Agnitio Framework Documentation

Agnitio A/S

Table of Contents

Document changelog

Date of change	Document version	Framework version	Changes
13-12-2011	1.0b	2.0	Initial version of document
05-01-2012	1.0	2.0	Corrections made after QA report. Initial release version.
08-05-2012	1.1	2.2	Corrections made according to latest update on the Framework and Agnitio HTML5 Editor *Important: Editable presentation could longer be created on the Agnitio Manager, please use the "Blank" presentation template included in the framework source code.
01-10-2012	1.2	2.3.2	Updated the iPlanner API section with information about the agnitio.js file. Added brief explanation about agnitio.js and draggy.js files in the file organization section. Updated Agnitio logo.
24-02-2014	1.3	2.4.0	Removed section about upgrading from version 1.0 to 2.0. Updated link to content API wiki documentation.

Introduction

The **Agnitio HTML5 Framework** (Framework) provides content creators with utilities and an API to facilitate easier development of interactive content and structure. The **Agnitio HTML5 Editor** (Editor) is a software tool built into the Agnitio Manager that allow a user to create new iPad presentations and then update the content and structure through a user interface. The Framework is a stand-alone product that can be used with or without the Editor.

The Framework is specifically developed for creating content viewed through the Agnitio iPlanner apps (iPlanner), but should be usable in any Webkit browsers (i.e. Safari and Chrome).

Framework version

This version of the document (v.1.3) documents Framework v.2.4.0.

Target audience

This documentation is targeted at iPlanner content developers using webkit compatible technologies. The documentation will cover how to use the Framework to create presentation using the HTML5 Editor, or by manual setup. Basic understanding of web development with HTML, CSS, and JavaScript is required.

Framework concepts

Design & Purpose

The intention with the Framework is to make it easy to create interactive presentations that will run smoothly in the Agnitio iPlanner apps available in the Apple App store. The following design goals form the basis of the Framework:

- API that is easy to use and test
- Convention over configuration
- Lightweight core
- Modular functionality
- Flexible in terms of possible structures
- Content loaded on-demand

Structural components (building blocks)

There are four types of structural building blocks that the Framework use:

- Slides - individual pages
- Slideshows - linear collection of slides
- Inline slideshows - slideshows embedded in a slide
- Collections - linear collection of slideshows

All of these building blocks can be combined in a single presentation as many times as necessary. There are a few conventions to be followed when using the building blocks:

A slide always belong to a slideshow, unless it is loaded on top of other content (i.e. using the SlidePopup module).

A slideshow and a collection cannot have the same ID.

Important events

The Framework will dispatch custom events when something important happens that other parts of the application need to know about. This will make it easy to initiate functionality whenever a slideshow is loaded for example. To successfully use the Framework it is important to know which custom events are available. Currently the following events are dispatched:

- presentationInit
- slideshowLoad

- slideshowUnload
- collectionLoad
- collectionUnload
- contentLoad
- contentUnload
- slideEnter
- slideExit
- sectionEnter
- sectionExit

For more information regarding the custom events please refer to the API documentation, which is bundled with the code download.

Framework API

Great care has been taken to create an easy to use and testable API. The API methods and properties are accessed through the global namespace *app*. The following example will navigate to the next slide (if available) in the current slideshow:

```
app.slideshow.next();
```

Understanding and knowing the API is essential to successfully build applications using the Framework. For more information please refer to the API documentation.

Presentation types

Currently there are two types of presentations based on how it store and read its structural configuration (i.e. what slides belong to what slideshow). When created, the presentation uses JSON (*presentation.json*) to configure the presentation, while a JavaScript object could be optionally supplied as a parameter when creating a new presentation manually.

A property is set in the presentation configuration to tell the framework to use one type or another. Possible types are ‘dynamic’ and ‘json’. If no type is supplied then it will be treated as ‘dynamic’ by default.

Slide JavaScript objects

Each slide loaded has a corresponding JavaScript object provided by the Framework. By default, each slide object has two empty methods: onEnter and onExit. A slide object is referenced through the following API: *app.slide.slideId.* In order to attach behavior to the slide, the two methods are overwritten with necessary JavaScript code. The Framework supplies the methods with the slide’s article element as a parameter. In the following example an event listener is added when entering a slide with id “efficacy”, and then removed when exiting it:

```
(function() {
    app.slide.efficacy = {

        onEnter: function(ele) {

            this.btn = ele.querySelector('#saveBtn');

            this.btn.addEventListener('tap', this.save);

        },

        onExit: function(ele) {

            this.btn.removeEventListener('tap', this.save);

        },

        save: function(event) {

            // TODO: add save functionality

        }

    };

})();
```

New presentation

Getting the code

Obtaining the Framework code:

- Download from GitHub: https://github.com/Agnitio/Framework

File organization

When creating a new presentation it will have the following files and folders:

```
_framework
    |-- lib
        |-- agnitio.js
        |-- draggy.js
        |-- touchy.js
        |-- util.js
    |-- styles
        |-- core.css
        |-- reset.css
    |-- templates
        |-- error_missing_slide.html
        |-- error_missing_structure.html
        |-- new_slide.html
    |-- core.js
code
    |-- css
    |-- js
content
    |-- img
modules
slides
    |-- placeholder.html
all.css
index.html
presentation.json
setup.js
```

_framework
This folder will contain the necessary files that make the Framework API and other features work properly. **It is important that the files in this folder are not edited as that might break functionality.**

_framework > lib
Bundled files offering special functionality used by the framework or by Agnitio modules.

_framework > lib > agnitio.js
The Agnitio Content API that allow the content to communicate with the Agnitio platform (iPlanner). This is a copy of the file bundled in the iPlanner *viewer/js* folder.

_framework > lib > draggy.js
A library that enables dragging of HTML elements. Great for creating sliders and other functionality.

_framework > lib > touchy.js
This library dispatches custom events based on mouse or touch events, i.e. swipes, longTouch. See API documentation for more information. Required if using the Slidescroller module.

_framework > lib > util.js
A small library containing some often used functionality, i.e. addClass and getPosition. Not required by the framework but used by some of the Agnitio-authored modules.

_framework > styles > core.css
Contains some basic styles for placeholder and error slides, and to position slides horizontally and vertically.

_framework > styles > reset.css
A CSS stylesheet resetting the default browser styles.

_framework > templates
Template html files to create placeholder and error slides.

_framework > templates > error_missing_slide.html
Template for referencing a non-existent slide.

_framework > templates > error_missing_structure.html
Template for referencing a non-existent structure.

_framework > templates > new_slide.html
Template used when creating a new slide through the Editor.

_framework > core.min.js
The actual framework code responsible for the API and custom events.

code
Folder holding the custom code used in the presentation that are not modules.

code > css
Custom CSS files are put here.

code > js
Folder for custom JavaScript files.

content
Folder for adding various files used by the presentation, i.e. images, PDFs, videos.

content > img
Folder to contain images used in the presentation.

modules
Folder for reusable modules used in the presentation.

slides
Folder containing the slides available to the presentation. When using the Editor it is required that the slide html files is stored here. If not using the Editor the framework can be configured to fetch slides from any folder.

slides > placeholder.html
A placeholder slide used to display some content when presentation is first created.

all.css
A container CSS file importing all other CSS files used in presentation. Makes it easy to combine and minify all CSS files using a builder tool like Juicer. It is a convention used in the framework but is not required.

index.html
The application entry point. Link to all CSS and JavaScript files. Contains some basic container html elements by default.

presentation.json
The presentation configuration file updated by the Editor. **Should generally not be edited manually.**

setup.js
JavaScript file used to initialize the presentation. Should be updated with necessary configuration and modules to customize presentation.

Presentation setup

Linking CSS and JavaScript

CSS and JavaScript is linked from the index.html file. By convention individual CSS files are imported to all.css, which is linked from the default index.html file. JavaScript files are referenced below the markup but before the closing *body* tag.

JavaScript setup

The file *setup.js* contain code that is run when the presentation is first loaded. In this file two JavaScript calls have to be present to make use of the framework and load the content:

- Call to new Presentation(config)
- Call to app.init()

Most presentations will also have the following in the *setup.js* file:

- Call to new Slidescroller (enabling swipe of slides)
- Initialization of modules

Presentation configuration

It is possible to pass an optional configuration object when calling *new Presentation* in *setup.js*.This configuration object will tell the presentation to behave in certain ways. Below is a outline of possible configuration options.

type

Presentation type can be set to either “json” or “dynamic” (default). Type *json* is automatically provided as presentation type when creating presentation.

orientation

Default orientation of presentation. Defaults to ‘landscape’, other possible value is ‘portrait’.

pathToSlides

A string containing an alternate path to the slide files, i.e. ‘content/slides/’.

globalElements

An array of DOM element ids. The Framework will add the corresponding DOM elements to the *app.elements* object.

manageMemory

Set it to *true* if framework should set display:none on slides not currently being viewed. Will wrap slides in a div tag with class *slideWrap*.

slideshows

JavaScript object mapping slideshow id to array of slides. Used by the framework to register slideshows if type is *dynamic*. Not used if the presentation type is *json*, as the corresponding data is stored in *presentation.json*.

collections

JavaScript object mapping collection id to array of slideshows. Used by the framework to register collections if type is *dynamic*. Not used if the presentation type is *json* as the corresponding data is then stored in *presentation.json*.

Creating structures

Two types of structures can be created using the framework: slideshows and collections. Slideshows are populated with slides and collections are populated with slideshows. How they are created depends on the presentation type.

When created using the "Blank" template (type *json*) the structures are created by clicking “Add structure” and then selecting either “Slideshow” or “Collection”. Each created structure is then populated with either slides (slideshows) or slideshows (collections).

When creating presentations using type *dynamic*, the structures are created manually in the presentation configuration. A slideshow is created by adding a unique name and an array of slides in the slideshows object, and a collection is similarly created with an array of slideshows in the collections object The following is an example creating two slideshows and one collection:

```
new Presentation({

    slideshows: {

        intro:     [“Introduction”, “table_of_contents”],

        efficacy:  [“ef_overview”, “ef_details”, “ef_references”]

    },

    collections: {

        main:      [“intro”, “efficacy”]

    }

});
```

Adding modules

Modules should be initialized in the setup.js file, after call to *new Presentation* and before call to *app.init*. Modules should contain no hard-coded references to the rest of the presentation, ensuring reusability between presentations.

Loading initial content

The framework need to know what content to load initially. If the presentation is created using the "Blank" template (type json), the framework will automatically load the first structure in the storyboard. This can be manually overridden by supplying parameters to the call *app.init* in *setup.js* (see below example for *dynamic* presentations).

If manually creating the presentation structures with type dynamic then parameters have to be given to the *app.init* call (in *setup.js*), telling what content to load. The following example would load the collection “placebo” when starting up:

```
app.init(‘placebo’);
```

Adding and editing content

Creating slides

Slides are either created by manually uploading new slide html files via FTP or by clicking “Add slide” in the Editor. Regardless, content is added by manually inserting the necessary markup in the slide file and uploading via FTP. Slides are by default created in the root “slides” folder, but can through presentation configuration be set to any folder. The following requirements need to be met to successfully create a new slide:

- Root level html tag is <article>
- The root level article tag has an unique id
- The root level article tag has a class “slide”

When manually creating slides to be used with the HTML5 Editor, it is recommended to also add an Agnitio data slide name attribute to the article tag (see example below).

Following is an example of a properly created slide:

```
<article id=”efficacy_overview” class=”slide”

         data-ag-slide-name=”Efficacy overview”>

    <h2>Efficacy overview</h2>

</article>
```

Making content editable (Agnitio HTML5 Editor)

The Editor use HTML5 attributes to identify the elements that should be possible to update through the user interface. To make an element editable add the following attribute to it: *data-ag-editatble*. The value of the attribute will serve as an identifier for the content in the element. The example below will make a header editable:

```
<h2 data-ag-editable=”efficacy_header”>Text that could be updated</h2>
```

iPlanner API

Bundled files

The Agnitio iPlanner app comes bundled with a file that allow the content to communicate with the app. This can be used to open PDFs, send emails and save data from the content.

To get access to the APIs it is necessary to link to the agnitio.js file from index.html:

```
<script src= "../viewer/js/agnitio.js"></script>
```

For full documentation please visit the following entry in the Agnitio wiki:

http://wiki.agnitio.com/index.php/Agnitio_Content_API_(iPad)

Error handling

Error handling

Currently there are two error scenarios that the Framework will handle internally.

Missing slide

If a slide is added to a slideshow but it is not actually present in the presentation a placeholder file will be inserted with an error message specifying the id of the slide missing. Apart from being a useful message when a slide has mistakenly been deleted or never added, this also makes it possible to create the structure before creating the actual content.

Missing structure

If the Framework tries to load a structure that does not exist, the presentation cannot run. In that case a placeholder slide is displayed with a message specifying the id of the missing structure.