This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n: 761999

EasyTV: Easing the access of Europeans with disabilities to converging media and content.

D5.9 Final release of the EasyTV Service Development Kit

EasyTV Project

H2020. ICT-19-2017 Media and content convergence. – IA Innovation action.

Grant Agreement n°: 761999

Start date of project: 1 Oct. 2017

Duration: 33 months

Document. ref.: D5.9

Disclaimer

The EasyTV Consortium consists of the following partners:

	Partner Name	Short name	Country
1	Universidad Politécnica de Madrid	UPM	ES
2	Engineering Ingegneria Informatica S.P.A.	ENG	IT
3	Centre for Research and Technology Hellas/Information Technologies Institute	CERTH	GR
4	Mediavoice SRL	MV	IT
5	Universitat Autònoma Barcelona	UAB	ES
6	Corporació Catalana de Mitjans Audiovisuals SA	CCMA	ES
7	ARX.NET SA	ARX	GR
8	Fundación Confederación Nacional Sordos España para la supresión de barreras de comunicación	FCNSE	ES
9	Unione Italiana dei ciechi e degli ipovedenti	UICI	IT

PROGRAMME NAME:	**H2020. ICT-19-2017 Media and Content Convergence – IA Innovation Action**
PROJECT NUMBER:	761999
PROJECT TITLE:	EASYTV
RESPONSIBLE UNIT:	ARX
INVOLVED UNITS:	ENG, MV
DOCUMENT NUMBER:	D5.9
DOCUMENT TITLE:	Final release of the EasyTV Service Development Kit
WORK-PACKAGE:	WP 5
DELIVERABLE TYPE:	Demonstrator
CONTRACTUAL DATE OF DELIVERY:	31-01-2020
LAST UPDATE:	31-01-2020
DISTRIBUTION LEVEL:	PU

Distribution level:

PU = *Public,*

RE = *Restricted to a group of the specified Consortium,*

PP = *Restricted to other program participants (including Commission Services),*

CO = *Confidential, only for members of the LASIE Consortium (including the Commission Services)*

Document History

VERSION	DATE	STATUS	AUTHORS, REVIEWER	DESCRIPTION
v.0.1	05/11/2019	Draft	Stavros Skourtis ARX Chrysostomos Bourlis ARX Athanasios Chatziathanasiou ARX Christos-Menelaos Vlemmas ARX Zisis Kolias ARX	Table of Contents definition and document structure
v.0.2	12/12/2019	Draft	Stavros Skourtis ARX Athanasios Chatziathanasiou ARX Chrysostomos Bourlis ARX Christos-Menelaos Vlemmas ARX Zisis Kolias ARX Nicolamaria Manes MV	First draft
v.0.3	20/12/2019	Draft	Stavros Skourtis ARX Christos-Menelaos Vlemmas ARX Chrysostomos Bourlis ARX Zisis Kolias ARX Athanasios Chatziathanasiou ARX	Service Manager SDK and Service Registration tool
V0.4	24/12/2019	Draft	Francisco Moreno UPM	Accessibility services SDK
v.0.5	23/01/2020	Draft	Zisis Kolias ARX Stavros Skourtis ARX Chrysostomos Bourlis ARX Christos-Menelaos Vlemmas ARX Serafeim Kosyvakis ARX Athanasios Chatziathanasiou ARX	First full version ready for internal review

V0.6	28/01/2020	Draft	Francisco Moreno UPM	Review
V0.7	30/01/2020	Draft	Giuseppe Vitolo ENG	Review

Definitions, Acronyms and Abbreviations

ACRONYMS / ABBREVIATIONS	DESCRIPTION
SDK	Service Development Kit
API	Application Programming Interface
CS	Companion Screen
HbbTV	Hybrid Broadcast Broadband TV
VOD	Video on Demand
ASR	Automatic Speech Recognition
TTS	Text to Speech

Table of Contents

List of Figures

List of Tables

Executive Summary

The present document describes the design and architecture of the EasyTV SDK (Service Development Kit) that is part of Task 5.4 “EasyTV Service Development Kit”. This task focuses on providing open source tools, libraries, repositories and catalogues assisting the development and the integration of services within the EasyTV technical platform.

The document will cover the following topics:

Chapter 1 will provide an introduction and an overview of this document, as well as the purpose of the SDK.

Chapter 2 will cover the natures and forms, that the SDK can take and the use cases, that it can cover.

Chapter 3 will describe all the components that compose the SDK. The components can be software libraries or tools.

1. Introduction

A **Service Development Kit** (SDK) is a collection of software development libraries and tools, that allows developers to create applications for certain software or hardware platforms. Some of the most common examples of SDKs, that can be found, are the iOS and Android SDK, which respectively allow developers to create applications for the iOS and Android operating systems.

The **EasyTV Service Development Kit** (EasyTV SDK) is a toolkit for the development of services for the EasyTV platform. This toolkit contains open source command line tools and software libraries, that allow third-party developers to use or create EasyTV Services.

There can be two distinct use cases of the EasyTV SDK for software developers. In the first case, they can use the EasyTV SDK to include EasyTV services in third party products. In the second case, they can create new services using the EasyTV SDK and integrate them in the platform, by using the tools provided by the EasyTV SDK.

The SDK is designed to have a modular architecture, so that the developers can use part of it as needed. Some of the components of the EasyTV SDK, that will be described in detail in the next sections, are the following.

- Integration of the EasyTV HbbTV [1] Companion Screen application into a third party HbbTV terminal application
- Integration of an HbbTV Companion Screen application with HbbTV terminal applications that are compatible with the EasyTV SDK
- Accessiblity features for HbbTV Companion Screen applications
- Command line tool that registers services to the Service Manager (which is described in deliverable 1.4)
- SDK for the internal and public Web API of the Service Manager

2. Nature of the EasyTV SDK

There are two distinct use cases of the SDK, that a third party developer can use to develop new services. On one hand, a developer may wish to use the EasyTV services and integrate them into other third party products and on the other hand, they may wish to develop services that integrate and cooperate with the EasyTV platform.

The EasyTV SDK will cover both of these cases and in the following sections they will be described in detail.

2.1. Integration of EasyTV services in third party products

This is the most common form that an SDK can take, like, for example, the Android SDK developed by Google. This is the case that most developers will need. It involves a third party developer that has checked the features of the EasyTV platform and has decided that it would be beneficial to integrate some of them into their own third party products. The EasyTV SDK is designed and developed in a modular way in order to allow the third party developer to only use the components that they need and not be forced to include the entire SDK in their project.

After choosing a service that they want to integrate, they will start the process by including the corresponding packages into their codebase and, following the instructions as defined in the EasyTV SDK documentation, they will be able to use its features.

At this point, the third party developers have, successfully, developed a product that contains some of the EasyTV platform features. This product, though, is not part of the EasyTV platform and this is how it differentiates from the other use cases that are described in the next chapter.

An example of a component that falls under this use case is the one for the interoperability between an HbbTV application and the EasyTV Companion Screen application. In this case, the third party product is the HbbTV terminal application and the service is the EasyTV Companion Screen. By using the corresponding libraries the HbbTV terminal application will be able to communicate and work with the EasyTV Companion Screen application, so the users of the HbbTV terminal application will benefit from the accessibility features.

2.2. Integration of third party services in the EasyTV platform

In order to provide a way for the EasyTV platform to be extended with new services, the EasyTV SDK must, also, cover the use case where a third party developer wishes to integrate a third party service into the EasyTV platform. In this case, the EasyTV SDK will provide a set of command line tools and libraries in order to assist the development of new services and, also, to allow the integration of this new service into the EasyTV platform.

To complete this integration, the EasyTV SDK will communicate with the Service Manager, which is the entry point of every communication between the Content Owner and the EasyTV platform.

An example of a component that covers this use case is the registration tool that will make the new service and its capabilities known to the Service Manager.

3. Components

3.1. Introduction

The EasyTV SDK is divided in multiple components, that offer accessibility features supported by the platform. These components are designed to be as independent as possible, so it will be easier for developers to use them. Moreover, each component is documented separately and is contained and distributed through different solutions (different package managers depending on the technology used). In the following tables the components are listed according to their relation to HbbTV applications or the Service Manager.

Type of application it is intended for	Objective
Terminal	Achieves the interoperability with the EasyTV Companion Screen application.
Companion Screen	Achieves the interoperability with terminal HbbTV applications that use the EasyTV SDK.
Companion Screen	Offers accessibility features for buttons.

Table 1 SDK Components related to HbbTV applications

Name	Objective
Service Registration Tool	Allows the registration and the management of information about the services that the Service Manager will handle.
SDK for EasyTV services	A library wrapping the internal Web API of the Service Manager in an object oriented way. It is intended to be used by the services in the EasyTV platform.
SDK for the Content Owner	A library wrapping the public Web API of the Service Manager in an object oriented way. It is intended to be used by the Content Owner.

Table 2 SDK Components related to the Service Manager

3.2. SDK for HbbTV applications to integrate with the EasyTV Companion Screen

The EasyTV Companion Screen is an Android application that offers accessibility features to its users when used alongside a terminal HbbTV application. Some of its features are:

- Video synchronization between the HbbTV terminal application and the EasyTV HbbTV Companion Screen application.
- Navigation of the terminal HbbTV application from the Companion Screen application.
- Voice commands, using speech recognition technologies.
- Audio assistance, using Text-To-Speech technologies.
- Image enhancement and zoom

The Companion Screen application, though, will not work with any terminal HbbTV application directly, because the two applications need to communicate using the same message protocol. The purpose of this component of the SDK is to assist developers in making their terminal HbbTV application compatible with the EasyTV Companion Screen application.

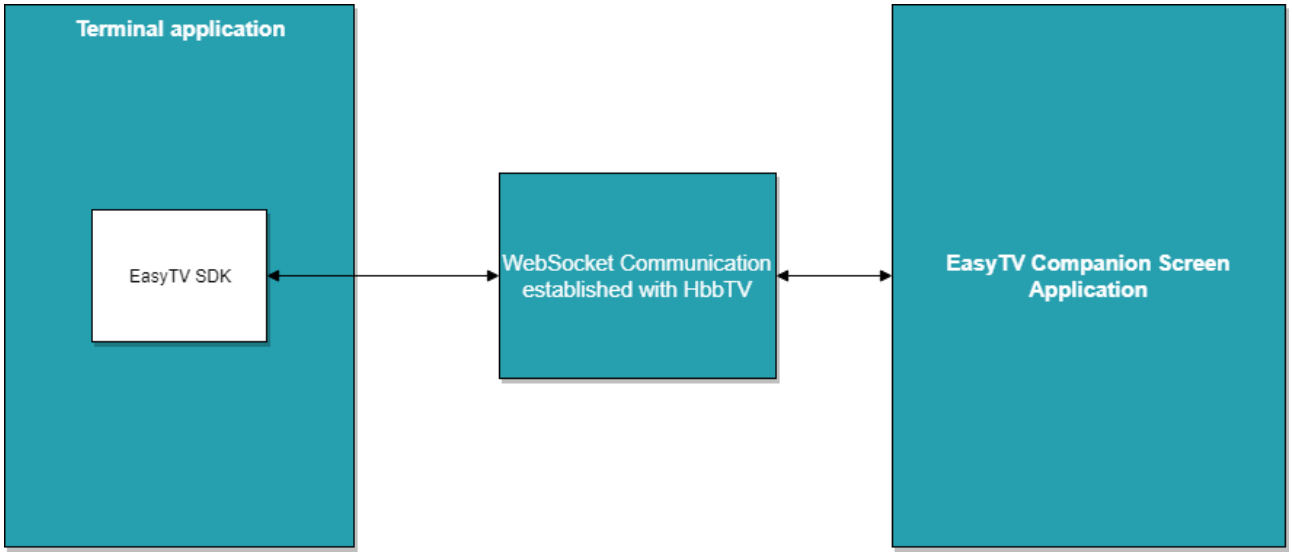

Figure 1 : Interoperability between the EasyTV Companion Screen application and a terminal HbbTV application

3.2.1. HbbTV Terminal library setup

This component is a Javascript library that can be loaded in the HbbTV application. It has a simple API for the necessary actions that the HbbTV application needs to do in order to be compatible with the EasyTV Companion Screen application.

The developer can install this library by downloading the library file and including it into their HTML page.

After loading the module, the API is accessible through an object derived from the *EasyTVHbbTVTerminal* class. Creating the object for this library can be done as follows:

let easytv_terminal = new EasyTVHbbTVTerminal(debug_enabled, sync_enabled);

The first parameter specifies if the library should be run in debug mode. When in debug mode, the library will print debugging information to the console. The second parameter specifies if the object should enable the video synchronization feature.

After setting up the library, the final step is to start listening for Companion Screen applications. This is done using the *startListening* method:

easytv_terminal.startListening()

After this point, the library will handle the connections to the Companion Screen applications automatically.

3.2.2. Navigation

With this feature the user can click on buttons in the HbbTV Companion Screen application and navigate the HbbTV terminal application. Using the library, the developer doesn't need to do any additional actions other than setting up the library. The navigation commands sent from the

Companion Screen application are received from the library and get converted to key presses.

- When the “navigate left” button is pressed on the Companion Screen application, the library emits a left arrow key press. Event with key code 37.
- When the “navigate right” button is pressed on the Companion Screen application, the library emits a right arrow key press. Event with key code 39.
- When the “navigate down” button is pressed on the Companion Screen application, the library emits a down arrow key press. Even with key code 40.
- When the “navigate up” button is pressed on the Companion Screen application, the library emits an up arrow key press. Event with key code 38
- When the “navigate enter/ok” button is pressed on the Companion Screen application, the library emits an enter key press. Event with key code 13.

If the terminal HbbTV application can be navigated by using the key presses, described above, then navigation by using the Companion Screen application will work automatically.

3.2.3. **Events**

The library uses an event driven system in order to get information needed from the terminal HbbTV application or to notify it for some actions that occured in the Companion Screen application. An event handler can be registered using the *event* method as in the following example:

```
easytv_terminal.event(“eventName”, function() {

});
```

The first parameter is the name of the event, that should be handled. The second parameter is the function, that will be called when the event occurs. Whether the callback function has any argument or not depends on the specific event that is handled.

3.2.4. **Video controls**

The Companion Screen application, has controls that are used to control the video playing on the terminal HbbTV application. This library supports this functionality, when using the *dash.js [2]* library for video playback. To enable this functionality on the terminal HbbTV application, the developers need to provide a reference to the *dash.js [2]* object that is used to play the video stream. This should be done as follows:

```
easytv_terminal.attachPlayer(player);
```

After that, the library will handle resuming, pausing, changing the position and changing the volume from the Companion Screen user interface. In case the terminal HbbTV application wants to act when some of these actions occur and execute custom code, the following events can be used.

3.2.4.1. onStreamPlaying event

This event is emitted when the stream has been resumed by the HbbTV Companion Screen application. The callback function for this event doesn’t have any arguments.

3.2.4.2. onStreamPaused event

This event is emitted when the stream has been paused by the HbbTV Companion Screen

application. The callback function for this event doesn't have any arguments.

3.2.4.3. onVolumeChanged event

This event is emitted when the volume has been changed by the HbbTV Companion Screen application. The callback function for this event doesn't have any arguments.

3.2.4.4. onStreamClose event

This event is emitted when the video stream has been requested to close by the HbbTV Companion Screen application. The callback function for this event doesn't have any arguments. The library doesn't actually close the video stream, so by listening to this event the terminal application should have custom code for handling the request by the Companion Screen application to close the stream.

3.2.5. **Information extraction**

The HbbTV Companion Screen application provides some accessibility features, that require some information from the terminal HbbTV application about its current status. This information is passed to the library through events as described in the following sections:

3.2.5.1. onGetVodTitle event

This event is emitted when the HbbTV Companion Screen application requests the title of the currently selected VOD item. The event callback has one argument, which is the locale in which the title should be. The title will be returned as the output of the callback function.

3.2.5.2. onGetVodDescription event

This event is emitted when the HbbTV Companion Screen application requests the description of the currently selected VOD item. The event callback has one argument which is the locale in which the description should be. The description will be returned as the output of the callback function.

3.2.5.3. onGetVodActors event

This event is emitted when the HbbTV Companion Screen application requests the actors of the currently selected VOD item. The event callback has one argument, which is the locale in which the actors should be returned. The actors will be returned in the callback function, the type of output should be a "string" with all the actors name separated by a comma.

3.2.5.4. onGetVodDuration event

This event is emitted when the HbbTV Companion Screen application requests the duration of the currently selected VOD item. The event callback is the locale in which the duration text should be returned. The duration should be returned as a string in natural language. For example a valid duration value in English is "1 hour, 50 minutes".

3.2.6. **Video Synchronization**

The video synchronization feature is about both the terminal HbbTV application and the Companion Screen application playing the same video at the same time. When the playing position is changed in one application, it needs to be changed in the other as well. Also, resuming and pausing must be synchronized. The library offers this functionality only when the dash.js [2] library is used. Also, as stated in subsection 3.2.4 the developers need to pass a reference of the player

object to the library in order to be able to complete the necessary actions.

3.2.6.1. Starting and stopping the synchronized stream.

Initially when the stream is setup to play at the terminal HbbTV application, the terminal application also needs to notify the Companion Screen application that it has started. This can be done with the following method.

```
easytv_terminal.startStream(stream_url, stream_title);
```

The first parameter “stream_url” is the URL of the dash stream, that should be played, whereas the second parameter “stream_title” is the stream’s title.

Having started the stream, the terminal HbbTV application, only, needs to do two things. The first is to notify the library when the player is buffering and the second is to notify the library when the player is ready to play. The actual synchronization logic is entirely handled by the library.

So, for example, in order to notify the library that the player is buffering, the following code sample can be used. In this example the “BUFFER_EMPTY” event of the dash.js [2] library is used to determine whether the player is buffering.

```
player.on(dashjs.MediaPlayer.events["BUFFER_EMPTY"], function () {
    easytv_terminal.onPlayerBufferEmpty();
});
```

And to notify when the player is ready to play, the following example can be used. In this example the “CAN_PLAY” event of the dash.js library is used.

```
player.on(dashjs.MediaPlayer.events["CAN_PLAY"], function () {
    easytv_terminal.onPlayerCanPlay();
});
```

The reason why the library doesn’t register these dash.js events directly is to give the developers more flexibility and allow them to define their own event handler for the dash.js [2] events in the case they need to do more actions. The non-usage of the dash.js events directly, is guaranteed by the library as a whole and not just for the synchronization feature.

When the stream must close from the terminal application’s user interface or if it has been completed on its own the library must be notified in order to do the necessary actions on the Companion Screen application. To do this, the *onPlayerEnded* method must be called in the same way as in the following example.

```
easytv_terminal.onPlayerEnded();
```

3.2.6.2. Loading icons when buffering

A major part of the user interface of a video player is the loading animation that takes place when the player does not have enough data to show to the user. The player is in a state that waits for more data to be downloaded. When the synchronization of the stream is enabled, there are some extra steps that need to be done for this functionality to work as expected. The reason for this is that there will be times when one of the applications is paused and waits for the other application to be ready to play. If the terminal application depends on the dash.js events only, it will not be able to tell if the loading animation should be shown when the stream is paused, because a paused state in this case can both mean, that it is waiting for the other application to get ready and, also, that the user paused the stream manually. In one case the loading animation should be shown and in the

other case it should not.

To fix this issue, it is recommended to use two events provided by this library for this exact purpose. The event “onStreamLoading” will be emitted when one of the application is loading and the event “onStreamFinishedLoading” will be emitted when both of the applications have finished loading. Both of these events have no parameters and expect no value to be returned.

3.2.6.3. User actions from the terminal HbbTV application

After following the instructions from section 3.2.4, the terminal HbbTV application will be able to handle the video controlling actions from the Companion Screen user interface. If video synchronization is enabled, the terminal application needs to notify the library when the video stream has been paused, resumed or changed its playing position from the terminal’s user interface. This is done using the following methods of the library:

- The “play()” method will notify the Companion Screen application to start playing.
- The “pause()” method will notify the Companion Screen application to pause the video.
- The “updateSeekLocation(position)” method will notify the Companion Screen application to change the playing position of the video. The only parameter of this method is the position to change to in seconds.

3.3. SDK with accessibility feature for Companion Screen applications

3.3.1. SDK for integrating Companion Screen with Terminal HbbTV applications

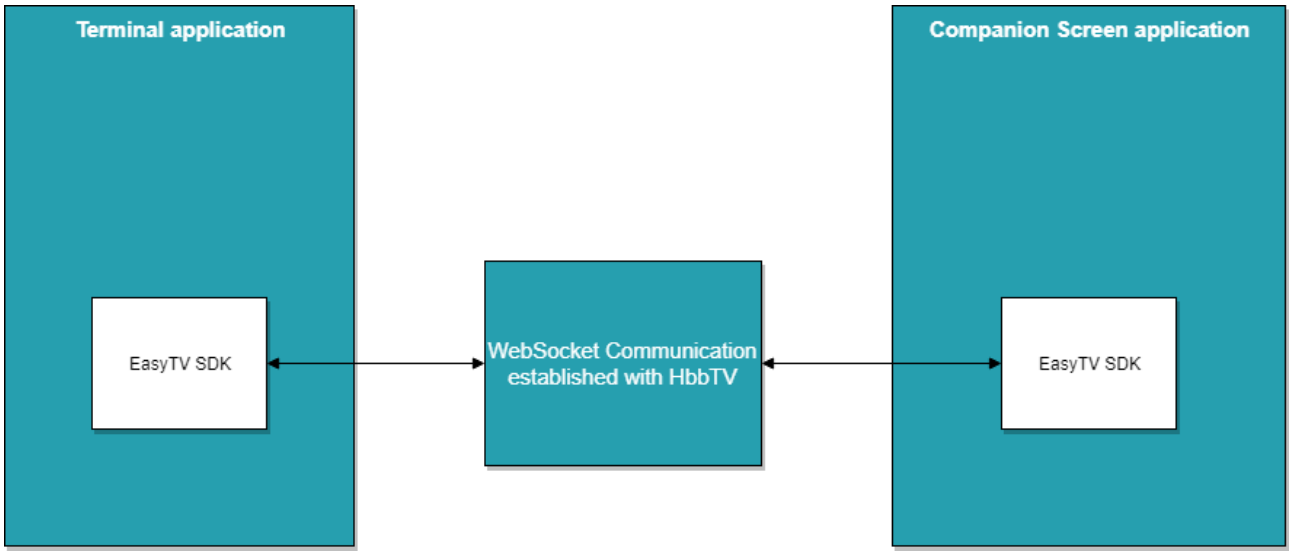

Figure 2 Integration of Terminal and Companion Screen application using the EasyTV SDK

This part of the SDK is a library that assists the integration of HbbTV Companion Screen applications with HbbTV Terminal applications. It is written in javascript and for applications developed with the Apache Cordova framework [3]. A similar library exists in the SDK for the HbbTV terminal applications that is described in section 3.2. These two libraries are designed to have perfect compatibility with each other. So, in order for the following features to work, the HbbTV terminal application must either use the EasyTV SDK as described in section 3.2 or implement the HbbTV websocket communication in the exact same way.

The features that are offered are the following:

- Navigating the HbbTV terminal application from the companion screen application.
- Extracting information from the HbbTV terminal application to be used for any purpose in the companion screen application.
- Video synchronization of the content being played in the terminal application with the Companion Screen application.

This library has 3 dependencies, that must be present in order for it to be able to be used. The first is, as already described, the Apache Cordova framework [3]. The second dependency is the "cordova-plugin-hbbtv" [4] plugin for Apache Cordova, that is necessary, in order to establish a websocket connection between the two applications (Companion Screen and terminal). The third dependency is the dash.js [2] library, that is used when the synchronization of the video between the two applications is enabled.

To start using the library it must first be included into the html page as follows:

```
<script type="text/javascript" src="js/easyTVHbbTVCS.js"></script>
```

After that, the developer must create a new instance of the "EasyTVHbbTVCS" class after the "deviceready" event has been fired from the Cordova framework. It is important to wait for the "deviceready" event, otherwise some of the necessary objects may not exist. The first parameter of the construtor of the "EasyTVHbbTVCS" class declares if debugging messages should be printed to the console by the library.

```
document.addEventListener('deviceready', function(){
      var easyTVCS = new EasyTVHbbTVCS(true);
}, false);
```

In the next sections, it will be described how each feature of the library can be used. For the purposes of this document, it will be assumed that the "easyTVCS" object has been already created and is available to the rest of the application.

3.3.1.1. Connecting to a terminal

The API to search, connect and disconnect to/from a terminal is very simple and straightforward.

To search for available terminal devices in the network the "searchForDevices()" method should be used. The method takes no parameters and returns a promise. When the promise is resolved, an array of terminal objects is returned to the calling code.

```
easyTVCS.searchForDevices().then(function(terminals) {
      // the "terminals" argument is an array of terminal objects
      // to get the name of the terminal use the 'friendly_name' attribute
      // eg terminal.friendly_name
});
```

After showing the terminals to the user and after they select one, the "connect(terminal)" method should be called. The only parameter that it takes is the terminal object. This is the same object that was contained in the terminals array.

```
easyTVCS.connect(terminal)
```

In a similar way, in order to disconnect from a terminal the developer should call the "disconnect()" method. The terminal object doesn't need to get passed to the method.

```
easyTVCS.disconnect()
```

3.3.1.2. Navigation

An HbbTV Terminal application runs on a TV device and the user can navigate it using the arrow buttons on the remote control. This library offers a way to simulate this action from a Companion Screen application. Of course, the library has to be first connected to a terminal application. After that, the developer can then call the following methods that correspond to navigation actions.

- "easyTVCS.navigateLeft(count)", send a navigate left command. Has an optional parameter about the number of times this command should be executed. Default is 1.
- "easyTVCS.navigateRight(count)", send a navigate right command. Has an optional parameter about the number of times this command should be executed. Default is 1.
- "easyTVCS.navigateUp(count)", send a navigate up command. Has an optional parameter about the number of times this command should be executed. Default is 1.
- "easyTVCS.navigateDown(count)", send a navigate down command. Has an optional parameter about the number of times this command should be executed. Default is 1.
- "easyTVCS.navigateEnter()", send a navigate enter command. Expects no parameters.

3.3.1.3. Extracting information

There are cases where the Companion Screen application will need some information about what is shown on the screen. For example, in the case of the EasyTV companion screen application, it requests information and reads them to the user using Text-To-Speech technologies.

At this version of the library, the information that can be extracted is about the currently selected VOD item on the HbbTV terminal screen, specifically its title, description, the actors and the duration.

The API of the library follows the same pattern for all the information. The method that starts the request expects a parameter, which is the language in which the information should be returned. It returns a promise, which when resolved will return the requested information. The following code examples are given.

```
easyTVCS.getSelectedTitle("en-GB").then(function(title) {
      // "title" parameter is of type string.
});
easyTVCS.getSelectedActors("en-GB").then(function(actors) {
      // "actors" parameter is a string that contains the actors in a comma seperated
format.
});
easyTVCS.getSelectedDescription("en-GB").then(function(description) {
      // "description" parameter is of type string.
});
easyTVCS.getSelectedDuration("en-GB").then(function(duration) {
      // "duration" parameter is of type string.
});
```

3.3.1.4. Video Synchronization

The video synchronization feature is initiated by the Terminal application when the user selects a stream to play. To enable the feature in the Companion Screen, the "setupSync()" method must be called once. It takes two parameters, the first being a callback function that will be executed when the playback of the stream has been started by the Terminal, second being, also, a callback

function that will be executed when the playback stops from the Terminal.

```
easyTVCS.setupSync(function() {
      // Playback started from the Terminal
}, function() {
      // Playback stopped from the Terminal
});
```

After receiving the event that the playback of the video has started, the Companion Screen needs to provide the library with an html video element to show the video. This is done as follows:

```
easyTVCS.sync.startSyncing(document.getElementById("stream-element"))
```

In the example above, "stream-element" is the id of the video HTML element. After that, the stream will start and the library will handle all the synchronization logic. In order to control the playing video, the following methods exist in the library:

To resume the playback of the video the "play()" method should be called on both applications.

```
easyTVCS.sync.play();
```

To pause the playback of the video the "pause()" method should be called on both applications.

```
easyTVCS.sync.pause();
```

To change the position of the video the "seek()" method should be called on both applications. It takes a parameter with the position in seconds.

```
easyTVCS.sync.seek(156); // Moves the video to the 156th second
```

To close the video the "close()" method should be called on both applications.

```
easyTVCS.sync.close();
```

To mute or unmute the volume on the HbbTV terminal application, the "muteTV()" method should be called. It takes one parameter of boolean type that specifies whether the volume should be muted or unmuted.

```
easyTVCS.sync.muteTV(true); // mutes the volume
easyTVCS.sync.muteTV(false); // unmutes the volume
```

To change the volume of the HbbTV terminal application, the "setTVVolume()" method should be called. It takes one parameter of type float that ranges from 0.0 to 1.0.

```
easyTVCS.sync.setTVVolume(0.5); // Set volume at 50%
easyTVCS.sync.setTVVolume(1.0); // Set volume at 100%
```

To get the duration of the video, the "getDuration()" method should be called. It return the duration in number of seconds.

```
easyTVCS.sync.setDuration();
```

To get the position of the player, the "getPlayerTime()" method should be called. It returns the position in number of seconds.

```
easyTVCS.sync.getPlayerTime();
```

To get the list of video qualities that the video stream supports, the "getVideoTracks()" method should be used. It returns an array with all the video qualities as javascript objects, the first quality always being the automatic adaption quality.

```
qualities = easyTVCS.sync.getVideoTracks();
qualities.forEach(function(quality){
      quality.label; // a descriptive name
      quality.id; // an id used to change the quality
      quality.selected; // a boolean flag that specifies if it is currently selected
});
```

To change the video quality, the “setVideoTrack()” method should be used. The method expects a parameter to be the quality id.

```
easyTVCS.sync.setVideoTrack(quality_id);
```

To get the list of the audio tracks that the video stream supports, the “getAudioTracks()” method should be used. It returns an array with all the audio tracks as javascript objects.

```
tracks = easyTVCS.sync.getAudioTracks();
tracks.forEach(function(track) {
      track.label; // a descriptive name
      track.id; // an id used to change the track
      track.selected; // a boolean flag that specifies if it is currently selected
});
```

To change the audio track, the “setAudioTrack()” method should be used. The method expects a parameter to be the quality id.

```
easyTVCS.sync.setAudioTrack(audio_track_id);
```

To get the list of the text tracks that the video stream supports, the “getTextTracks()” method should be used. It returns an array with all the text tracks as javascript objects.

```
tracks = easyTVCS.sync.getTextTracks();
tracks.forEach(function(track) {
      track.label; // a descriptive name
      track.id; // an id used to change the track
      track.selected; // a boolean flag that specifies if it is currenlty selected
});
```

To change the text track the “setTextTrack()” method should be used. It expects the track id as a parameter.

```
easyTVCS.sync.setTextTrack(text_track_id);
```

3.3.2. **EasyTV accessible buttons**

This is a javascript library that adds accessibility features to all the buttons in a Companion Screen application. There are two accessibility features offered. The first is that it makes the device vibrate when the user touches a button, giving a blind or visually impaired user a physical feedback that they indeed touched a button. The second accessibility feature is audio assistance using Text-To-Speech technologies. What this means is that, the first time, a user touches a button, but instead of executing the button’s action, the application will read out loud a description of its action. On the first touch the button’s action will not be performed at all. Afterwards, when the user touches again the same button, its action will be performed and the description will not be read again. If the user touches another button, this process will start again.

The requirement for using this library is that Apache Cordova must be also used to develop the

application. The reason for this is because there are two Cordova plugins that are needed to implement this functionality: the first plugin is “cordova-plugin-vibration” [5], that allows the library to control the device’s vibration and the second is “cordova-plugin-tts” [6], that offers Text-To-Speech features.

There are three steps to integrate this library into a Companion Screen application.

The first step is to download and include the library in the html document.

```
<script type="text/javascript" src="easyTVAccessibleButtons.js"></script>
```

The second step is to mark the button that should be affected by the library and provide a text description of its actions. This is done by adding the “acc_button” class to the html element and, also, by providing a description in the “alt” attribute of the html element. For example, the following html code will read out lout “Pause the video” the first time the button is pressed.

<button class=”acc_button” alt=”Pause the video”>Pause</button>

The third and final step is optional. It is about configuring the behaviour of the library by modifying the object *easyTVAccessibleButtons.options.* This object has the following properties:

- “easyTVAccessibleButtons.options.vibrationEnabled”: controls whether the vibration should be enabled or not. The default value is set to “true”.
- “easyTVAccessibleButtons.options.vibrationTime”: is the duration of the vibration in milliseconds. The default value is set to “50” milliseconds
- “easyTVAccessibleButtons.options.ttsEnabled”: controls whether the audio assistance using Text-To-Speech technologies should be enabled or not. The default value is set to “true”.
- “easyTVAccessibleButtons.options.lang”: this is the language in which the Text-To-Speech technology will try to read the buttons description.

One important technical detail is that the library adds an event listener for the click event of the html document when it is loaded. This event listener must be able to cancel the other event listeners for the case of the audio assistance feature. So it needs to always be executed first before all the others listeners in javascript, that means the library’s event listener must be registered first.

3.3.3. **Accessibility Services SDK**

This part of the SDK is developed in angularJS and it contains an angular factory to integrate the accessibility services. The services included in this factory are:

- **MPD Analysis**: the service analyzes the mpd looking for the urls of the Json files which contains the information of face detection, text detection, sound detection and character recognition services in order to parse their information and make the services available.
- **Face magnification**: This service uses the information retrieved on the face magnification json from the mid to magnify the detected faces on the screen.
- **Custom Video Magnification**: this service allow the user to custom magnify a scene on the screen by pinching on the device
- **Character Recognition**:This service uses the information retrieved from the character recognition json (gathered from the mpd file) to present on the screen and read out loud the character presented on the requested scene.

- **Text Detection**: This service uses the information retrieved from the text detection json (gathered from the mpd file) to present the recognised text on the screen. The presentation of the text is customizable (font, color, background).
- **Sound Detection**: This service uses the information retrieved from the sound detection json (gathered from the mpd file) to present the recognised sound on the screen. The presentation of the text is customizable (font, color, background).
- **Subtitle customization**: This service is to customize the font size, color and background color of the subtitles.

3.3.3.1. Dependencies

The sdk needs angularJS version >=1.6.* and jQuery version >= v1.11.*. The player used on this app is Dash JS version 2.8.0 (https://github.com/Dash-Industry-Forum/dash.js?) and it is a key dependency.

3.3.3.2. Integration

The first step is to include the SDK javascript on an Angular application by inserting the script tag on the html.

```
<script type="text/javascript" src="easytv.sdk.min.js"></script>
```

Once the javascript is loaded, developers can include the accessibility services factory on an angular controller.

```
angular.module("app")
      .controller("playerController",["AccesibilityService"],
function(AccesibilityService){
            //Use the service
      })
```

When the factory is injected on the controller it can be initialized with:

AccesibilityService.init(mpd_url, videoID, dashPlayer, options);

The parameters are the following:

- mpd_url: Url of the mpd file, where it contents the accessibility tag services
- VideoID: the id of the html5 video tag.
- dashPlayer: the dash Js player object.
- options: is a JavaScript objet containing the preferences of the accessibility services described below.

Options		
soundDetection	True or False	Enable or disable the sound detection service

Options		
textDetection	True or False	Enable or disable the text detection service
readAloud	True or False	Enable or disable the read aloud option for text detection or character recognition
enhancement.enabled	True or False	Enable or disable the Video enhacment option
enhancement.type f	face or magnification	Describe the type of enhancement to be done, face magnification or custom magnification
enhancement.magnification.scale	[1.5 , 3.5]	scale of the magnification to be done on custom magnification
subtitles.audio.track	en, it, es, cargo	Language in which the audio subtitle should be read
subtitles.audio.enabled	True or False	Enable or disabled the audio subtitles service
subtitles.size	[10,100]	font size of subtitles, text and sound detection captions
subtitles.background	Hexadecimal color	background color of subtitles, text and sound detection captions
subtitles.color	Hexadecimal color	font color of subtitles, text and sound detection captions
equalization	Equalization Object	JS Object with the equalization parameters, explained in 3.3.3.3

It is not necessary to initialize the service again to change these values on the fly while watching content, there is a function to update the options called “updateOptions”. Combined with the angular $watch function, below there is an example of how to enable, disable, change options on the fly on an app.

```
$scope.$watch("options", function (newValue) {
                        AccesibilityService.updateOptions(newValue);
                }, true);
```

Most of the services are automatically enabled or disabled just by changing this options, and updated them by calling updateOptions, but the character recognition needs an active action from the user to be triggered, the function checkCharacters needs to be called as shown here:

```
AccesibilityService.checkCharacters( )
```

3.3.3.3. Equalization Services SDK

The equalization service is an accessibility service that has its own factory, so can be used separately from the other accessibility services .This service equalize the audio of the content with the users preferences to fit their needs. This service creates a tool to set up the audio equalization

parameters, and the changes of this parameters will be reflected on the html5 sample audio and optionally in a html5 target video.

Developers can include the equalization services factory on an angular controller.

```
angular.module("app")
      .controller(“playerController”,[“EqualizationService”],
function(EqualizationService){
            //Use the service
      })
```

When the factory is injected on the controller it can be initialized with:

EqualizationService.init(playerId, equalizationObject, videoId)

The parameters are the following:

- playerId: the id of the html5 audio tag, where the sample audio is playing,
- videoId:(optionally) the id of the html5 video tag, target of the equalization.
- equalizationObject: is a JavaScript objet containing the preferences of the accessibility services .

equalizationObject		
lowShelf		
frequency	[35, 220]	Equalization Low Shelf frequency value
gain	[-50, 50]	Equalization Low Shelf gain value
lowPass		
frequency	[80, 1600]	Equalization Low Pass frequency value
qFactor	[0.7, 12]	Equalization Low Pass qFactor value
highPass		
frequency	[800, 5900]	Equalization High Pass frequency value
qFactor	[0.7, 12]	Equalization High Pass qFactor value
highShelf		

equalizationObject		
frequency	[4700, 2200]	Equalization High self frequency value
gain	[-50, 50]	Equalization High self gain value

Example of an equalization Object:

```
{
      highShelf:{frequency:4700,gain:0},
      lowShelf:{frequency:35,gain:0},
      highPass:{frequency:800,qFactor:0.7},
      highShelf:{frequency:880,qFactor:0.7},
}
```

As in the main accessibility service there is a function to update the parameters of the equalization without initialising the object again. Normally the accessibility services contains the equalization object, so the example below shows how to handle them when are included there, but the service is prepared to independent equalization parameters.

```
$scope.$watch("options", function (newValue) {
      EqualizationService.updateParameters(newValue.equalization);

                  }, true);
```

3.4. Speech SDK, DialogManager plugin

3.4.1. Introduction

The **dialog manager plugin** is the Cordova plugin developed to manage text to speech engine, speech recognition engine and speech interpretation. When Cordova ‘deviceready’ event is fired a dialogmanagerplugin object has to be declared:

```
dlgmanager = new cordova.plugins.dialogmanagerplugin();
```

Then InititDialogManager function must be called to initialize ASR and TTS engines with the language desired:

```
dlgmanager.initDialogManager(‘en-Gb’);
```

Now everything is ready to manage the dialog flow and to receive events from the EasyTV Speech Platform.

A list of all the events with a detailed description for each one is reported later in this document.

3.4.2. Methods

3.4.2.1. InitDialogManager

initDialogManager(String language): initialize text to speech engine and speech recognition engine. At the end of init process a *onPlatformInitialized* event is fired.

- *language:* parameter is used to set voice and recognition engine.

3.4.2.2. start

start(): start speech recognition engine. If recognition has been performed, an *onResult* event is fired.

3.4.2.3. stop

stop (): stop speech recognition engine.

3.4.2.4. speech

speech (String text, boolean queue): send text to TTS engine.

- *text:* string value
- *queue:* boolean value. If true the text is queued; false to stop TTS engine and to start new text immediately.

3.4.2.5. speechOut

speechOut (boolean queue): send one of the text in tts_out list (choosen randomly) to TTS engine

- *queue:* boolean value. If true the text is queued; false to stop TTS engine and to start new text immediately.

3.4.2.6. speechIn

speechIn (boolean queue): send one of the text in tts_in list (choosen randomly) to TTS engine

- *queue:* boolean value. If true the text is queued; false to stop TTS engine and to start new text immediately.

3.4.2.7. createVoiceStates

createVoiceStates (VoiceState[] states, Callback success, Callback error): create a list of voice states in dialogmanager. Each voice state in the list can be called using *setVoiceStateById* function described above.

- *states:* An array of voice state objects to be set.
- *success:* callback function called if the creation of the voice states process has been

completed successfully

- *error:* callback function called if creating the voice states is interrupted by an error

3.4.2.8. addVoiceState

addVoiceState (VoiceState voiceState, Callback success, Callback error): add a voice state to voice state list. See createVoiceStates function.

- *voiceState:* VoiceState value. Is a voice state object to be added.
- *success:* callback function called if the addition of a voice state process has been ended successfully
- *error:* callback function called if the addition of a voice state process has been interrupted by an error

3.4.2.9. setVoiceState

setVoiceState (VoiceState voiceState, boolean speechIn, boolen queue, Callback success, Callback error): set a voice state.

- *voiceState:* VoiceState value. Is a voice state object to be set.
- *speechIn:* boolean value. True if one of the speechIn text is chosen (randomly) and is sent to TTS engine when voiceState is activated.
- *queue:* boolean value. True if speechIn text has to be queued to current text used by TTS engine
- *success:* callback function called if setting voice state process has ended successfully
- *error:* callback function called if setting voice state process has been interrupted by an error

3.4.2.10. setVoiceStateById

setVoiceStateById (String voiceStateId, boolean speechIn, boolen queue, Callback success, Callback error): set a voice state.

- *voiceStateId:* String value. Is the id of the state to be set from the voice states list, see createVoiceStates function.
- *speechIn:* boolean value. True if one of the speechIn text is chosen (randomly) and is sent to TTS engine when voiceState is activated.
- *queue:* boolean value. True if speechIn text has to be queued to current text used by TTS engine
- *success:* callback function called if setting voice state process is ended successfully
- *error:* callback function called if setting voice state process is interrupted by an error

3.4.3. VoiceState class

The **VoiceState** class represents a single state in the flow of a state machine which represents the dialog between the user and the app. A list of VoiceStates objects can be prepared and can be sent to voicedialogmanager using function createVoiceStates (see above voicedialogmanager specs). Then during voice interaction between user and app, depending on context, a state of the

list can be set using setVoiceStateById (see above voicedialogmanager specs). When a state is set a tts_in can be played by TTS engine. When a state is changed and a new state is set a tts_out can be played by TTS engine. Tts_in and tts_out are chosen randomly in the corresponding list.

3.4.3.1. Constructor

constructor(String stateId, NLP[] resultManagers, String[] tts_in, String[] tts_out)

- *stateId:* String value. The voice state id
- *resultManagers:* List value. A list of NLP classes used to get semantic value from recognized text using NLP module
- *tts_in:* List value. A list of texts. When a voice state is activated one of the texts in the list (chosen randomly) is used by TTS engine
- *tts_out:* List value. A list of texts. When a new voice state is activated one of the texts in the list (chosen randomly) can be used calling by dialogmanagerplugin using “speechOut” function.

3.4.3.2. setTts_in

setTts_in(String[] tts_in): set a list of tts_in text. This list replace the list of text set in constructor.

- *tts_in:* List value. A list of texts. When a voice state is activated one of the texts in the list (chosen randomly) is used by TTS engine

3.4.3.3. setTts_out

setTts_out(String[] tts_out): set a list of tts_out text. This list replace the list of text set in constructor.

- *tts_out:* List value. A list of texts. When a voice state is activated one of the texts in the list (chosen randomly) is used by TTS engine

3.4.3.4. addResultManager

addResultManager(NLP_recognition): add a NLP class to list of NLP classes passed in the constructor

- *recognition:* a NLP class

3.4.3.5. addTTSInRange

addTTSInRange(String[] tts_list): add a list of texts to tts_in list.

- *tts_list:* List value. A list of texts to add to tts_in list.

3.4.3.6. AddTTSOutRange

addTTSOutRange(String[] tts_list): add a list of texts to tts_out list.

- *tts_list:* List value. A list of texts to add to tts_out list.

3.4.4. NLP Classes

NLP classes are the classes used by dialogmanager to get a semantic from text recognized by

ASR engine. In onResult event of dialog manager a list of results is returned. Each result contains semantic and confidence returned by NLP classes elaboration computation. The NLP classes used are those in the resultMangers list passed in the constructor of VoiceState object. Each class has a priority, the lower this value is, the earlier the class will be used by the dialog manager during interpretation phase. In this way when a result is matched the utterance interpretation is ended and a result is returned to app by onResult event.

3.4.4.1. ExactTextRecognition Class

The **ExactTextRecognition** class executes an exact match between recognized text (utterance) and the items added to the class using the "add" function: if a phrase is matched in the onResult event a result is returned with a confidence value of 1 (100%).

- **constructor()**
- **add(String semantic, String utterance):** add items to list. Utterance is the exact text to be matched with text recognized by ASR engine.
- **setPriority(int priority):** set the priority. The lower this value is, the earlier the class will be used by the dialog manager during interpretation phase.

3.4.4.2. Dication Class

The **Dictation** class is used to return utterance without interpretation.

- **constructor()**
- **setPriority(int priority):** set the priority. The lower this value is, the earlier the class will be used by the dialog manager during interpretation phase.

3.4.4.3. Levenshtein Class

The **Levenshtein** class executes a match calculated with the Levenshtein algorithm between the recognized text and the items added to the list using the add function.

- **constructor()**
- **add(String semantic, String utterance)**: add items to the list. Utterance is the string to be matched with the recognized text.
- **setPriority(int priority):** set the priority. The lower this value is, the earlier the class will be used by the dialog manager during interpretation phase.

3.4.4.4. RegularExpression Class

The RegularExpression class executes a match using regular expressions between the recognized text and the items added to the list using the add function.

- **constructor()**
- **add(String semantic, String utterance)**: add items to the list. Utterance is the string holding the regular expression to be matched with the recognized text.
- **setPriority(int priority):** set the priority. The lower this value is, the earlier the class will be used by the dialog manager during interpretation phase.

3.4.5. Events

3.4.5.1. onResult

onResult (Event event): fired by ASR engine when a speech is recognized and transformed in a string.

- *event:* is an object that contains a list of results. Each result has three elements: semantic, utterance and confidence. As you can see in demo application each result can be accessed in onResult event in this way:

```
if (event.results.length > 0) {
      //first result
      var utterance = event.results[0][0].utterance);
}
```

3.4.5.2. onParialResult

onPartialResult (Event event): fired by ASR engine when partial speech is recognized and transformed in a string.

- *event:* is an object that contains a list of results. Each result has three elements: semantic, utterance and confidence. As you can see in demo application each result can be accessed in onPartialResult event in this way:

```
if (event.results.length > 0) {
      //first result
      var partial_utterance= event.results[0][0].utterance);
}
```

3.4.5.3. onTTSStart

onTTSStart (String tts): fired by TTS engine at the start of speech synthesis.

- *tts:* string value. Text that is reproduced by TTS engine.

3.4.5.4. OnPlatformInitialized

onPlatformInitialized (): this event is fired after a call to initDialogManager function. If all went right, ASR engine, TTS engine and dialog manager are initialized and is possible to start speech flow.

3.4.5.5. onASREndOfSpeech

onASREndOfSpeech: fired when end of speech is detected by ASR engine.

3.4.5.6. onASRError

onASRError: fired when an error occurred during speech recognition phase. In this version of the platform no other information are sent.

3.5. Service Manager

The Service Manager is the gateway of the EasyTV platform and is the way that the Content

Owner will access and use any of the services that are offered by EasyTV. Its purpose is not simply to redirect actions to the others services, but, also, to handle jobs, tasks and the files that the Content Owner wants to pass to the service they want to use.

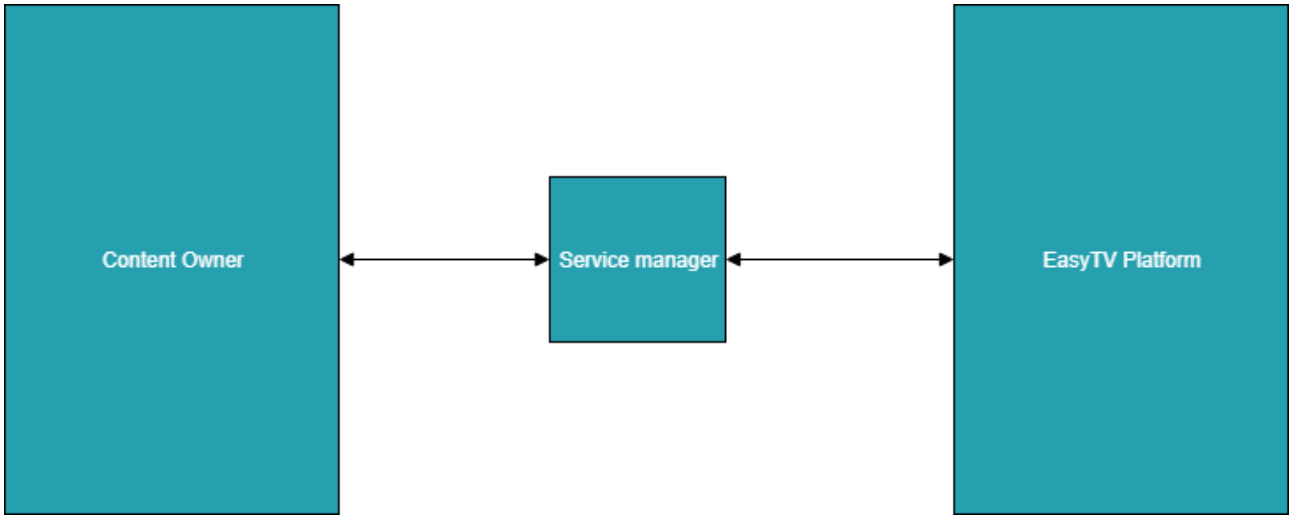

Figure 3 The Service Manager as a gateway to the EasyTV platform

From the above, it is clear that in order for new services to be integrated into the EasyTV platform, they need to be able to communicate with the Service Manager. This section will provide information about the components of the EasyTV SDK regarding the cooperation with the Service Manager.

3.5.1. Service Registration Tool

The Service Manager stores detailed information about each EasyTV service in order to know how to communicate with them and what are the tasks that they support. So, it is important to provide a way for other services to register themselves to the Service Manager and declare their capabilities.

The Service Registration Tool is a command line tool that runs on the machine that the service manager is deployed to. With it, the user, which in this case is the administrator that manages the EasyTV platform, can register or unregister services and define new tasks that are supported by the service. When the tasks for a service are defined, the user should, also, specify a description of the inputs and outputs of the task and, also, if this task should forward its results to some other task. In this way, a chain of tasks can be defined that can form more complex jobs. So, as described the Service Manager can handle the services and the tasks in a powerful generic way and it is not tied to implementation details of the services that it manages. So in conclusion this tool provides a way to control the capabilities of the Service Manager.

The Service Registration Tool is composed of multiple sub-commands, each responsible for a specific entity in the Service Manager. This command line tool also offers help pages for all the subcommands in order to assist the user. To show these help pages, the command should be executed with the “-h” flag parameter. In the following sections the output of the help page for each subcommand is shown.

3.5.1.1. Service Registration Tool main help page

```
$ srt -h
NAME:
   srt - Service Registration Tool
```

```
USAGE:
   srt [global options] command [command options] [arguments...]

VERSION:
   1.0.0

COMMANDS:
     admin, a           actions about the admin user
     service, s         actions about the services
     task, t            Actions about the task
     content-owner, co  actions about the content-owner user
     help, h            Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --lang value   language for the greeting (default: "english")
   --help, -h     show help
   --version, -v  print the version
```

3.5.1.2. Service Registration Tool admin help page

```
$ srt admin -h
NAME:
   srt admin - actions about the admin user

USAGE:
   srt admin command [command options] [arguments...]

COMMANDS:
     change-password, cw

OPTIONS:
   --help, -h  show help
```

3.5.1.3. Service Registration Tool service help page

```
$ srt service -h
NAME:
   srt service - actions about the services

USAGE:
   srt service command [command options] [arguments...]

COMMANDS:
     list, ls
     create, c
     update, u
     set-availability, sa
     renew-api-key
```

```
OPTIONS:
   --help, -h  show help
```

3.5.1.4. Service Registration Tool task help page

```
$ srt task -h
NAME:
   srt task - Actions about the task

USAGE:
   srt task command [command options] [arguments...]

COMMANDS:
     create, c             Create a new task
     list, ls              List the tasks of a service
     set-availability, sa  Change the availability of the task
     update, u             Update the task
     remove, rm            Remove the task
     set-vars              Change the input or output of the task

OPTIONS:
   --help, -h  show help
```

3.5.1.5. Service Registration Tool content owner help page

```
$ srt content-owner -h
NAME:
   srt content-owner - actions about the content-owner user

USAGE:
   srt content-owner command [command options] [arguments...]

COMMANDS:
     reset-password, rp
     create, c
     list, ls
     update, u

OPTIONS:
   --help, -h  show help
```

3.5.2. Service Manager SDK

The Service Manager provides two Web API, one for the internal services (Internal API) and one for the Content Owner (public API). In order to ease the development of new services in the EasyTV platform and of the Content Owner, this component offers a javascript library that abstracts the service manager API in an object oriented way. By employing this component the user, which in this case is the developer of the new service, doesn't have to work with HTTP requests and the API directly. Instead, the user can work with objects in the programming language he uses and is

familiar with. Unlike the Service Registration Tool, this component is not mandatory to use and developers can still use the web API directly if they wish so.

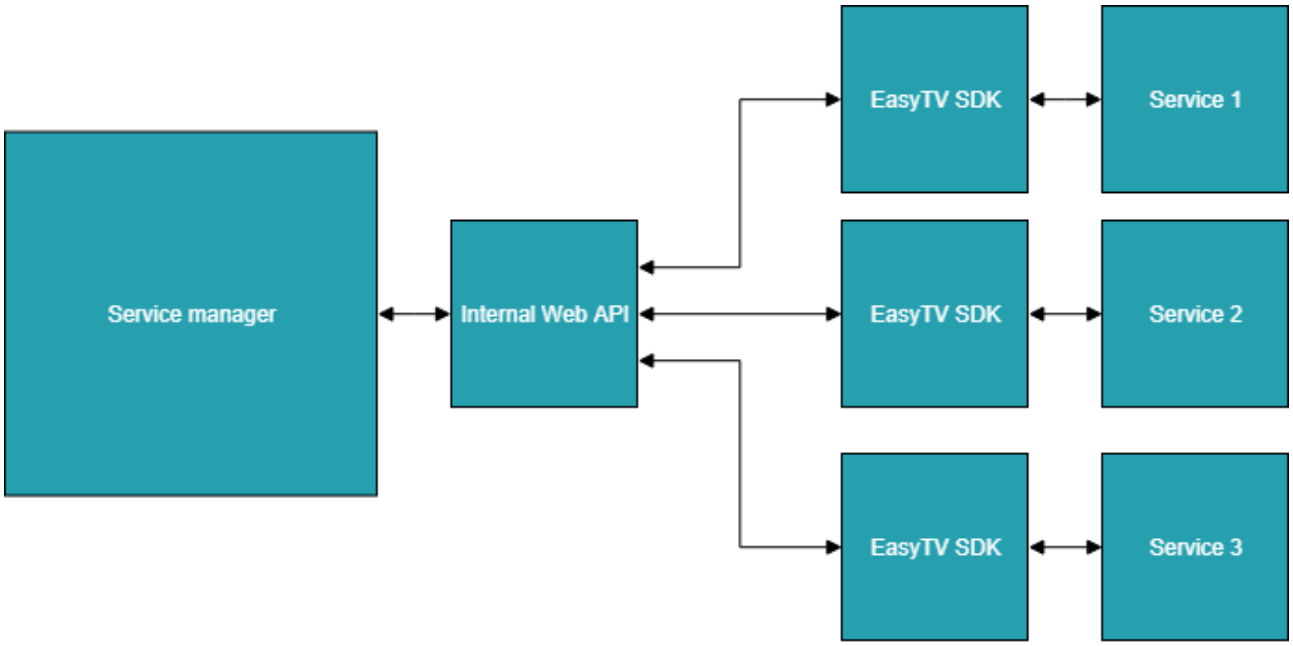

Figure 4 EasyTV SDK of the internal API

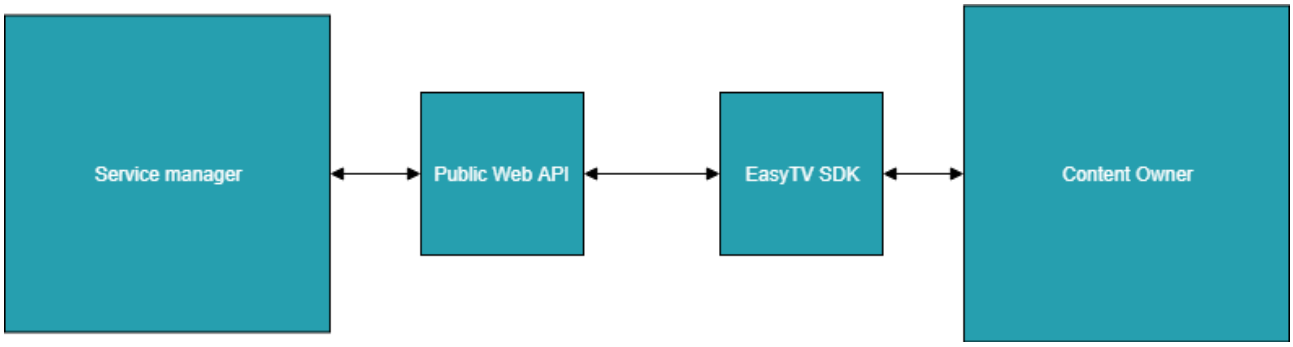

Figure 5 EasyTV SDK for the public API

The first step for a developer to install this library is to add it to their project dependencies using the **npm** package manager. To do this they must retrieve the library source code and then execute the following command on their project directory.

```
npm install --save <library_path>
```

Where <library_path> is the path that the library source code exists in. Afterwards the library can be used with the *require* command.

```
var smsdk = require("easytv-sm-sdk")
```

The above command will return the library object which contains:

- **smsdk.ContentOwnerClient**: This is the client class to be used to access the public service manager API.
- **smsdk.ServiceClient**: This is the client class to be used to access the internal API.
- **smsdk.Code**: This is a collection of error codes that are included in the *ServiceManagerError* exception that is thrown when something goes wrong with the API

call.

3.5.2.1. ContentOwnerClient

The *ContentOwnerClient* is described in the following table. All the methods return their data inside a javascript *Promise* object.

Method	**Description**

constructor(baseUrl = "https://sm-api.easytv.eng.it")	Creates a new instance of ContentOwnerClient class:
	`var sdk = require("easytv-sm-sdk");`
	`var client = new sdk.ContentOwnerClient();`
	
	"baseUrl": Is the base url of the Service Manager endpoint to use. The default "https://sm-api.easytv.eng.it"" is the service manager running on the EasyTV cloud.

login(username, password)	Login with content owner credentials.
	Example:
	`client.login("arx", "arxPwd007")`
	`.then(data => {`
	`...`
	`});`
	
	Returned object on success:
	
	`{`
	`code: 200,`
	`description: "Success, hello USER",`
	`session_token: "<TOKEN>"`
	`}`
	
	Throws **ServiceManagerError**

logout()	Clears the current session
	Example:
	`client.logout()`
	
	Returned object on success:
	`true`

	Throws **NotLoggedInError, ServiceManagerError**

ping()	Checks if the session is ok
	Example:
	`client.ping()`
	Returned object on success:
	`{ code: 200, description: 'Pong' }`
	Throws **NotLoggedInError, ServiceManagerError**

changePassword(oldPassword, newPassword, newPasswordVerification)	Changes the users password
	Example:
	`client.changePassword("oldPass", "newPass", "newPass")`
	Returned object on success:
	`{ code: 200, description: 'password is changed' }`
	Throws **NotLoggedInError, ServiceManagerError**

getServices()	Retrieves the available services
	Example:
	`client.getServices()`
	Returned object on success:
	`{ code: 200, description: 'Success', services: [ { description: 'ENG test service', enabled: true, id: 3, name: 'ENG Service', tasks: [] },`

	<pre>{
	description: 'Test',
	enabled: true,
	id: 2,
	name: 'Test Service',
	tasks: [Array]
	},
	{
	description: '"Subtitle production"',
	enabled: true,
	id: 1,
	name: '"Subtitle Production Module"',
	tasks: [Array]
	}
	]
	}</pre>
	Throws **NotLoggedInError, ServiceManagerError**

getService(id)	Returns the service with a specific id
	Example:
	<pre>client.getService(123)</pre>
	Returned object on success:
	<pre>{
	code: 200,
	description: 'Success',
	service: {
	description: 'Test',
	enabled: true,
	id: 2,
	name: 'Test Service',
	tasks: []
	}
	}</pre>
	Throws **NotLoggedInError, ServiceManagerError**

postJob(publicationDate, expirationDate, tasks)	Posts a new job with the given tasks *Example:* <pre>client.postJob(1578955296, 1578958296, [</pre>

	<pre>{
	  task_id: 2,
	  input: {
	  text: "Sample Input"
	  }
	}
	])</pre>
	Returned object on success:
	<pre>{ code: 200, description: 'Job created', job_id: 790 }</pre>
	Throws **NotLoggedInError, ServiceManagerError**

cancelJob(jobId)	Cancels a job with a specific id *Example:* <pre>client.cancelJob(562)</pre> *Returned object on success:* <pre>{ code: 200, description: 'Job cancelled'}</pre> Throws **NotLoggedInError, ServiceManagerError**

getJobs(limit= 50, beforeJobId=null)	Get the latest jobs that were posted
	Example:
	<pre>client.getJobs()</pre>
	Returned object on success:
	<pre>{
	  code: 200,
	  description: 'Success',
	  jobs: [
	  {
	  completion_date: 1578950478,
	  creation_date: 1578950478,
	  current_task: null,
	  expiration_date: 1578958296,
	  id: 790,
	  is_canceled: false,
	  is_completed: true,</pre>

<table>
<tr><td></td><td>

```
      output: [Object],
      publication_date: 1578955296,
      status: 'Completed',
      tasks: [Array]
    },
    {
      completion_date: 1578950309,
      creation_date: 1578950309,
      current_task: null,
      expiration_date: 1578958296,
      id: 789,
      is_canceled: false,
      is_completed: true,
      output: [Object],
      publication_date: 1578955296,
      status: 'Completed',
      tasks: [Array]
    }
  ]
}
```

Throws **NotLoggedInError, ServiceManagerError**

</td></tr>
<tr><td>getJob(id)</td><td>

Get the latest jobs that were posted

Example:

```
    client.getJob(789)
```

Returned object on success:

```
{
  code: 200,
  description: 'Success',
  job: {
    completion_date: 1578950309,
    creation_date: 1578950309,
    current_task: null,
    expiration_date: 1578958296,
    id: 789,
    is_canceled: false,
    is_completed: true,
    output: { text: 'SAMPLE INPUT' },
    publication_date: 1578955296,
    status: 'Completed',
    tasks: [ [Object] ]
```

</td></tr>
</table>

	<pre> } }</pre> Throws **NotLoggedInError, ServiceManagerError**

3.5.2.2. ServiceClient

The *ServiceClient* class is described in the following table. All the methods return the data inside a javascript *Promise* object.

Method	**Description**
constructor(apiKey, baseUri = "https://sm-api.easytv.eng.it")	Create new instance for a specific service API key. *Example:* *var client = new sdk.ServiceClient("<API_KEY>");*

registerTask(name, description, startUrl, cancelUrl, input, output, useRestCancel = false)	Cancels a job with a specific id
	Example:
	<pre>client.registerTask(
	    "Name",
	    "Description",
	    "http://start_url",
	    "http://cancel_url",
	    {
	        "arg1": "string",
	        "arg2": "int"
	    },
	    {
	        "out1": "string"
	    }
	)</pre>
	Returned object on success:
	<pre>{
	  code: 200,
	  description: 'The task was registered successfully',
	  task_id: 20
	}</pre>

	Throws **ServiceManagerError**

setTaskAvailability(taskId, availability)	Change the given tasks availability
	Example:
	`client.setTaskAvailability(12, false)`
	Returned object on success:
	`{ code: 200, description: 'success'}`
	Throws **ServiceManagerError**

deleteTask(id)	Deletes the given task
	Example:
	`client.delete(20)`
	Returned object on success:
	`{ code: 200, description: 'success'}`
	Throws **ServiceManagerError**

getTasks()	Get alls the tasts for this service
	Example:
	`client.getTasks()`
	Returned object on success:
	`{ code: 200, description: 'Success', tasks: [ { cancel_url: '<CANCEL_URL>', description: 'Converts the given text to have all upper case letters', enabled: true, id: 2, input: [Object],`

<table>
<tr>
<td></td>
<td>
<pre>
      name: 'Upper Case',
      output: [Object],
      start_url: '<START_URL>'
    },
    {
      cancel_url: '<CANCEL_URL>',
      description: 'Converts the given text to
have all lower case letters',
      enabled: true,
      id: 3,
      input: [Object],
      name: 'Lower Case',
      output: [Object],
      start_url: '<START_URL>'
    },
    {
      cancel_url: '<CANCEL_URL>',
      description: 'This task replaces some
characters with some other characters in the
text',
      enabled: true,
      id: 4,
      input: [Object],
      name: 'Replace',
      output: [Object],
      start_url: '<START_URL>'
    },
    {
      cancel_url: '<CANCEL_URL>',
      description: 'bla',
      enabled: false,
      id: 5,
      input: [Object],
      name: 'blablablabla',
      output: [Object],
      start_url: '<START_URL>'
    }
  ]
}
</pre>

Throws ServiceManagerError
</td>
</tr>
<tr>
<td>getJobs(limit=50, before=null)</td>
<td>Get the latest job posted for this service
Example:</td>
</tr>
</table>

	`client.getJobs()`
	
	Returned object on success:
	`{ code: 200, description: 'success', jobs: [{ completion_date: 1578950478, content_owner: 'ARX', creation_date: 1578950478, expiration_date: 1578958296, id: 791, is_canceled: false, is_completed: true, publication_date: 1578955296, status: 'Completed' }, { completion_date: 1578950309, content_owner: 'ARX', creation_date: 1578950309, expiration_date: 1578958296, id: 790, is_canceled: false, is_completed: true, publication_date: 1578955296, status: 'Completed' }] }`
	
	Throws **ServiceManagerError**

getJob(id)	Gets the job with that specific id
	Example:
	`client.getJob(790)`
	
	Returned object on success:
	`{ code: 200, description: 'success',`

	<pre> job: {
	    completion_date: 1578950309,
	    content_owner: 'ARX',
	    creation_date: 1578950309,
	    expiration_date: 1578958296,
	    id: 790,
	    is_canceled: false,
	    is_completed: true,
	    publication_date: 1578955296,
	    status: 'Completed'
	  }
	}</pre>
	Throws **ServiceManagerError**

setStatus(jobId, status)	Changes the status text of a job *Example:* <pre> client.setStatus(20, “In progress”)</pre> *Returned object on success:* <pre> { code: 200, description: 'success'}</pre> Throws **ServiceManagerError**
cancelJob(id)	Cancel the job with that specific id *Example:* <pre> client.cancelJob(20)</pre> *Returned object on success:* <pre> { code: 200, description: 'success'}</pre> Throws **ServiceManagerError**
getAssets(id)	Gets the assets upload for that specific job *Example:* <pre> client.getAssets(20)</pre>

	Returned object on success: `{ assets: [asset_id: 2, asset_url: "/assets/file1.png"], code: 200, description: 'Success' }` Throws **ServiceManagerError**

uploadAsset(asset, jobId)	Uploads a new asset for this job.
	Example:
	`// Get the "asset" object from html <input>`
	`client.uploadAsset(asset, 20)`
	
	Returned object on success:
	`{ asset_id: 2, asset_url: "/assets/file1.png", code: 200, description: 'Success' }`
	
	Throws **ServiceManagerError**

finishJob(jobId, output)	Finish a job in progress
	Example:
	`client.finish(20, { "out1": "Result", "out2": "Another Result"})`
	
	Returned object on success:
	`{ code: 200, description: 'success'}`
	
	Throws **ServiceManagerError**

3.5.2.3. Codes

The *Codes* object contains the following values.

Name	Value
Success	200

MissingInput	-400
Unauthorized	-401
NotFound	-404
InternalServerError	-500
TaskNotDisabled	-1
TaskHasActiveJobs	-2
JobAlreadyCanceled	-3
JobAlreadyCompleted	-4
EmptyAsset	-5
TaskAlreadyExists	-8
TaskNoInputParameter	-9
TaskNoOutputParameter	-10
InvalidStartUrl	-11
InvalidCancelUrl	-12
InvalidInput	-13
InvalidPublicationdate	-14
InvalidExpirationDate	-15
JobStatusNotUpdatable	-16
ForbiddenAsset	-17
InvalidOutput	-18

NotCompletable	-19
LinkedParameterNotTheSameType	-20
LinkedOutputNotFound	-21
ServiceNameInUse	-22
ContentOwnerNameExists	-23
ContentOwnerUsernameExists	-24
ContentOwnerEmailExists	-25
JobWithDisabledTasks	-26
PasswordIsTooShort	-27
InvalidCredentials	-28
NewPasswordDoesntMatchVerification	-29

4. Conclusion and future work

The architecture of the EasyTV Service Development Kit was described in detail in the present document. It was noted that the EasyTV SDK can have two major use cases. In the first case a third party developer may want to use the features of the EasyTV SDK in order to integrate them in third party products. The second case is about third party developers that want to create a new service and integrate it to the EasyTV platform.

The SDK contains components which deal with the development and integration of HbbTV terminal and Companion Screen applications. On one hand a terminal HbbTV application can use the SDK and enable the interoperability with the EasyTV Companion Sceen application. Some of the features that will be available to the user are video synchronization, image enhancement, navigation, speech recognition and accessibilty options using Text-To-Speech technologies. On the other hand Companion Screen applications can, also, benefit by using the EasyTV SDK. By using the library described in 3.2.1 a Companion Screen application can be made compatible with any HbbTV application, that uses the EasyTV SDK.

Moreover, the SDK contains components related to the Service Manager. The service registration tool will help the platform administrator to register and configure the services of the EasyTV platform. The SDK for the public Service Manager Web API will assist the Content Owner in using the platform in a more automated way. In addition, the SDK for the internal Service Manager Web API can support the development of the services themselves.

5. References

[1] HbbTV [Online]. https://www.HbbTV.org/ (last accessed 24 Oct 2018)

[2] Dash.js Javascript library [Online]. https://github.com/Dash-Industry-Forum/dash.js (last accessed 24 Oct 2018)

[3] Apache Cordova framework [Online]. https://cordova.apache.org/ (last accessed 24 Oct 2018)

[4] HbbTV plugin for Apache Cordova [Online]. https://github.com/fraunhoferfokus/cordova-plugin-HbbTV (last accessed 24 Oct 2018)

[5] Apache Cordova plugin to control device vibration [Online]. https://github.com/apache/cordova-plugin-vibration (last accessed 24 Oct 2018)

[6] Apache Cordova plugin for Text-To-Speech functionality [Online]. https://github.com/vilic/cordova-plugin-tts (last accessed 24 Oct 2018)