HTML5 storage.

### Public functions

## Skin – skin

Handles different sets of stylesheets and images.

### Public functions

## WebServiceClient – wsc

Handles the communication between your app and one or more servers.

### Public functions

## WebServiceError – wse

Takes control of handling webservice results and handles user defined and common errors.

### Public functions

# Workflow