# WebOS JS Playback Engine - High Level Specifications

V.1.06

[**WebOS JS Playback Engine - High Level Specifications**](#_66wufv3gxahy) **1**

[**I. Project Scope**](#_1crb7o3s79ol) **2**

[I.1 In Scope](#_u2f2wsy3yeon) 2

[I.2 Out of Scope](#_pis136ln52qz) 3

[I.3 Risk Assessment](#_jcz5zd61lyhc) 3

[**II. Application Global Architecture**](#_7axybr1sfc50) **4**

[II.1 General Class Diagram](#_xw22hr6nps9b) 4

[II.2 “Core” Services](#_p3umv8ljxjds) 4

[II.2. Rendering Services](#_8e2b8n4wpd4h) 5

[II.3. Non-Rendering Services](#_uzn93ppdw2z0) 8

[II.4. The “Configuration” Services](#_1ebjtgb2ytnw) 9

[II.5. “Transversal” Services](#_5dxdd1gi5v6k) 11

[II.5.1 “SDK” Services](#_hh3ja3kyg5bs) 12

[**III. Implementation Guidelines**](#_4gaswikw3xow) **16**

[III.1. General Approach](#_t6mpvklh03lq) 16

# 

# I. Project Scope

## I.1 In Scope

**Main Scope:**

The main scope of the project is to implement a Playback Engine on WebOS LG TV Signage platform.

**Additional Requirements:**

1. Modular integration of a new/additional TV OS that will support this Playback Engine - without to affect the existing components

(ex. adding support for Panasonic Signage TVs, etc)

1. Modular integration of a new/additional Firmware Versions that will support this Playback Engine without to affect the existing components

(ex. adding support for Tizen SSP5)

3. Modular integration of new/additional Rendering/Non-Rendering Services without to affect the existing components.

(ex. adding an Audio Playlist Service)

4. Modular integration of a new Media Type with a minimal impact on the existing components.

(ex. adding support for Streams - a new media-item type)

5. Supply new versions of Rendering and/or Non-Rendering services without to affect the existing versions

If needed, this new service versions can run in parallel with the existing ones

The reasons for such implementations can be:

* a native service for a specific firmware version
* a customized service for a specific customer
* an enhanced version of a service that add new functionalities

6. Different Rendering or Non-Rendering services dedicated to the same specific scope can be plugin and run in a customized way (depending on a set of properties)

(ex.1. one Video Renderer plays a set of video types, another Video Service play another set of videos, ones play 4K, others specific video types etc).

(ex.2. adding a Youtube video Renderer)

7. Adding additional properties to the existing Playlist / Media Items or Services that will trigger new functionalities - with a minimal impact on the existing components.

8. The implementation in the scope of this project, will leverage and will reuse (as much as possible) the existing SOC application code, by maintaining the modularity and the flexibility of the architecture described in this document.

9. The proposed architecture should be a (new) step in the direction of a “A Common Playback Engine Architecture”

Note : a dedicated chapter (“Chapter XX” - that will be added) will analyze how the proposed architecture respect the main requirements.

**Additional Scopes:**

By implemented the “Playback Engine on WebOS LG TV Signage”, we will also obtain the following 3 additional implementation.

1. a “Tizen Playback Engine on Samsung Signage TV” - This means the existing SOC application will be transposed inside the new architecture.
2. A Java Script Playback Engine that based on Node.js platform.
3. A Java Script Playback Engine that will work in browsers with a minimal setup (local/remote web server and/or local file system)

## I.2 Out of Scope

Out of Scope 2:

A list of service (VOD, Audio, HDMI Pass through, Sync Playback were included here) but will be in the scope of the next release

Out of Scope 1:

The automatic plugin of different components based on an xml/json description (that later can arrive from a server) is out of the scope of this project.

This can be achieved in a future phase if the decisions will be in that direction.

Even so, different platform specific components will be instantiated (or not) based on different Target TV OS, Firmware Versions, media types, additional properties

Also the specific services that will form the PlaybackEngine for a targeted platform will be plugin in the application based on an ApplicationConfigurator Service

## I.3 Risk Assessment

The team (=”Windows” team) has no experience in WebOS or Tizen OS SDK.

So we are expected that the initial phases in the implementation to advance slower in comparison with an experienced team. For this reason, t5he global risk project was fixed at 15% from total.

# II. Application Global Architecture

## 

## II.1 General Class Diagram

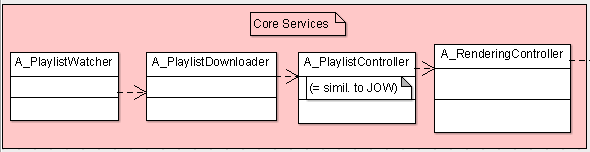
Link:

<https://dochub.com/m-alexandru-zu4dxi/raEv8l/playbackenginewebosgeneralclassdiagram1-06-png>

## II.2 “Core” Services

The “Core Services” are components that are used to implement the playback engine main functionalities. We named them “Core Services” because they represent “the backbone“ of the playback engine.

This group of services contains the following services:



1. **“Playlist Watcher” Service** - is a service in charge to monitor from time to time if a new playlist become available on the server.

2. **“Playlist Downloader” Service** - is the service responsible to download (whenever is necessary) a new playlist from the server.

3. **“Playlist Controller”** **Service** (that is the equivalent of JOW on windows/android platform) will implement all the Playlist Logic

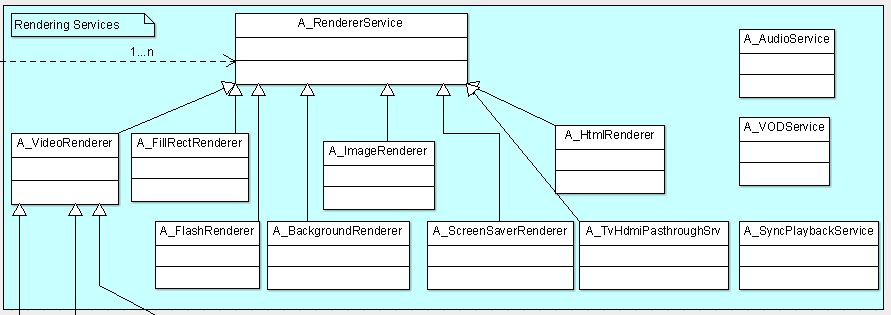
This is a CRITICAL Component of the application.(critical = needs maximum fiability, needs to be stable on the long runs of the application, can be subject to error prone, etc..)

4.  **“Rendering Controller”** **Service** - is the service that controls the Rendering Services based on the commands that it received from the PlaylistController.

This is another CRITICAL Component of the application.

## II.2. Rendering Services

The rendering services are a hierarchy of services that are able to render on the screen different media types ( the audio playback is considered as a rendering service too).



1. **“The Renderer Service”** is the root service of this hierarchy (A\_RendererService). The service is needed in order to control the rendering services in a uniform way.

2. **“Image Renderer” Service”**  perform the rendering of different types of images based on different parameters (stretch, fit, fill, center, normal)

3. **“Video Renderer” Service Hierarchy**  performs the rendering of different types of videos based on different setup parameters (like : keep aspect ratio etc). Usually it covers the playback of the rendering streaming too.

Usually each TV Signage OS supplies a native api for video rendering (that also evolved during different firmwares lifecycles).

For this reason we are talking (from the beginning about a hierarchy of service: the Html5VideoRenderer (general available), plus different video renderers supplied by each TV OS that will also have their versions and their evolutions)

4. **“HTML Renderer” Service**  performs the rendering of different types of html pages and html5 templates based on different setup parameters (like : keep aspect ratio etc). Usually it covers ¨the playback” both of the HTML Pages and of HTML 5 Templates.

5. **“Flash Renderer” Service**  performs the rendering of Flash media type. The Flash rendering is not supported by different TV OS, so this type of renderer can be absent on some OS TV playback engines. However the html5 flash renderer will be implemented.

6. **“RenderingZone”** **Service** is a rendering service that is implemented by a hidden container (/windows). His existence is important in order to manage the positions and the z-order on the screen of different rendering services

7. **“Background Renderer” Service** is a rendering service that manages the display of the playback background (on one or multiple screens)

8. **“Screen Saver Renderer” Service** is a rendering service that manages the display of a “screen-saver” when there are no valid media types to be played via the current playlist. Its implementation leverage the implementation of the other services

9. **“TvHdmiPassThroughService”**

The HDMI Pass-Through passes video and audio signals from a high definition (HD) source such as a Blu-ray player or a HD set top box to a home theatre system (HTS) via a HDMI cable. The home theatre system plays the audio from the HD source through its own speakers and sends the unaltered video signal to a TV via a second HDMI cable. Alternatively the original source can be a signal TV from a high definition (HD) source having a TV Tuner.

A home theatre system that supports HDMI pass-through will have one HDMI OUT port to send the HDMI signal to a TV, and at least one HDMI IN port to receive a HDMI signal from the connected external device.

Note: The implementation was scheduled for a future Release 2. To discuss about his capabilities before the implementation

**Special Renderer Services:**

10.  **“Sync Playback” Service** - different playlists on different media-players can run in sync-playback. There is a playlist naming convention that indicate the sync-master and the sync-clients. There is also an UDP Broadcasting service that established the communication between master and clients. The module is implemented on Android and on Windows. The Sync Playback part on videos is implemented inside Video Player (inside Video Renderer)

Note: Currently, this renderer module (that have a command/controller part too) is not implemented in SOC. The implementation was scheduled for a future Release 2.

11.  **“VOD” Service** - the service displays all the existing media items of the player-box and allows the user to make a playlist with all of them. The created playlist can be played independently from the playback and is controlled via a Remote Control

Note: This service doesn’t exist in SOC. The implementation was scheduled for a future Release 2.

(To discuss the need for its implementation)

12.  **“Audio” Service** - the audio service is a dedicated service that supplies an Audio Renderer that can play the audio Media Items. It can also exposed a set of interfaces to access the Mood Media Audio solution (to be discussed).

Note: This service doesn’t exist in SOC (but the audio element is available on the “Server Preview” that is a SOC module. The implementation was scheduled for a future Release 2.

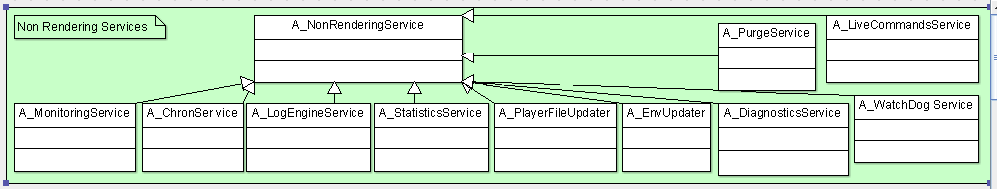
(To discuss the need for its implementation)

## 

## II.3. Non-Rendering Services

The Non-Rendering Services are services that do not render a Media Item Type on the screen.

Even the current non-rendering services are mainly related to monitoring/controlling/logging task, in the future additional non-rendering services can be added that will implement specific functionalities.



1. The **“NonRenderingService”** is the root service of this hierarchy (A\_NonRendererService). The service is needed in order to control the non-rendering services in a uniform way.

2. **The “MonitoringService”** - is a service that supplies information to the server regarding the good or bad functionality of the Playback Engine.

It sent also additional information regarding the context in which the Playback Engine run: CPU, memory, storage (hdd, usb), networking, software version etc…

It is based also on a **“Capture Screen” Service** that captures from time to time the playback screen (or multiscreen)

3.  **The “ChronService”** - is a service that schedule the periodical execution of other services based on a supplied configuration

Note: This service doesn’t exist in SOC. The equivalent is a Set of Timeouts used by each service (to schedule himself)

4. **The “LogEngineService”** - is a service that set the log periodically the log file to the server

5.  **The “StatisticsService”** - is a service that sent statistics with the media-items that have completely played in the last 24h.

5.a The Statistics Service can be later extended to become an **“Analytics” Service**

6.  **The “PlayerFilesUpdater” Service** - is a service that verifies from time to time if a set of specified player files are in sync with the similar reference files hosted on the server.

7.  **The “Env.Updater” Service** - is a service that on Windows/Android platforms verifies from time to time if a new Software / Hardware and/or Client Updates exist on the server and need to be downloaded by the client. If exist, they are downloaded and installed.

Note: This service doesn’t exist in SOC.

(To discuss the need for its implementation)

8. The **“DiagnosticsService” (= “PlayerConfig” on Windows) -** is a service that on Windows/Android platforms presents (on the screen), different information regarding the current status of the playback system

Note: This service doesn’t exist in SOC. (To discuss the need for its implementation)

9.  **“The WatchDog Service” -** is the service that verifies from time to time if the playback engine is up a running. In case that the Playback Engine is not running it reboots the player-box from a specified number of time in order to restart the Playback Engine on a clean context.

Note: This service doesn’t exist actually in the current version of SOC

10. The **“StartupSettings”** Service - This service will perform some configuration settings during the startup of the playback engine (like: Audio levels, etc.)

11. The **“LiveCommands”** Service - This service will perform the execution of the live commands that can be received from the server

12. The **“Purge Server”** Service - This service will perform some purges on the File System (usually on the USB) of TV Signage OS.

## II.4. The “Configuration” Services

The “Configuration” services are represented by the services that ensure ***the Creation*** of each specific service and the ***Injection of Dependencies*** among these services, based on a PlaybackEngine configuration file supplied for each Targeted Platform/SDK/specific Customers etc...

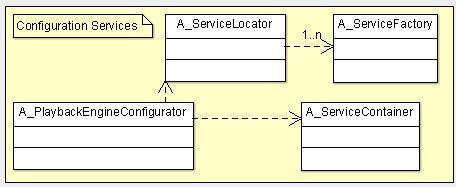
Why these services are needed?

By externalising the above operations we ensure the modularity of the application by decoupling dependencies between the services that composed a PlaybackEngine solution. In this way, the specific services will interact only via an abstraction layer (see ¨Implementation Guides” chapter for more details) .

Without this set of services that ensure the decoupling of the dependencies and the modularity of the application, the requirements specified in the application Scope cannot be fulfilled.

So, these services are needed in relation to the dependency inversion principle, which concerns itself with decoupling dependencies between high-level and low-level layers through shared abstractions.

The “Configuration” services are represented by the following services:



1. The **“PlaybackEngineConfigurator” -**  the PlaybackEngineConfigurator load the configuration of the PlaybackEngine (a .js or .xml file) for a target platform (like: WebOS, Tizen, Node.js) and for a specified Firmware(/SDK) (like Tizen SSP3 , Tizen SSSP4, WebOS 4.06 etc..and - if needed - for a specified subset of customers, etc.).

This loaded PlaybackEngine Configuration is supplies next to the ServiceContainer and ServiceLocator services in order to create the specific services of the application and to inject dependencies among these services.

This services is that one that load a PlaybackEngine solution for a dedicated platform.

*Examples* of such configurations files will be:

* ***PlaybackEngine\_DefaultConfiguration\_forTizenOs\_SDKSSSP4.js*** *or* ***.xml*** where inside we find all the services that are used for Tizen\_SSSP4\_PlaybackEngine or
* ***PlaybackEngine\_DefaultConfiguration\_forWebOs\_SDK4.06.js*** *or* ***xml*** where inside we find all the services that are used for Tizen\_SSSP4\_PlaybackEngine

1. The **“Service Container” -**  is a service that will contains a list with the instances of all the main services that are used by the application

Why is needed?

The service is needed in relation to the dependency inversion principle, which concerns itself with decoupling dependencies between high-level and low-level layers through shared abstractions. It is, not only, the service that hosts all the other services but is the service that injects dependencies among the services that composed a PlaybackEngine solution for a dedicated platform.

1. The **“Service Locator” -**  is a service that contains ***a set of Service Factories*** in order to create outside the implementation of any specific service, any new instance of any service of a specific platform.
2. A set of **“Service Factories”**  that create the new instances on the conditions specified in different **ServiceCreation** contexts

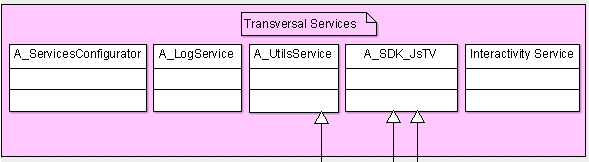
## II.5. “Transversal” Services

“Transversal” Services are components that are used across all the other services. For this reason we named them “transversal”.

They are needed in order to :

* encapsulate the TV OS specificity included the evolutions of each TV SDK (supplied inside each Firmware version of the TV),
* also they offer the reusable code (like. time/string/number functions, html download/upload etc..),
* together with the “log” functionality supplies to can track the execution of each service.

This group of services contains the following services:



1. **“SDK” Services** - is a hierarchy of SDK services having as root class A\_SDK\_JsTV class (with A - meaning “*A*bstract”). This hierarchy will encapsulate the specificity of each TV Signage OS and also (via additional subclasses) the specificity of each new Firmware inside each TV OS.

2.  **“Utils” Service** - In addition to the hierarchy of SDK Services a “Utils” Service will be in charge to encapsulate the reusable code (like time/string/numbers functionality, html download/upload, files common processing functions). So the “Utils” Services offers reusable code that is constructed on top of “SDK Services” an/or based on external libraries.

3. **“Log” Service** - The log service is another service that is used across all the other components of the application. It implements the log functionality across all the log services.

Note: not to be confounded with the “LogEngine” service that is a component that sent periodically log information to the Server.

4. **“ServicesConfigurator”**  - is the service that load different configuration files (like: network\_settings.js or . xml etc.). It is composed by a specified set of “FileConfigurator” Services that loads a single specified .xml file.

A specific service will configure and will request one or more FileConfigurators in order to obtain the configuration information.

5. **“Interactivity” Service**  - is the service that will ensure the interactivity of the PlaybackEngine.

Note: This service doesn’t exist in SOC.

(This service will be implemented in a Release 2 of the project)

### II.5.1 “SDK” Services

Why are needed?

The implementation of the SDK Services is needed in order to encapsulate all the specificity of different WebTV OS Systems.

The other services will work only with the abstract service “A\_SDK\_JsTV” in order

This “A\_SDK\_JsTV” (A is the notation for abstract) will offer a set of interfaces that will not depend on the specificity of each system

General Diagram

There are 2 types of Java Script/TV “system specificity” :

* A major one : TV OS Types / Node.js / etc.. (ex. Tizen OS, WebOS, Node.js etc...)
* A minor one : The “”ongoing” versions inside a specific OS (= firmware/SDK versions)

This situation is modeled via a hierarchy a SDK classes that will have on top the abstract service A\_SDK\_JsTV

**II.5.1.1 The Root JavaScript TV SDK Service**

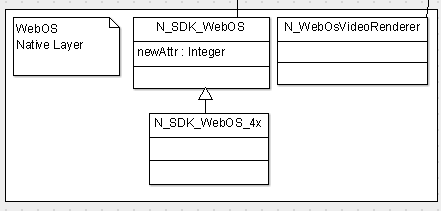
General Description:

An abstract SDK root class (A\_SDK\_JsTV), that will offer a set of SDK interfaces (one for each SDK used package) will encapsulate the specificity of all the TV Signage that are based on Javascript SDK.

All the other services will work directly with this abstract SDK layer (A\_SDK\_JsTV) service that will encapsulate via a set of SDK-interfaces all the needed specifics of each TV and of the Node.js functionality.

**II.5.1.2 The Native WebOS SDK Service**

N\_SDK\_WebOS Service - The Native Service that will encapsulate the WebOS SDK specificities

****

Note-1: The N\_SDK\_WebOS class will have a set of subclasses that will encapsulate different SDK packages (the specification of these subclasses will be subject of the Details Specifications).

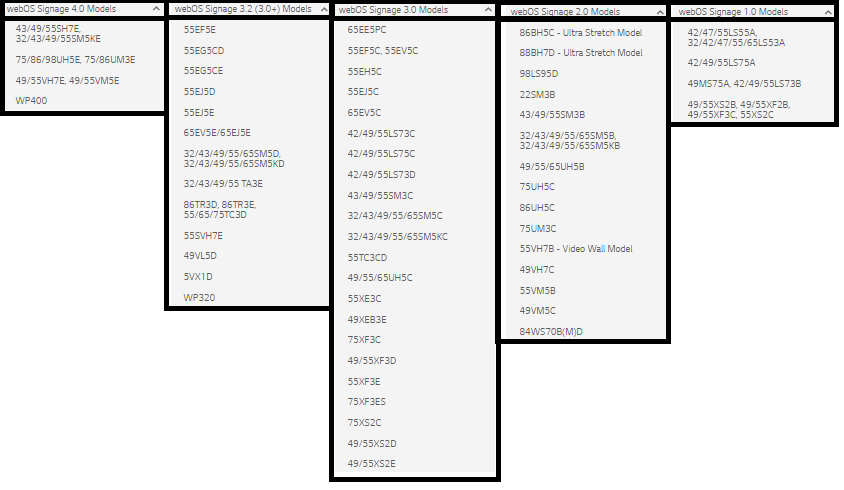
Note-2: Only the needed SDK packages (and inside a SDK package only the needed SDK functions) will be encapsulated to the native and abstract SDK layers

Note-3: In Addition to the N\_SDK\_WebOS class, a native class N\_SDK\_WebOS\_4x will encapsulate the specificity of Web OS SDK 4.06

Note-4: The native subcomponent “Sync Video Playback” relative to the Mixed Service “Sync Playback” is not part of this diagram (but will be added during the Specification phase of Sync Playback Service, subject of a future Release 2)

Note-5: The specificity (if any) of each WebOS version (= Firmware/SDK versions) will be encapsulated in a derived class N\_SDK\_WebOS\_<yyy> . The first targeted SDK version will be the last available one at this moment (= WebOS SDK 4.06)

The links between Firmware Versions and TV Models are indicated below



*Question: What are the current TV Models that we are targeting?*

The answer to this question will trigger the number of WebOS versions that we need to manage

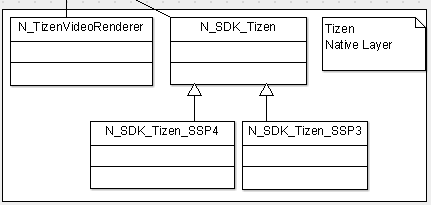
**II.5.1.3 The Native Samsung Tizen SDK Service**

N\_SDK\_Tizen Service - The Native Service that will encapsulate the Samsung Tizen SDK specificities

Similar with the Web OS SDK encapsulation, a specific class N\_SDK\_Tizen (having a set of subclasses for each needed SDK package) will encapsulate the Samsung Tizen SDK.

The specificity of different firmware/SDK versions will be encapsulated in the derived classes: N\_SDK\_Tizen\_SSP4, N\_SDK\_Tizen\_SSP3, etc.

In addition, the N\_TizenVideoRenderer service a Native Video Renderer will be implemented, that will encapsulate the Native Video Renderer api supplied by Samsung Tizen OS



**II.5.1.3 The Native Node.js SDK Service**

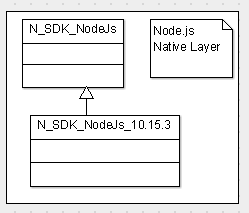
In addition to the encapsulation of the SDKs for WebOS and Tizen the first implementation will encapsulate the needed packages of Node.js

This will facilitate:

- the debugging and the development speed of the application (by supplying a full Windows environment with the exception of Native/Mixed services that need to be developed and tested on the emulators and on Signage TV)

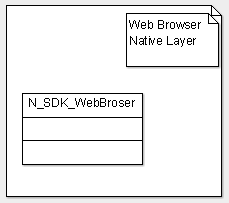
and also

* will supply a full Java Script Playback Engine (based on Node.js) on Windows and other operating systems



**II.5.1.4 The WebBrowser SDK Service**

In addition, the JavaScript Playback Engine will work on the WebBrowsers (like Chromium, Chrome, Microsoft Edge), with a minimum access on a local/remote Web Server or to the local file system (to discuss on these aspects)



# III. Implementation Guidelines

## III.1. General Approach

1. Each Service will be defined first in the Abstract Layer

as <A\_Component\_K> having a set of interfaces <I\_Interface\_K1>...<I\_Interface\_Kn>

b. Next, each Abstract Service will have one or more Implementations either in the Reusable Layer and/or in the Native Layer.

c. In some special cases (ex. Sync Playback Service) we can have Mixed services that will have some subcomponents implemented in the Reusable Layer and some subcomponents implemented in the Native Layer (like the Native Sync Video playback)

Note-1: This special cases should be avoided as much as possible.

Note-2: If not possible, the subcomponents of a Mixed Service needs to be either Reusable, either Native (but dot a mixture of two)

d. The above implementations (specified at b. and c.), are instantiated in the ServiceFactory classes of the ServiceLocator, based on some specified conditions, and next injected by the ServiceContainer service as Abstractions

inside each specific Service that used them. With this approach the decoupling among specific implementations is realized 100%.

d. Finally, for each targeted platform (WebOS, Tizen, NodeJs) and for each targeted Firmware (like SSP3, SSP4, WebOS4.06) the PlaybackEngineConfigurator service will load a PlaybackEngine configuration file that describes the service-implementation list that composed a specific PlaybackEngine for each TV Signage Model.