

# Getting started

## Overview

HED applications framework is a new-generation multimedia device middleware designed to provide an interoperability layer between the specific hardware implementation of the device and the applications running on it, which means that a HED application can run in any HED supported device without change, where the unique limitations are the hardware capabilities.

In addition to the default HED applications (for IPTV and Consumer Electronics environments), the framework provides an easy way to integrate third party applications through its API. Nowadays this API is an own implementation of the new market trend for the design of interactive applications over IP, based on HTML5/CSS-TV/Javascript rendering the GUI using a Webkit browser. LambdaStream is following the evolution of HBB-TV and Open-IPTV Forum draft specifications of this technologies in order to include support for it as soon they are available.

There are two different kind of applications that can run over the HED API:

- **Simple apps.** All the application functionality runs over the HTML/CSS-TV/Javascript code running on the browser and accessing hardware capabilities using the provided API.
- **Advanced apps.** In an advanced application, a service (capability) can run in native code (C++, Erlang, etc,) in the box and communicate with the GUI in an asynchronous way using the API.

This document covers the Normal apps. development framework and describes the HED Javascript development Kit (**hed\_jsdk**).

## Base Technologies

There are several base technologies which are strongly recommended to review in order to develop HED applications:

•**JQuery**. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

References:

- Official sites: <http://jquery.com/>, <http://jquery.org/>
- General information: <http://en.wikipedia.org/wiki/JQuery>

•**AJAX**. Ajax (shorthand for asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create interactive web applications. References:

- General information: [http://en.wikipedia.org/wiki/Ajax\\_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))

•**JSON**. JSON (JavaScript Object Notation) is a lightweight data-interchange format, easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. References:

- Official site: <http://www.json.org/>
- General information: <http://en.wikipedia.org/wiki/JSON>

•**DOM**. DOM the Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. References:

- Official site: <http://www.w3.org/DOM/>
- General information: [http://en.wikipedia.org/wiki/Document\\_Object\\_Model](http://en.wikipedia.org/wiki/Document_Object_Model)

•**Javascript**. JavaScript is a small, lightweight, object-oriented, cross-platform scripting language.

References:

- Official site: <https://developer.mozilla.org/en/JavaScript>
- General information: <http://en.wikipedia.org/wiki/JavaScript>

## Introducing HED Applications

HED applications are composed of a set of HTML and JavaScript files running on the WebKit-based browser provided by the set-top box. The applications are based on web standards: HTML, CSS, JavaScript and HTTP. The interaction/events model is the HTML/JavaScript one.

The communication with the system is performed using the `hed_jsdk` API available on the system. This API provides a set of functions defined in the `HED` object. Most of this functions are asynchronous, which means that the result of the operations is returned using the callback concept.

The `hed_jsdk` API makes use of the jQuery library, which is available on the system in two different versions, v1.2.6 and v1.3.2.

## Application development

An application is basically a set of HTML and Javascript archives that can be stored locally or accessed through HTTP. In its simplest configuration it is composed by a configuration file and an entry point file. Additionally, several files can be included in order to modularize the application development or to include images or icons. The most common way to distribute an application is putting all these files together in one directory with the following application structure:

```
app_dir
|-- app.info (configuration file)
|-- app.html (entry point)
|-- app.js (additional Javascript)
`-- app_icon.png (application icon)
```

### Configuration file

The first and mandatory file is the configuration one, it is always named `app.info[app_name]`, usually just `app.info`. An `app.info` file is a set of attributes (key-value pairs) which carry all information needed to proper execute of the application. Some of the attributes defined here are mandatory, other are optional and other are defined by the developer and are application specific.

App.info example:

```
{ "version", "1.0" }.
{ "id", "app.api.example" }.
{ "name", "Example app" }.
{ "init_url", "app.html" }.
{ "description", "Example application." }.
{ "icon", "app_icon.png" }.
{ "type", "internal" }.
{ "position", "applications" }.
```

The complete list of the predefined attributes, both mandatory and optional, as well as its explanation is below:

- version**: application version.
- id**: application identifier, it must be unique.
- name**: is the name which the set-top box will show in the “Applications” menu. The maximum length allowed for this name is 25 characters.
- position**: this attribute indicates that the application will be present on the “Applications” menu.
- description**(Optional): application description.
- type** (Optional): this attribute establishes some default parameters so it's much easier to obtain the application desired behaviour. This parameters are referred to application position, size, opacity and focus availability. Allowed values are: default, dock and internal.
  - **default**: default application type. This is the most flexible way of application development as it gives the developer complete control over application size, position and opacity. This kind of applications can, as well, gain focus and implement interactivity. This type of application is selected automatically when the “type” attribute is not defined. Multiple applications can be running simultaneously.
  - **dock**: this offers to developers the same level of customization as “default” ones but is not focusable. The main intention for this applications is to provide information services (for example, an stock market viewer).
  - **internal**: this type of applications sets automatically size, position and opacity to fit exactly inside HED “Applications” box as native applications do. This applications are focusable so developers can add interoperability to them. Only one application can be running at an specific moment.
- init\_url**: this attribute specifies the entry point file for the application. This file is an HTML which contains the initial application code. This URL can be relative to the `app.info` file location or absolute (external web server). For example:
  - `{"init_url","app.html"}`. % Entry point file is in the same folder as `app.info`.
  - `{"init_url","http://server:port/path/to/app.html"}`. % Entry point file located in an external web server
- onload**: this attribute specifies a js file that will be loaded and evaluated on the application html context. This URL can be relative to the `app.info` file location or absolute (external web server). For example:
  - `{"onload","hook.js"}`. % Hook file is in the same folder as `app.info`
  - `{"onload","http://server:port/path/to/hook.js"}`. % Hook file located in an external web server
- icon**(Optional): this is the relative path to the icon image. This image will be shown in the “Applications” menu. The supported formats for this icon are png, jpg and gif and its size is 168x86px.
- window\_xpos** (Optional): this attribute is used to set the window horizontal position in pixels relative to the top left corner of the screen. This attribute is valid for “default” or “dock” applications and it has no effect in “internal” ones. Default value is 0.
- window\_ypos** (Optional): this attribute is used to set the window vertical position in pixels

relative to the top left corner of the screen. This attribute is valid for “default” or “dock” applications and it has no effect in “internal” ones. Default value is 0.

•**window\_width** (Optional): window width in pixels. This attribute is valid for “default” or “dock” applications and it has no effect in “internal” ones. Default value is 1280.

•**window\_height** (Optional): window height in pixels. This attribute is valid for “default” or “dock” applications and it has no effect in “internal” ones. Default value is 720.

•**window\_opacity** (Optional): opacity level, allowed values are from 0 to 255. 0 means completely transparent and 255 completely opaque. This attribute is valid for “default” or “dock” applications and it has no effect in “internal” ones. Default value is 200.

•**window\_userAgent** (Optional): custom user agent for the application window, if not specified default user agent is used.

•**debug** (Optional): if true the debug mode is enabled, on this mode shows a debug window when the application is running. The debug window is intended to show the console messages and the javascript Errors. The debug mode can be enabled or disabled on demand using *HED.window.enable\_debug* and *HED.window.disable\_debug*, should take into account, that on debug mode enabled exists an extra consumption of resources, so should only be enabled for developing purposes.

•**debug\_xpos** (Optional): this attribute is used to set the window horizontal position in pixels relative to the top left corner of the screen for the debug window. Default value is 0.

•**debug\_ypos** (Optional): this attribute is used to set the window vertical position in pixels relative to the top left corner of the screen for the debug window. Default value is 500.

•**debug\_width** (Optional): debug window width in pixels. Default value is 1280.

•**debug\_height** (Optional): debug window height in pixels. Default value is 220.

The `app.info` file encoding has to be UTF-8.

## The application entry point file

The application entry point file for a HED application is a plain HTML file. The developer has to import some mandatory Javascript HED libraries and register some callbacks in order to get the application running properly. These libraries are:

- <http://localhost:8080/hedjsdk/jquery/core/v1.3.2/jquery.min.js>
- <http://localhost:8080/hedjsdk/jquery/plugins/json/v1.0/jquery.json.js>
- <http://localhost:8080/hedjsdk/jquery.hed.js>

And the callbacks needed to be registered can be split into two parts:

- Application status management
- User interaction

The application must call the function `HED.ready` to ensure the system is ready to process events and execute the code properly.

```
HED.ready(function() {
    $('#message').html('Hello World!!!');
});
```

The user interaction is managed by registering the application to the Javascript `key` and `focus` events. Focus events can not be registered in the “dock” type applications. There are two reserved key events for “internal” and “default” application types:

- default: HED.KC\_MENU gives the control (focus) to the main menu while the application keeps running in background. Application focus is recovered by selecting again the application from the “applications” menu.
- Internal: HED.KC\_MENU hides the menu and HED.KC\_RETURN closes the application and returns focus to the main menu.

## Entry point file example

The following code is a sample application that obtains the system MAC address and prints it on a div with id `mac`; then, when the `HED.KC_RETURN` key is pressed, the application is closed:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <script type="text/javascript"
    src="http://localhost:8080/hedjsdk/jquery/core/v1.3.2/jquery.min.js"></script>
  <script type="text/javascript"
    src="http://localhost:8080/hedjsdk/jquery/plugins/json/v1.0/jquery.json.js"></script>
  <script type="text/javascript"
    src="http://localhost:8080/hedjsdk/jquery.hed.js"></script>
</head>
<body>
  <script type="text/javascript">
    /*
     * Check middleware state before requesting any data.
     */
    HED.ready(function() {
      /*
       * Print a text indicating that the request is in process.
       */
      $('#mac').html('Requesting mac addresses...');

      /*
       * Call get_mac_addres function.
       */
      HED.configuration.get_mac_address(
        function(oResult) {
          /*
           * Print result when success.
           */
          $('#mac').html('Your MAC addresses are:');
          for(var i in oResult){
            $('#mac').append('<div>'+i+' - ' + oResult[i]+'</div>');
          }
        },
        function(oError) {
          /*
           * Print result when an error occurs.
           */
          $('#mac').html('MAC address could not be retrieved');
        }
      );
    });
  </script>

  <div id="mac">
    Loading application...
  </div>
</body>
</html>
```

The system call `HED.configuration.get_mac_address()` is a function available in the system library <http://localhost:8080/hedjsdk/jquery.hed.js>. This is an asynchronous call and two functions are passed to it to handle the response:

- function(oResult)
- function(oError)

The first one is invoked when `HED.Configuration.get_mac_address()` gets a correct response and the second one when an error occurs. In both cases, `oResult` and `oError` are the `HED.Configuration.get_mac_address()` JSON responses.

```
/*
 * Call get_mac_addres function.
 */
HED.configuration.get_mac_address(
  function(oResult) {
    /*
     * Print result when success.
     */
    $('#mac').html('Your MAC addresses are:');
    for(var i in oResult){
      $('#mac').append('<div>'+i +' - ' + oResult[i]+'</div>');
    }
  },
  function(oError) {
    /*
     * Print result when an error occurs.
     */
    $('#mac').html('MAC address could not be retrieved');
  }
);
});
```

## Executing a HED application in a set-top box

The fastest way of getting an application running in a set-top box is using an USB pen drive. The root folder of the USB memory must contain a folder called “applications”. You can put all your application files inside this folder but it's much easier for application management to create a different subfolder for each application. The folder structure then should look like this (you may have any other auxiliary files):

```
USB
|-- Applications
    |-- app_dir
        |-- -- app.info
        |-- -- app.html
        |-- `-- app_icon.png
    `-- other_app
```

Once all the files has been copied to the USB memory, it has to be plugged to the set-top box. HED system will automatically update the available applications every time “applications” menu is entered, so the new application can be executed as any other application under the “applications” menu. To force an update of the application list just press the option button and select update list.