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.

D5.4 First 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: 30 months

Document. ref.: D5.4

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:	D5.4
DOCUMENT TITLE:	First release of the EasyTV Service Development Kit
WORK-PACKAGE:	WP 5
DELIVERABLE TYPE:	Demonstrator
CONTRACTUAL DATE OF DELIVERY:	31-10-2018
LAST UPDATE:	25-10-2018
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	05/09/2018	Draft	Stavros Skourtis ARX Athanasios Chatziathanasiou ARX Chrysostomos Bourlis ARX Zisis Kolias ARX	Table of Contents definition and document structure
v.0.2	25/09/2018	Draft	Stavros Skourtis ARX Athanasios Chatziathanasiou ARX Chrysostomos Bourlis ARX Zisis Kolias ARX Fivos Asimakopoulos ARX	First draft
v.0.3	19/10/2018	Draft	Serafeim Kosyvakis ARX Ioannis Doudoulakis ARX Christos Vlemmas ARX Pavlos Girsas ARX	First full version ready for internal review
V.0.4	23/10/2018	Draft	Kosmas Dimitropoulos CERTH	Review
V.0.5	24/10/2018	Draft	Giuseppe Vitolo ENG	Review
V.0.6	24/10/2018	Draft	Silvia Uribe UPM	Review
V.0.7	25/10/2018	Final	Chrysostomos Bourlis ARX Stavros Skourtis ARX Serafeim Kosyvakis ARX Ioannis Doudoulakis ARX Fivos Asimakopoulos ARX Christos Vlemmas ARX Pavlos Girsas ARX Athanasios Chatziathanasiou ARX Zisis Kolias ARX	Final deliverable preparation

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

Table of Contents

Executive Summary **9**
1. Introduction **10**
2. Nature of the EasyTV SDK **11**
2.1. Integration of EasyTV services in third party products 11
2.2. Integration of third party services in the EasyTV platform 11
3. Components **12**
3.1. Introduction 12
3.2. SDK for HbbTV applications to integrate with the EasyTV Companion Screen 12
3.2.1. HbbTV Terminal library setup 13
3.2.2. Navigation 13
3.2.3. Events 14
3.2.4. Video controls 14
3.2.5. Information extraction 15
3.2.6. Video Synchronization 15
3.3. SDK with accessibility feature for Companion Screen applications 17
3.3.1. SDK for integrating Companion Screen with Terminal HbbTV applications 17
3.3.2. EasyTV accessible buttons 21
3.4. Service Manager 22
3.4.1. Service Registration Tool 22
3.4.2. Service Manager SDK for services in EasyTV platform 22
3.4.3. Service Manager SDK for the content owner 23
4. Conclusion and future work **24**
5. References **25**

List of Figures

Figure 1 : Interoperability between the EasyTV Companion Screen application and a terminal HbbTV application..... 13
Figure 2 Integration of Terminal and Companion Screen application using the EasyTV SDK..... 17
Figure 3 The Service Manager as a gateway to the EasyTV platform..... 22
Figure 4 EasyTV SDK of the internal API..... 23
Figure 5 EasyTV SDK for the public API..... 23

List of Tables

Table 1 SDK Components related to HbbTV applications..... 12
Table 2 SDK Components related to the Service Manager..... 12

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 to 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

This deliverable contains many technical details in order to describe the usage and the features of the various components of the EasyTV SDK. For this reason, the reader of this document must have sufficient technical knowledge about software development in order to be able to properly understand it.

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 consist of command line tools and libraries in order to assist the development of new services and, also, 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 specifically.
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 indented 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 indented 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.

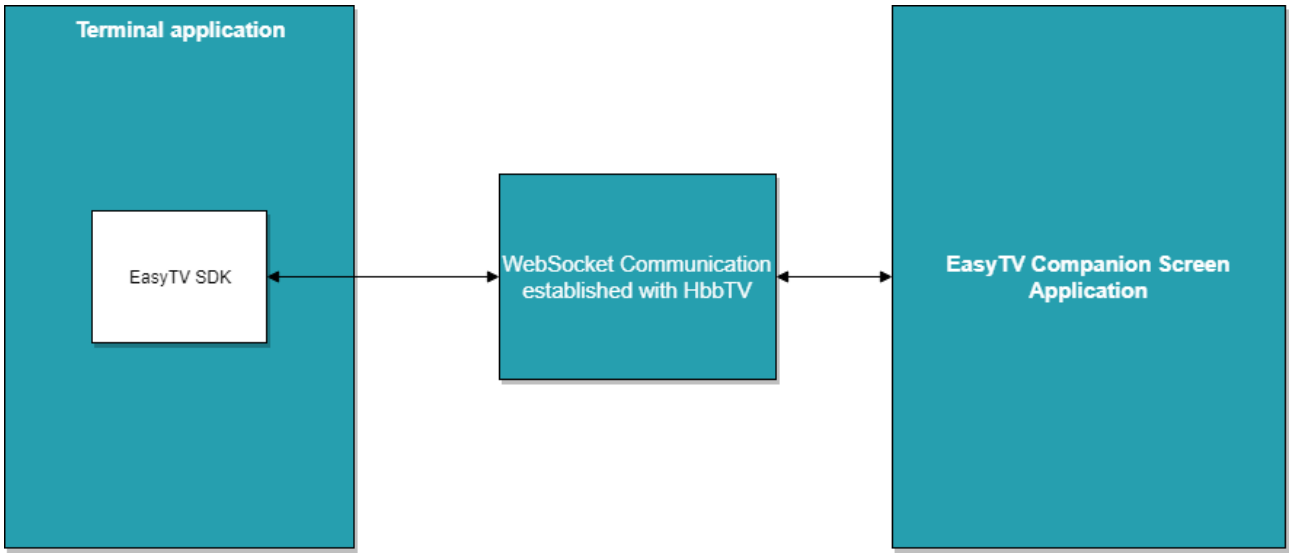

Figure 1 : Interoperability between the EasyTV Companion Screen application and a terminal HbbTV application

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 using the key presses, that are 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 like 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 arguments depends on the specific event that is handled.

3.2.4. **Video controls**

The Companion Screen application, also, 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 the case that 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 using the *return* keyword.

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 using the *return* keyword.

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 using the return keyword, 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 using the return keyword. 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 logic about the synchronization is handled entirely 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 that 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 that they need to do more actions. The non-usage of the dash.js events directly, is guaranteed by the library in its entirety 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 mean, both, 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 the two events, that are 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

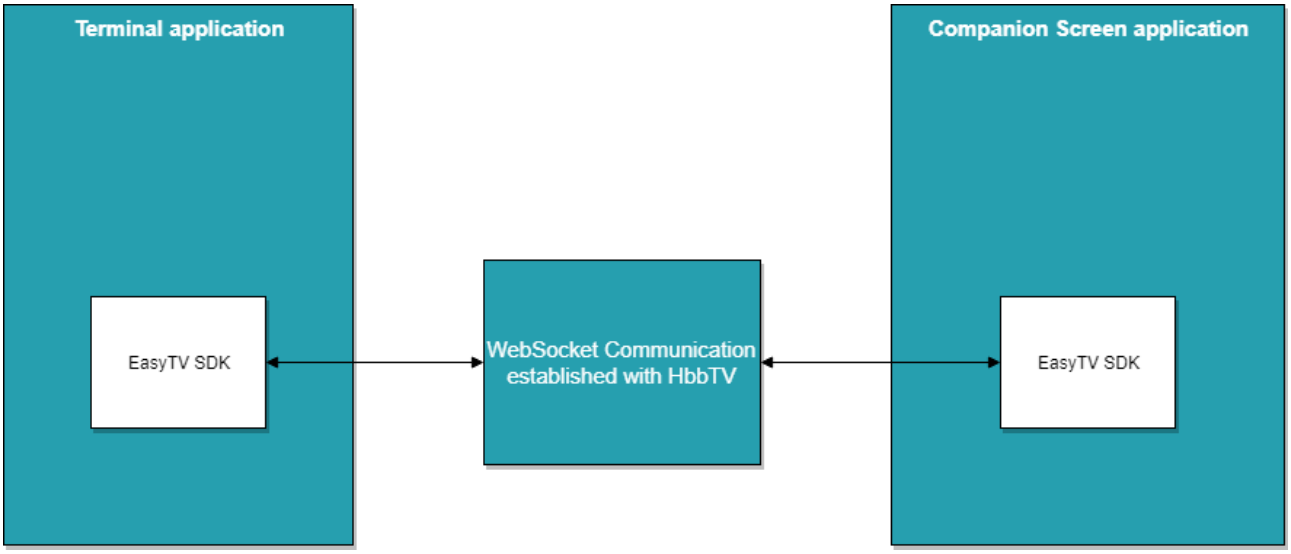

Figure 2 Integration of Terminal and Companion Screen application using the EasyTV SDK

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 to 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 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 nuber 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 it 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 stoped 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 sychronization 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 has values 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 that it offers. The first is that it makes the device vibrate when the user touches a button, giving a blind or visually impaired user a physical verification 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, instead of doing 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 is 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 behavior 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 to have in mind 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. For javascript, that means the library's event listener must be registered first.

3.4. 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.

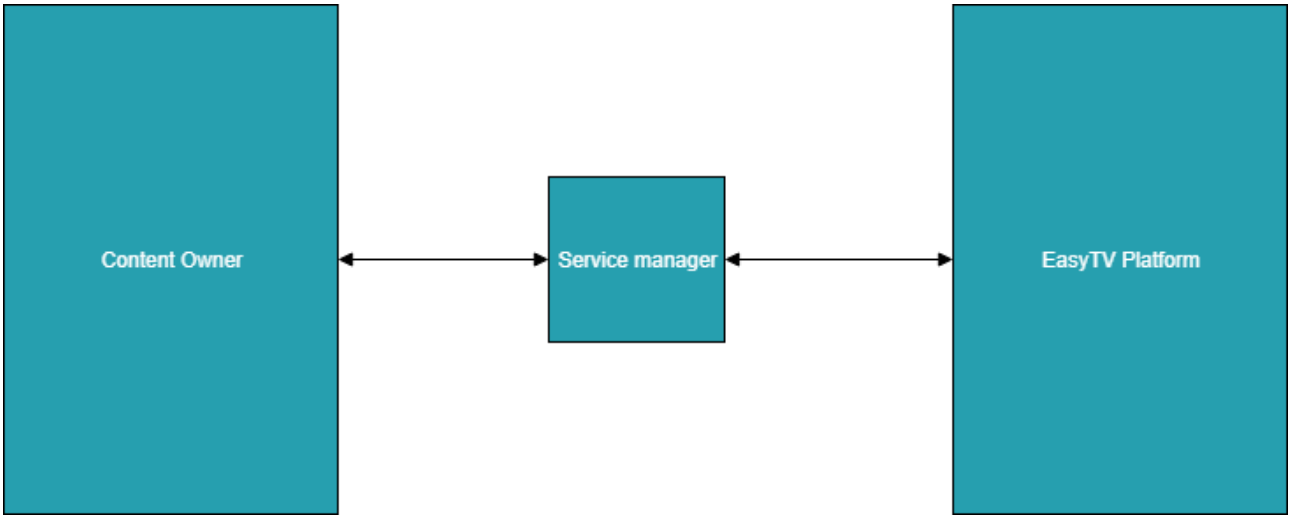

Figure 3 The Service Manager as a gateway to the EasyTV platform

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.

The development of the Service Manager at the time that this document is written is at a very early stage, so the description of the components of the SDK about it is done at a very high level. Having that in mind, some design decisions that are made for this part of the SDK may change. Nevertheless, the final version of the EasyTV SDK will be accurately defined in the final release.

3.4.1. 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 power of the Service Manager.

3.4.2. Service Manager SDK for services in EasyTV platform

With the Service Registration Tool, it is described how new services are registered to the Service Manager. In this section, it will be described how the Service Manager communicates with the

other services.

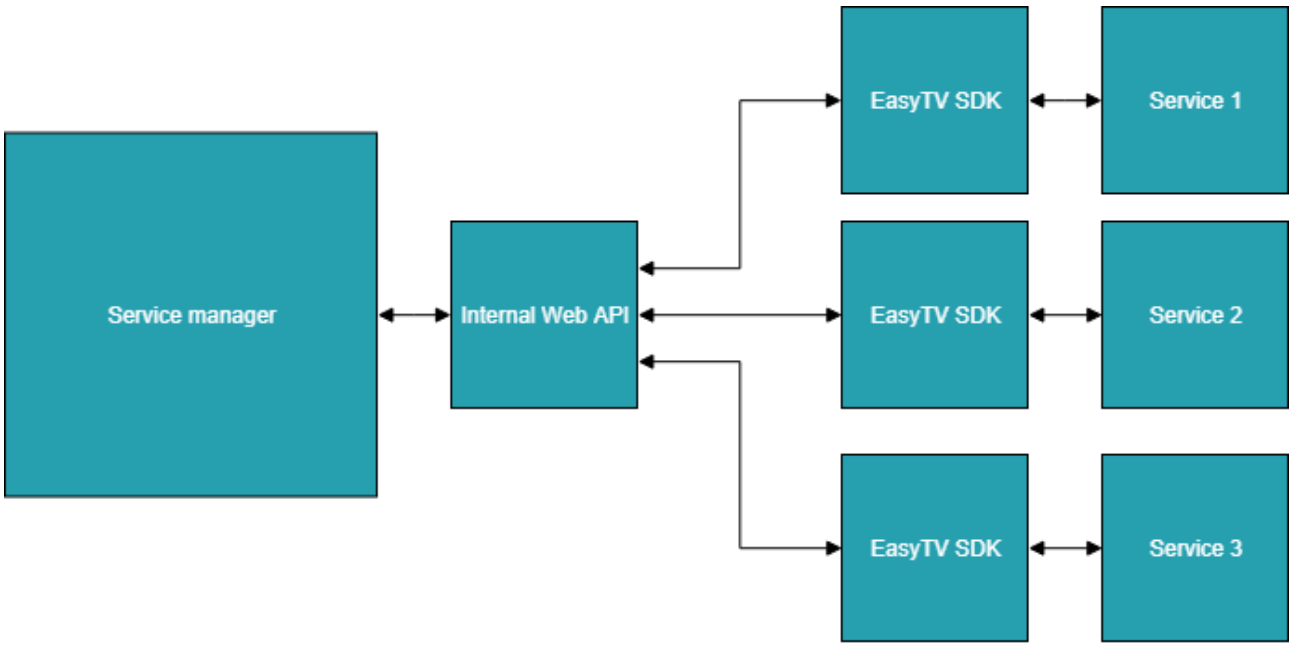

Figure 4 EasyTV SDK of the internal API

The Service Manager provides an internal Web API. Internal, in this case, means that it is to be used only from services inside the EasyTV platform. In order to ease the development of new services, this component of the SDK consists of libraries for programming languages that abstract the internal Web API in an object oriented way. With 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. Contrary with the Service Registration Tool, this component is not mandatory to use and the developer can still use the web API directly if they wish so.

3.4.3. **Service Manager SDK for the content owner**

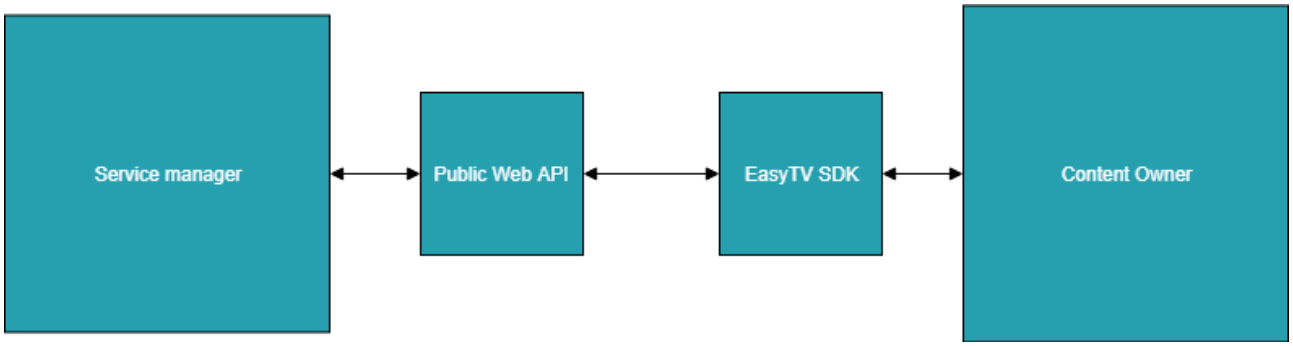

Figure 5 EasyTV SDK for the public API

The Service Manager provides a Web user interface for the Content Owner in order to manage jobs. It, also, provides a Web API that has, almost, the same capabilities as the Web UI and allows the Content Owner to automate their procedures. This component has the same purpose as the SDK for services described in 3.3.2. The only thing that changes is the user, which is the Content-Owner and the target Web API, which is the public Service Manager Web API.

4. Conclusion and future work

The architecture of the EasyTV Service Development Kit was described in detail in this 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 about 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 became compatible with any HbbTV application, that is using the EasyTV SDK.

Moreover, the SDK contains components related to the Service Manager. The service registration tool will help the platform administration 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. Also, in order to assist the development of the services themselves, the SDK for the internal Service Manager Web API exist.

In this first release of the EasyTV SDK its components are described in an early stage. As the development of the rest of the EasyTV services continues, the components of the SDK will become more clear and mature.

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)