AXEMAS Documentation

Release 1.0

AXANT

Sep 27, 2017

Contents

Development Framework for MultiPlatform hybrid mobile applications.

AXEMAS handles the whole navigation of the application and transition between views, while it permits to implement the views content in HTML itself.

AXEMAS works using `sections`, each `Section` represents the content of the view and is loaded from an HTML file or from an external URL.

Whenever native code requires to be attached to a section, it is possible to attach a `SectionController` to a `Section` itself.

CHAPTER 1

Getting Started

To Install AXEMAS you need `Python 2.7` with the `pip` package manager installed as AXEMAS uses Python to generate project skeletons. To install `pip` follow the Pip Install Guidelines. Then you can install AXEMAS toolkit using:

```
$ pip install axemas
```

To create a new AXEMAS project you can then use the `axemas-quickstart` command, it will automatically create a new AXEMAS project:

```
$ gearbox axemas-quickstart -n ProjectName -p com.company.example
```

See *Quickstarting a New Application* for additional details on the `gearbox` command.

Basic Project Introduction

By default AXEMAS will create for you a basic application for **iOS** and **Android**. Content of the application will be available inside `www` directory and the application will load `www/sections/index/index.html` on startup.

The application can be run by simply opening in *Android Studio* or *XCode* the `android` and `ios` projects inside the newly created application directory and then pressing the **Run** button inside the IDE.

To start customizing the application and providing your own code, you can open the `www` directory in your favourite editor and start editing sections.

From the **index.html** section, you can then use the *JavaScript API* to push and pop additional sections and implement your whole Application.

Binding to Native Code

The previous code shows how to load `HTML` based sections and rely on the *JavaScript API* to implement your web application. When more advanced features or interaction with the hardware is needed you might need to get to native code level. AXEMAS has been designed specifically to make it as easy as possible to work with native code, the main difference with frameworks like Cordova is explicitly that AXEMAS makes native a first citizen of your application.

HTML sections loaded by your application are explicitly declared inside the application code itself and the application window is explicitly create using `makeApplicationRootController` from the `NavigationSectionsManager`

Inside your `AppDelegate` for **iOS**:

```
@implementation AppDelegate

- (BOOL)application:(UIApplication *)application␣
↪didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window.rootViewController = [NavigationSectionsManager␣
↪makeApplicationRootController:@[@{
        @"title": @"Home",
        @"url": @"www/home.html",
    }]];

    [self.window makeKeyAndVisible];
    return YES;
}

@end
```

Or in your `AXMActivity` subclass `onCreate()` method for **Android**:

```
public class MainActivity extends AXMActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (savedInstanceState == null) {
            JSONObject data = new JSONObject();
            try {
                data.put("url", "www/home.html");
                data.put("title", "Home");
            } catch (JSONException e) {
                e.printStackTrace();
            }

            NavigationSectionsManager.makeApplicationRootController(this, data);
        }
    }
}
```

The binding between the `HTML Sections` and native code is performed using `SectionControllers`, to link a section to a section controller is as easy as registering the controller class for the specified route:

```
[NavigationSectionsManager registerController:[MySecionController class] forRoute:@
↪"www/mysection.html"];
```

```
NavigationSectionsManager.registerController(this, MySecionController.class, "www/
↪mysection.html");
```

To get started using `SectionControllers` read the *iOS API* and *Android API*.

Contents:

iOS API

Declaring Sections

The main application controller (`window.rootViewController`) must be created using `[NavigationSectionsManager makeApplicationRootController]`.

`makeApplicationRootController` accepts an array of section data, each data will be used to create a tab with a section inside.

To create an application with a stack of sections (using a *Navigation Controller*), and not tabs, just pass data for a single section data:

```
[NavigationSectionsManager makeApplicationRootController:@[@{
        @"url":@"www/index.html",
        @"title":@"Home",
    }]
];
```

The created section will contain content of `www/index.html` and will be titled `Home`. Further sections can be pushed onto the navigation stack using `axemas.goto(dictionary)` from Javascript.

To create an application with a TabBar just pass data for multiple sections into the `makeApplicationRootController` array, each section must have an `url` pointing to the section path and can have a `title` and `icon` which will be used as title and icon for the TabBar tabs.

An application with sidebar can also be created by passing a section data as sidebar to the `makeApplicationRootController`:

```
[NavigationSectionsManager
    makeApplicationRootController:@[@{
        @"url":@"www/index.html",
        @"title":@"Home",
        @"toggleSidebarIcon":@"reveal-icon"}]
    withSidebar:@{@"url":@"www/sidebar.html"}
];
```

The sidebar will be created with content from the section data passed in `withSidebar` parameter, sections that have a `toggleSidebarIcon` value in section data will provide a button to open and close the sidebar with the given icon. If the value is omitted, even when the sidebar is enabled, there will be no button to show it.

Section Controllers

Section controllers permit to attach native code to each section, doing so is as simple as subclassing section controllers and providing `sectionWillLoad` and `sectionDidLoad` methods.

Inside those methods it is possible to register additional native functions on the javascript bridge.

Inside `viewWillLoad` method of `SectionController` subclass it is possible to register handlers which will be available in Javascript using `axemas.call`:

```
@implementation HomeSectionController

- (void)sectionWillLoad {
    [self.section.bridge registerHandler:@"openMap" handler:^(id data,␣
↪WVJBResponseCallback responseCallback) {
        UINavigationController *navController = [NavigationSectionsManager␣
↪activeNavigationController];
```

```
        [navController pushViewController:[[MapViewController alloc] init]␣
↪animated:YES];

        if (responseCallback) {
            responseCallback(nil);
        }
    }];
}

@end
```

Registering the `SectionController` for a section can be done using the `NavigationSectionsManager`:

```
[NavigationSectionsManager registerController:[HomeSectionController class] forRoute:@
↪"www/index.html"];
```

Calling JS from native code is also possible using the section bridge, after you registered your handlers in JavaScript with `axemas.register`:

```
axemas.register("handler_name", function(data, callback) {
    callback({data: data});
});
```

Calling `handler_name` from native code from a `SectionController` is possibile using the javascript bridge `callHandler`:

```
[self.section.bridge callHandler:@"handler_name"
                            data:@{@"key": @"value"}
                responseCallback:^(id responseData) {
        NSLog(@"Callback with responseData: %@", responseData);
}];
```

`SectionController` available callbacks:

- *sectionDidLoad* triggered when the webpage finished loading
- *sectionWillLoad* just before the webpage will start to load
- *sectionViewWillAppear* when the section is going to be displayed to the user.
- *sectionViewWillDisappear* when the section is going to disappear to the user.
- *sectionOnViewCreate:(UIView)*view* when the section view is first created.
- *(BOOL)isInsideWebView:(CGPoint)point withEvent:(UIEvent)*event* whenever a touch event for the webview happens, can be used to return block events to be trapped by webview.
- *navigationbarRightButtonAction* Triggered whenever the right button in the navigationBar is pressed.

NavigationSectionsManager

The `NavigationSectionsManager` manages the whole AXEMAS navigation system, creates the sections and keeps track of the current *Navigation Controller*, *TabBar Controller* and *Sidebar Controller* which are exposed through `NavigationSectionsManager`.

(void)`registerDefaultController:(Class)controllerClass`

Registers a given *Section Controllers* for the specified route (html file).

(void)`registerController:(Class)controllerClass forRoute:(NSString*)path`

Registers a given *Section Controllers* as the default controller which is used for all the sections that do not provide a specific section controller.

(UIViewController*)`makeApplicationRootController:(NSArray*)tabs`

Creates one or more *Section Controllers*. The first controller specified in the array is considered the root controller. If more than one controller is provided a `TabBar` is created with each controller being a Tab.

The `tabs` list should contain dictionaries in the format:

```
@{
    @"url": @"www/index.html",
    @"title": @"Home",
    @"toggleSidebarIcon": @"reveal-icon"
}
```

(UIViewController*)`makeApplicationRootController:(NSArray*)tabs withSidebar:(NSDictionary*)sideb`

Creates one or more *Section Controllers*. The first controller specified in the array is considered the root controller. If more than one controller is provided a `TabBar` is created with each controller being a Tab.

This also creates a `SideBar` with the *Section Controllers* described by `sidebarData` as the sidebar content.

The `tabs` list and `sidebarData` should contain dictionaries in the format:

```
@{
    @"url": @"www/index.html",
    @"title": @"Home",
    @"toggleSidebarIcon": @"reveal-icon"
}
```

(UINavigationController*)`activeNavigationController`

Returns the UINavigationController of the application. This is the object that manages the navigation stack (pushing and popping section controllers). See reference for a list of provided methods.

(UIViewController*)`activeController`

Returns the current *Section Controllers* on top of the navigation stack. This is usually the view that the user is currently looking at.

(id)`activeSidebarController`

Returns the `AXMSidebarController` of the application. This is the object that manages the sidebar of the application if available. It also provides the following methods to manage the sidebar:

- `(IBAction)revealToggle:(id)sender`
- `UIViewController *rearViewController`
- `FrontViewPosition frontViewPosition`
- `(void)setFrontViewPosition:(FrontViewPosition)frontViewPosition animated:(BOOL)animated`

(void)`setSidebarButtonVisibility:(BOOL)visible`

Hides/Shows the sidebar button in the navigationbar

(void)`goto:(NSDictionary*)data animated:(BOOL)animated`

Pushes on the view navigation stack the given *Section Controllers*. This works like *goto* and accepts `data` as `NSDictionary` with the same data as the related Javascript Object.

(void)`showProgressDialog`

Displays a spinner on top of the application. This is automatically called whenever a new section is loaded.

(void)`hideProgressDialog`
Hides the currently displayed spinner.

(void)`store:(NSString*)value withKey:(NSString *)key`
Stores a new value in the application persistent storage.

(NSString *)`getValueFrom:(NSString*)key`
Retrieves a previously stored value from the application persistent storage.

(void)`removeValueFrom:(NSString*)key`
Deletes a value from the application persistent storage.

Android API

Declaring Sections

The application MainActivity must extend `AXMActivity` which creates the application main structure. In the `onCreate` of your MainActivity initialize the root section by calling `NavigationSectionsManager.makeApplicationRootController()` The `makeApplicationRootController()` accepts a JSONObject containing the section data:

```
JSONObject data = new JSONObject();
try {
    data.put("url", "www/index.html");
    data.put("title", "Home");
} catch (JSONException e) {
    e.printStackTrace();
}
NavigationSectionsManager
        .makeApplicationRootController(this, data);
```

The created section will contain content of `www/index.html` and will be titled `Home`. Further sections can be pushed onto the navigation stack using `axemas.goto(dictionary)` from Javascript.

An application with sidebar can also be created by passing a section data as sidebar to the `makeApplicationRootController`:

```
JSONObject data = new JSONObject();
try {
    data.put("url", "www/index.html");
    data.put("toggleSidebarIcon", "reveal-icon");
    data.put("title", "Home");
} catch (JSONException e) {
    e.printStackTrace();
}
NavigationSectionsManager
        .makeApplicationRootController(this, data, "www/sidebar.html");
```

The sidebar will be created with content from the section data passed as `sidebarURL`` parameter, sections that have a ``toggleSidebarIcon` value in section data will provide a button to open and close the sidebar with the given icon. If the value is omitted, even when the sidebar is enabled, there will be no button to show it.

Section Controllers

Section controllers permit to attach native code to each section, doing so is as simple as subclassing section controllers and providing `sectionWillLoad` and `sectionDidLoad` methods.

Inside those methods it is possible to register additional native functions on the javascript bridge.

Inside `sectionWillLoad` method of `SectionController` subclass it is possible to register handlers which will be available in Javascript using `axemas.call`:

```
this.getSection().getJSBridge().registerHandler("openMap", new JavascriptBridge.
↪Handler() {
    @Override
    public void call(Object data, JavascriptBridge.Callback callback) {

        String uri = "https://maps.google.com/maps";
        Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
        section.startActivity(i);

    }
});
```

Registering the `SectionController` for a section can be done using the `NavigationSectionsManager`:

```
NavigationSectionsManager
            .registerController(this,HomeSectionController.class, "www/index.html");
```

Calling JS from native code is also possible using the section bridge, after you registered your handlers in JavaScript with `axemas.register`:

```
axemas.register("handler_name", function(data, callback) {
    callback({data: data});
});
```

Calling `handler_name` from native code from a `SectionController` is possibile using the javascript bridge `callHandler`:

```
this.getSection().getJSBridge().callJS("send-passenger-count", data, new␣
↪JavascriptBridge.AndroidCallback() {
    @Override
    public void call(JSONObject data) {
        Log.d("axemas", "Callback with responseData: "+ data.toString());
    }
});
```

`SectionController` available callbacks:

- *sectionDidLoad* triggered when the webpage finished loading
- *sectionWillLoad* just before the webpage will start to load
- *sectionOnViewCreate(ViewGroup view)* when the fragment is first created
- *boolean isInsideWebView(MotionEvent ev)* whenever a touch event for the webview happens, can be used to return block events to be trapped by webview.
- *sectionFragmentWillPause* triggered by fragment's onPause
- *sectionFragmentWillResume* triggered by fragment's onResume
- *sectionFragmentOnActivityResult* triggered by fragment's onActivityResult

- *sectionFragmentOnSaveInstanceState* triggered by fragment onSaveInstanceState
- *sectionFragmentOnCreateView* triggered by fragment View Creation during inflation
- *actionbarRightButtonAction* triggered whenever the right button is pressed in the actionbar

NavigationSectionsManager

The `NavigationSectionsManager` manages the whole AXEMAS navigation system, creates the sections and keeps track of the current *Fragment Stack*, *Action Bar* and *Sidebar* which are exposed through `NavigationSectionsManager` methods.

public static void **registerController** (Context *context*, Class *controllerClass*, String *route*)
Registers a given *Section Controllers* for the specified route (html file).

public static void **registerDefaultController** (Context *context*, Class *controllerClass*)
Registers a given *Section Controllers* as the default controller which is used for all the setions that do not provide a specific section controller.

public static void **makeApplicationRootController** (Context *context*, JSONObject *data*)
Creates a new application root *Section Controllers* (must be called from `MainActivity.onCreate`). `data` is the details of the section controller as you would pass them to `goTo`.

public static void **makeApplicationRootController** (Context *context*, JSONObject *data*, String *sidebarUrl*)
Creates a new application root *Section Controllers* (must be called from `MainActivity.onCreate`). `data` is the details of the section controller as you would pass them to `goTo`. This method also adds a sidebar, `sidebarUrl` is the path of the section html file that should be loaded inside the sidebar.

public static void **makeApplicationRootController** (Context *context*, JSONObject *data*, JSONObject... *tabs*)
Creates a new application root *Section Controllers* (must be called from `MainActivity.onCreate`). `data` is the details of the section controller as you would pass them to `goTo`. This method also provides additional **tabs** to the application, the root section controller is placed in the first tab, while the other `tabs` are also additional section controllers data used to fill additional tabs in the tabbar.

public static void **makeApplicationRootController** (Context *context*, JSONObject *data*, String *sidebarUrl*, JSONObject... *tabs*)
Creates a new application root *Section Controllers* (must be called from `MainActivity.onCreate`). `data` is the details of the section controller as you would pass them to `goTo`. This method also adds a sidebar, `sidebarUrl` is the path of the section html file that should be loaded inside the sidebar. This method also provides additional **tabs** to the application, the root section controller is placed in the first tab, while the other `tabs` are also additional section controllers data used to fill additional tabs in the tabbar.

public static void **goTo** (Context *context*, JSONObject *data*)
Pushes on the view navigation stack the given *Section Controllers*. This works like *goto* and accepts `data` as `JSONObject` with the same data as the related Javascript Object.

public static AXMNavigationController **getActiveNavigationController** (AXMActivity *activity*)
Returns the `AXMNavigationController` of the application. This is the object that manages the navigation stack (pushing and popping section controllers) and provides the following methods to manage the navigation stack:

- `void popFragments(final int fragmentsToPop)` -> Pops up to `fragmentsToPop` fragments (sections) from the navigation stack.
- `void popFragmentsAndMaintain(final int maintainedFragmentsArg)` -> Pops until only `maintainedFragmentsArg` fragments (sections) are left on the stack.

- `void pushFragment(final Fragment fragment, final String tag)` -> Pushes a new `Fragment` on the navigation stack.

public static SectionFragment **`getActiveFragment`**(Context *context*)
Returns the current *Section Controllers* on top of the navigation stack. This is usually the view that the user is currently looking at.

public static AXMTabBarController **`getTabBarController`**(AXMActivity *activity*)
Returns the `AXMTabBarController` of the application. This is the object that manages the application tabs if available. It also provides the following methods to manage the tabs:

- `int getSelectedTab()` -> gets the index of the currently selected tab.
- `void setSelectedTab(int idx)` -> sets the currently selected tab.

public static AXMSidebarController **`getSidebarController`**(AXMActivity *activity*)
Returns the `AXMSidebarController` of the application. This is the object that manages the sidebar of the application if available. It also provides the following methods to manage the sidebar:

- `AXMSectionController getSidebarSectionController()` -> Retrieves the *Section Controllers* bound to the section loaded into the sidebar.
- `void setSidebarButtonVisibility(boolean visible)` -> Hides/Shows the sidebar button in the actionbar
- `void setSideBarButtonIcon(String resourceName)` -> Sets the sidebar button icon from a project resource
- `void setSidebarAnimationConfiguration(float alpha, int duration, String hexColor)` -> change the sidebar animation configuration.
- `View enableFullSizeSidebar()` -> Switches to full size sidebar mode. This moves the actionbar inside the sidebar instead of being on top of both the sidebar and the content. It returns the actionbar View.
- `boolean isOpening()` -> Whenever the sidebar is open or not.
- `void toggleSidebar(boolean visible)` -> Sets sidebar visibility.
- `void toggleSidebar()` -> Toggles sidebar visibility.

public static void **`showProgressDialog`**(Context *context*)
Displays a spinner on top of the application. This is automatically called whenever a new section is loaded.

public static void **`hideProgressDialog`**(Context *context*)
Hides the currently displayed spinner.

public static void **`showDismissibleAlertDialog`**(Context *context*, String *title*, String *message*)
Displays an alert message with the specified `title` and `message`. By default only a dismiss button is provided.

public static void **`showDismissibleAlertDialog`**(Context *context*, AlertDialog.Builder *builder*)
New alert message built with the user provided `AlertDialog.Builder` dialog builder.

public static void **`enableBackButton`**(Context *context*, boolean *toggle*)
Enables/disables the back button in the application.

public static void **`store`**(Context *context*, String *key*, String *value*)
Stores a new value in the application persistent storage.

public static String **`getValueForKey`**(Context *context*, String *key*)
Retrieves a previously stored value from the application persistent storage.

public static void **`removeValue`**(Context *context*, String *key*)
Deletes a value from the application persistent storage.

JavaScript API

The JavaScript module `axemas.js` permits interaction with the native code of the application:

- goto
- gotoFromSidebar
- call
- alert
- dialog
- showProgressHUD
- hideProgressHUD
- getPlatform
- platformDetails
- storeData
- fetchData
- removeData
- log

goto

Pushes new `section` on the navigation stack. It is the equivalent of the iOS `[NavigationSectionsManager goto]` and Android's `NavigationSectionsManager.goTo()`. All three functions accept a dictionary as **payload** which defines the extra actions the `goto` call must execute:

```
axemas.goto(
    {"url":"www/home.html",
    "title":"HOME",
    "toggleSidebarIcon":"slide_icon",
    "stackMaintainedElements": 0,
    "stackPopElements": 0}
);
```

The **payload** structure is shared between JavaScript, Objective C and Java, and accepts the following parameters:

- `url` contains the local or remote address from which the WebView must load the content
- `title` (optional) is the tile show in the application's ViewController / Action Bar.
- `toggleSidebarIcon` (optional) is the sidebar's icon to be displayed and if missing a button to open the sidebar will not be created
- `actionbarRightIcon` (optional) icon to display as a button on the right of the actionbar inside the target section, pressing the button triggers the `navigationbarRightButtonAction` event on section controllers.
- `stackMaintainedElements` (optional) instructs the navigation stack to pop all views and maintain the last X `sections` indicated on the bottom of the stack; it is ill advised to use in conjunction with `stackPopElements`
- `stackPopElements` (optional) instructs the navigation stack to pop the first X `sections`; it is ill advised to use in conjunction with `stackMaintainedElements`

- `animation` (optional) only supported on iOS platform, can be used to change the default push/pop animation in one of `"fade"` or `"slidein"` values

gotoFromSidebar

Same as `goto` but closes the sidebar and must be used only inside the sidebar `section`. Refer to `goto`:

```
axemas.gotoFromSidebar(
    {"url":"www/home.html",
    "title":"HOME",
    "toggleSidebarIcon":"slide_icon",
    "stackMaintainedElements": 0,
    "stackPopElements": 0}
);
```

call

The `call` enables JavaScript to execute a native registered handler inside a `SectionController`:

```
axemas.call('openNativeController');

axemas.call('execute-and-return', '{"payload": "something"}', function(result) {
    alert(JSON.strgify(result));
});
```

alert

Creates a native dismissible alert dialog with a title and a message:

```
axemas.alert('Alert title', "Alert message");
```

dialog

Generates a native dialog with a title, a message and a maximum of three buttons. When pressing a button a callback returns the button's value as integer, range [0-3]:

```
axemas.dialog('Dialog title', 'Dialog display message', ['Cancel', 'Ok'],
↪function(data) {
    axemas.alert('Pressed button', data.button);
});
```

showProgressHUD

Locks interface interaction by displaying a spinner on the screen. The same spinner is always displayed when lading the contents of a page inside a `section`:

```
axemas.showProgressHUD();
```

hideProgressHUD

Used to dismiss a previously displayed progressHUD:

```
axemas.hideProgressHUD();
```

getPlatform

Uses the `navigator.userAgent` object to determine if the current platform. Returns `Android`, `iOS` or `unsupported`:

```
if (axemas.getPlatform() == 'your_platform') {
    //do something
}
```

platformDetails

Getting information about device: `model`, `systemName` and `systemVersion`.

For example:

```
axemas.platformDetails(function(device_info) {
    console.log(device_info);
});

// Example of the device_infor

//iOS Device
{model: "iPhone", systemVersion: "9.3", systemName: "iPhone OS"}

//Android Device
{model: "Nexus 5", systemName: "Android", systemVersion: "6.0.1"}
```

storeData

Uses the Native/WebView's `localSotrage` for key/value storing. Data stored will be available next time the application is launched:

```
axemas.storeData("key","only_string_values");
```

fetchData

Returns a previously stored `value` providing a `key`:

```
var value = axemas.fetchData("key");
```

removeData

Permanently removes the previously saved data from the locationStorage:

```
axemas.removeData("key")
```

log

Utility for use native and javascript log system:

```
axemas.log("Hello World");
```

or:

```
axemas.log({'tag': 'CustomTAG', 'message': "Hello World"});
```

- `tag` is the tag for Android, default is AXEMAS_LOG
- `message` is the message of the log as String

Utilities

Crash Logging

Android

Splunk MINT is used to log crash reports. To enable it, it's necessary to be registered on their service (https://mint.splunk.com/) Next to the app creation on their portal initiate this code inside the Activity wich extends AXMActivity with:

```
this.startCrashReporter();
```

The next step is to add this config string inside strings.xml file inside res/values folder inside your Android project. This String resource contains the api-key got by Splunk MINT Portal for your application.

> <string name='splunk_api_key'></string>

In the end you have to configure gradle to work with the new Mint dependency required. Add this code to your build.gradle project file:

```
repositories {
    maven {
        url "https://mint.splunk.com/gradle/"
    }
}

dependencies {
    compile 'com.splunk.mint:mint:4.2'
}
```

Once these 3 steps are done you have a fully implemented crash reporter on your Axemas Application.

AXEMAS CookBook

Updating Sidebar Whenever it appears

By default content of a section is only loaded once, the first time the section itself appears. The Sidebar specifically is only loaded when the application starts, and is never reloaded again.

In case you need to display content that might change during the application lifetime, you will need to somehow receive a notification each time the sidebar appears so that you can actually force its update.

IOS

On **iOS** you can easily achieve this, by calling a custom *handler* from your sidebar `AXMSectionController` each time the `sectionViewWillAppear` is performed:

```
[NavigationSectionsManager registerController:[SidebarSectionController class]␣
↪forRoute:@"www/sidebar.html"];

@implementation SidebarSectionController
    - (void)sectionViewWillAppear {
        [self.section.bridge callHandler:@"on-sidebar-appears"];
    }
@end
```

This will correctly call the `on-sidebar-appears` handler on Javascript both when the sidebar is displayed the first time and each time its opened/closed by the user.

Android

On **Android** you can achieve a similar result by relying on the `sectionFragmentWillResume` method:

```
NavigationSectionsManager.registerController(this, SidebarSectionController.class,
↪"www/sidebar.html");

public class SidebarSectionController extends AXMSectionController  {
    @Override
    public void sectionFragmentWillResume() {
        super.sectionFragmentWillResume();
        section.getJSBridge().callJS("on-sidebar-appears", new JSONObject(), null);
    }
}
```

While this is enough to update the sidebar each time the user switched the application in/out of background, it won't update it when the sidebar is opened/closed through the sidebar button. So to achieve the same behaviour we had on iOS it is required to also `@Override` the `onSidebarOpened` method inside the `MainActivity` and forcefully trigger a `sectionFragmentWillResume` each time the sidebar is opened/closed:

```
public class MainActivity extends AXMActivity {
    @Override
    public void onSidebarOpened() {
        SectionFragment sidebarFragment = (SectionFragment)getFragmentManager().
↪findFragmentByTag("sidebar_fragment");
        ((SidebarSectionController)sidebarFragment.getRegisteredSectionController()).
↪sectionFragmentWillResume();
    }
}
```

Disabling Sidebar current menu to reload view

Js

In order to disable view reloading when click on sidebar menù you must do the check section side, you can achieve using html tag attribute and check the attribute when you click the menù item:

```
<div class="menu-item" active="true" id="dashboard-menu-item">Dashboard</div>
<div class="menu-item" id="profile-menu-item">Profile</div>
<div class="menu-item" id="contacts-menu-item">Contacts</div>

<script>
var dashboardMenuItem = $('#dashboard-menu-item')
dashboardMenuItem.on('singletap', function () {
        if (! dashboardMenuItem.attr("active")) {
                axemas.goto({
                        'url': 'www/sections/dashboard/section.html',
                        'title': 'Dashboard'
                });
                $(".menu-item").removeAttr("active");
                dashboardMenuItem.attr("active", "true");
        }
});

</script>
```

Quickstarting a New Application

AXEMAS provides a `Gearbox` extension that will generate a new `Axemas` project. Gearbox is a skeleton generation tool for Python

Installaton

Create a new virtual environment and install `axemas` with `pip`. This will include all needed dependencies like `gearbox`:

```
$ virtualenv --no-site-packages axemas_builder
$ . axemas_builder/bin/activate
(axemas_builder)$ pip install axemas
```

Usage

Run the following command to generate a project called `new_project` inside the directory where you want to create the new project:

```
(axemas_builder)$ gearbox axemas-quickstart -n ProjectName -p com.company.example
```

Project structure

The resulting project structure will be as follows:

```
/                   root of the project
----/android        Axemas Android native Java code
----/axemas-js      Axemas library JavaScript code imported by www
```

```
----/ios            Axemas iOS native Objective-C code
----/www            common share html/css/js/resource files
```

Maintain AXEMAS

AXEMAS framework

Repositories Structure

The project is divided into 5 repositories:

```
website
examples
framework
releases
doc
```

Inside framework repo (https://github.com/AXEMAS/framework) you will find the `AXEMAS` base library, composed by:

- Android library project
- iOS library project
- HTML library project

When you are done modifing the library please remember to update the debug projects and the release repository with the following commands.

Debug Projects Update

To update the `Android` and the `iOS` demo projects use the following command (examples folder must be in the same parent folder or will be cloned inside ../examples):

```
./update_debug
```

Il will delete all the old files in the iOS and Android projects and copy the new library files; native binaries and HTML. If you want to release demo apps remember to make necessary fixes, tag version and push everything.

Release Repository Update

Same as for the Debug Project Update, this will update the repository that the `gearbox axemas-quickstart` command needs to clone in order to quickstart a new project (releases folder must be in the same parent folder or will be cloned inside ../releases, be careful if you have pending changes):

```
./update_release
```

If you want to release new version remember to commit, tag version and push everything.

Android AXEMAS library

This library project is used to build the axemas.aar used inside the Android application.

How to use

After modifiying the library please inside this project's root folder:

```
./gradlew clean assemble
```

You will find the `app-release.aar` inside the `app/build/outputs/aar/` folder. Copy this file inside the `axemas-android` project in the `libs` folder.

iOS AXEMAS library

This project is used to build the axemas iOS library. Please use the following instructions to make a new release.

Setup

Install the command line tools, following the instructions at this link:

```
https://developer.apple.com/library/ios/technotes/tn2339/_index.html
```

New Release

Use the following command in the :: cp ios ./build_libaxemas

In the release directory you will find all the neccessary files to import the project in Xcode.

Indices and tables

- genindex
- modindex
- search

Index