D3.3 3rd Edition: HRADIO mobile and HMTL client API implementations

Editor: Alexander Erk (IRT)

This deliverable includes the software documentation of the HRADIO Client libraries in its 3rd edition. Newly added sections describe the new RadioDNS and ServiceLookup and Metadata search libraries. Substantial amendments have been made with the implementation of server-based time shift into the DAB-IP splitter and the player libraries for web and Android. Additionally, the basic design and build model of the iOS version of the HRADIO DAB library have been added in this 3rd iteration.
### D3.3: HRADIO mobile and HTML client API implementations

#### Basic Information

<table>
<thead>
<tr>
<th>Work package</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Due date</td>
<td>30/09/2019</td>
</tr>
<tr>
<td>Submission date</td>
<td>31/03/2020</td>
</tr>
<tr>
<td>Deliverable lead</td>
<td>IRT</td>
</tr>
<tr>
<td>Version</td>
<td>3.0</td>
</tr>
<tr>
<td>Authors</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Klaas Baert (VRT), Maximilian Knop (Konsole)</td>
</tr>
</tbody>
</table>

#### Document Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of change</th>
<th>List of contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>19/07/2018</td>
<td>Initial version</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>V0.2</td>
<td>20/07/2018</td>
<td>Updates from review</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>V0.9</td>
<td>27/07/2018</td>
<td>Word editing</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>V1.0</td>
<td>30/07/2018</td>
<td>Coordinator review</td>
<td>Simon Delaere (imec)</td>
</tr>
</tbody>
</table>

**Work on 2nd iteration**

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of change</th>
<th>List of contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.1</td>
<td>07/01/2019</td>
<td>Initial version</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>V1.2</td>
<td>04/02/2019</td>
<td>Final draft for review</td>
<td>Alexander Erk (IRT), Fabian Sattler (IRT), Benedikt Vogel (IRT)</td>
</tr>
<tr>
<td>V1.9</td>
<td>28/02/2019</td>
<td>Updates from review and final Word editing</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>V2.0</td>
<td>28/02/2019</td>
<td>Coordinator review</td>
<td>Simon Delaere (imec)</td>
</tr>
</tbody>
</table>

**Work on 3rd iteration**

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description of change</th>
<th>List of contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2.1</td>
<td>07/02/2020</td>
<td>Final version for internal review</td>
<td>Alexander Erk (IRT)</td>
</tr>
<tr>
<td>V2.2</td>
<td>04/03/2020</td>
<td>Initial internal review</td>
<td>Klaas Baert (VRT)</td>
</tr>
<tr>
<td>V2.3</td>
<td>31/03/2020</td>
<td>Final internal review</td>
<td>Maximilian Knop (Konsole)</td>
</tr>
<tr>
<td>V3.0</td>
<td>31/03/2020</td>
<td>Final version after coordinator review</td>
<td>Simon Delaere (imec)</td>
</tr>
</tbody>
</table>
Disclaimer

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731677. This document reflects only the authors’ views and the Commission is not responsible for any use that may be made of the information it contains.

<table>
<thead>
<tr>
<th>Project co-funded by the European Commission in the H2020 Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nature of the deliverable:</strong></td>
</tr>
<tr>
<td>PU</td>
</tr>
<tr>
<td>CL</td>
</tr>
<tr>
<td>CO</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

This is the 3rd iteration of the WP3 HRADIO client libraries deliverable. Main focus of this 3rd release is the addition of new libraries and platforms to the HRADIO core technology. In the 1st iteration of this document an Android based OMRI implementation was introduced and provided to the HRADIO partners. Basic components such as the time shift player have been developed and the general setup of the development workflow was described.

In the 2nd iteration of the deliverable, the following new components and platforms, according to the technical architecture are added:

- **The SPI fetching and parsing libraries** form a set of two libraries for the android OMRI implementation, which helps the application developers to simply fetch (download) and parse RadioDNS/WorldDAB SPI information. These two additions help application developers significantly to avoid repeating tasks and concentrate similar functionality into shared libraries.

- **The DAB-IP Splitter and OMRI IP player integrations** section introduces the concept and reasoning for the use of the DAB-EDI specification for the transport of single service IP radio streams. And mentions for implementation of ShoutCast and DAB-IP based IP player integrations for OMRI.

- **The JavaScript based DAB Player implementation** completes the concept of using DAB data streams also for IP delivered radio services. This section introduces the WP3 work on the WEB based DAB-IP player which enables the reception of full DAB Services over IP in a standard web browser.

- **The RadioWEB View component** provides the partner with an easy but yet powerful and reusable component, to enrich the radio program with interactive elements such a votings, quizzes or program related additional information. The sections describe the signaling, life cycle and API of the RadioWeb View.

In this 3rd iteration of the deliverable, the following components have been extended with additional features or have been added to the set of client libraries for the HRADIO application development:

- **The TimeShift Player Component** has been extended by the server-based components for time shifting the radio programme.

- **The DAB-IP Splitter and OMRI Player** integration has been extended for the server-based time shift functionalities.
- The **Service Lookup and Search Library** for accessing the HRADIO Metadata Platform via Android applications has been added.

- An initial implementation and build model for the **iOS platform**

- Added support for the server-based time shift in the **JavaScript based DAB Player**

- A new **RadioDNS library** replacing the SPI fetching and parsing lib now provides a comprehensive implementation of the RadioDNS services.
# TABLE OF CONTENTS

1. **INTRODUCTION** ........................................................................................................... 13

1.1. Amendments for the 2\textsuperscript{nd} iteration of D3.3 ........................................ 14

1.2. Amendments for the 3\textsuperscript{rd} iteration of D3.3 ........................................ 15

2. **OMRI OPEN MOBILE RADIO INTERFACE** ............................................................ 17

2.1. System Overview ...................................................................................................... 17

2.2. API Overview ........................................................................................................... 19

2.2.1. Packages ............................................................................................................. 20

2.2.2. Object Diagram .................................................................................................. 20

2.2.3. Radio State Model .............................................................................................. 21

2.2.4. Tuner State Model ............................................................................................. 22

3. **TIMESHIFT PLAYER COMPONENT** ........................................................................ 25

3.1. Classes (public methods only): ............................................................................... 25

3.1.1. TimeshiftPlayer.java ......................................................................................... 25

3.1.2. SkipItem.java ...................................................................................................... 26

3.2. Interfaces ................................................................................................................ 27

3.2.1. TimeshiftListener.java ...................................................................................... 27

4. **RADIODNS LIB** ........................................................................................................ 28

5. **DAB–IP SPLITTER AND OMRI IP PLAYER INTEGRATION** .................................... 31

5.1. DAB IP Splitter .......................................................................................................... 33

5.1.1. Server Based Time shift (SBT) ......................................................................... 34

5.1.2. SBT REST API ................................................................................................. 35

5.1.3. Custom EDI TAG ............................................................................................... 38

5.2. IP Tuner in Android OMRI Libraries ...................................................................... 39

5.2.1. Server Based Time shift .................................................................................... 39

6. **JAVASCRIPT BASED DAB PLAYER** ....................................................................... 40

6.1. SharedArrayBuffer .................................................................................................. 41

6.2. The use of libFAAD for audio decoding ................................................................. 42
6.3. Software Interface for web developers ........................................ 43
6.4. Hello world Demo .................................................................. 47
7. RADIOWEB VIEW COMPONENT ................................................... 48
7.1. Application Discovery .............................................................. 50
7.2. Document retrieval ............................................................... 50
7.3. Compression ......................................................................... 51
7.4. Language .............................................................................. 51
7.5. Application Information List (AIL) ......................................... 52
7.5.1. Contents ........................................................................... 52
7.5.2. Location .......................................................................... 53
7.6. Application Life-cycle ............................................................ 54
7.6.1. Application Identification .................................................. 54
7.6.2. Application life-cycle diagram .......................................... 54
7.7. API .................................................................................... 56
7.7.1. General API .................................................................... 56
7.7.2. Time shift API .................................................................. 59
7.7.3. Server Based Time shift ................................................... 61
8. SERVICE LOOKUP AND SEARCH LIBRARY .............................. 62
9. REPOSITORIES AND DEVELOPMENT SETUP FOR ANDROID .... 66
9.1. Repositories ........................................................................ 66
9.2. IDE Android Studio ............................................................ 67
9.3. Library Management ............................................................ 68
9.4. Simple developer workflow example ...................................... 69
10. IOS DEVELOPMENT .................................................................. 71
10.1. iOS Build Phases ................................................................. 72
11. CONCLUSIONS ........................................................................ 73
APPENDIX A ............................................................................... 74
A.1 Package org.omri.radio .......................................................... 74
A.1.1 Radio .............................................................................. 74
D3.3: HRADIO mobile and HTML client API implementations

A.1.2 RadioErrorCode........................................................................................................76
A.1.3 RadioListener................................................................................................................76
A.1.4 RadioStatus..................................................................................................................77
A.2 Package org.omri.radioservice.......................................................................................77
A.2.1 RadioService................................................................................................................77
A.2.2 RadioServiceAudiodataListener .................................................................................78
A.2.3 RadioServiceDab...........................................................................................................79
A.2.4 RadioServiceDabComponent.......................................................................................80
A.2.5 RadioServiceDabComponentListener .........................................................................82
A.2.6 RadioServiceDabUserApplication..............................................................................82
A.2.7 RadioServiceFm.........................................................................................................83
A.2.8 RadioServiceFmPty......................................................................................................83
A.2.9 RadioServiceIelp........................................................................................................84
A.2.10 RadioServiceIelpStream............................................................................................84
A.2.11 RadioServiceListener...............................................................................................85
A.2.12 RadioServiceMimeType..............................................................................................85
A.2.13 RadioServiceType.......................................................................................................86
A.3 Package org.omri.radioservice.metadata......................................................................86
A.3.1 Group .........................................................................................................................86
A.3.2 Location.......................................................................................................................87
A.3.3 ProgrammeInformation ..............................................................................................88
A.3.4 ProgrammeInformationType .......................................................................................88
A.3.5 ProgrammeServiceMetadataListener .........................................................................89
A.3.6 ServiceInformation .....................................................................................................89
A.3.7 SpiProgrammeInformation ........................................................................................89
A.3.8 TermId ........................................................................................................................90
A.3.9 Textual ........................................................................................................................90
A.3.10 TextualDabDynamicLabel .........................................................................................91
A.3.11 TextualDabDynamicLabelPlusContentType ..............................................................91
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1</td>
<td>HRADIO CLIENT LIBRARY ARCHITECTURE</td>
<td>13</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>RADIODNS HYBRID RADIO</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>OMRI SOFTWARE STACK</td>
<td>19</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>OMRI OBJECT DIAGRAM</td>
<td>21</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>RADIO STATE MODEL</td>
<td>22</td>
</tr>
<tr>
<td>FIGURE 6</td>
<td>TUNER STATE MODEL</td>
<td>23</td>
</tr>
<tr>
<td>FIGURE 7</td>
<td>SCREENSHOT “WHAT IS IT ABOUT” PILOT WITH PI DATA</td>
<td>29</td>
</tr>
<tr>
<td>FIGURE 8</td>
<td>USING DAB+ FOR BROADCAST AND IP DISTRIBUTION OF RADIO SERVICES</td>
<td>33</td>
</tr>
<tr>
<td>FIGURE 9</td>
<td>SERVER BASED TIME SHIFT SEGMENTATION</td>
<td>34</td>
</tr>
<tr>
<td>FIGURE 10</td>
<td>JAVASCRIPT DAB PLAYER ARCHITECTURE</td>
<td>41</td>
</tr>
<tr>
<td>FIGURE 11</td>
<td>DAB–IP SERVICE PLAYING IN JAVASCRIPT DAB PLAYER</td>
<td>42</td>
</tr>
<tr>
<td>FIGURE 12</td>
<td>JS DAB PLAYER „HELLOWORLD”</td>
<td>47</td>
</tr>
<tr>
<td>FIGURE 13</td>
<td>RADIOWEB OVERVIEW</td>
<td>49</td>
</tr>
<tr>
<td>FIGURE 14</td>
<td>LIFE–CYCLE OF A RADIOWEB APPLICATION</td>
<td>56</td>
</tr>
<tr>
<td>FIGURE 15</td>
<td></td>
<td>62</td>
</tr>
<tr>
<td>FIGURE 16</td>
<td>DEVELOPMENT WORKFLOW</td>
<td>66</td>
</tr>
<tr>
<td>FIGURE 17</td>
<td>HRADIO GROUP LANDING PAGE ON GITLAB–EXT.IRT.DE</td>
<td>67</td>
</tr>
<tr>
<td>FIGURE 18</td>
<td>ANDROID STUDIO WITH TIMESHIFTPLAYER EXAMPLE</td>
<td>68</td>
</tr>
<tr>
<td>FIGURE 19</td>
<td>ARTIFACTORY FOR HRADIO</td>
<td>69</td>
</tr>
<tr>
<td>FIGURE 20</td>
<td>SUCCESSFUL APK BUILD FOR “TIMESHIFTSAMPLEAPP”</td>
<td>70</td>
</tr>
<tr>
<td>FIGURE 21</td>
<td>IOS BUILD STRUCTURE</td>
<td>72</td>
</tr>
</tbody>
</table>
## ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Prot</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DAB</td>
<td>Digital Audio Broadcasting</td>
</tr>
<tr>
<td>DAB-DL</td>
<td>DAB Dynamic Label</td>
</tr>
<tr>
<td>DAB-SPI</td>
<td>DAB Service and Programme Information</td>
</tr>
<tr>
<td>DAB-SLS</td>
<td>DAB Slideshow</td>
</tr>
<tr>
<td>DNS</td>
<td>Dynamic Name Server</td>
</tr>
<tr>
<td>EPG</td>
<td>Electronic Programme Guide</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency modulation</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IDAG</td>
<td>International DMB Advancement Group</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>PI</td>
<td>Programme Information</td>
</tr>
<tr>
<td>MOT</td>
<td>Multimedia Object Transfer</td>
</tr>
<tr>
<td>SI</td>
<td>Service Information</td>
</tr>
<tr>
<td>SPI</td>
<td>Service and Programme Information</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

In deliverable D2.3 the selected HRADIO user scenarios have been analysed and technical system requirements have been derived. In these system requirements DAB/Radio centric technologies (e.g. DAB-DL+, DAB-SPI, DAB-SLS) play an important role and will be an absolute critical factor in successful pilot development. Therefore, WP3 decided to concentrate its first effort on the implementation of a comprehensive set of the OMRI API for the widely used Android system.

![HRADIO Client Library Architecture](image)

**Figure 1: HRADIO Client Library Architecture**

In March 2016 LG introduced a Smartphone with built-in DAB+ receiver for the European and Australian market. Smartphones with built-in radio receivers (mostly FM) are no novelty, but a new feature of the LG presentation was that for the first time, application developers were granted access to the hardware through a proprietary but publicly available API.

In order to enable the development of cross-system applications, independent of manufacturers or different tuner concepts, the IDAG (International DMB Advancement Group) decided to develop and standardize an API precisely for this purpose. The result of these efforts is OMRI (Open Mobile Radio Interface), which was first designed by IDAG members (IRT, Fraunhofer, BBC...) and will then be handed over for standardisation in WorldDAB.
One of OMRI’s main objectives was independence from transmission technologies. Whether the tuner to be used uses DAB, FM, HD radio or even AM as radio reception technology is of no importance in OMRI for the application developer. Although initially defined as Java API, care was taken in the design phase to ensure that equivalent API functions can also be implemented in C++/Swift for the iOS platform and JavaScript for common HTML5 browsers.

At the start of the HRADIO project, OMRI was only an API proposal with the idea to provide an easy to use application layer interface for radio applications. One of the development goals in work package 3 of the HRADIO project is the implementation of this OMRI API for the intended platforms. Based on the existing C++ DAB library developments of the project partner IRT, a Java JNI layer was implemented to make the functions for demultiplexing and decoding of digital DAB+ services realized in C++ accessible from the Java layer in Android.

For the iOS platform the situation is a bit more straightforward because the iOS development platform and it’s tools allow the inclusion of C++ code directly. It might become necessary to develop a thin C++ to Swift layer, but this layer’s complexity would be lower than a JNI layer for Android.

On top of that, browser platforms play an important role of radio listening. To support these platforms, the common C++ Layer has been successfully transcompiled using the emscripten/webassembly tools into a JavaScript library running directly inside common browser sandboxes.

Based on the OMRI API, a player component was also developed to temporarily store radio services and their data components and play them back at different times – i.e. a time shift component. To make it easier for application developers to get started with the libraries and components, sample code was written and a Maven repository (Artifactory) was set up to deploy the OMRI implementation and the player components.

### 1.1. AMENDMENTS FOR THE 2ND ITERATION OF D3.3

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI Fetching and parsing libraries</td>
<td>New chapter describing the aim and the basic usage of the HRADIO SPI fetching and parsing libraries.</td>
</tr>
</tbody>
</table>
D3.3: HRADIO mobile and HTML client API implementations

<table>
<thead>
<tr>
<th>DAB–IP Splitter and OMRI Integration</th>
<th>New chapter describing the reasoning and implementation details of the DAB-IP distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript based DAB player</td>
<td>New chapter describing the development process for the JavaScript based DAB–IP player.</td>
</tr>
<tr>
<td>RadioWEB View Component</td>
<td>New chapter describing the RadioWEB component, its signalization, application lifecycle and JavaScript API</td>
</tr>
<tr>
<td>Source code repositories and development setup for Android</td>
<td>Small changes due to the change of the GitLab server to gitlab.irt.de</td>
</tr>
</tbody>
</table>

Table 1: Changes and additions in version 2.

Besides the new components and libraries in the table above, the overall stability was improved and some minor bugs in the existing OMRI Android implementations, where fixed.

1.2. AMENDMENTS FOR THE 3RD ITERATION OF D3.3

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeShift Player component (Section 3)</td>
<td>Updated with the changes needed for support of server–based time shift.</td>
</tr>
<tr>
<td>SPI Fetching and Parsing renamed to RadioDNS Lib (Section 4)</td>
<td>New fully comprehensive RadioDNS Library for SPI, RadioVIS (including STOMP) and RadioWEB resolution.</td>
</tr>
<tr>
<td>DAB–IP Splitter and OMRI Integration (Section 5)</td>
<td>Updated chapter for the integration of the server–based time shift into the DAB–IP splitter</td>
</tr>
<tr>
<td>JavaScript based DAB player (Section 6)</td>
<td>Updated chapter describing the amendment in the REST APIs for</td>
</tr>
</tbody>
</table>
### Table 2: Changes and additions in version 3.

<table>
<thead>
<tr>
<th>Component/Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RadioWEB View Component (Section 7)</td>
<td>Updated chapter describing the amendment for server-based time shift functionality.</td>
</tr>
<tr>
<td>HRADIO Service and Metadata lookup API (Section 8)</td>
<td>New chapter for the application level access to the HRADIO Metadata platform.</td>
</tr>
<tr>
<td>Source code repositories and development setup for iOS (Section 10)</td>
<td>New chapter describing the iOS development setup.</td>
</tr>
</tbody>
</table>
2. OMRI OPEN MOBILE RADIO INTERFACE

2.1. SYSTEM OVERVIEW

Normally mobile devices have the capability to connect to IP based networks either over integrated WiFi or cellular communications systems (3G, 4G). This connectivity however only allows for point to point connectivity. Access to broadcast services such as DVB for TV or DAB/FM for Radio is often not possible. Even if mobile devices are equipped with broadcast receivers, application developers do not have access to the hardware tuner resources and therefore they cannot enhance their media centric applications with access to broadcast services.

Technologies such as IP audio streaming and Podcasts have enabled service offerings for on-demand experiences in radio consumption. Specifications such as RadioDNS [ref] will allow a combination of broadcast and IP based services today known as Hybrid Radio. In order to utilize this potential, it is important to combine broadcast media with individual accessed on-demand content in a seamless User eXperience (UX).

The following drawing depicts a system overview of mobile devices accessing Radio and hybrid services using RadioDNS as the “pathfinder” between them.
While application developers understand very well how to access and implement IP based services on target mobile platforms, such an easy access does not exist for broadcast media. This OMRI API specification closes this gap, by providing a standardized, technology agnostic API for application developers to develop hybrid radio applications.
The receiver product receives broadcast data via the tuner hardware which includes the audio services as well as additional metadata such as MOT-Slideshow, DL/DL+, MOT-EPG and other information. The tuner hardware usually is integrated into the device’s OS via a driver software usually provided by the manufacturer of the tuner hardware itself. A middleware software layer uses the data provided by the driver software to perform a range of tasks to extract the audio and metadata for presentation to a User Application (the App). Such tasks can include demodulation, demultiplexing and decoding of the different service components. Currently different manufacturers provide custom APIs for access to their tuner hardware and middleware software and consequently the user application must conform to individual tuner solutions. The OMRI API standardizes the access to tuner solutions and enables the development of comprehensive radio applications.

2.2. API OVERVIEW

The following section gives an overview of the OMRI API. Additionally the core Classes state models (org.omri.radio and org.omri.tuner) are described. For a detailed analysis, the complete definition of the API can be found in Annex A of this document.
2.2.1. Packages

The OMRI API currently consists of three different packages:

**org.omri.radio:**

The radio package acts as the entry point into the API for the developer. The main class in org.omri.radio is the Radio class which is designed as a singleton and provides a simple getInstance method for the application developer to obtain the Radio instance for further usage. Additionally in the org.omri.radio package Enumerations for error and status codes are defined. The access to broadcast data is highly asynchronous, therefore the org.omri.radio package defines the base class of all further interfaces of listeners in the OMRI API.

**org.omri.radioservice:**

The org.omri.radioservice package contains all the necessary definitions for applications developers to access radio service information such as Service labels, descriptions, logos, and many more. While the general radio service model in OMRI is agnostic to the underlying broadcasting technology, the org.omri.radioservice package contains the necessary sub-interfaces derived from the org.omri.radioservice. The RadioService interface reveals broadcast system specific information and metadata to the developer. Derived from the RadioListener interface, org.omri.radioservice and its sub-package metadata defines specific listener interface definitions for service data components such as DynamicLabel/DynamicLabel+, visuals (Slideshow) and programme information.

**org.omri.tuner:**

The org.omri.tuner package defines the abstract Tuner interface which enables the developer to access radio functionalities such as service scan. The OMRI API is designed to be able to handle devices which include multiple tuners even for different transmission technologies (e.g. DAB, FM and IP). The same package contains the TunerListener interface which is used to deliver highly asynchronous information (e.g. service scan status, signal levels).

2.2.2. Object Diagram

The following drawing shows an example OMRI instance diagram. Beginning with the singleton instance of the Radio class, Radio.getAvailableTuners returns a list of Tuner instances. Querying the TunerStatus reveals that one of the tuners is in TUNER_STATUS_INITIALIZED state and can return an instance of RadioServiceDab. Subscribed to the RadioServiceDab instance are two RadioServiceListener subclasses receiving events from the arrival of new Visual and Textual metadata.
Figure 4: OMRI Object Diagram

2.2.3. Radio State Model

The following drawing depicts the state model of the Radio class. The Radio can have three different states:
**D3.3: HRADIO mobile and HTML client API implementations**

**Figure 5: Radio State Model**

- **not initialized**: This is the initial state of the Radio class. This means that when the OMRI application is started and a Radio instance is obtained through the getInstance() method the call to getRadioStatus() returns STATUS_RADIO_UNINITIALIZED. In this state, calls to getAvailableTuners() and/or getRadioServices() will return empty lists. Therefore, also no Tuners can be initialized nor RadioServices can be started.

- **running**: Calling one of the initialize() or resume() methods brings the Radio object into running status. If an ERROR_INIT_OK or ERROR_RESUME_OK is returned, the Radio object is in running state and ready for tuner initialisation and/or service selections.

- **suspended**: Calling suspend() on a running Radio object brings it into suspended status. All activities by the Radio class will be suspended until its status changes back to running.

**2.2.4. Tuner State Model**

The following drawing depicts the state model of the Tuner object.
**Figure 6: Tuner State Model**

**not initialized**: This is the initial state of the Tuner object.

**initialized**: Calling the initializeTuner() or resumeTuner() methods brings the Tuner object into initialized status. When in the initialized state a service scans or service selection can be performed. When the tuner is scanning, the stopServiceScan() method can be used to terminate the scan and return to the initialized state.

**suspended**: Calling suspendTuner() on a running Tuner object brings it into suspended status. All activities by the Tuner will be suspended until its status changes back to initialized.

**scanning**: Calling the startServiceScan() method while in the initialize state will start a service scan on the Tuner. When finished the Tuner goes back into initialized state automatically. While in scanning state, the method getCurrentRunningRadioService will return null.

**error**: For many reasons the device or underlying driver software can cause a Tuner to go into error status. By calling the delinitializeTuner() method, the developer can try to bring the Tuner back into a defined initial state. However, if the error cause is
still valid the Tuner can go instantly into an error state again. Tuner implementations shall provide a meaningful status description in the newStatus parameter when calling the tunerStatusChanged() method.
3. TIMESHIFT PLAYER COMPONENT

The time-delayed consumption of media is a widely used service in the TV sector. TV devices with hard disk storage allow a local buffer on the end device. Technologies such as HbbTV also provide mechanisms to enable network-based time shift implementations.

Time shift also makes sense in the radio sector. In deliverable D2.3 the technical requirements of the HRADIO use cases were examined and it was determined that time shift components play a central role for many of the selected HRADIO use cases. Six of the scenarios required the availability of a local time shift player, in order to pause the live radio program.

Therefore, the development of a reusable local time shift player component has become a high priority in the early phase of T3.3.

For the Application developers the time-shift player API defines two classes and one interface for its functionality. Basically the implementation uses an android.media.AudioTrack object to play back the audio on the device and maintains a local buffer for time shifting this data.

When the TimeshiftPlayer detects a DynamicLabel+ Item.toggle bit change, a new SkipItem object is created and added. These SkipItems provide a content-based segmentation of the time shift buffer.

3.1. CLASSES (PUBLIC METHODS ONLY):

3.1.1. TimeshiftPlayer.java

```java
/**
 * Creates a {link TimeshiftPlayer}
 * @param context the Application {link Context}. Must not be {code null}.
 */
public TimeshiftPlayer(Context context)

/**
 * Prepares the {link TimeshiftPlayer}
 * @param timeshiftService the {link RadioService} to timeshift
 * @throws IllegalArgumentException if the timeshiftService is not a DAB+ service. May change in further updates
 */
public void prepare(RadioService timeshiftService) throws IllegalArgumentException

/**
 * Stops the {link TimeshiftPlayer}. You can reuse the instance by calling prepare.
 * @param deleteTmp {code true} if you want to delete the previously created timeshift file,
 * {code false} otherwise
 */
public void stop(boolean deleteTmp)

/**
 * Call it when you wish that playback starts immediately without calling play()
 */
public void setPlayWhenReady()
```
3.1.2. SkipItem.java

/**
 * Returns the number of previous saved AUs
 * @return the number of previous saved AUs
 */
public long getWrittenAus()

/**
 * Returns the skip point
 * @return the skip point
 */
public long getSkipPoint()

/**
 * Returns the time point relative to the start timepoint in milliseconds
 * @return the time point relative to the start timepoint in milliseconds
 */
public long getRelativeTimepoint()

/**
 * Returns the Textual at the point of the skip item
 * @return the Textual at the point of the skip item
 */
public Textual getSkipTextual()

/**
 * Returns a Visual at the point of the skip item or {code null} if not available
 * @return a Visual at the point of the skip item or {code null} if not available
 */
public Visual getSkipVisual()

## 3.2. INTERFACES

### 3.2.1. TimeshiftListener.java

void progress(long curPos, long totalDuration);
void sbtRealTime(long realTimePosix, long streamTimePosix, long curPos, long totalDuration);
void started();
void paused();
void stopped();
void textual(Textual timeshiftLabel);
void visual(Visual timeshiftVis);
void skipItemAdded(SkipItem addedItem);
void skipItemRemoved(SkipItem removedItem);

Together with the component implementation an example app has been developed, which uses the OMRI library to receive DAB+ Radio services (after a successful service scan), time shifts the selected service and provides a GUI with the received SkipItems (DL+ text messages and slide show images).
4. RADIODNS LIB

In the 2nd iteration of this deliverable, the SPI Fetching and Parsing Lib was introduced. This lib enabled a convenient access to basic RadioDNS/RadioEPG information for the application developer. In the technical work after the 2nd iteration of this document, a fully comprehensive RadioDNS library including support for the following specification has been implemented.

- RadioDNS-Core Lookup: You need the core lookup specification to create a DNS lookup to find the FQDN (Fully Qualified Domain Name) for a radio station, so you can make an IP connection. TS 103 270 v1.3.1

- RadioVIS-Slideshow (SLS): Needed to deliver clickable visual content over IP alongside broadcast radio.

- RadioEPG (SPI): Needed to describe meta-data about a radio station (including logos) and its individual programmes and schedule, as well as an index of on-demand/podcast material. TS 101 499 v3.1.1 TS 102 818 v3.2.1

The OMRI API supports different types of tuners. Besides the broadcast technologies such as DAB and FM, also IP based tuners are possible. One simple solution for such an IP tuner implementation is to use a set of radio services from a previous DAB scan and perform a RadioDNS/SPI lookup for these services. An analysis of this SPI SI.xml service information not only reveals the IP bearers for the already known DAB services, it also reveals so-called sub brands. Sub brands are linear radio services (often with special interest) which are only IP streamed. For the purpose of finding RadioDNS/SPI based IP-only streams, an OMRI IP tuner has been implemented to download and parse necessary SI.xml files and to add and match the newly discovered radio service into the existing (DAB only) service list.

Additionally, a second component of the library has been developed which, once a RadioDNS/SPI lookup has been performed, downloads and parses the available program information file of the radio services. Programme information files contain the future radio schedule of at least a single day into the future. The German public broadcasters include a 7-day schedule in their RadioDNS/SPI files.

The RadioDNS RadioVIS specifications allows the service provider to signal images and text messages along with their radio services. While in DAB provides a native support for images (Slideshow) and Textmessages (DL/DL+) the RadioVIS specification adds such support to services not based on DAB or even for DAB services which don’t want to spend data capacities in the broadcast stream.
This functionality was developed for the "What is it about" scenario tested in pilot phase 1 together by Konsole and IRT.

![Screenshot](image-url)

**Figure 7:** Screenshot "What is it about" Pilot with PI data.

The complete sample code is available on the HRADIO project repositories described in section 8 of this deliverable.

Users of the library create an instance via the constructor of the RadioDNSCore class.

```java
RadioDnsCore(@NonNull RadioService lookupSrv)
```

The parameter to this call is a OMRI RadioService object for which the core lookup should be performed.
As data retrieval over a network connection are usually block the calling application to long, the lib is designed to perform all possible blocking tasks in an asynchronous way. That means for the developer, that data which is retrieved over the library is send back into the calling application via callback objects passed into the according methods.

```java
/**
 * Initiates the core RadioDNS lookup
 * @param callback the {@link RadioDnsCoreLookupCallback}
 */
public void coreLookup(@NonNull RadioDnsCoreLookupCallback callback, @NonNull final Context context)
```

Once finished the applications callback object get notifies through the methods defined in the RadioDNSCoreLookupCallback interface.

```java
package eu.hradio.core.radiodns;
import org.omri.radioservice.RadioService;
import java.util.List;
public interface RadioDnsCoreLookupCallback extends RadioDnsCallback {
    void coreLookupFinished(RadioService lookupSrv, List<RadioDnsService> foundServices);
}
```

The list with resolved RadioDNSService objects can then be used by the application developer to get access to the RadioDNS service data itself.

Implementations of the abstract class RadioDNSService exist for the following services:

- RadioDNSServiceEPG
- RadioDNSServiceVIS
- RadioDNS ServiceWEB (see section RadioWEB view component)

The java package `eu.hradio.core.radiodns` then contains the implementation of the necessary functions to receive STOMP messages and create objects representing the OMRI classes `import org.omri.radioservice.metadata.Textual` and `org.omri.radioservice.metadata.VisualIpRdnsRadioVis`.

The java package `eu.hradio.core.radiodns.radioepg` then contains the implementations which parse the network retrieved data into classes representing the RadioDNS/WorlDAB SPI data model.

For a detailed usage example refer to the HRADIO platform app or the according sample apps in the HRADIO github repository.
5. DAB-IP SPLITER AND OMRI IP PLAYER INTEGRATION

For the transparent consumption of radio services either via DAB or IP, the same level of metadata and data services for both of the transmissions is necessary. Otherwise the listener would notice the (even temporary) switch over form DAB+ to IP and back.

Rebuilding the same level of audio- and data services in IP delivered scenarios could be difficult for the following topics:

**Seamless audio switching:** Switching between DAB+ based audio (HE-AAC) and IP based audio (often mp3 or HE-AAC) is rather difficult because of the fact that normally the same audio signal from the studio is going through different processing workflows for either DAB transmission or IP streaming, often even on different locations. Different delay times and different audio levels are the consequences.

**Data services (DL+ and SLS):** In traditional DAB/DAB+ services the provision of a slideshow service as well as DynamicLabel+ is more or less obligatory. DynamicLabel+ provides short text messages to the client, which contain additional metadata with title.artist or title.album information about the current song and exact timing information regarding the start, stop or pause status on a program event. Slideshow provides little jpg or PNG encoded images with program related information e.g. album art. However, slideshow (with categorization) can provide links to web pages or alternative (location based or device specific) content related to the actual slide. These two data services are transmitted “in Band” as additional information to the audio packets (PAD) which synchronizes them perfectly to the audio signal they relate to. Although RadioVIS provides an alternative of a text message and slideshow specification, opening the necessary private ports for the required STOMP protocol is difficult (esp. in cars) and RadioVIS cannot provide the same tight synchronization between the data and the audio service as its DAB counterparts.

**Announcement support:** As a radio technology the support of announcement signalization (Traffic, Alarm, Emergency) always have been in the core DNA of DAB. On FM, a similar technology in the form of RDS exists. Surely, server push information to the clients exists, however compared to the “in Band” broadcast announcements either additional IP ports or technologies such as web sockets or (for some mobile platforms i.e. iOS or Android) even proprietary systems.

For broadcasters and service providers this means additional development effort and additional operational resources.
As a solution, an alternative model has been developed. Using DAB data for streaming over IP.

The DAB Specifications also describe the encapsulation of DAB ensemble data into IP packets (DAB EDI). This encapsulation of a whole DAB ensemble (usually 10-12 services in a 1.8MBit/s multiplex) is used normally for the IP based contribution of DAB signals from the multiplexer to the transmission sites. The HRADIO project uses this specification but only for the transmission of a single radio service.

Benefits of the solution are:

- No additional work for broadcasters, who anyway produce the DAB ensemble for the broadcast transmission.
- All the signalization and data services supported in the DAB broadcast are available “in Band” for the IP based clients.
- No audio correlation and audio processing when implementing service following.
- For application developers everything becomes DAB.
- Well known and standardized solution.
Therefore, the following components have been developed for the HRADIO project:

### 5.1. DAB IP Splitter

The DAB-IP-Splitter receives a full multiplexed DAB EDI stream, which is also used for DAB broadcast contribution, via UDP unicast. The received multiplex stream is analyzed for how many DAB services are multiplexed. Every single DAB service is then split out of the full multiplex and the necessary parameters in the EDI-AF (Application frame) layer and the FIGs (Fast Information Group) contained in the EDI-DETI tag are adjusted to generate a standard compliant EDI stream.

The following FIGs are adjusted for a single service EDI stream:

- FIG 00 Extension 01 - Basic Subchannel Organization
- FIG 00 Extension 02 - Basic service and service component definition
- FIG 00 Extension 13 - User application Information
- FIG 00 Extension 14 - FEC sub-channel organization

**Figure 8:** Using DAB+ for broadcast and IP distribution of radio services.
The service payload contained in EDI-MST tags stay untouched. The stream is then recompiled into a standard compliant single service EDI stream. For every single service stream, the DAB-IP-Splitter creates an entry in the HTTP REST-service it provides with the following scheme:

http://<server-url>/services/<subchannelid>

The services can now be obtained via HTTP and/or via HTTPS. The whole DAB-IP-Splitter is designed as a micro-service. Therefore, the rollout in distributed networks with load balancing is no problem.

5.1.1. Server Based Time shift (SBT)

The functionality of a Server-Based Time shift was added to the DAB IP splitter. This enables the broadcaster to offer some, or all, of its services with a configurable time shift-buffer. The listener of a server-based time shifted service gains a whole new radio experience. With this feature enabled the listener can immediately start listening from the start of a radio show or from the exact beginning of a segmented element of the radio programme. Also, the listener can seek freely in the range of the time shift-buffer.

Figure 9: Server Based Time shift segmentation
For a quick access to the underlying radio stream data the DAB IP splitter keeps it in its random-access memory. Further implementations may also decide to persist the data on storage systems.

The DAB IP splitter automatically segments the programme via the Dynamic Label Plus item toggle functionality. To achieve this functionality a full DAB and PAD decoder stack was implemented. It extracts the DL+ information from each subchannel contained in the full DAB multiplex. DL+ contains the information about the segmentation of the radio programme. The Slideshow decoder extracts visual content from the radio subchannel to enhance the visual representation of segments.

To indicate the availability of the server-based time shift functionality to a client, additional HTTP headers are sent by the DAB IP splitter:

- time shift-token: a unique id for an EDI streaming session
- time shift-max: indicates the maximum time shift range in milliseconds

The content-type HTTP header field is set to "audio/edi".

5.1.2. SBT REST API

An SBT streaming session is a constant stream. This means that the IP connection receiving the stream will not be closed when an action is issued. Only the status, e.g. the playback position, on the DAB IP splitter changes. Therefore the “time shift-token” field is necessary to target the belonging streaming session. The time shift-token is also persisted on the SBT server as long as the status of the session is valid. This can be used to resume a previously stopped or paused session.

The server-based time shift functionality of the DAB IP splitter is easily accessible via the following described REST API.

To get all the segmented items which are currently available in the buffer a client sends a HTTP POST request to the stream URL with following JSON encoded body data.

```
{
   "action": "items"
}
```

The DAB IP splitter answers the request with the following exemplarily JSON encoded body data.
A client who wants to seek in the buffer of its current streaming session has the following possibilities:

- seeking relative to the real time:

```json
{
    "action": "seek",
    "wantedPos": 15000,
    "timeshiftToken": "n-Tj9bggDLMzu-AMV6oEzg"
}
```
This request seeks 15000 milliseconds into the past from the current real-time position for the client with the given time shift-token.

- seeking to a fixed timepoint in form of a Unix timestamp

```json
{
  "action": "seek",
  "wantedUts": 1578321000,
  "timeshiftToken": "n-Tj9bggDLMzu-AMV6oEzg"
}
```

This request seeks to the Unix timestamp 1578321000, which translates to 6. January 2020 14:30:00.

- seeking directly to an item

```json
{
  "action": "toggle",
  "wantedPos": 29958,
  "timeshiftToken": "n-Tj9bggDLMzu-AMV6oEzg"
}
```

This request seeks to the item with id 29958. In this case this is the item “The Chainsmokers & Coldplay: Something just like this” which was obtained from the previous “items” action.

To pause or resume a streaming session the client sends the respective HTTP POST request

```json
{
  "action": "pause",
  "timeshiftToken": "n-Tj9bggDLMzu-AMV6oEzg"
}
```

```json
{
  "action": "play",
  "timeshiftToken": "n-Tj9bggDLMzu-AMV6oEzg"
}
```

In case of success the DAB IP splitter answers each request with the usual HTTP status codes. HTTP 200 on success and HTTP 400 on a failed request.

Some REST calls are also available with URL encoded parameters for the HTTP GET command at the start of the streaming session.

The available parameters are:
An example to start a session with a previously obtained time shift token would look like:

http://<server-url>/services/<subchannelid>?timeshiftToken=n-Tj9bggDLMzu-AMV6oEzg

### 5.1.3. Custom EDI TAG

To inform a time shifted client of newly available segmented items received from the real time stream a custom EDI TAG element was introduced. This is still full EDI standard compliant as any unknown TAG shall be ignored by a standard compliant receiver.

The TAG name is “dlpt”, short for dynamic label plus toggle. The payload of such a TAG contains a JSON encoded string as described previously in the “items” action.

The fact that a custom EDI TAG is still standard compliant emphasizes the flexibility of the EDI format.

### 5.2. IP TUNER IN ANDROID OMRI LIBRARIES

An IP-Tuner for receiving online radio streams has been added to the OMRI library implementation. With this implementation it is now possible to receive Shoutcast HTTP streaming stations. It also extracts the so-called ‘ICY-tags’ in-band metadata from the streams. These ICY-tags may contain textual metadata for e.g. the currently running song title and artist. Depending on the end user’s device capabilities it is able to receive and decode Shoutcast AAC-ADTS and MP3 streams. Other formats as OGG and FLAC are possible to be implemented in a next iteration of the IP-Tuner.

Also, an IP-Tuner for receiving DAB-EDI-Streams (as described previously) via HTTP/HTTPS was added to the OMRI implementation. For the EDI-IP-Tuner a new component for decoding the AF-Layer of the EDI-stream was developed. It decodes...
and extracts the data in the various EDI-Tag items and feeds this into the DAB decoding library. As the decoded EDI stream corresponds to a standard DAB signal, no changes have to be made to the DAB decoding library to decode all currently supported components like DAB Slideshow (SLS) with categorization and DAB Dynamic Label Plus.

5.2.1. **Server Based Time shift**

The IP-Tuner for receiving EDI streams was extended to use the server based time shift functionalities. The needed REST calls, as described in 5.1.2, were implemented.
6. JAVASCRIPT BASED DAB PLAYER

According to Image 1 (HRADIO Client Library Architecture) the development goal for enabling the playback of DAB-IP http/https stream in browser environments was to reuse the existing C++ implementation of the DAB demultiplexing and decoding libraries. To be used in JavaScript, the C++ parts of the OMRI USB Library were translated to JavaScript. The Emscripten\(^1\) compiler generates JavaScript from C++ code, which can be executed in any web browser.

The JavaScript based OMRI USB Library processes FIC and MSC packets from an EDI\(^2\) stream and creates and forwards the MSC packets which contain the encoded audio data into a libFaad based audio decoder, which returns decoded PCM samples can be played with the HTML audio element in a web browser using the Media Source Extensions\(^3\).

In addition to the audio data, the metadata distributed by the DAB signal is also decoded by the JavaScript based OMRI USB Library and can be displayed in a web browser. These are mainly the Slide Show Service incl. the categorized Slide Show Service and the Dynamic Label Service, respectively the Dynamic Label Plus Service.

The following drawing shows the architecture.

---

\(^1\) https://emscripten.org
\(^2\) https://www.etsi.org/deliver/etsi_ts/102600_102699/102693/01.01.01_60/ts_102693v010101p.pdf
\(^3\) https://developer.mozilla.org/en-US/docs/Web/API/Media_Source_Extensions_API
The benefit for the HRADIO project is the fact that for all three targeted platforms (Android, iOS and HTML/JavaScript), the same core code base can be used.

### 6.1. SHAREDARRAYBUFFER

The `SharedArrayBuffer` object is needed for multithreading scenarios of C++ applications that have been made executable by Emscripten on web browser platforms.

---

The SharedArrayBuffer object in the web browser Environment is used to represent a generic, fixed-length raw binary data buffer, similar to the ArrayBuffer object, but in a way that they can be used to create views on shared memory.

SharedArrayBuffer was disabled by default in all major browsers on 5 January 2018 in response to Spectre. Chrome re-enabled it in v67 on platforms where its site-isolation feature is enabled to protect against Spectre-style vulnerabilities.

In current (January 2019) Chrome Desktop Browsers the SharedArrayBuffer Object is activated by default again, but not on Mobile Devices. On mobile devices the SharedArrayBuffer Object must still be activated manually, which is possible under chrome://flags/#shared-array-buffer

The following picture shows the browser window while decoding a DAB-IP https stream.

![DAB-IP Service playing in JavaScript DAB Player](image)

**Figure 11:** DAB-IP Service playing in JavaScript DAB Player

### 6.2. THE USE OF LIBFAAD FOR AUDIO DECODING

Since the OMRI USB Library only returns raw AAC packets without header information in the former version of this deliverable the implementation strategy was, to add ADTS headers, so that the Media Source Extension in the browser has the necessary
information to process the data stream correctly. The ADTS header mainly contains information about audio sampling frequency, and the channel configuration. This header is added to the audio data within the DAB-JS Library. While this concept worked in general, the 960-frame mode of the AAC encoding used in DAB+ couldn’t be decoded correctly by the common web browsers. The final result appeared to be pitched down by the amount of the missing data compared to the 1024-frame mode commonly used. Other tests with more detailed header and audio configuration options revealed implementation gaps in the browser platforms. Therefore, the HRADIO project decided to additionally include the libFaad audio decoder into the input code for the Emscripten compiler. This results in a perfectly pitched playback of the DAB+ audio data.

6.3. SOFTWARE INTERFACE FOR WEB DEVELOPERS

The EDI web streaming library has a simple API with just a few functionalities.

Developers find the API implementation under:

https://gitlab.irt.de/hradio/Libraries/dab-over-ip-webplayer

Simply cloning this repository into own web projects enables to start receiving real DAB radio services in the web environment.

The Library provides a ready to use object called ediPlayer. This Object will be available in the global scope as a property of the window. The ediPlayer Object has some properties described in the Table below.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Possible Values/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start (url)</td>
<td>starts streaming data and audio decoding, when you call the function for the first time, an url must be passed. After that, this is no longer necessary. For compatibility reasons, startStream() also exists.</td>
</tr>
<tr>
<td>stop()</td>
<td>stops audio decoding and streaming data</td>
</tr>
<tr>
<td>seek(ms)</td>
<td>if time shift is supported by the EDI server (the timeShiftControllerAvailable is thrown), the EDI stream can be set to a time relative to now in milliseconds. ediPlayer.seek(3600e3) sets the edistream an hour ago</td>
</tr>
</tbody>
</table>
seekUts(uts)

the same as ediPlayer.seek() only that the time is specified as absolute Unix time in Milliseconds.

ediPlayer.seekUts(new Date() - 1e3605) jumps 5sec back

toggleId(id)

the same as ediPlayer.seek() only that the jump marker is passed over an ID to jump to defined positions. these positions come from the items-array that is available through the items event

next()

same as ediPlayer.seek(), only that the time you jump to is from the list of toggle items and is relative to the current position of the stream. the stream is set to the beginning of the following item. However, this only works, if the service delivers toggles

prev()

same as ediPlayer.next(), only if an item ran longer than 3 seconds, it jumps to the beginning of the current item, otherwise to the beginning of the previous item

<table>
<thead>
<tr>
<th>Properties</th>
<th>Possible Values/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>represents the current player state. it can be running or stopped</td>
</tr>
<tr>
<td>audioCtx</td>
<td>this lib uses in the background the web Audio API. To increase control possibilities, this property provides the corresponding baseAudioContext</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>events</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>sls:</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>&quot;contentName&quot;: &quot;f851.jpg&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;clickThroughUrl&quot;: &quot;,&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;contentSubType&quot;: 1,</td>
</tr>
<tr>
<td></td>
<td>&quot;triggerTime&quot;: &quot;now&quot;,</td>
</tr>
<tr>
<td></td>
<td>&quot;categoryId&quot;: 0,</td>
</tr>
</tbody>
</table>

Table 3: API for HTML/Javascript Dab player
<table>
<thead>
<tr>
<th><strong>D3.3: HRADIO mobile and HTML client API implementations</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dls</strong>: Dynamic Label text received</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;dlsObject&quot;: {</td>
</tr>
<tr>
<td>&quot;dynamicLabel&quot;: &quot;Lana Del Rey - Video Games&quot;</td>
</tr>
<tr>
<td>&quot;charset&quot;: 0,</td>
</tr>
<tr>
<td>&quot;itemToggle&quot;: true,</td>
</tr>
<tr>
<td>&quot;itemRunning&quot;: true,</td>
</tr>
<tr>
<td>&quot;dlPlusTags&quot;: {}</td>
</tr>
<tr>
<td>},</td>
</tr>
<tr>
<td>&quot;dls&quot;: &quot;Lana Del Rey - Video Games&quot;</td>
</tr>
<tr>
<td>&quot;dlsp&quot;: {</td>
</tr>
<tr>
<td>&quot;ITEM_TITLE&quot;: &quot;Video Games&quot;</td>
</tr>
<tr>
<td>&quot;ITEM_ARTIST&quot;: &quot;Lana Del Rey&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td><strong>stateChange</strong>: Information about EDI player state</td>
</tr>
<tr>
<td>running or stopped</td>
</tr>
<tr>
<td><strong>missingSharedArrayBuffer</strong></td>
</tr>
<tr>
<td>No arguments. This function is just a convenience method for</td>
</tr>
<tr>
<td>a quick notification the the necessary shared object array</td>
</tr>
<tr>
<td>buffer is missing.</td>
</tr>
<tr>
<td><strong>msg</strong>: Some system messages such as notifications and error</td>
</tr>
<tr>
<td>messages.</td>
</tr>
<tr>
<td>Possible codes are:</td>
</tr>
<tr>
<td>afBroken,</td>
</tr>
<tr>
<td>noStreamData,</td>
</tr>
<tr>
<td>audioZero,</td>
</tr>
<tr>
<td>networkError</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>code: &quot;noStreamData&quot;</td>
</tr>
<tr>
<td>msg: &quot;No data has been received from the streaming server</td>
</tr>
<tr>
<td>for a few seconds. Is there a problem?&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>Or:</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>code: &quot;afBroken&quot;</td>
</tr>
<tr>
<td>msg: &quot;af seq counter broken&quot;,</td>
</tr>
<tr>
<td>seq: 60447,</td>
</tr>
<tr>
<td>lastSeqCounter: 60445,</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td><strong>serviceInfo</strong>: Is thrown if the current service name and</td>
</tr>
<tr>
<td>ensemble name is available</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>ensembleLabel: 'Antenne',</td>
</tr>
</tbody>
</table>
|   servlabel: 'Bayern'
| }                                                           |
| **timeShiftControllerAvailable**: Is thrown if the current  |
| edi server supports TimeShift.                              |
| {                                                           |
|   'Timeshift-Max': 360000                                   |
|   'Timeshift-Token': 'dsfh2345kh3'
| }                                                           |
token can be used to design multidevice scenarios by transferring the token to another device.
To start the same stream at the same position on another device, the time shift token is appended to the stream URL as get parameter. If the timeShiftControllerAvailable-event returns the same time shift token, it is possible to control the stream from both devices.

**items:**

An array of available items elements

```json
[
  {
    "label": "Talisco - Sun",
    "tags": [
      {
        "type": 1,
        "typeDescription": "ITEM_TITLE",
        "text": "Sun"
      },
      {
        "type": 4,
        "typeDescription": "ITEM_ARTIST",
        "text": "Talisco"
      }
    ],
    "toggleState": true,
    "runningState": true,
    "time": 1574668398548,
    "id": 314010,
    "slidePath": "",
    "slideMime": ""
  },
  ...
]
```

**liveItem:**

Items that are transferred at runtime. The structure corresponds to the elements from the items-array. (see items event)

**unhandledEdiTag:**

An EDI tag was found in the stream that was not further handled. This is especially the case for custom edi tags the case.

```json
{
  "tagName",
  "payload"
}
```

**dabTime:**

JavaScript Date Object
6.4. HELLO WORLD DEMO

This is a Hello World Demo of using the ediPlayer Library. It contains some MaterializeCss calls. Remember that you will need also a DOM containing an audio element.

You find this demo on: https://github.com/hradio

```javascript
// playBtn
const playPauseBtn = document.querySelector('#startAudioContext');
playPauseBtn.addEventListener('click', () => {
    // play or pause track depending on state
    if (playPauseBtn.dataset.playing === 'false') {
        ediPlayer.start({url: 'https://radio.stream:443/services/13'})
        playPauseBtn.dataset.playing = 'true';
        playPauseBtn.firstChild.innerHTML = 'access_time';
    } else if (playPauseBtn.dataset.playing === 'true') {
        ediPlayer.stop();
        playPauseBtn.dataset.playing = 'false';
        playPauseBtn.firstChild.innerHTML = 'access_time';
    }
}, false);

ediPlayer.addEventListener('stateChange', (state) => {
    switch (state) {
    case 'running':
        playPauseBtn.firstChild.innerHTML = 'stop';
        break;
    case 'stopped':
        playPauseBtn.firstChild.innerHTML = 'play_arrow';
        break;
    }
});

const prettyPrintJson = (json) => {
    M.toast({html: `<pre>${JSON.stringify(json, null, 10)}</pre>`});
}

ediPlayer.addEventListener('sla', (obj) => {
    M.toast({html: `<img src="" + obj.url + ">"});
    prettyPrintJson(obj);
});

ediPlayer.addEventListener('msg', (obj) => {
    prettyPrintJson(obj);
});

ediPlayer.addEventListener('dirs', (obj) => {
    prettyPrintJson(obj);
});
```

Figure 12: JS DAB player „HelloWorld“
7. RADIOWEB VIEW COMPONENT

In the TV world, HTML based interactive content for broadcast signals is a common way for broadcasters to enhance their viewers experience (HbbTV). It is therefore amongst the most popular services: according to the HbbTV website\(^5\), over 44 million HbbTV enabled devices are deployed in 35 countries worldwide. In 2016 IRT began to develop a similar technology called RadioWEB for smart radio devices for which a draft specification has been developed and published internally in the RadioDNS developer groups\(^6\).

The draft focused on the following aspects of a RadioWEB technology:

Section 1-3 “Application signalling and discovery”: These parts specify, how RadioWEB applications are signalled to the end user device. If necessary, during the service discovery process, different device options (e.g. display size) can be signalled and negotiated.

Section 4 “Application Life-Cycle”: This part defines the live-cycle of a RadioWEB application. This includes, when to start the web application, when to stop the web application. Furthermore, can web applications stay alive over service boundaries while preserving their state?

Section 5-6 “Content formats and Application API”: This part will specify the amendments to HTML5/CSS and JavaScript, which will enable the RadioWEB application to utilize the built-in radio device in order to select other services, access in band data and in band signalisations. Primary goals of this task are:

---

\(^5\) https://www.hbbtv.org
\(^6\) https://groups.google.com/forum/#!forum/radioweb-developers
Enable Web developers to easily develop HTML/CSS/JavaScript applications with the focus on accompanying the selected radio programme with additional data, content and links.

The developed APIs must be agnostic to the underlying broadcast technology (FM, DAB, HD, ...)

Make sure that the developed APIs are aligned with the work coming out of other groups – probably the Universal Smartphone Radio project – so there could be a clear mapping between the JavaScript functions and the corresponding native functions.

A similar architecture to the existing HbbTV technology has been proposed.

In HRADIO D2.3, RadioWEB was identified as a system requirement for different user scenarios (S17 Personalised Interactive Advertising, S19 Push the Button, S20 Traffic Jam Quiz, S21 More info, S25 Visual Stories, S26 Subtitles and S27 Lyrics). In WP3, for the existing draft specification an Android based WebView component was developed and the following definitions of the aforementioned RadioWEB draft have been included.
7.1. APPLICATION DISCOVERY

A device must be capable of resolving the authoritative FQDN for a service via the methodology defined in the RadioDNS specification.

Application lookup may then be performed against this FQDN by means of a DNS SRV Record request for the RadioWEB application by using the service name: radioweb

If at least one SRV record is successfully resolved, this service supports the RadioWEB application, accessed on the host and port indicated in the relevant SRV record. For example, for a query made to:

_radioweb._tcp.< authoritative FQDN >

Using the nslookup tool, this would yield the following SRC record:

service = 0 100 80 radioweb.hbbradio.de

This indicates that the RadioWEB application signaling can be accessed on the FQDN radioweb.hbbradio.de, port 80.

Note that more than one SRV record may be returned with different values. This can be used for load balancing purposes by providing different FQDNs/Ports with different priorities/weightings.

7.2. DOCUMENT RETRIEVAL

The host and port returned by the SRV record(s) refer to an HTTP server from which to acquire XML documents. The following sections detail these documents and how they may be retrieved and parsed.

Because the resource is returned via HTTP, the HTTP specification [3] should be correctly implemented. Attention is particularly drawn to the status codes section, which may be used to indicate problems and failures during attempts to retrieve documents.

In particular, the following behaviors must be adhered to:
A device **MUST** correctly follow any HTTP redirects that are returned when retrieving a document.

A device **MUST** respect any indicated document expiry in the HTTP response.

It is **RECOMMENDED** that devices cache retrieved documents, as per the HTTP specification.

It must also be noted that any file names given in this document should be treated as case sensitive in order to support different web servers.

The HTTP specification allows for the client to send additional parameters in the request header to a service provider in order for a more appropriate resource to be returned. The following subsections give a few examples of HTTP request parameters that may be used for specific purposes. Note that other parameters may be sent by devices and clients, and a service provider may optionally choose to vary the response as a result.

If a different response is sent back from a service provider based on the request headers, it is **RECOMMENDED** that the HTTP response include the Vary header to indicate to any intermediate caching layers that the response has been varied because of the originating request.

### 7.3. COMPRESSION

Should a client request indicate that they accept compressed documents using the Accept-Encoding header, the service provider may respond with a compressed document using one of the indicated encodings.

The service provider **MUST** include the compression method of the response within the Content-Encoding header in the HTTP response, as per standard HTTP behavior.

If requesting any compression, the client **MUST** then inspect the response to determine if, and how, any returned documents have been compressed and expand accordingly.

### 7.4. LANGUAGE

Should a client request indicate desired languages using the Accept-Encoding header, the service provider may respond with a version of the document more appropriate to that language.
The language of the element data should be correctly indicated using the xml:lang attribute, where applicable (see the DAB EPG XML specification for more information)

7.5. APPLICATION INFORMATION LIST (AIL)

The Application Information List (AIL) document is intended to detail application information for a service provider.

7.5.1. Contents

It holds a definition of applications, including any relevant information about the single applications, such as:

- nameGroup (shortName, mediumName, longName)
- MediaDescription (short/long description, multimedia type e.g. logos)
- ApplicationID
- ApplicationPriority
- Application control codes (e.g. autostart)
- Bearer information (where the application can be accessed)

Example:

```xml
<applicationInformation creationTime="2014-06-12T14:00:00+02:00" originator="IRT Institut fuer Rundfunktechnik(c) 2014" xml:lang="en">  
<application control="autostart" applicationID="1" applicationPriority="20">  
<applicationScope id="de0.d350.d001.0"/>
<shortName>IRT Radio App 1</shortName>
<mediumName>1st IRT Radio application full name</mediumName>
<mediaDescription>
<shortDescription>This is demo application for the brand new RadioWEB user application. </shortDescription>
</mediaDescription>
<mediaDescription>
<multimedia url="http://logos.irt.de/logos/irt1/320x240.png" mime="image/png" height="240" width="320"/>
```

7 https://www.etsi.org/deliver/etsi_ts/102800_102899/102818/03.01.01_60/ts_102818v030101p.pdf
D3.3: HRADIO mobile and HTML client API implementations

Possible control values:

- **autostart** = Receiver shall start the signaled application immediately after the signaling has been discovered.
- **enabled** = Receiver shall start the signaled application on user request only
- **disabled** = Receiver shall not start the application at all. But shall keep running instances alive until they are quit by the user.

The bearer element points to the application location. In general, this will be a http URL, where the device can access the application contents. However, for a possible inclusion of applications into the broadcast multiplex, locations pointing to MOT carousels shall be possible. The cost field gives the broadcaster the possibility to provide preferences to the RadioWEB device for heavy traffic applications.

7.5.2. Location

The document is acquired using a URL constructed using the format:

http://<host>[:<port>/radiodns/web/<broadcast parameters>/AIL.xml

Where host and port are populated by the host and port values obtained from the SRV record lookup for the RadioWEB application.

AIL documents are stored as a single file.
The broadcast parameters are based on the bearer of the service being consumed and specified in the following subsections, specific to each bearer.

**VHF/FM**

The broadcast parameters value for a VHF/FM service PI request URI is constructed as follows:

```
fm/(<gcc>|<country>)/<pi>/<frequency>
```

**DAB/DAB+ Digital Radio**

The broadcast parameters value for a DAB/DAB+ Digital Radio service PI request URI is constructed as follows:

```
dab/<gcc>/<eid>/<sid>/<scids>[(<appty-uatype>|<pa>)]
```

**IP-delivered Audio Service**

The broadcast parameters value for a request URI when receiving IP-delivered audio is constructed as follows:

```
id/<fqdn>/<sid>
```

### 7.6. APPLICATION LIFE-CYCLE

The present approach defines a single application model. This means that on a RadioWEB enabled device there can be only one single application active. Available RadioWEB applications are signaled via the AIL document retrieved over the existing RadioDNS mechanisms (see section above).

#### 7.6.1. Application Identification

RadioWEB applications are identified uniquely under the namespace of the FQDN, where the AIL.xml file has been retrieved. All given ApplicationIDs must be unique or refer to the same application.

#### 7.6.2. Application life-cycle diagram

Two main types of Application control codes are defined:

- autostart
- enabled
Autostart applications shall be started immediately after the application signaling has been detected.

Enabled application shall not be started automatically by the device. However, these applications may be listed as available via a UI of the RadioWEB device. This type of application can either be started by the end user manually or by other RadioWEB applications.

The following Figure shows the definition of the life cycle in a flowchart.
7.6.3. API

Regarding the JavaScript interface of RadioWEB HTML/JavaScript based interactive content, the originally proposed API now seems too complicated and too feature rich for the envisaged HRADIO use cases. Therefore, in order to speed up the development process, a simplified approach fitting to the requirements of broadcasters has been implemented for the 1st release.

7.6.4. General API

The main object when loading the RadioWEB-App is the `radioWeb` object.

A developer shall register at first a callback to get notified when the `radioWeb` object is ready with:

```javascript
window.addEventListener('radioWebReady', callBackFunction);
```

The `radioWeb` object has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Possible values/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>String</td>
<td>STATUS_RADIO_RUNNING, STATUS_RADIO_SUSPENDED</td>
</tr>
<tr>
<td>volume</td>
<td>Int</td>
<td>percent value 0-100</td>
</tr>
<tr>
<td>active services</td>
<td>Service object array</td>
<td>array of currently active services</td>
</tr>
<tr>
<td>Services</td>
<td>Service object array</td>
<td>array of all available services</td>
</tr>
</tbody>
</table>

![Table 4: RadioWeb object properties](image)

The `radioWeb` object has the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Possible values/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addEventListener(type, callback)</td>
<td>stype is one of 'serviceStarted', 'serviceStopped', 'servicesUpdated', the callback function shall look like function</td>
<td></td>
</tr>
</tbody>
</table>

![Table 4: RadioWeb object properties](image)
D3.3: HRADIO mobile and HTML client API implementations

### function to call on eventString
```javascript
cb(startedServiceObject){
}
```

### removeEventListener(type, callback)
- type is one of 'serviceStarted', 'serviceStopped', 'servicesUpdated', the callback function to remove
- the callback function shall look like function `cb(startedServiceObject){
}
```

### setVolume(newVolume)
- newVolume is an integer value between 0 and 100
- Sets the playback volume

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Possible values/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>RADIOSERVICE_TYPE_DAB, RADIOSERVICE_TYPE_IP, RADIOSERVICE_TYPE_FM, RADIOSERVICE_TYPE_SIRIUS, RADIOSERVICE_TYPE_HDRADIO</td>
</tr>
<tr>
<td>serviceLabel</td>
<td>String</td>
<td>The service label.</td>
</tr>
<tr>
<td>timeshifted</td>
<td>Boolean</td>
<td>Indicates if the service is time shifted or not (currently only DAB or EDI services could be time shifted.</td>
</tr>
</tbody>
</table>

**Table 5: RadioWeb object methods**

The service object has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Possible values/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceID</td>
<td>Int</td>
<td>The DAB service ID</td>
</tr>
</tbody>
</table>

**Table 6: Service object properties**

If the service.type is RADIOSERVICE_TYPE_DAB the service object has the following additional properties:
D3.3: HRADIO mobile and HTML client API implementations

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ensembleID</td>
<td>Int</td>
<td>The DAB ensemble ID</td>
</tr>
<tr>
<td>ensembleECC</td>
<td>Int</td>
<td>The extended country code for this ensemble</td>
</tr>
<tr>
<td>ensembleLabel</td>
<td>String</td>
<td>The DAB ensemble label</td>
</tr>
<tr>
<td>ensembleFrequency</td>
<td>Int</td>
<td>The ensemble frequency in kHz</td>
</tr>
<tr>
<td>isProgramme</td>
<td>Bool</td>
<td>Indicates if the service is a programme (audio) or data service.</td>
</tr>
</tbody>
</table>

Table 7: Service object properties for DAB

The service object has the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Possible values/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start()</td>
<td>--</td>
<td>Starts the service</td>
</tr>
<tr>
<td>stop()</td>
<td>--</td>
<td>Stops the service</td>
</tr>
<tr>
<td>addEventListener (type, callback)</td>
<td>type is one of 'sls', 'dls', 'epg', 'state', callback the function to call on event</td>
<td>the callback function shall look like function cb(eventJson)()</td>
</tr>
<tr>
<td>removeEventListener (type, callback)</td>
<td>type is one of 'sls', 'dls', 'epg', 'state', callback the function to call on event</td>
<td>the callback function shall look like function cb(eventJson)()</td>
</tr>
</tbody>
</table>

Table 8: Service object methods

The JSON Objects on service events look like the following examples:

DLS JSON:

```json
{
    "textualType": "METADATA_TEXTUAL_TYPE_DAB_DLS",
    "dls": "Sprechstunde - Hörertelefon 00800 44644464",
    "itemRunning": true,
```
"itemToggled": false,
"dlPlusItems": [
    {
        "contentType": "PROGRAMME_NOW",
        "contentCategory": "PROGRAMME_NOW",
        "contentTypeDescription": "Now",
        "dlPlusText": "Sprechstunde - Hörerfon 00800 44644464"
    }
]
}

SLS JSON:
{
    "visualType": "METADATA_VISUAL_TYPE_DAB_SLS",
    "contentName": "0701.png",
    "slideId": 1,
    "triggerTime": "NOW",
    "mimeType": "image/png",
    "isCategorized": true,
    "categoryId": 7,
    "categoryName": "Kontakt",
    "clickthroughUrl": "http://www.deutschlandfunk.de",
    "alternativeLocationUrl": "",
    "expiryTime": 0,
    "visualData": "binary BASE64 encoded image data"
}

7.6.5. Time shift API

If the RadioWebView is running as a component of an App which has time shift functionalities and the RadioWebView has a reference to the running TimeshiftPlayer, the currently times shifted RadioService has an additional object time shift.

The timeshift object has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Possible values/description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentPosition</td>
<td>int</td>
<td>the current position in the timeshift buffer in seconds</td>
</tr>
<tr>
<td>totalDuration</td>
<td>int</td>
<td>the total duration of the timeshift buffer in seconds since the start of timeshift</td>
</tr>
<tr>
<td>paused</td>
<td>boolean</td>
<td>indicates the current state of the timeshiftplayer. true if the player is paused, false otherwise</td>
</tr>
<tr>
<td>skipItems</td>
<td>Array of objects</td>
<td>array of the available skipItems</td>
</tr>
<tr>
<td>timeShiftToken</td>
<td>string</td>
<td>a unique token string for the server based timeshift (SBT)</td>
</tr>
</tbody>
</table>
### sbtMax

**Type:** int  
**Description:** indicates the maximum SBT buffer in milliseconds

---

The **timeshift** object has the following properties:

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pause(pauseUnpause)</td>
<td>pauseUnpause boolean value. true to pause the player, false to make it play again</td>
<td>pauses / unpauses the timeshift player</td>
</tr>
<tr>
<td>seek(seekMS)</td>
<td>seekMs the position in the timeshift buffer to seek to in milliseconds</td>
<td>seeks the timeshift player to a new position</td>
</tr>
<tr>
<td>skipTo(skipToItem)</td>
<td>skipToItem, a skipItem object to skip to its start position</td>
<td>skips the timeshift player to the beginning of the given skipItem</td>
</tr>
<tr>
<td>addTimeshiftListener(type, callback)</td>
<td>type is one of 'state', 'skipitemadded', 'skipitemremoved', 'progress', 'sbtprogress', 'visual', 'textual', callback the function to call on event</td>
<td>the callback function shall look like function cb(eventJson)()</td>
</tr>
<tr>
<td>removeTimeshiftListener(type, callback)</td>
<td>type is one of 'state', 'skipitemadded', 'skipitemremoved', 'progress', 'sbtprogress', 'visual', 'textual', callback the function to remove from event notifications</td>
<td>removes the previously registered callback function</td>
</tr>
</tbody>
</table>
7.6.6. Server Based Time shift

If a started radioservice has support for server-based time shift (SBT) there may be SkipItems right from the start or added rapidly via the 'skipitemadded' callback.

For the SBT functionality you should register a timeshiftlistener with:

```
addTimeshiftListener(sbtprogress, callback)
```

The registered callback gives a JSON in the following form:

**SBT JSON:**

```
{
  "realTime": 1575376717895,
  "streamTime": 1575373118860,
  "currentPosition": 3600,
  "totalDuration": 7200
}
```

`realTime` is the current real-world POSIX timestamp in milliseconds. `streamTime` is the current POSIX timestamp of the stream in milliseconds. `currentPosition` is the current relative position in the SBT buffer in seconds. `totalDuration` is the total relative SBT buffer in seconds.

Also a 'skipitemremoved' callback should be registered to be notified when skip items are not valid anymore.
8. SERVICE LOOKUP AND SEARCH LIBRARY

For easy access to the search and recommendation platform of the HRADIO system, a library was developed that greatly simplifies access to the REST APIs of the content platform servers for the application developer. The developed library enables the developers to fully concentrate on the content functions in the application. Network specific functions and the error handling in case of an insufficient network connection, e.g. connection aborts, are handled accordingly by the library. The next image is extracted from deliverable D3.2v3.0 and shows the overall architecture of the HRADIO communication platform.

Figure 15: The architecture of the HRADIO Communication Platform

While the HRADIO communication platform provides an extensive set of functionalities for the insertion, enrichment, processing and retrieval of radio specific metadata, the core components for the library described here in this section are the following publicly available REST endpoints (see HRADIO deliverable D3.2):

- **Search REST**: The search REST service can be used for service, service use (station listening trends) and programme search. It follows RESTful service principles, uses JSON as transport format and is pageable. The service is
publicly available and only HTTPS access is enabled. Some methods are secured using the security module.

- **The recommendations REST** service endpoint with which recommendations can be requested for a specific service hash. The recommendation engine is mostly content-based and offers 6 recommenders which can be combined to deliver service recommendations based on different data sources.

- **Statistics REST** is used to send User data (arbitrary user data or radio service usage) to the platform via a REST service interface. The data is then aggregated and stored in the document database and the relational database.

The basic concept of REST APIs is the statelessness of the client and server communication. This, and the easy access through the common HTTP methods GET, POST or PUT, make this API technology so successful. Therefore, the basic task for the described library is the wrapping of the HTTP requests to the Java language used for the Android application development. The library package structure is as follows:

```java
class java.eu.hradio.httprequestwrapper {
    ...annotations/json
    ...dtos
    ...exception
    ...listener
    ...parser
    ...query
    ...service
    ...util
}
```

The entry point for the application developer is the:

- **ServiceSearchClient** interface for the search of Radio services,

- **MetaDataSearchClient** interface for the search in schedule data and the

- **RecommendationClient** interface for requesting service recommendations.
The **ServiceSearchClient** provides the following API:

```java
void asyncGetAllServices(OnSearchResultListener<ServiceList> listener,
OnErrorListener errorListener);

void asyncGetAllEDIServices(OnSearchResultListener<ServiceList> listener,
OnErrorListener errorListener);

void asyncServiceSearch(Map<String, String> params,
OnSearchResultListener<ServiceList> listener, OnErrorListener errorListener);

void asyncServiceSearchByExactName(String name, OnSearchResultListener<ServiceList>
listener, OnErrorListener errorListener);

void asyncServiceSearchByName(String name, OnSearchResultListener<ServiceList>
listener, OnErrorListener errorListener);

void asyncServiceSearch(Map<String, String> params,
OnSearchResultListener<ServiceList> listener, OnErrorListener errorListener,
boolean returnAll);

void asyncServiceSearchByExactName(String name, OnSearchResultListener<ServiceList>
listener, OnErrorListener errorListener, boolean returnAll);

void asyncServiceSearchByName(String name, OnSearchResultListener<ServiceList>
listener, OnErrorListener errorListener, boolean returnAll);
```

The **MetaDataSearchClient** provides the following API:

```java
void asyncProgrammeSearch(Map<String, String> query,
OnSearchResultListener<ProgrammeList> listener, OnErrorListener errorListener,
boolean returnAll);

void asyncProgrammeSearchForServiceHash(String hash,
OnSearchResultListener<ProgrammeList> listener, OnErrorListener errorListener,
boolean returnAll);
```

And finally, for the **RecommendationClient** the following API was defined and implemented:

```java
void asyncRecommendationRequestName(List<WeightedRecommender> recommenders,
String name, Context context, OnSearchResultListener<ServiceList> listener,
OnErrorListener errorListener, boolean filter, double threshold);

void asyncRecommendationRequestName(List<WeightedRecommender> recommenders,
String name, Context context, OnSearchResultListener<ServiceList> listener,
OnErrorListener errorListener, boolean filter);

void asyncRecommendationRequestName(List<WeightedRecommender> recommenders,
String name, Context context, OnSearchResultListener<ServiceList> listener,
OnErrorListener errorListener);

void asyncRecommendationRequest(List<WeightedRecommender> recommenders,
StandaloneService service, Context context, OnSearchResultListener<ServiceList>
listener, OnErrorListener errorListener, boolean filter, double threshold);

void asyncRecommendationRequest(List<WeightedRecommender> recommenders,
StandaloneService service, Context context, OnSearchResultListener<ServiceList>
listener, OnErrorListener errorListener, boolean filter);
The following code snippet from the HRADIO platform application is showing the necessary steps a developer has to perform for a service name-based search in the HRADIO metadata platform.

```java
/**
 * Browse for service matching a given query
 * @param params - the query
 * @param listener - result callback
 * @param errorListener - error callback
 */
public void serviceSearch(Map<String, String> params,
OnSearchResultListener<ServiceList> listener, OnErrorListener errorListener) {
    if (params.containsKey(ESQuery.Keys.NAME)) {
    }
    if (params.size() == 1 && params.containsKey(ESQuery.Keys.NAME)){
        serviceSearchName(params.get(ESQuery.Keys.NAME), listener, errorListener);
    } else {
        service.asyncServiceSearch(params, listener, errorListener);
    }
}
```

Request to the program data or for recommendations follow the same pattern. For a set of comprehensive examples and usage on the whole library, please refer to the HRADIO platform app, which is released as open source.
9. REPOSITORIES AND DEVELOPMENT SETUP FOR ANDROID

This section describes technical details for developers about the availability, usage and feedback regarding this initial release of the HRADIO client libraries.

Figure 16: Development workflow

The drawing above shows the various tools and platforms that allow developers to use the existing libraries and examples in their own applications as easily as possible.

9.1. REPOSITORIES

Source code for all developments in HRADIO is provided on a GitLab instance available on the network:

https://gitlab-ext.irt.de

https://jfrog.com

https://developer.android.com/studio/

Contains all the HRADIO Sources for Applications and libraries.

Serves binary Java (Android) libraries for the pilot developments.

Development IDE

Get dependencies/Push updates

Pull/Push application and libraries source code

Distribute apps (closed or open)

GitLab is a well-known platform for distributed software development projects. Amongst many other powerful features, it provides central access to the source repositories, an issue tracking and WIKI Pages for simple documentation purposes. For the HRADIO project a closed group HRADIO with members from all the project partners has been set up. This group will not only be the umbrella for the client library development tasks, but also host the developments for Metadata and Communication platform, for the pilots and the Lab-Playout systems.
9.2. **IDE ANDROID STUDIO**

For development software for the Android platform, Android Studio will be used.

https://developer.android.com/studio/

Android Studio is the first stop for development on this platform and provides not only source code editing, compilation and debugging tools. Additionally, Android Studio supports the developer with advanced support and tools for:
9.3. LIBRARY MANAGEMENT

To facilitate the setup of a complete development environment for HRADIO application developers, in addition to GitLab and Android Studio a system that manages binary (fully compiled) dependencies has been set up. The JFrog Artifactory automatically resolves the required dependencies for the developer, loads and makes them automatically available to the build system.
Like on the GitLab platform all necessary HRADIO project members have accounts on the system to access or manage the different artifacts for the different products to build. This setup makes it extremely easy to switch between different library version during development, e.g. stable and beta releases.

![JFrog Artifactory](image)

**Figure 19: Artifactory for HRADIO**

### 9.4. SIMPLE DEVELOPER WORKFLOW EXAMPLE

The following section walks through the necessary steps a novice developer has to perform, in order to come to a fully working HRADIO development setup.

1. Create accounts to access GitLab and Artifactory

2. Clone the GitLab Source project

   ```bash
   git clone git@gitlab-ext.irt.de:hradio/TimeshiftPlayerExample.git
   ```
Note: It’s not necessary to also clone the library projects in order to fulfil build dependencies, as this functionality is provided by the Artifactory.

3: In the “Authentication Settings “ of the Artifactory profile generate an encrypted API Key for access.

4: Configure the .gradle properties file with the credentials created in step 3.

5: Open the cloned Android Studio project from step2 and build. During this 1st build process, the project configuration and the access to the Artifactory take care, that:

- the required OMRI API definition and API implementation is loaded and
- that the required TimeshiftPlayer library is loaded.

![Build APK(s)](image)

Figure 20: Successful APK build for "timeshifsampleapp"
10. IOS DEVELOPMENT

From the beginning of iOS application development, Apple allowed developers to use C/C++ libraries in iOS app which could be easily connected with Objective-C code, the generally used programming language for Macs and iOS devices. Due to its origins in NextStep, applications for MacOS as well as for iOS are heavily oriented on the MVC (Model View Controller) paradigm. In 2014, with the release of XCode 6, Apple introduced Swift as new kid on the block of future iOS and Mac development, which has been open sourced in December 2015. Swift and Objective-C code can also be bridged easily so that for the usage of the HRADIO C++ libraries a simple Swift-Objective-C-C++ chain is suitable and leads to the following architecture.

![iOS implementation architecture](image)

Figure 21: iOS implementation architecture

The MVC-Designed Swift GUI part takes care of all UI rendering and app initialization (FirstViewController). Once the source of a DAB-EDI stream is set, the ServiceController passes the source information into the Objective-C bridge code, where no additional processing or logic is applied. The ediinput_objcpp.h defines an EdiInputImpl struct, which delegates the calls directly into the DAB-C++ library.
10.1. IOS BUILD PHASES

The iOS build for an HRADIO application completely takes place in the XCode development tool. A script calling the common CMAKE tools is used to compile the necessary IRT DAB+ sources into object files for later linking. The Swift GUI and Objective-C Bridge build as well as the linking of all dependencies into a running iOS application is then handled by the build system in XCode.

![iOS Build Structure Diagram]

Figure 22: iOS Build structure.
11. CONCLUSIONS

This deliverable is the second iteration of three deliverables, documenting the final state of work in T3.3 “Client libraries”.

In the first phase of T3.3 the goal was to quickly enable applications developers to use the OMRI API for their developments. Therefore T3.3 focused on the well-known and widely available Android platform. A fully working Android implementation of OMRI has been delivered. Additionally, a Time Shift player component has been developed and provided to the project partners. In order to enable a quick adoption of this technologies by the developers, an easy to setup but powerful working platform (GIT, Android Studio, Artifactory…) has been set up.

In the 2nd phase of T3.3 focus was to add additional functionality (SPI fetching and parsing) and RadioWEB component which is required for the implementation of the HRADIO pilots. Furthermore, a main step towards a single DAB service layer for all HRADIO platforms (Android, iOS and HTML/JavaScript) has been made with the development of the DAB over IP server solution and the related players for Android and HTML/JavaScript.

In this final 3rd phase of library implementation, the biggest step forward is the implementation of a server-based time shift support for the Android and Web platforms. The second big step forward is the integration of the HRADIO Service and Metadata Search into a library for application development. The former libraries for RadioDNS SPI fetching and parsing have been replaced by a new full featured RadioDNS library covering now the full set of RadioDNS functionalities for application developers. Finally the project also showed the portability of the OMRI Libraries to the iOS platform.
APPENDIX A

The following section lists the formal API definition in Java.

Developers will find the Java skeletons of the org.omri packages on the specifications Git repository hosted on:

https://github.com/ebu

A.1 PACKAGE ORG.OMRI.RADIO

A.1.1 Radio

```java
package org.omri.radio;
import java.util.List;
import org.omri.radio.impl.RadioImpl;
import org.omri.radioservice.RadioService;
import org.omri.tuner.Tuner;
import org.omri.tuner.TunerListener;
import org.omri.tuner.TunerType;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * The main Radio class
 * The implementer has to implement {@link RadioImpl}implementation
 * @author FabianSattler, IRT GmbH
 */
public abstract class Radio {
    /** the singleton instance of Radio */
    private static Radio INSTANCE = new RadioImpl();
    /**
     * Returns the {@link Radio} instance or null if not implemented
     * @return the {@link Radio} instance or null if not implemented
     */
    public static Radio getInstance() {
        return INSTANCE;
    }
    /**
     * Initializes the {@link Radio} instance
     * @return the {@link RadioErrorCode} indicating the success of init.
     */
    public abstract RadioErrorCode initialize();
    /**
     * Initializes the {@link Radio} instance with an Java Object. Intention
     * of the appContext parameter is, that applications get the opportunity to
     * pass necessary platform specific objects into the OMRI implementation
     * i.e. Android {@link Context}
     * @param appContext the AppContext
     */
    public abstract Radio ErrorCode initialize();
}
```
public abstract RadioErrorCode initialize(Object appContext);

/**
 * Suspends the [Radio] and with it all [Tuner]s
 * @return {RadioErrorCode} indicating the success of the suspend.
 */
public abstract RadioErrorCode suspend();

/**
 * Resumes the [Radio] to the previous state before it was suspended.
 * @return {RadioErrorCode} indicating the success of the resume
 */
public abstract RadioErrorCode resume();

/**
 * Indicates the current status of the [Radio]
 * @return the current [RadioStatus]
 */
public abstract RadioStatus getRadioStatus();

/**
 * Deinitializes the Radio and all [Tuner]s
 */
public abstract void deInitialize();

/**
 * Returns the available [Tuner]s or an empty list
 * @return the available [Tuner]s or an empty list
 */
public abstract List<Tuner> getAvailableTuners();

/**
 * Returns the available [Tuner]s for a specific [TunerType] or an empty list
 * @return the available [Tuner]s for a specific [TunerType] or an empty list
 */
public abstract List<Tuner> getAvailableTuners(TunerType tunerType);

/**
 * Retrieve the currently known [RadioService]s of this [Radio] device
 * The method here is for the convenience of the application developer.
 * @return list of [RadioService]s or an empty list
 */
public abstract List<RadioService> getRadioServices();

/**
 * Start a [RadioService] on an available tuner
 * The method here is for the convenience of the application developer.
 * @param radioService the [RadioService] to start
 */
public abstract void startRadioService(RadioService radioService);

/**
 * Scans using all tuners and builds the combined service list.
 * The method here is for the convenience of the application developer.
 * In the case the Radio exposes more than one [Tuner] instances, it's recommended
 * to use the dedicated method calls in the [Tuner] objects.
 */
public abstract void startRadioServiceScan();

/**
 * Stops the possible running service scan on all available tuners.
 * The method here is for the convenience of the application developer.
 * If the application developer wants to perform service scans in the background
 * In the case the Radio exposes more than one [Tuner] instances, it's recommended
 * to use the dedicated method calls in the [Tuner] objects.
 */
public abstract void stopRadioServiceScan();
A.1.2 RadioErrorCode

```java
package org.omri.radio;

/**< *
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Error codes for the \{@link Radio\}
* @author FabianSattler, IRT GmbH */

public enum RadioErrorCode {
    ERROR_INIT_OK(0, "No Error"),
    ERROR_INIT_NOT_OKAY(1, "Init error"),
    ERROR_INIT_FATAL_ERROR(2, "Fatal error"),
    ERROR_SUSPEND_OK(3, "Radio suspended"),
    ERROR_SUSPEND_FAILED(4, "Radio could not be suspended"),
    ERROR_RESUME_OK(5, "Radio resumed"),
    ERROR_RESUME_FAILED(6, "Radio could not be resumed");

    private final int errorCode;
    private final String errorDescription;

    private RadioErrorCode(int errorCode, String errorDescription) {
        this.errorCode = errorCode;
        this.errorDescription = errorDescription;
    }

    public int getErrorCode() {
        return this.errorCode;
    }

    public String getErrorCodeDescription() {
        return this.errorDescription;
    }
}
```

A.1.3 RadioListener

```java
package org.omri.radio;
import org.omri.tuner.TunerListener;
/**< *
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* The basic \{@link Radio\} listener interface
* @author FabianSattler, IRT GmbH */

public interface RadioListener {
}
```
A.1.4 RadioStatus

```java
package org.omri.radio;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Status codes for {@link Radio}
 * @author FabianSattler, IRT GmbH
 */
public enum RadioStatus {
    STATUS_RADIO_UNINITIALIZED(0, "Radio is uninitialized"),
    STATUS_RADIO_RUNNING(1, "Radio is running"),
    STATUS_RADIO_SUSPENDED(2, "Radio is suspended"),
    ...

    private final int statusCode;
    private final String statusDescription;

    private RadioStatus(int statusCode, String statusDescription) {
        this.statusCode = statusCode;
        this.statusDescription = statusDescription;
    }

    public int getStatusCode() {
        return this.statusCode;
    }

    public String getStatusDescription() {
        return this.statusDescription;
    }
}
```

A.2 PACKAGE ORG.OMRI.RADIOSERVICE

A.2.1 RadioService

```java
package org.omri.radioservice;
import java.util.List;
import org.omri.radioservice.metadata.Group;
import org.omri.radioservice.metadata.Location;
import org.omri.radioservice.metadata.TermId;
import org.omri.radioservice.metadata.Visual;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Abstract base class for a radio service
 * @author FabianSattler, IRT GmbH
 */
```
public interface RadioService {
    /*
     * Indicates the type of this RadioService
     * @return the {@link RadioServiceType} of this RadioService
     */
    public RadioServiceType getRadioServiceType();
    /*
     * Returns the label of this {@link RadioService}
     * @return the label of this {@link RadioService}
     */
    public String getServiceLabel();
    /*
     * Returns the short description of this {@link RadioService} as {@link String}
     * @return the short description of this {@link RadioService} as {@link String}
     */
    public String getShortDescription();
    /*
     * Returns the long description of this {@link RadioService} as {@link String}
     * @return The long description of this {@link RadioService} as {@link String}
     */
    public String getLongDescription();
    /*
     * Returns the available {@link Visual}s for this {@link RadioService} or an empty list
     * @return the available {@link Visual}s for this {@link RadioService} or an empty list
     */
    public List<Visual> getLogos();
    /*
     * Returns the available {@link TermId}s for this {@link RadioService} or an empty list
     * @return the available {@link TermId}s for this {@link RadioService} or an empty list
     */
    public List<TermId> getGenres();
    /*
     * Returns the available Links for this {@link RadioService} or an empty list
     * @return the available Links for this {@link RadioService} or an empty list
     */
    public List<String> getLinks();
    /*
     * Returns the available {@link Location}s for this {@link RadioService} or an empty list
     * @return the available {@link Location}s for this {@link RadioService} or an empty list
     */
    public List<Location> getLocations();
    /*
     * Returns the available keywords for this {@link RadioService} or an empty list
     * @return the available keywords for this {@link RadioService} or an empty list
     */
    public List<String> getKeywords();
    /*
     * Returns the available {@link Group}s for this {@link RadioService} or an empty list
     * @return the available {@link Group}s for this {@link RadioService} or an empty list
     */
    public List<Group> getMemberships();
    /*
     * Subscribe a {@link RadioServiceListener} to receive updates from this {@link RadioService}
     * @param radioServiceListener the {@link RadioServiceListener} to subscribe
     */
    public void subscribe(RadioServiceListener radioServiceListener);
    /*
     * Unsubscribe a {@link RadioServiceListener} from this {@link RadioService}
     * @param radioServiceListener the {@link RadioServiceListener} to unsubscribe
     */
    public void unsubscribe(RadioServiceListener radioServiceListener);
}

A.2.2 RadioServiceAudiodataListener

package org.omri.radioservice;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * *
 */

/* you may not use this file except in compliance with the License.
*/
public interface RadioServiceAudiodataListener extends RadioServiceListener {
    /**
     * PCM audio data interface
     *
     * @param pcmData the pcm data, encoded as interleaved signed 16 bit little endian PCM
     * @param numChannels the number of audio channels
     * @param samplingRate the sampling rate
     */
    public void pcmAudioData(byte[] pcmData, int numChannels, int samplingRate);
}

A.2.3 RadioServiceDab

package org.omri.radioservice;
import java.util.List;

/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * Abstract class for a DAB {@link RadioService}
 * @author FabianSattler, IRT GmbH
 */
public interface RadioServiceDab extends RadioService {
    /**
     * Returns the extended country code (ECC) for the DAB Ensemble this {@link RadioServiceDab} belongs to
     */
    public String getEnsembleEcc();

    /**
     * Returns the DAB Ensemble ID, this {@link RadioServiceDab} belongs to, as hex-string
     */
    public String getEnsembleId();

    /**
     * Returns the label of the DAB Ensemble, this {@link RadioServiceDab} belongs to
     */
    public String getEnsembleLabel();

    /**
     * Returns the frequency in kHz of the DAB Ensemble, this {@link RadioServiceDab} belongs to
     */
    public String getEnsembleFrequency();
public long getEnsembleFrequency();

/**
 * Returns the short label of this {@link RadioServiceDab}
 */
public String getShortLabel();

/**
 * Returns the service id as hex-string
 * @return the service id as hex-string
 */
public String getServiceId();

/**
 * Returns the service extended country code
 * @return the service extended country code
 */
public String getServiceEcc();

/**
 * Indicates if this {@link RadioServiceDab} is a DAB programme or a DAB data service
 * @return indication for programme or data service
 */
public boolean isProgrammeService();

/**
 * Returns a list with the {@link RadioServiceDabComponent}s associated with this {@link RadioServiceDab}
 * @return a list with the {@link RadioServiceDabComponent}s associated with this {@link RadioServiceDab}
 */
public List<RadioServiceDabComponent> getServiceComponents();
public int getScChannelId();

/**
 * Indicates if the DG flag is set
 * @return indication for DG
 */
public boolean isScDgFlagSet();

/**
 * Returns the {@link RadioServiceDabComponentID}
 * @return the {@link RadioServiceDabComponentID}
 */
public int getScId();

/**
 * Returns the label for this {@link RadioServiceDabComponent}
 * @return the label for this {@link RadioServiceDabComponent}
 */
public String getScLabel();

/**
 * Returns the packet address
 * @return the packet address
 */
public int getScPacketAddress();

/**
 * Indicates if this {@link RadioServiceDabComponent} is the primary component of this
 * {@link RadioServiceDab}
 * @return if this {@link RadioServiceDabComponent} is the primary component of this
 * {@link RadioServiceDab}
 */
public boolean isScPrimary();

/**
 * Returns the service component ID
 * @return the service component ID
 */
public int getScServiceComponentId();

/**
 * Returns the Transport Mode (TM) ID
 * @return the Transport Mode (TM) ID
 */
public int getTmId();

/**
 * Returns the service component type
 * @return the service component type
 */
public int getScType();

/**
 * Returns a list with {@link RadioServiceDabUserApplication}s for this SC
 * @return a list with {@link RadioServiceDabUserApplication}s for this SC
 */
public List<RadioServiceDabUserApplication> getScUserApplications();

/**
 * Subscribe a {@link RadioServiceDabComponentListener} to receive updates from this
 * {@link RadioServiceDabComponent}
 * @param dabComponentListener the {@link RadioServiceDabComponentListener} to subscribe
 */
public void subscribe(RadioServiceDabComponentListener dabComponentListener);

/**
 * Unsubscribe a {@link RadioServiceDabComponentListener}
 * @param dabComponentListener the {@link RadioServiceDabComponentListener} to unsubscribe
 */
public void unsubscribe(RadioServiceDabComponentListener dabComponentListener);
A.2.5 RadioServiceDabComponentListener

```java
package org.omri.radioservice;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Interface to receive raw data from a RadioServiceDabComponent (e.g. Journaline data)
 * @author FabianSattler, IRT GmbH
 */
public interface RadioServiceDabComponentListener {
    /**
     * Called when new data from a specific DAB Service Component was received
     * @param serviceComponentChannelId the Service component id
     * @param scData raw data from the service component
     */
    public void newServiceComponentData(int serviceComponentChannelId, byte[] scData);
}
```

A.2.6 RadioServiceDabUserApplication

```java
package org.omri.radioservice;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Abstract class containing informations about one user application for a RadioServiceDabComponent
 * @author FabianSattler, IRT GmbH
 */
public interface RadioServiceDabUserApplication {
    /**
     * Returns the user application label
     * @return the user application label
     */
    public String getUappLabel();

    /**
     * Returns the user application type as numerical value
     * @return the user application type as numerical value
     */
    public int getUappType();

    /**
     * Returns the user application type as hex-string
     * @return the user application type as hex-string
     */
    public String getUappTypeString();
}
```
A.2.7 RadioServiceFm

```java
package org.omri.radioservice;

/**< *
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * 
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * 
 * http://www.apache.org/licenses/LICENSE-2.0
 * 
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * */

public interface RadioServiceFm extends RadioService {
    /** *
     * Returns the frequency in kHz for this RadioService (e.g. 99300000 for a
     * frequency of 99.3 MHz)
     * 
     * @return the frequency in kHz
     */
    public int getFrequency();

    /** *
     * Returns the RDS PI (ProgrammeIdentification) code for this Service if available (e.g. 'D412'), or an empty String
     * 
     * @return the RDS PI code as a hexadecimal String representation
     */
    public String getRdsPiCode();

    /** *
     * Returns a RadioServiceFmPty for this RadioService if available, or null
     * 
     * @return a RadioServiceFmPty for this RadioService
     */
    public RadioServiceFmPty getRdsPty();
}
```

A.2.8 RadioServiceFmPty

```java
package org.omri.radioservice;

/**< *
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * 
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * 
 * http://www.apache.org/licenses/LICENSE-2.0
 * 
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * 
 * Abstract class representing a RDS PTY (ProgrammeType)
 * 
 * @author FabianSattler, IRT GmbH
 */

public interface RadioServiceFmPty {
    /** *
     * Returns the PTY code
     * 
     * @return the PTY code
     */
    public int getPtyCode();
}
```
D3.3: HRADIO mobile and HTML client API implementations

public String getPtyDescription();

A.2.9 RadioServiceIp

package org.omri.radioservice;
import java.util.List;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 */
public interface RadioServiceIp extends RadioService {
    /**
     * The list of available @link RadioServiceIpStream}s for this IP delivered @link RadioService
     * @return a list of available @link RadioServiceIpStream}s for this @link RadioService
     */
    public List<RadioServiceIpStream> getIpStreams();
}

A.2.10 RadioServiceIpStream

package org.omri.radioservice;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 */
public interface RadioServiceIpStream {
    /**
     * Returns the URL of this stream (e.g. 'http://somewebstream:1337/genre')
     * @return the URL of this stream
     */
    public String getUrl();
    /**
     * Returns the bitrate in kbit/s for this stream
     * @return the bitrate in kBIT/s
     */
    public int getBitrate();
}
D3.3: HRADIO mobile and HTML client API implementations

A.2.11 RadioServiceListener

```java
package org.omri.radioservice;
import org.omri.radio.RadioListener;
import org.omri.radioservice.metadata.ProgrammeServiceMetadataListener;
import org.omri.radioservice.metadata.TextualMetadataListener;
import org.omri.radioservice.metadata.VisualMetadataListener;

public interface RadioServiceListener extends RadioListener {
}
```

A.2.12 RadioServiceMimeType

```java
package org.omri.radioservice;

public enum RadioServiceMimeType {
    UNKNOWN("mime/unknown"),
    /* MIME type not known */
}
/** MIME type for MPEG 1 Layer 1, 2, 3 **/ AUDIO_MPEG("audio/mpeg"),
/** MIME type for OGG Vorbis audio **/ AUDIO_OGG_VORBIS("audio/ogg"),
/** MIME Type for Free Lossless Audio Codec **/ AUDIO_FLAC("audio/flac"),
/** MIME type ADTS AAC **/ AUDIO_AAC("audio/aacp");

private final String contentTypeString;

private RadioServiceMimeType(String contentTypeString) {
    this.contentTypeString = contentTypeString;
}

public String getMimeTypeString() {
    return this.contentTypeString;
}

A.2.13 RadioServiceType

package org.omri.radioservice;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * @link RadioService) type definitions
 * @authorFabianSattler, IRT GmbH
 */
public enum RadioServiceType {
    /** RadioService type DAB **/ RADIOSERVICE_TYPE_DAB,
    /** RadioService type IP **/ RADIOSERVICE_TYPE_IP,
    /** RadioService type FM **/ RADIOSERVICE_TYPE_FM,
    /** RadioService type SiriusXM **/ RADIOSERVICE_TYPE_SIRIUS,
    /** RadioService type iBiquity HD Radio **/ RADIOSERVICE_TYPE_HDRADIO;
}

A.3 PACKAGE ORG.OMRI.RADIOSERVIICE_Metadata

A.3.1 Group

package org.omri.radioservicemetadata;
import java.util.List;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,  
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
* See the License for the specific language governing permissions and  
* limitations under the License.  
* Abstract base class for a radio service  
* @author Fabian Sattler, IRT GmbH  
* @author Erk, IRT GmbH  
*/
public interface Group {
    /**
     * Returns the CRID of this @link Group
     * @return the CRID of this @link Group
    */
    public String getCRID();

    /**
     * Returns the short name of this @link Group as @link String
     * @return The short name of this @link Group as @link String
    */
    public String getShortName();

    /**
     * Returns the medium name of this @link Group as @link String
     * @return The medium name of this @link Group as @link String
    */
    public String getMediumName();

    /**
     * Returns the long name of this @link Group as @link String
     * @return The long name of this @link Group as @link String
    */
    public String getLongName();

    /**
     * Returns the short description of this @link Group as @link String
     * @return The short description of this @link Group as @link String
    */
    public String getShortDescription();

    /**
     * Returns the medium description of this @link Group as @link String
     * @return The medium description of this @link Group as @link String
    */
    public String getMediumDescription();

    /**
     * Returns the available @link Visuals for this @link Group or an empty list
     * @return the available @link Visuals for this @link Group or an empty list
    */
    public List<Visual> getLogos();

    /**
     * Returns the available @link TermId's for this @link Group or an empty list
     * @return the available @link TermId's for this @link Group or an empty list
    */
    public List<TermId> getGenres();

    /**
     * Returns the available links for this @link Group or an empty list
     * @return the available links for this @link Group or an empty list
    */
    public List<String> getLinks();

    /**
     * Returns the available keywords for this @link Group or an empty list
     * @return the available keywords for this @link Group or an empty list
    */
    public List<String> getKeywords();
}
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
public interface Location {
}

A.3.3 ProgrammeInformation

package org.omri.radioservice.metadata;
/**
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Abstract base class for programme information (e.g. DAB EPG, RadioDNS EPG, ...)
* @author Fabian Sattler, IRT GmbH
*/
public interface ProgrammeInformation {
    /**
     * Returns the type of this ProgrammeInformation metadata
     * @return the type of this ProgrammeInformation metadata
     */
    public ProgrammeInformationType getType();
}

A.3.4 ProgrammeInformationType

package org.omri.radioservice.metadata;
/**
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Programmemetadata type definitions
* @author Erk, IRT GmbH
*/
public enum ProgrammeInformationType {
    /** Programmemetadata received via RadioDNS/DAB SPI **/
**A.3.5 ProgrammeServiceMetadataListener**

```java
package org.omri.radioservice.metadata;
import org.omri.radioservice.RadioServiceListener;
/**
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Interface to receive {@link ProgrammeInformation} and ServiceInformation
* @author FabianSattler, IRT GmbH
*/
public interface ProgrammeServiceMetadataListener extends RadioServiceListener {
    /**
     * New @link ProgrammeInformation was received
     * @param programmeInformation the @link ProgrammeInformation received
     */
    public void newProgrammeInformation(ProgrammeInformation programmeInformation);

    /**
     * New @link ServiceInformation was received
     * @param serviceInformation the @link ServiceInformation received
     */
    public void newServiceInformation(ServiceInformation serviceInformation);
}
```

**A.3.6 ServiceInformation**

```java
package org.omri.radioservice.metadata;
/**
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Abstract base class for (Extended) Service Information (e.g. DAB EPG SI, RadioDNS RadopEPG XSI)
* @author FabianSattler, IRT GmbH
*/
public interface ServiceInformation {
}
```

**A.3.7 SpiProgrammeInformation**

```java
package org.omri.radioservice.metadata;
import org.w3c.dom.Document;
/**
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
*/
```
D3.3: HRADIO mobile and HTML client API implementations

* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
  * http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.

* Abstract base class for a radio service
* @author Erk, IRT GmbH
*
public interface SpiProgrammeInformation extends ProgrammeInformation {
  /**
   * Returns the SPI as a DOM object
   * @return a SPI as a {@link Document}
   */
  public Document getSpiDocument();
}

A.3.8 TermId

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
public interface TermId {
  /**
   * Returns the href of this termID
   * @return the href of this termID
   */
  public int getHref();
  /**
   * Returns text of this termID.
   * @return text of this termID
   */
  public String getText();
}

A.3.9 Textual

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
public interface Textual {
    /**
     * Returns the type of this textual metadata
     * @return the type of this textual metadata
     */
    public TextualType getType();

    /**
     * Returns the String representation of this textual metadata
     * @return the String representation of this textual metadata
     */
    public String getText();
}

A.3.10 TextualDabDynamicLabel
package org.omri.radioservice.metadata;
import java.util.List;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * Abstract class for a DAB Dynamic Label (Plus) (Textual) metadata
 * @author FabianSattler, IRT GmbH
 */
public interface TextualDabDynamicLabel extends Textual {
    /**
     * Indicates if this TextualDabDynamicLabel has DL+ tags
     * @return indication of DL+ tags
     */
    public boolean hasTags();

    /**
     * Returns the number of tags. Only applicable for DL+. Check 'hasTags()'
     * @return the number of tags
     */
    public int getTagCount();

    /**
     * Returns a list of TextualDabDynamicLabelPlusItem or an empty list
     * @return a list of TextualDabDynamicLabelPlusItem or an empty list
     */
    public List<TextualDabDynamicLabelPlusItem> getDLPlusItems();
}

A.3.11 TextualDabDynamicLabelPlusContentType
package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 */
Dynamic Label Plus Content type definitions

@Author FabianSattler, IRT GmbH

```java
public enum TextualDabDynamicLabelPlusContentType {
    DUMMY(0, "Dummy"),
    ITEM_TITLE(1, "Title"),
    ITEM_ALBUM(2, "Album"),
    ITEM_TRACKNUMBER(3, "Tracknumber"),
    ITEM_ARTIST(4, "Artist"),
    ITEM_COMPOSITION(5, "Composition"),
    ITEM_MOVEMENT(6, "Movement"),
    ITEM_CONDUCTOR(7, "Conductor"),
    ITEM_COMPOSER(8, "Composer"),
    ITEM_BAND(9, "Band"),
    ITEM_COMMENT(10, "Comment"),
    ITEM_GENRE(11, "Genre"),
    INFO_NEWS(12, "News"),
    INFO_NEWS_LOCAL(13, "Local News"),
    INFO_STOCKMARKET(14, "Stockmarket"),
    INFO_SPORT(15, "Sport"),
    INFO_Lottery(16, "Lottery"),
    INFO_HOROSCOPE(17, "Horoscope"),
    INFO_DAILY_DIVERSION(18, "Daily Diversion"),
    INFO_HEALTH(19, "Health"),
    INFO_EVENT(20, "Event"),
    INFO_SCENE(21, "Scene"),
    INFO_CINEMA(22, "Cinema"),
    INFO_TV(23, "TV"),
    INFO_DATE_TIME(24, "Date and Time"),
    INFO_WEATHER(25, "Weather"),
    INFO_ALARM(26, "Alarm"),
    INFO_ADVERTISEMENT(27, "Advertisement"),
    INFO_URL(28, "URL"),
    INFO_OTHER(29, "Other"),
    STATIONNAME_SHORT(30, "Short Stationname"),
    STATIONNAME_LONG(31, "Long Stationname"),
    PROGRAMME_NOW(32, "Now"),
    PROGRAMME_NEXT(33, "Next"),
    PROGRAMME_PART(34, "Part"),
    PROGRAMME_HOST(35, "Host"),
    PROGRAMME_EDITORIAL_STAFF(36, "Editorial Staff"),
    PROGRAMME_FREQUENCY(37, "Frequency"),
    PROGRAMME_HOMEPAGE(38, "Homepage"),
    PROGRAMME_SUBCHANNEL(39, "Subchannel"),
    PHONE_HOTLINE(40, "Hotline"),
    PHONE_STUDIO(41, "Studio Telephone"),
    PHONE_OTHER(42, "Other Telephone"),
    SMS_STUDIO(43, "Studio SMS"),
    SMS_OTHER(44, "SMS"),
    EMAIL_HOTLINE(45, "Hotline Email"),
    EMAIL_STUDIO(46, "Studio Email"),
    EMAIL_OTHER(47, "Email"),
    MMS_OTHER(48, "MMS"),
    CHAT(49, "Chat"),
    CHAT_CENTER(50, "Chat Center"),
    VOTE_QUESTION(51, "Vote Question"),
    VOTE_CENTRE(52, "Vote Centre"),
    RFU_1(53, "RFU1"),
    RFU_2(54, "RFU2"),
    PRIVATE_CLASS_1(55, "Private Data 1"),
    PRIVATE_CLASS_2(56, "Private Data 2"),
    PRIVATE_CLASS_3(57, "Private Data 3"),
    DESCRIPTOR_PLACE(58, "Place"),
    DESCRIPTOR_APPOINTMENT(59, "Appointment"),
    DESCRIPTOR_IDENTIFIER(60, "Identifier"),
    DESCRIPTOR_PURCHASE(61, "Purchase"),
    ...}
```
DESCRIPTOR_GET_DATA(63, "Get Data");

private final int contentTypeId;
private final String contentTypeString;

private TextualDabDynamicLabelPlusContentType(int contentTypeId, String contentTypeString) {
    this.contentType = contentTypeId;
    this.contentTypeString = contentTypeString;
}

public int getContentTypeId() {
    return this.contentTypeId;
}

public String getContentTypeString() {
    return this.contentTypeString;
}

A.3.12 TextualDabDynamicLabelPlusItem

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * Abstract class containing information about one DL+ tag
 * @author FabianSattler, IRT GmbH
 */
public interface TextualDabDynamicLabelPlusItem {
    /**
     * Returns the {@link TextualDabDynamicLabelPlusContentType}
     * @return the {@link TextualDabDynamicLabelPlusContentType}
     */
    public TextualDabDynamicLabelPlusContentType getDynamicLabelPlusContentType();

    /**
     * Returns a textual description of the content type
     * @return textual description of the content type
     */
    public String getDlPlusContentTypeDescription();

    /**
     * Returns the content category of this tag
     * @return the content category of this tag
     */
    public String getDlPlusContentCategory();

    /**
     * Returns the text of this DL+ tag
     * @return the text of this DL+ tag
     */
    public String getDlPlusContentText();
}

A.3.13 TextualFmRdsRadioText

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 */
D3.3: HRADIO mobile and HTML client API implementations

* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*    http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Abstract class for a FM RDS Radiotext{@link Textual}metadata
* @author FabianSattler, IRT GmbH
*/
publicinterface TextualFmRdsRadiotext extends Textual {
}

A.3.14 TextualIpRdnsRadioVis

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *    http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Abstract class for IP delivered RadioDNS RadioVIS {@link Textual}metadata
* @author FabianSattler, IRT GmbH
*/
publicinterface TextualIpRdnsRadioVis extends Textual {
}

A.3.15 TextualMetadataListener

package org.omri.radioservice.metadata;
import org.omri.radioservice.RadioServiceListener;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *    http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Interface to receive {@link Textual}programme associated metadata
* @author FabianSattler, IRT GmbH
*/
publicinterface TextualMetadataListener extends RadioServiceListener {
    /**
     * New {@link Textual}metadata was received
     * @param textualMetadata the {@link Textual}received
     */
    public void newTextualMetadata(Textual textualMetadata);
A.3.16 **TextualType**

```java
package org.omri.radioservice.metadata;

/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Textual metadata type definitions
 *
 * @author FabianSattler, IRT GmbH
 *
 */

public enum TextualType {
    /** Textual metadata received via DAB Dynamic Label service **/
     METADATA_TEXTUAL_TYPE_DAB_DLS,
    /** Textual metadata received via RadioDNS RadioVIS Text service **/
     METADATA_TEXTUAL_TYPE_RADIODNS_RADIOVIS,
    /** Textual metadata received via FM Radiotext **/
     METADATA_TEXTUAL_TYPE_FM_RDS_RADIOTEXT,
    /** Textual metadata received via ID3 parsing **/
     METADATA_TEXTUAL_TYPE_ID3_TEXT,
    /** Textual metadata received via Shoutcast ICY parsing **/
     METADATA_TEXTUAL_TYPE_ICY_TEXT;
}
```

A.3.17 **Visual**

```java
package org.omri.radioservice.metadata;

/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * Abstract base class for visual metadata (e.g. DAB SLS, RadioDNS RadioVIS, ID3 Coverart)
 * @author FabianSattler, IRT GmbH
 */

public interface Visual {
    /**
     * Returns the type (source) of this visual metadata (e.g. DAB Slideshow, RadioDNS RadioVIS, etc.)
     * @return the type of this visual metadata
     */
    public VisualType getVisualType();
    /**
     * Returns the type of this visual metadata
     * @return the type of this visual metadata
     */
    public VisualMimeType getVisualMimeType();
    /**
     * Returns the actual image data
     */
}
```
D3.3: HRADIO mobile and HTML client API implementations

A.3.18 VisualDabSlideShow

```java
package org.omri.radioservice.metadata;
import java.net.URI;
import java.util.Calendar;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * Abstract class for a DAB Slideshow metadata
 * @author FabianSattler, IRT GmbH
 */
public interface VisualDabSlideShow extends Visual {
    /**
     * Indicates if this VisualDabSlideShow has categorization information
     * @return indication for categorization
     */
    public boolean isCategorized();
    /**
     * The content name.
     * @return the content name
     */
    public String getContentName();
    /**
     * The ID of this VisualDabSlideShow
     * @return the ID
     */
    public int getSlideId();
    /**
     * The Triggertime
     * @return a Calendar for the Triggertime or null if the tumgertime is now
     */
    public Calendar getTriggerTime();
    /**
     * The ExpiryTime
     * @return a Calendar for the Expirytime or null
     */
    public Calendar getExpiryTime();
    /**
     * Returns the category description of this VisualDabSlideShow. Only applicable
     * for a DAB Categorized Slideshow (check 'isCategorized()')
     * @return the category text or an empty String if it's not a categorized
     * VisualDabSlideShow
     */
    public String getCategoryText();
    /**
     * Returns the category id of this VisualDabSlideShow. Only applicable for a
     * DAB Categorized Slideshow (check 'isCategorized()')
     * @return the category id or -1 if it's not a categorized VisualDabSlideShow
     */
}
```
public int getCategoryId();
/**
 * Returns the link associated with this {@link VisualDabSlideShow}. Only applicable for
 for a DAB Categorized Slideshow (check 'isCategorized()')
 * @return the link associated with this {@link VisualDabSlideShow} or an empty String
 */
public URI getLink();
/**
 * Returns the click through link associated with this {@link VisualDabSlideShow}.
 * @return the click through link associated with this {@link VisualDabSlideShow} or an
 empty String
 */
public URI getClickThroughUrl();
/**
 * Returns the alternative location link associated with this {@link VisualDabSlideShow}.
 * @return the alternative location link associated with this {@link VisualDabSlideShow} or an
 empty String
 */
public URI getAlternativeLocationURL();
/**
 * The MOT object Content type of this {@link VisualDabSlideShow}
 * @return the Content Type
 */
public int getContentType();
/**
 * The MOT object content subtype of this {@link VisualDabSlideShow}
 * @return the content subtype
 */
public int getContentSubType();

A.3.19 VisualIpRdnsRadioVis

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License").
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * Abstract class for a IP delivered RadioDNS RadioVis {@link Visual} metadata
 * @author FabianSattler, IRT GmbH
 */
public interface VisualIpRdnsRadioVis extends Visual {
/**
 * Returns the trigger time as POSIX time (seconds elapsed since 1.1.1970)
 * @return the trigger time as POSIX time
 */
public long getTriggerTime();
}

A.3.20 VisualMetadataListener

package org.omri.radioservice.metadata;
import org.omri.radioservice.RadioServiceListener;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
D3.3: HRADIO mobile and HTML client API implementations

* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
  * http://www.apache.org/licenses/LICENSE-2.0
* Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.

  * Interface to receive {@link Visual}programme associated metadata
  * @author FabianSattler, IRT GmbH
  */
public interface VisualMetadataListener extends RadioServiceListener {
  /**
   * New {@link Visual}metadata was received
   * @param visualMetadata the {@link Visual}received
   */
  public void newVisualMetadata(Visual visualMetadata);
}

A.3.21 VisualMimeType

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
  * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
 * @author FabianSattler, IRT GmbH
 */
public enum VisualMimeType {
  METADATA_VISUAL_MIMETYPE_UNKNOWN,
  METADATA_VISUAL_MIMETYPE_JPEG,
  METADATA_VISUAL_MIMETYPE_PNG,
  METADATA_VISUAL_MIMETYPE_TIFF,
  METADATA_VISUAL_MIMETYPE_BMP,
  METADATA_VISUAL_MIMETYPE_WEBP,
  METADATA_VISUAL_MIMETYPE_SVG,
  METADATA_VISUAL_MIMETYPE_GIF,
  METADATA_VISUAL_MIMETYPE_ANIMATED_GIF;
}

A.3.22 VisualType

package org.omri.radioservice.metadata;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
  * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
D3.3: HRADIO mobile and HTML client API implementations  

public enum VisualType {
    /** Visual metadata received via DAB Slideshow service **/
    METADATA_VISUAL_TYPE_DAB_SLS,
    /** Visual metadata received via RadioDNS RadioVIS service **/
    METADATA_VISUAL_TYPE_RADIODNS_RADIOVIS,
    /** Visual metadata received via ID3 tag **/
    METADATA_VISUAL_TYPE_ID3_COVERART;
}

A.4 PACKAGE PACKAGE ORG.OMRI.TUNER

A.4.1 Tuner

package org.omri.tuner;
import java.util.List;
import org.omri.radioservice.RadioService;
/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 *
 Licensed under the Apache License, Version 2.0 (the "License);
 you may not use this file except in compliance with the License.
 You may obtain a copy of the License at
 *
 http://www.apache.org/licenses/LICENSE-2.0
 *
 Unless required by applicable law or agreed to in writing, software
 distributed under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
 *
 Abstract Tuner class
 * @author Fabian Sattler, IRT GmbH
 */
public interface Tuner {
    /**
     * Initializes the Tuner. This call is asynchronous. Register a
     * [TunerListener] to receive status updates and error notifications.
     */
    public void initializeTuner();

    /**
     * Suspends the tuner. Keeping its current status to be resumed later.
     */
    public void suspendTuner();

    /**
     * Resume the suspended tuner to the last state.
     */
    public void resumeTuner();

    /**
     * De-Initializes the tuner.
     */
    public void deInitializeTuner();

    /**
     * Indicates the type of tuner
     * @return the [TunerType] of this [Tuner]
     */
    public TunerType getTunerType();

    /**
     * Indicates the current status of the [Tuner] device
     * @return the current [TunerStatus]
     */
public TunerStatus getTunerStatus();
/**
 * Retrieve the currently known {@link RadioService}s of this {@link Tuner}
 * @return list of {@link RadioService}s or an empty list
 */
public List<RadioService> getRadioServices();
/**
 * Start a scan for available {@link RadioService}s
 */
public void startRadioServiceScan();
/**
 * Stops a currently running {@link RadioService} scan
 */
public void stopRadioServiceScan();
/**
 * Start a {@link RadioService} on this tuner
 * @param radioService the {@link RadioService} to start
 */
public void startRadioService(RadioService radioService);
/**
 * Stop the currently running {@link RadioService} on this tuner
 */
public void stopRadioService();
/**
 * Retrieve the currently running {@link RadioService}
 * @return the currently running {@link RadioService} or (null) if no {@link RadioService} is running
 */
public RadioService getCurrentRunningRadioService();
/**
 * Subscribe a {@link TunerListener} to receive updates of this tuner
 * @param tunerListener the {@link TunerListener} to subscribe
 */
public void subscribe(TunerListener tunerListener);
/**
 * Unsubscribe a {@link TunerListener}
 * @param tunerListener the {@link TunerListener} to unsubscribe
 */
public void unsubscribe(TunerListener tunerListener);
D3.3: HRADIO mobile and HTML client API implementations

* @param newStatus the new {@link TunerStatus} */
public void tunerStatusChanged(Tuner tuner, TunerStatus newStatus);  
/*
* Tuner scan progress indicator
* @param percentScanned the percentage finished so far */
public void tunerScanProgress(Tuner tuner, int percentScanned);  
/**
* Tuner has found a service during scanning
* @param foundService the {@link RadioService} which has been found */
public void tunerScanServiceFound(Tuner tuner, RadioService foundService);  
/**
* A {@link RadioService} started
* @param startedRadioService the {@link RadioService} which has started */
public void radioServiceStarted(Tuner tuner, RadioService startedRadioService);  
/**
* A {@link RadioService} stopped
* @param stoppedRadioService the {@link RadioService} which has stopped */
public void radioServiceStopped(Tuner tuner, RadioService stoppedRadioService);  
/**
* Updates on RF reception statistics
* @param rfLock RF tuner frontend gained lock
* @param rssi the Received Signal Strength Indicator (in dbuV[LS1] ???) */
public void tunerReceptionStatistics(Tuner tuner, boolean rfLock, int rssi);  
/**
* Implementation and {@link TunerType} dependent raw data (e.g. in case of a DAB Tuner
raw Fast Information Blocks)
* @param tuner the {@link Tuner} from which the raw data was received
* @param data the raw data */
public void tunerRawData(Tuner tuner, byte[] data);

A.4.3 TunerStatus

package org.omri.tuner;
/**
* Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
* Status codes for the {@link Tuner}
* @author Fabian Sattler, IRT GmbH */
public enum TunerStatus {
/** Tuner is not initialized **/
TUNER_STATUS_NOT_INITIALIZED(0, "Tuner not initialized"),  
/** Tuner is ready **/
TUNER_STATUS_INITIALIZED(1, "Tuner ready"),  
/** Tuner is in an error state **/
TUNER_STATUS_ERROR(2, "Tuner is in an error state"),  
/** Tuner is in suspended state **/
TUNER_STATUS_SUSPENDED(3, "Tuner is suspended"),  
/** Tuner started scan for services **/
TUNER_STATUS_SCANNING(4, "Tuner is scanning for services")
};

private final int statusCode;
private final String statusDescription;
private TunerStatus(int statusCode, String statusDescription) {
    this.statusCode = statusCode;
    this.statusDescription = statusDescription;
}

custom int getStatusCode() {
    return this.statusCode;
}

custom String getStatusDescription() {
    return this.statusDescription;
}

A.4.4 TunerType
package org.omri.tuner;

/**
 * Copyright (C) 2016 Open Mobile Radio Interface (OMRI) Group
 * Licensed under the Apache License, Version 2.0 (the "License";
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
@link Tuner}type definition enum
@authorFabianSattler
public enum TunerType {
    /** DAB Tuner **/
    TUNER_TYPE_DAB,
    /** IP Tuner **/
    TUNER_TYPE_IP,
    /** FM Tuner **/
    TUNER_TYPE_FM,
    /** SiriusXM Tuner **/
    TUNER_TYPE_SIRIUS,
    /** iBiquity HD Radio Tuner **/
    TUNER_TYPE_HDRADIO;
}