ETSI TS 103 632 V1.1.1 (2018-10)

Digital Audio Broadcasting (DAB);
Open Mobile Radio Interface (OMRI);
Application Programming Interface (API)

EBU
OPERATING EUROVISION

DAB
Digital Audio Broadcasting

Reference
DTS/JTC-DAB-91

Keywords
application, audio, broadcasting, digital, interface, radio

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE 1: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

The Eureka Project 147 was established in 1987, with funding from the European Commission, to develop a system for the broadcasting of audio and data to fixed, portable or mobile receivers. Their work resulted in the publication of European Standard, ETSI EN 300 401 [i.1], for DAB (see note 2) which now has worldwide acceptance.

NOTE 2: DAB is a registered trademark owned by one of the Eureka Project 147 partners.

The DAB family of standards is supported by WorldDAB, an organization with members drawn from broadcasting organizations and telecommunication providers together with companies from the professional and consumer electronics industry.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Introduction

The OMRI API is designed to allow developers to gain access to broadcast radio tuners in consumer electronic devices such as smartphones, tablets and/or other devices, and allows the execution of program code often referred to as apps. Device manufacturers, who embed tuner hardware in their devices, should implement the OMRI API and enable the development of individual, rich and sophisticated radio applications. The Java® programming language was selected because of its clear and well known syntax, its implementation of all necessary programming paradigms (e.g. Object Orientated, Generics, Data encapsulation) and its use in the main target platform of possible devices, the Android™ platform.

NOTE: Oracle and Java are registered trademarks of Oracle and/or its affiliates. Android is a trademark of Google LLC.

1 Scope

The present document specifies an Application Programming Interface (API) for the Open Mobile Radio Interface (OMRI) which can be used by application developers to gain access to broadcast radio tuners in consumer electronic devices such as smartphones, tablets and/or other devices, and which allows the execution of program code often referred to as apps.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI EN 300 401: "Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers".

[i.2] ETSI TS 101 499: "Hybrid Digital Radio (DAB, DRM, RadioDNS); SlideShow; User Application Specification".

[i.3] ETSI TS 102 818: "Hybrid Digital Radio (DAB, DRM, RadioDNS); XML Specification for Service and Programme Information (SPI)".

[i.4] ETSI TS 103 270: "RadioDNS Hybrid Radio; Hybrid lookup for radio services".

[i.5] ETSI TS 102 980: "Digital Audio Broadcasting (DAB); Dynamic Label Plus (DL Plus); Application specification".

[i.6] ISO EN 62106: "Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87,5 MHz to 108,0 MHz".

3 Definition of terms and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

app: small software program providing a dedicated function typically found on a smart device

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AAC	Advanced Audio Coding
ADTS	Audio Data Transport Stream
API	Application Programming Interface
CRID	Content Reference IDentifier
DAB	Digital Audio Broadcasting
DL	Dynamic Label
DLS	Dynamic Label Segment
EPG	Electronic Programme Guide
FIC	Fast Information Channel
FM	Frequency Modulation
ICY	I Can Yell
IP	Internet Protocol
MMS	Multimedia Message Service
MOT	Multimedia Object Transfer
MPEG	Moving Picture Experts Group
MSC	Main Service Channel
OMRI	Open Mobile Radio Interface
OS	Operating System
POSIX	Portable Operating System Interface
PTY	Programme TYpe
RDS	Radio Data Service
RF	Radio Frequency
SBR	Spectral Band Replication
SI	Service Information
SLS	SLideShow
SMS	Simple Message Service
SPI	Service and Programme Information
UI	User Interface
URI	Uniform Resource Identifier
URL	Universal Resource Locator
UX	User eXperience
XSI	eXtended Service Information

4 API Architecture

4.1 Introduction

In recent years, the class of so called smart devices has shown an impressive growth in market share. Initially designed primarily as mobile phones (cell phones) and for accessing internet services such as e-mail and the World Wide Web, the versatility of these devices has provided developers with the ability to implement small software programs called apps for a huge amount of different use cases and services. The present document addresses such apps which want to make use of built-in broadcast radio tuners.

4.2 System overview

Normally mobile devices have the capability to connect to IP based networks either over integrated wifi or mobile communications systems. This connectivity, however, only allows for point-to-point connectivity. Access to broadcast services such as DAB or FM for radio is often not possible. Even if mobile devices are equipped with broadcast receivers, app developers do not have access to the hardware tuner resources and therefore they cannot enhance their media centric Apps 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 a RadioDNS [i.4] allow a combination of broadcast and IP based services, known as hybrid radio. In order to utilize this potential, it is important to combine broadcast media with individually accessed on-demand content in a seamless user experience.

Figure 1 depicts a system overview of mobile devices accessing radio and hybrid services using RadioDNS as the "pathfinder" between them.

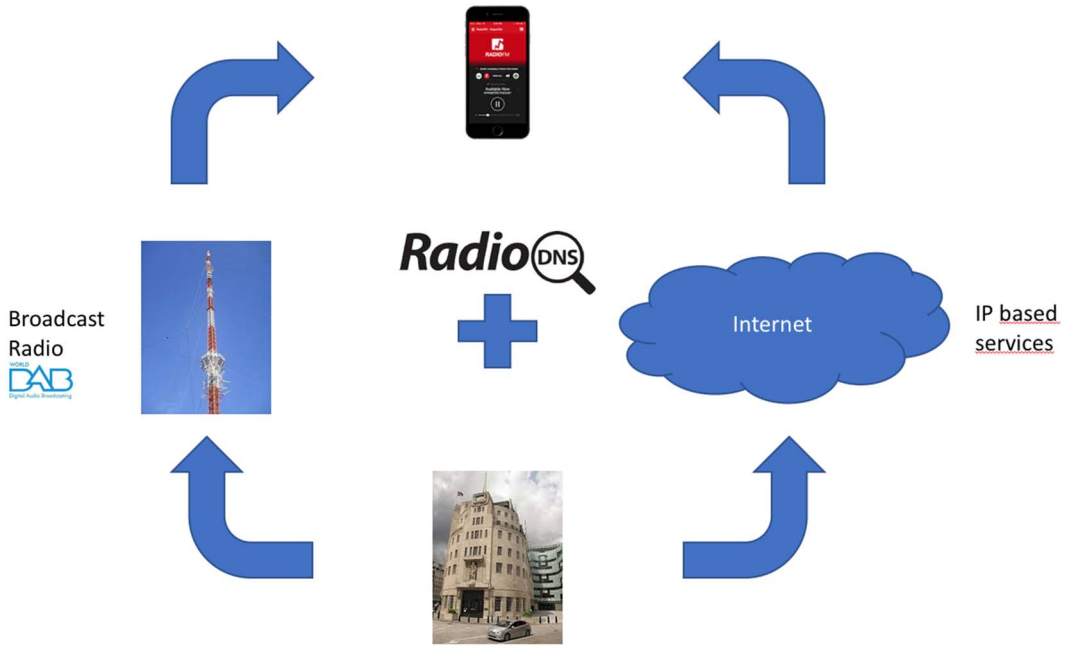

Figure 1: System overview

While App developers understand very well how to access and implement IP based services on target mobile platforms, such easy access has not been possible for broadcast media. The OMRI API closes this gap, by providing a standardized, technology agnostic API for App developers to develop hybrid radio Apps.

The device receives broadcast data via the tuner hardware which includes the audio services as well as additional metadata such as dynamic label [i.1], SlideShow [i.2], Service and Programme Information (SPI) [i.3], and other information. The tuner hardware is usually integrated into the OS of the device via a driver software generally 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 an 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 app has to be adapted to conform to individual tuner solutions. The OMRI API standardizes the access to tuner solutions and enables the development of comprehensive radio apps.

4.3 API overview

4.3.1 OMRI packages

The OMRI API currently consists of three main 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 app 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 app developers to access radioservice 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 define specific listener interface definitions for service data components such as dynamic label [i.1] and DL Plus [i.5], SlideShow [i.2] and programme information (SPI) [i.3].

`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 [i.1], FM-RDS [i.6]). The same package contains the TunerListener interface which is used to deliver highly asynchronous information (e.g. service scan status, signal levels).

4.3.2 OMRI Object diagram

Figure 2 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 in in TUNER_STATUS_INITIALIZED state and returns an instance of RadioServiceDab. Subscribed to the RadioServiceDab instance are two RadioServiceListener subclasses receiving events from the arrival of new Visual and Textual metadata.

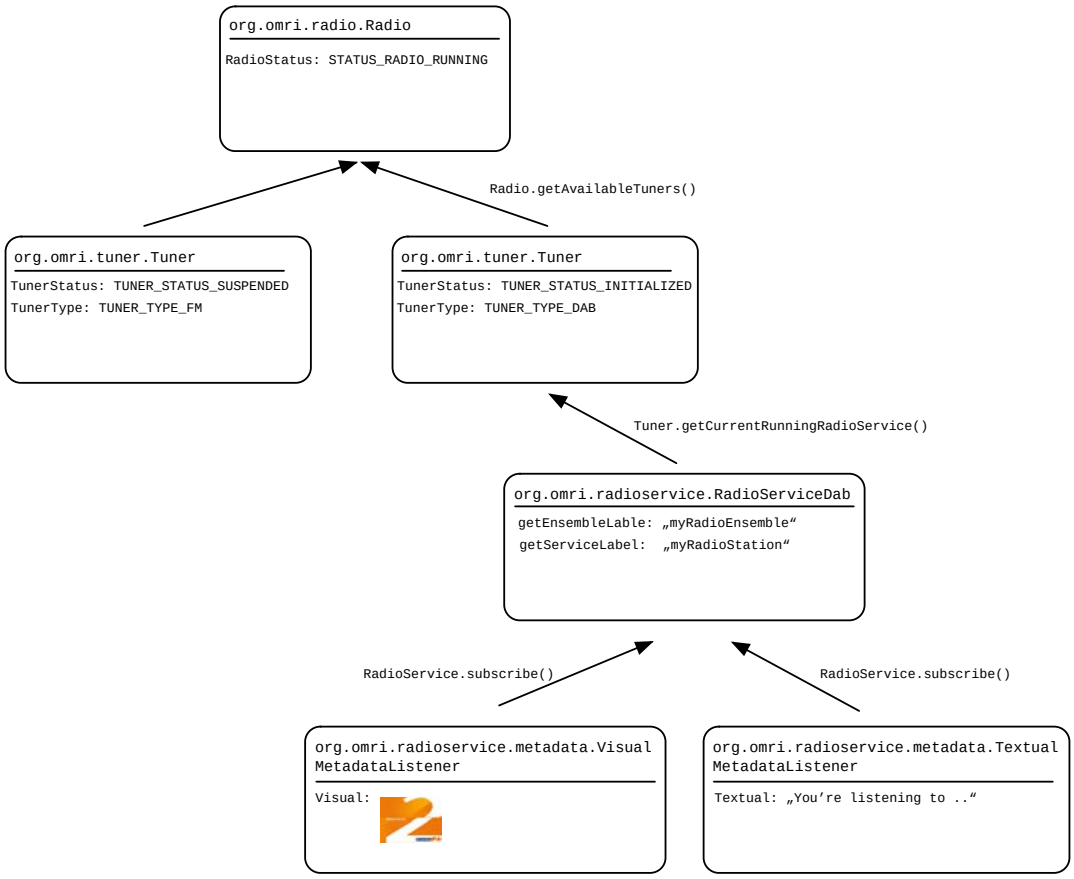

Figure 2: OMRI instance

4.3.3 Radio state model

Figure 3 depicts the state model of the Radio class. The Radio can have three different states:

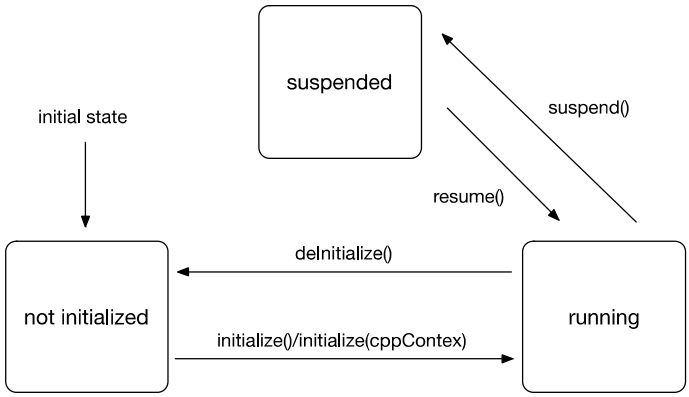

Figure 3: Radio class state model

not initialized: This is the initial state of the Radio class. This means that when the OMRI app 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 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 initialization and/or service selections.

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

4.3.4 Tuner state model

Figure 4 depicts the state model of the Tuner object.

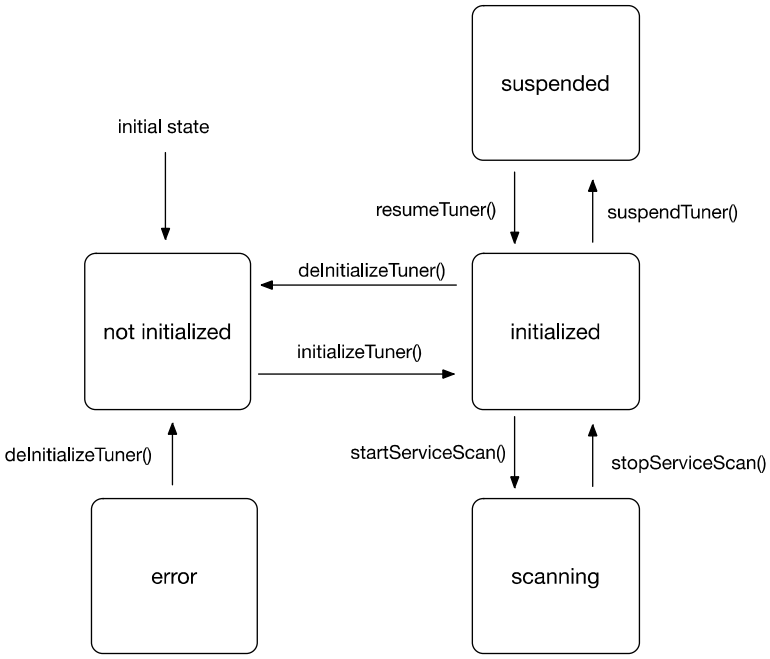

Figure 4: Tuner object state model

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 an 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 deInitializeTuner() 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 (see clause A.5.3) in the newStatus parameter when calling the tunerStatusChanged() method.

5 Examples of use of OMRI API

5.1 Developer experience

In order to gain a better understanding of the steps necessary to use the OMRI API for a minimal radio playing app, an example is provided. This example is an extraction of source code fragments from a real OMRI sample app's MainActivity. The full source code of this app is given in annex C. The example is quite basic and does not show implementations for sophisticated functions such as persistent storage of service and programme information or favourite lists. The sample allows the user to scan for services, tune to a service and display the dynamic label [i.1], DLplus [i.5] and SlideShow [i.2] information if present.

The sample code shown in clause 5 is highly Android specific.

The example shows the following steps:

1) Getting a Radio instance and initializing and registering the minimum listeners.

2) Implementing a TunerListener.

3) Implementing a VisualMetadataListener.

4) Implementing a TextualMetadataListener.

5.2 Getting a Radio instance and initializing and registering the minimum listeners

This is an excerpt of the onCreate method of the App's main activity. For the full code see annex C. Here only the OMRI relevant code fragments are shown.

```
/*
 * Ok, here is the real entry point for the RadioAPI
 * Ask the API for its status and handle it.
 * It will be in the state 'STATUS_RADIO_SUSPENDED' when it is not initialized. You have to
 * initialize it, optionally with the Android App Context,
 * allowing you e.g. to persist the scanned services to the private App data directory
 * without explicitly asking the WRITE_INTERNAL_ or WRITE_EXTERNAL_STORAGE
 * permission in your app manifest.
 * Be aware that this is a blocking call. If your Radio takes a long time to initialize you
 * should do this in a separate thread or AsyncTask.
*/
RadioStatus stat = Radio.getInstance().getRadioStatus();
switch (stat) {
    case STATUS_RADIO_SUSPENDED: {
        RadioErrorCode initCode = Radio.getInstance().initialize(this);
        if(initCode == RadioErrorCode.ERROR_INIT_OK) {
            Log.d(TAG, "Radio successfully initialized!");
        }
        break;
    }
    case STATUS_RADIO_RUNNING: {
        Log.d(TAG, "Great, the Radio is already running.");
        for(Tuner tuner : Radio.getInstance().getAvailableTuners()) {
            if(tuner.getCurrentRunningRadioService() != null) {
                tuner.getCurrentRunningRadioService().subscribe(mServiceSlideshowListener);
                tuner.getCurrentRunningRadioService().subscribe(mServiceDynamicLabelListener
                );
            }
        }
        break;
    }
    default: {
        break;
    }
}

mServiceList.clear();
mServiceList.addAll(Radio.getInstance().getRadioServices());
mServicelistFragment.updateServiceList(mServiceList);
```

```
        /* The Radio is running, now we are getting the available Tuners.
         * Before we initialize a Tuner we should subscribe a TunerListener to it to get callbacks
         * on Tuner state changes.
         */
        List<Tuner> tunerList = Radio.getInstance().getAvailableTuners();
        Log.d(TAG, "Found " + tunerList.size() + " Tuners!");
        for(Tuner tuner : tunerList) {
            Log.d(TAG, "TunerType: " + tuner.getTunerType().toString());
            Log.d(TAG, "TunerStatus: " + tuner.getTunerStatus().toString());

            tuner.subscribe(mTunerListener);

            if(tuner.getTunerStatus() == TunerStatus.STATUS_TUNER_NOT_INITIALIZED) {
                tuner.initializeTuner();
            }
        }
    }
```

Firstly the app should check on the status of the Radio object. Options are either:

- STATUS_RADIO_SUSPENDED: Then the initialize method should be called. NOTE: For simplicity, no error handling is implemented; or
- STATUS_RADIO_RUNNING: The Radio is already up and running. Here the app developer needs the get the current running radio service from one of the available tuners.

NOTE: For simplicity, the possibility of multiple running tuners is not handled.

The example does not implement code for persistent storage of service lists or other metadata, so the service list is deleted and a new service list is required from the Radio object.

Finally the example adds a TunerListener for each available tuner.

At this stage, the example is in the state to start a service scan and display the found radio services or in the case of the STATUS_RADIO_RUNNING the actual selected service is playing and available text and visual data are displayed.

5.3 Implementing a TunerListener

The TunerListener Interface is the central element for the app developer to get live data from the different available physical tuners known to the receiver device's OMRI implementation. Most important for the developer is the handling of the different tuner statuses during its life cycle.

The app's sample here takes care of the following statuses:

- STATUS_TUNER_SCAN_STARTED: Disable the scan button;
- STATUS_TUNER_SCAN_FINISHED: Update the service list UI element and enable the scan button. Additionally the example here shows how to iterate over the found DABservices and stores the type (Programme or Data service) and the service label;
- STATUS_TUNER_INITIALIZED: Update the service list UI element.

The tuner ScanProgress returns to the % of the DAB frequencies that have been tested for ensemble presence.

The tunerReceptionStatistics returns the rssi value in dBm.

When a service gets started or stopped, using the radioServiceStarted and radioServiceStopped methods are called by the implementation and the necessary listeners are subscribed or unsubscribed.

The implementation of the tunerRawData method is intentionally left blank here. It is intended to allow the app developer to get technology specific metadata e.g. FIC or MSC data from a DAB TunerType.

```
    /*
     * A listener for tuner messages.
     */
    private TunerListener mTunerListener = new TunerListener() {
```

```
@Override
public void tunerStatusChanged(final Tuner tuner, TunerStatus tunerStatus) {
    Log.d(TAG, "tunerStateChanged to: " + tunerStatus.toString());

    switch (tunerStatus) {
        case STATUS_TUNER_SCAN_STARTED: {
            Log.d(TAG, "ServiceScan started!");
            if(mScanButton != null) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mScanButton.setEnabled(false);
                    }
                });
            }
            break;
        }
        case STATUS_TUNER_SCAN_FINISHED: {
            Log.d(TAG, "ServiceScan finished!");

            mServiceList = Radio.getInstance().getRadioServices();
            if(mServicelistFragment != null) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mServicelistFragment.updateServiceList(mServiceList);
                    }
                });
            }

            if(mScanButton != null) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mScanButton.setEnabled(true);
                    }
                });
            }

            List<RadioService> singleTunerServices = tuner.getRadioServices();
            for(RadioService service : singleTunerServices) {
                switch (service.getRadioServiceType()) {
                    case RADIOSERVICE_TYPE_DAB: {
                        Log.d(TAG, "Found DAB " +
                            (((RadioServiceDab)service).isProgrammeService() ? "Programme" :
                         "Data" ) + " Service: " +
                         ((RadioServiceDab)service).getServiceLabel());
                        break;
                    }
                    default: {
                        break;
                    }
                }
            }
        }
        case STATUS_TUNER_INITIALIZED: {
            if(mServicelistFragment != null) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        mServicelistFragment.updateServiceList(mServiceList);
                    }
                });
            }
        }
        default: {
            break;
        }
    }
}

@Override
public void tunerScanProgress(Tuner tuner, int percentScanned) {
    Log.d(TAG, "tunerScanProgress: " + percentScanned + "%");
}

@Override
public void radioServiceStarted(Tuner tuner, RadioService radioService) {
```

```
            Log.d(TAG, "radioServiceStarted: " + radioService.getRadioServiceType().toString());

            radioService.subscribe(mServiceSlideshowListener);
            radioService.subscribe(mServiceDynamicLabelListener);
        }

        @Override
        public void radioServiceStopped(Tuner tuner, RadioService radioService) {
            Log.d(TAG, "radioServiceStopped: " + radioService.getRadioServiceType().toString());

            radioService.unsubscribe(mServiceSlideshowListener);
            radioService.unsubscribe(mServiceDynamicLabelListener);
        }

        @Override
        public void tunerReceptionStatistics(Tuner tuner, boolean locked, int rssi){
            Log.d(TAG, "tunerReceptionStatistics: Locked: " + locked + " Rssi: " + rssi);
        }

        @Override
        public void tunerRawData(Tuner tuner, byte[] data) {
            //Do something useful with this later
        }
    };
```

5.4 Implementing a VisualMetadataListener

When tuned to a service, an app shall provide a proper implementation of a VisualMetadataListener in order to get updates when new SlideShow [i.2] images are received. According to the interface definition for the VisualMetadataListener only one method (public void newVisualMetadata()) needs to be implemented. In the example above, the implementation checks if the necessary UI elements are not null and takes care that the display of the new SideShow image will happen in the UI thread of the app. When called by the OMRI implementation, metadata of the Visual (e.g. Type and DataSize) are logged, a Bitmap is generated and finally the Bitmap is passed to the View UI element.

```
    /*
     * A listener for Slideshow. Will be registered with a service on a serviceStarted callback and
     * unregistered at serviceStopped
     */
    private RadioServiceListener mServiceSlideshowListener = new VisualMetadataListener() {
        @Override
        public void newVisualMetadata(final Visual visual) {
            if(mSlsView != null) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Log.d(TAG, "New Visual Received: " + visual.getVisualType().toString() +
                           " with DataSize of: " + visual.getVisualData().length);
                        Bitmap sls = BitmapFactory.decodeByteArray(visual.getVisualData(), 0,
                           visual.getVisualData().length);
                        Log.d(TAG, "Bitmap: " + sls.getWidth() + ":" + sls.getHeight());
                        mSlsView.setImageBitmap(sls);
                    }
                });
            }
        }
    };
```

5.5 Implementing a TextualMetadataListener

When tuned to a service, an app shall provide a proper implementation of a TextualMetadataListener in order to get updates when new textual labels (e.g. from dynamic label [i.1], DLPlus [i.5], RDS RadioText [i.6] or RadioDNS [i.4]) are received. According to the interface definition for the TextualMetadataListener only one method (public void newTextualMetadata()) needs to be implemented. In the example above the implementation checks if the necessary view is not null. When called by the OMRI implementation, the text content is logged and the textual UI element is updated.

```
    /*
     * A listener for Dynamic Labels. Will be registered with a service on a serviceStarted callback
     * and unregistered at serviceStopped
```

```
 */
private RadioServiceListener mServiceDynamicLabelListener = new TextualMetadataListener() {
    @Override
    public void newTextualMetadata(Textual textual) {
        Log.d(TAG, "New Text Received: " + textual.getText());
        if(mDlsView != null) {
            mDlsView.setText(textual.getText());
        }
    }
};
```

Annex A (normative): Java API Interface and Class definitions

A.1 Introduction

This annex lists the formal API definition in Java.

The Java skeletons of the org.omri packages on the specifications Git repository are hosted at:

https://github.com/ebu.

A.2 Package org.omri.radio

A.2.1 Radio

```
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 Fabian Sattler, IRT GmbH
 */
public abstract class Radio {

   /** the singleton instance of Radio */
    private static Radio INSTANCE = new RadioImpl();

    /**
     * Returns the {@link Radio} instance or {@code null} if not implemented {@link Radio} instance
is set
     * @return the {@link Radio} instance or {@code null} if not implemented {@link Radio} instance
is set
     */
    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 plattform specific objects into the OMRI implementation
     * i.e. Android {@link Context}
     * @param appContext the App Context
     * @return the {@link RadioErrorCode} indicating the success of init.
     */
    public abstract RadioErrorCode initialize(Object appContext);

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

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

    /**
     * Indicates the current status of the {@link Radio}
     * @return the current {@link RadioStatus}
     */
    public abstract RadioStatus getRadioStatus();

    /**
     * Deinitializes the Radio and all {@link Tuner}s
     */
    public abstract void deInitialize();

    /**
     * Returns the available {@link Tuner} devices or an empty list
     * @return the available {@link Tuner} devices or an empty list
     */
    public abstract List<Tuner> getAvailableTuners();

    /**
     * Returns the available {@link Tuner} devices for a specific {@link TunerType} or an empty list
     * @return the available {@link Tuner} devices for a specific {@link TunerType} or an empty list
     */
    public abstract List<Tuner> getAvailableTuners(TunerType tunerType);

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

   /**
   * Start a {@link RadioService} on an available tuner
   * The method here is for the convenience of the application developer.
   * @param radioService the {@link 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.
     * If the application developer wants to perform service scans in the background
     * (in the case the Radio exposes more than on {@link Tuner} instances), it's recommended
     * to use the dedicated method calls in the {@link 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 on {@link Tuner} instances), it's recommended
     * to use the dedicated method calls in the {@link Tuner} objects.
   */
   public abstract void stopRadioServiceScan();

  /**
    * Initializes a specific {@link Tuner} device. You should listen to status changes with a
registered {@link TunerListener}.
```

```
    * @param tuner the {@link Tuner} to initialize
    */
   public abstract void initializeTuner(Tuner tuner);

  /**
    * Deinitializes a specific {@link Tuner} device
    * @param tuner the {@link Tuner} to deinit
    */
   public abstract void deInitializeTuner(Tuner tuner);

  /**
    * Registers a {@link RadioStatusListener}
    * @param listener the {@link RadioStatusListener} to register for callbacks
    */
   public abstract void registerRadioStatusListener(RadioStatusListener listener);

/**
   * Unregisters a {@link RadioStatusListener}
   * @param listener the {@link RadioStatusListener} to unregister for callbacks
  */
  public abstract void unregisterRadioStatusListener(RadioStatusListener listener);

}
```

A.2.2 RadioErrorCode

```
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 Fabian Sattler, 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.2.3 RadioListener

```
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 Fabian Sattler, IRT GmbH
 *
 * @see TunerListener
 */
public interface RadioListener {

}
```

A.2.4 RadioStatus

```
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 Fabian Sattler, 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.2.5 RadioStatusListener

```
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 Fabian Sattler, IRT GmbH
 */

import org.omri.tuner.Tuner;

public interface RadioStatusListener extends RadioListener {

   /**
    * Signals that a new {@link Tuner} device was attached
    * @param attachedTuner the new attached {@link Tuner}
    */
   void tunerAttached(Tuner attachedTuner);

   /**
    * Signals that a {@link Tuner} device was detached and is not available anymore
    * @param detachedTuner the detached {@link Tuner} device
    */
   void tunerDetached(Tuner detachedTuner);
}
```

A.3 Package org.omri.radioservice

A.3.1 RadioService

```
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 Fabian Sattler, IRT GmbH
 * @author Erk, 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.3.2 RadioServiceAudiodataListener

```
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 the decoded PCM audio stream
 *
 * @author Fabian Sattler, IRT GmbH
 */
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.3.3 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 Fabian Sattler, IRT GmbH
 */
public interface RadioServiceDab extends RadioService {

   /**
   * Returns the extended country code (ECC) for the DAB Ensemble this {@link RadioServiceDab}
belongs to
   * @return 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
   * @return 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
```

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

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


}
```

A.3.4 RadioServiceDabComponent

```
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 containing informations for a service component (SC) for a {@link RadioServiceDab}
 * @author Fabian Sattler, IRT GmbH
 */
public interface RadioServiceDabComponent {

    /**
     * Returns the bitrate in kbit/s for this {@link RadioServiceDabComponent}
     * @return the bitrate in kbit/s for this {@link RadioServiceDabComponent}
     */
    public int getBitrate();
```

```
    /**
     * Indicates if the CA (Conditional Access) flag is set
     * @return indication for conditional access
     */
    public boolean isCaApplied();

    /**
     * Returns the {@link RadioServiceDabComponent}'s service ID
     * @return the {@link RadioServiceDabComponent}'s service ID
     */
    public int getServiceId();

    /**
     * Returns the {@link RadioServiceDabComponent} subchannel ID
     * @return the {@link RadioServiceDabComponent} subchannel ID
     */
    public int getSubchannelId();

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

    /**
     * Returns the packet address or -1 if it's not a packetmode service.
     * @return the packet address or -1 if it's not a packetmode service.
     */
    public int getPacketAddress();

    /**
     * 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 isPrimary();

    /**
     * Returns the service component ID within service (ScIDs)
     * @return the service component ID within service (ScIDs)
     */
    public int getServiceComponentIdWithinService();

    /**
     * Returns the start address of this {@link RadioServiceDabComponent} in the MSC
     * @return the start address of this {@link RadioServiceDabComponent} in the MSC
     */
    public int getMscStartAddress();

    /**
     * Returns the size of this {@link RadioServiceDabComponent} in capacity units
     * @return the size of this {@link RadioServiceDabComponent} in capacity units
     */
    public int getSubchannelSize();

    /**
     * Return the protectionlevel of this {@link RadioServiceDabComponent}
     * @return the protectionlevel of this {@link RadioServiceDabComponent}
     */
    public int getProtectionLevel();

    /**
     * Return the protectiontype of this {@link RadioServiceDabComponent}
     * @return the protectiontype of this {@link RadioServiceDabComponent}
     */
    public int getProtectionType();

    /**
     * Returns the unequal error protection table index of this {@link RadioServiceDabComponent}
     * @return the unequal error protection table index of this {@link RadioServiceDabComponent}
     */
    public int getUepTableIndex();

    /**
     * Indicates if a forward error correction (FEC) scheme is applied to this {@link
RadioServiceDabComponent}
```

```
    * @return a boolean indicating if a forward error correction (FEC) scheme is applied to this
{@link RadioServiceDabComponent}
    */
   public boolean isFecSchemeApplied();

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

   /**
    * Returns the service component type. Audioservice Component Type (ASCTy) for TmId{0};
Dataservice Component Type (DSCTy) for TmId{1} and TmId{3}.
    * @return the service component type.
    */
   public int getServiceComponentType();

   /**
    * Indicates if MSC datagroup transport is used for this component. Only applicable for
dataservices TmId{1} and TmId{3}.
    * @return if MSC datagroup transport is used for this component.
    */
   public boolean isDatagroupTransportUsed();

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

   /**
    * 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.3.5 RadioServiceDabComponentListener

```
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 {@link RadioServiceDabComponent} (e.g. Journaline
data)
 *
 * @author Fabian Sattler, IRT GmbH
 */
public interface RadioServiceDabComponentListener {

   /**
   * Called when new data from a specific DAB Service Component was received
   * @param serviceComponentChannelId th Service component id
   * @param scData raw data from the service component
   */
```

```
   public void newServiceComponentData(int serviceComponentChannelId, byte[] scData);
}
```

A.3.6 RadioServiceDabUserApplication

```
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 {@link
RadioServiceDabComponent}
 * @author Fabian Sattler, 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.3.7 RadioServiceFm

```
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 {@link RadioService} (e.g. 99300000 for a frequency of
99.3 MHz)
   * @return the frequency in kHz
   */
   public int getFrequency();
```

```
    /**
    * Returns the RDS PI (Programme Identification) 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 {@link RadioServiceFmPty} for this {@link RadioService} if available, or {@code null}
    * @return a {@link RadioServiceFmPty} for this {@link RadioService}
    */
    public RadioServiceFmPty getRdsPty();
}
```

A.3.8 RadioServiceFmPty

```
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 (Rogramme Type)
 * @author Fabian Sattler, IRT GmbH
 */
public interface RadioServiceFmPty {

    /**
    * Returns the PTY code
    * @return the PTY code
    */
    public int getPtyCode();

    /**
    * Returns a human readable representation of the PTY code
    * @return a human readable representation of the PTY code
    */
    public String getPtyDescription();
}
```

A.3.9 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
 *
 * 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 interface for an IP delivered {@link RadioService}
 * @author Fabian Sattler, IRT GmbH
 */
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} for this {@link RadioService}
    */
    public List<RadioServiceIpStream> getIpStreams();
}
```

A.3.10 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
 *
 * 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 interface which defines the necessary data for a single 'stream' of a {@link
RadioService}
 * @author Fabian Sattler, IRT GmbH
 */
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();

    /**
    * Return the MIME type of this stream
    * @return the MIME type of this stream
    */
    public RadioServiceMimeType getMimeType();

    /**
    * Indicates the cost for this stream. Higher means more 'expensive'
    * @return the cost for this stream
    */
    public int getCost();

    /**
    * Indicates the offset in seconds from the fastest corresponding broadcast bearer (e.g. FM).
    * @return the offset in seconds.
    */
    public int getOffset();
}
```

A.3.11 RadioServiceListener

```
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;

/**
 * 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 updates for {@link RadioService} related informations
 * @author Fabian Sattler, IRT GmbH
 *
 * @see ProgrammeServiceMetadataListener
 * @see TextualMetadataListener
 * @see VisualMetadataListener
 */
public interface RadioServiceListener extends RadioListener {

}
```

A.3.12 RadioServiceMimeType

```
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 enum RadioServiceMimeType {

   /** MIME type not known **/
   UNKNOWN("mime/unknown", -1),
   /** MIME type for MPEG 1 Layer 1, 2, 3 **/
   AUDIO_MPEG("audio/mpeg", 0),
   /** MIME type for OGG Vorbis audio **/
   AUDIO_OGG_VORBIS("audio/ogg", 1),
   /** MIME type for Free Losless Audio Codec **/
   AUDIO_FLAC("audio/flac", 2),
   /** MIME type ADTS AAC **/
   AUDIO_AAC("audio/aacp", 3),
   /** RAW DAB+ AAC Access Unit **/
   AUDIO_AAC_DAB_AU("audio/aac", 63),
   /** PCM Audio **/
   AUDIO_PCM("audio/wav", 99);

   private final String contentTypeString;
   private final int contentTypeId;

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

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

   public int getContententTypeId() {
      return this.contentTypeId;
   }
}
```

A.3.13 RadioServiceRawAudiodataListener

```
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 the encoded, radiosystem specific, raw audio stream
 *
 * @author Fabian Sattler, IRT GmbH
 */
public interface RadioServiceRawAudiodataListener extends RadioServiceListener {

   /**
    * Encoded raw audio data interface
    * @param rawData the raw audio data
    * @param sbr indicates if SBR is used
    * @param type the used audioType
    * @param numChannels number of channels contained in the raw audio stream
    * @param samplingRate the sampling rate of the raw audio stream
    */
   public void rawAudioData(byte[] rawData, boolean sbr, RadioServiceMimeType type, int numChannels,
int samplingRate);
}
```

A.3.14 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
 * @author Fabian Sattler, 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 Sirius XM **/
   RADIOSERVICE_TYPE_SIRIUS,
   /** RadioService type iBiquity HD Radio **/
   RADIOSERVICE_TYPE_HDRADIO;
}
```

A.4 Sub-package org.omri.radioservice.metadata

A.4.1 Group

```
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 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 Visual}s for this {@link Group} or an empty list
     * @return the available {@link Visual}s 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();
}
```

A.4.2 Location

```
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 Location {

}
```

A.4.3 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.4.4 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.
 *
 * Programme metadata type definitions
 *
 * @author Erk, IRT GmbH
 */
public enum ProgrammeInformationType {

   /** Programme metadata received via RadioDNS/DAB SPI **/
   METADATA_PROGRAMME_TYPE_SPI_EPG
}
```

A.4.5 ProgrammeServiceMetadataListener

```
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 {@link ServiceInformation} programme
associated metadata
 *
 * @author Fabian Sattler, 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.4.6 ServiceInformation

```
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 Fabian Sattler, IRT GmbH
 */
public interface ServiceInformation {

}
```

A.4.7 SpiProgrammeInformation

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

import org.w3c.dom.Document;

/**
 * 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 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.4.8 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.4.9 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
 * limitations under the License.
 *
 * Abstract base class for textual metadata
 * @author Fabian Sattler, IRT GmbH
 */
public interface Textual {

   /**
   * Returns the type of this textual metadata
   * @return the type of this textual metadata
   */
   public TextualType getType();

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

A.4.10 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) {@link Textual} metadata
 *
 * @author Fabian Sattler, IRT GmbH
 */
public interface TextualDabDynamicLabel extends Textual {

   /**
   * Indicates if this {@link 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 {@link TextualDabDynamicLabelPlusItem}s or an empty list
   * @return a list of {@link TextualDabDynamicLabelPlusItem}s or an empty list
   */
   public List<TextualDabDynamicLabelPlusItem> getDlPlusItems();

   /**
    * Indicates that the current programme item is running. May be false if the current programme is
interrupted for e.g. road traffic flash
    * @return {@code true} if the programme item is running, {@code false} otherwise
    */
   public boolean itemRunning();

   /**
    * Toggles when a new programme element, e.g. a new song, begins
    * @return the current toggle state. Default is {@code false}
    */
   public boolean itemToggled();
}
```

A.4.11 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
 *
 *      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.
 *
 * Dynamic Label Plus Content type definitions
 *
 * @author Fabian Sattler, IRT GmbH
 */
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_TRAFFIC(26, "Traffic"),
    INFO_ALARM(27, "Alarm"),
    INFO_ADVERTISEMENT(28, "Advertisement"),
    INFO_URL(29, "URL"),
    INFO_OTHER(30, "Other"),
    STATIONNAME_SHORT(31, "Short Stationname"),
    STATIONNAME_LONG(32, "Long Stationname"),
    PROGRAMME_NOW(33, "Now"),
    PROGRAMME_NEXT(34, "Next"),
    PROGRAMME_PART(35, "Part"),
    PROGRAMME_HOST(36, "Host"),
    PROGRAMME_EDITORIAL_STAFF(37, "Editorial Staff"),
    PROGRAMME_FREQUENCY(38, "Frequency"),
    PROGRAMME_HOMEPAGE(39, "Homepage"),
    PROGRAMME_SUBCHANNEL(40, "Subchannel"),
    PHONE_HOTLINE(41, "Hotline"),
    PHONE_STUDIO(42, "Studio Telephone"),
    PHONE_OTHER(43, "Other Telephone"),
    SMS_STUDIO(44, "Studio SMS"),
    SMS_OTHER(45, "SMS"),
    EMAIL_HOTLINE(46, "Hotline Email"),
    EMAIL_STUDIO(47, "Studio Email"),
    EMAIL_OTHER(48, "Email"),
    MMS_OTHER(49, "MMS"),
    CHAT(50, "Chat"),
    CHAT_CENTER(51, "Chat Center"),
    VOTE_QUESTION(52, "Vote Question"),
    VOTE_CENTRE(53, "Vote Centre"),
    RFU_1(54, "RFU1"),
    RFU_2(55, "RFU2"),
    PRIVATE_CLASS_1(56, "Private Data 1"),
    PRIVATE_CLASS_2(57, "Private Data 2"),
    PRIVATE_CLASS_3(58, "Private Data 3"),
    DESCRIPTOR_PLACE(59, "Place"),
    DESCRIPTOR_APPOINTMENT(60, "Appointment"),
    DESCRIPTOR_IDENTIFIER(61, "Identifier"),
    DESCRIPTOR_PURCHASE(62, "Purchase"),
    DESCRIPTOR_GET_DATA(63, "Get Data");

    private final int contentTypeId;
     private final String contentTypeString;

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

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

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

A.4.12 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 Fabian Sattler, 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 a 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.4.13 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");
 * 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 Fabian Sattler, IRT GmbH
 */
public interface TextualFmRdsRadiotext extends Textual {

}
```

A.4.14 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 delievered RadioDNS RadioVIS {@link Textual} metadata
 *
 * @author Fabian Sattler, IRT GmbH
 */
public interface TextualIpRdnsRadioVis extends Textual {

}
```

A.4.15 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 Fabian Sattler, IRT GmbH
 */
public interface TextualMetadataListener extends RadioServiceListener {

   /**
   * New {@link Textual} metadata was received
   * @param textualMetadata the {@link Textual} received
   */
   public void newTextualMetadata(Textual textualMetadata);
}
```

A.4.16 TextualType

```
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 Fabian Sattler, 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.4.17 Visual

```
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 Fabian Sattler, 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
   * @return the actual image data
   */
   public byte[] getVisualData();

   /**
    * Returns the width of the image or -1 if not known
    * @return the width of the image or -1 if not known
    */
   public int getVisualWidth();

   /**
    * Returns the height of the image or -1 if not known
    * @return the height of the image or -1 if not known
    */
   public int getVisualHeight();
}
```

A.4.18 VisualDabSlideShow

```
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 {@link Visual} metadata
 *
 * @author Fabian Sattler, IRT GmbH
 */
public interface VisualDabSlideShow extends Visual {

   /**
    * Indicates if this {@link 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 {@link VisualDabSlideShow}
    * @return the ID
    */
   public int getSlideId();

   /**
    * The Triggertime
    * @return a Calendar for the Triggertime or {@code 'null'} if the triggertime is {@code 'now'}
    */
   public Calendar getTriggerTime();

   /**
    * The ExpiryTime
    * @return a Calendar for the Expirytime or {@code 'null'}
    */
   public Calendar getExpiryTime();

   /**
    * Returns the category description of this {@link VisualDabSlideShow}. Only applicable for a DAB
Categorized Slideshow (check 'isCategorized()')
    * @return the category text or an empty {@link String} if it's not a categorized
VisualDabSlideShow
    */
   public String getCategoryText();

   /**
    * Returns the category id of this {@link VisualDabSlideShow}. Only applicable for 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 {@link 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
{@link 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 {@link 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.4.19 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 Fabian Sattler, 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.4.20 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");
 * 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 Fabian Sattler, 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.4.21 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.
 *
 * Visual metadata types
 * @author Fabian Sattler, 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.4.22 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
 * limitations under the License.
 *
 * Visual type definition
 * @author Fabian Sattler, IRT GmbH
 */
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.5 Package org.omri.tuner

A.5.1 ReceptionQuality

```
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.
 *
 * Abstract Tuner class
 * @author Fabian Sattler, IRT GmbH
 */

public enum ReceptionQuality {
   NO_SIGNAL,
   BAD,
   POOR,
   OKAY,
   GOOD,
   BEST
}
```

A.5.2 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
   * {@link 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 {@link TunerType} of this {@link Tuner}
    */
    public TunerType getTunerType();

    /**
    * Indicates the current status of the {@link Tuner} device
    * @return the current {@link TunerStatus}
    */
    public TunerStatus getTunerStatus();

    /**
    * Retrieve the currently known {@link RadioService}s of this {@link Tuner}
    * @return a 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 {@code 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);
}
```

A.5.3 TunerListener

```
package org.omri.tuner;

import org.omri.radio.RadioListener;
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.
 *
 * The {@link Tuner} listener interface
 * @author Fabian Sattler, IRT GmbH
 */
public interface TunerListener extends RadioListener {

   /**
    * Tuner changed its operating state
    * @param newStatus the new {@link TunerStatus}
    */
   public void tunerStatusChanged(Tuner tuner, TunerStatus newStatus);

   /**
    * Tuner started scan for services
    */
   public void tunerScanStarted(Tuner tuner);

   /**
    * Tuner scan progress indicator
    * @param percentScanned the percentage finished so far
    */
   public void tunerScanProgress(Tuner tuner, int percentScanned);

   /**
    * Tuner finished scan for services
    */
   public void tunerScanFinished(Tuner tuner);

   /**
    * 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 dbµV ???)
    */
   public void tunerReceptionStatistics(Tuner tuner, boolean rfLock, ReceptionQuality quality);

   /**
    * 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.5.4 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 is in scanning progress **/
    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;
    }

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

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

A.5.5 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
 * @author Fabian Sattler
 */
public enum TunerType {

   /** DAB Tuner **/
   TUNER_TYPE_DAB,
   /** IP Tuner **/
   TUNER_TYPE_IP,
   /** FM Tuner **/
   TUNER_TYPE_FM,
   /** Sirius XM Tuner **/
   TUNER_TYPE_SIRIUS,
   /** iBiquity HD Radio Tuner **/
   TUNER_TYPE_HDRADIO;
}
```

Annex B (informative): Package org.omri.radio.impl

B.1 Introduction

This package contains only an example implementation of the abstract Radio Class. Possible implementers may choose a different name or structure.

The Java skeletons of the org.omri packages on the specifications Git repository are hosted at:

https://github.com/ebu.

B.2 public class RadioImpl extends Radio

```
package org.omri.radio.impl;

import java.util.List;

import org.omri.radio.Radio;
import org.omri.radio.RadioErrorCode;
import org.omri.radio.RadioStatus;
import org.omri.radioservice.RadioService;
import org.omri.tuner.Tuner;
import org.omri.tuner.TunerType;

import android.content.Context;

/**
 * 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.
 *
 * This is the main entry class for anyone who wants to implement the Universal Radio API.
 * @author Fabian Sattler, IRT GmbH
 */
public class RadioImpl extends Radio {

   @Override
   public RadioErrorCode initialize() {
      // TODO Auto-generated method stub. To be implemented by API middleware provider.
      return null;
   }

   @Override
   public RadioErrorCode suspend() {
      // TODO Auto-generated method stub. To be implemented by API middleware provider.
      return null;
   }

   @Override
   public RadioErrorCode resume() {
      // TODO Auto-generated method stub. To be implemented by API middleware provider.
      return null;
   }

   @Override
   public RadioStatus getRadioStatus() {
      // TODO Auto-generated method stub. To be implemented by API middleware provider.
      return null;
```

```
    }

    @Override
    public void deInitialize() {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.

    }

    @Override
    public List<Tuner> getAvailableTuners() {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.
       return null;
    }

    @Override
    public List<Tuner> getAvailableTuners(TunerType tunerType) {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.
       return null;
    }

    @Override
    public List<RadioService> getRadioServices() {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.
       return null;
    }



    @Override
    public void deInitializeTuner(Tuner tuner) {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.

    }

    @Override
    public RadioErrorCode initialize(Context appContext) {
       // TODO Auto-generated method stub
       return null;
    }

    @Override
    public void initializeTuner(Tuner tuner) {
       // TODO Auto-generated method stub

    }

    @Override
    public void startRadioService(RadioService radioService) {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.

    }

    @Override
    public void startRadioServiceScan() {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.

    }

    @Override
    public void stopRadioServiceScan() {
       // TODO Auto-generated method stub

    }

    @Override
    public void registerRadioStatusListener(RadioStatusListener listener) {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.
    }

    @Override
    public void unregisterRadioStatusListener(RadioStatusListener listener) {
       // TODO Auto-generated method stub. To be implemented by API middleware provider.
    }

}
```

Annex C (informative): OMRI sample application

C.1 Introduction

This annex lists the complete source of the MainActivity of the OMRI sample application.

Developers will find the Java source code on the specifications Git repository hosted on:

https://github.com/ebu.

C.2 MainActiviy.java

```
package irt.de.universalradiosampleapp;

import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import org.omri.radio.Radio;
import org.omri.radio.RadioErrorCode;
import org.omri.radio.RadioStatus;
import org.omri.radioservice.RadioService;
import org.omri.radioservice.RadioServiceDab;
import org.omri.radioservice.RadioServiceListener;
import org.omri.radioservice.metadata.Textual;
import org.omri.radioservice.metadata.TextualMetadataListener;
import org.omri.radioservice.metadata.Visual;
import org.omri.radioservice.metadata.VisualMetadataListener;
import org.omri.tuner.Tuner;
import org.omri.tuner.TunerListener;
import org.omri.tuner.TunerStatus;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity implements
RadioServiceListFragment.ServicelistListener {

    private final String TAG = "UniversalRadioMain";

    private ImageView mSlsView = null;
    private TextView mDlsView = null;
    private Button mScanButton = null;

    private RadioServiceListFragment mServicelistFragment;
    static final String SERVICELISTFRAGMENT_TAG = "SERVICELISTFRAGMENT";

    private List<RadioService> mServiceList = new ArrayList<RadioService>();

    /*
     * A listener for Slideshow. Will be registered with a service on a serviceStarted callback and
     * unregistered at serviceStopped
     */
    private RadioServiceListener mServiceSlideshowListener = new VisualMetadataListener() {
        @Override
        public void newVisualMetadata(final Visual visual) {
            if(mSlsView != null) {
                runOnUiThread(new Runnable() {
```

```
                    @Override
                    public void run() {
                        Log.d(TAG, "New Visual Received: " + visual.getVisualType().toString() + "
with DataSize of: " + visual.getVisualData().length);
                        Bitmap sls = BitmapFactory.decodeByteArray(visual.getVisualData(), 0,
visual.getVisualData().length);
                        Log.d(TAG, "Bitmap: " + sls.getWidth() + ":" + sls.getHeight());
                        mSlsView.setImageBitmap(sls);
                    }
                });
            }
        }
    };

    /*
     * A listener for Dynamic Labels. Will be registered with a service on a serviceStarted callback
and unregistered at serviceStopped
     */
    private RadioServiceListener mServiceDynamicLabelListener = new TextualMetadataListener() {
        @Override
        public void newTextualMetadata(Textual textual) {
            Log.d(TAG, "New Text Received: " + textual.getText());

            if(mDlsView != null) {
                mDlsView.setText(textual.getText());
            }
        }
    };

    /*
     * A listener for tuner messages.
     */
    private TunerListener mTunerListener = new TunerListener() {

        @Override
        public void tunerStatusChanged(final Tuner tuner, TunerStatus tunerStatus) {
            Log.d(TAG, "tunerStateChanged to: " + tunerStatus.toString());

            switch (tunerStatus) {
                case STATUS_TUNER_SCAN_STARTED: {
                    Log.d(TAG, "ServiceScan started!");
                    if(mScanButton != null) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mScanButton.setEnabled(false);
                            }
                        });
                    }
                    break;
                }
                case STATUS_TUNER_SCAN_FINISHED: {
                    Log.d(TAG, "ServiceScan finished!");

                    mServiceList = Radio.getInstance().getRadioServices();
                    if(mServicelistFragment != null) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mServicelistFragment.updateServiceList(mServiceList);
                            }
                        });
                    }

                    if(mScanButton != null) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mScanButton.setEnabled(true);
                            }
                        });
                    }

                    List<RadioService> singleTunerServices = tuner.getRadioServices();
                    for(RadioService service : singleTunerServices) {
                        switch (service.getRadioServiceType()) {
                            case RADIOSERVICE_TYPE_DAB: {
```

```
                                Log.d(TAG, "Found DAB " + (
((RadioServiceDab)service).isProgrammeService() ? "Programme" : "Data" ) + " Service: " +
((RadioServiceDab)service).getServiceLabel());
                                break;
                            }
                            default: {
                                break;
                            }
                        }
                    }
                }
                case STATUS_TUNER_INITIALIZED: {
                    if(mServicelistFragment != null) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mServicelistFragment.updateServiceList(mServiceList);
                            }
                        });
                    }
                }
                default: {
                    break;
                }
            }
        }

        @Override
        public void tunerScanProgress(Tuner tuner, int percentScanned) {
            Log.d(TAG, "tunerScanProgress: " + percentScanned + "%");
        }

        @Override
        public void radioServiceStarted(Tuner tuner, RadioService radioService) {
            Log.d(TAG, "radioServiceStarted: " + radioService.getRadioServiceType().toString());

            radioService.subscribe(mServiceSlideshowListener);
            radioService.subscribe(mServiceDynamicLabelListener);
        }

        @Override
        public void radioServiceStopped(Tuner tuner, RadioService radioService) {
            Log.d(TAG, "radioServiceStopped: " + radioService.getRadioServiceType().toString());

            radioService.unsubscribe(mServiceSlideshowListener);
            radioService.unsubscribe(mServiceDynamicLabelListener);
        }

        @Override
        public void tunerReceptionStatistics(Tuner tuner, boolean locked, int rssi){
            Log.d(TAG, "tunerReceptionStatistics: Locked: " + locked + " Rssi: " + rssi);
        }

        @Override
        public void tunerRawData(Tuner tuner, byte[] data) {
            //Do something useful with this later
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.content_main);

        /* Getting the Views for DLS and SLS */
        mSlsView = (ImageView)findViewById(R.id.radio_metadata_visual_imageview);
        mDlsView = (TextView)findViewById(R.id.radio_metadata_textual_textview);

        /* Adding the Fragment for the servilist to the layout */
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        mServicelistFragment =
(RadioServiceListFragment)fragmentManager.findFragmentByTag(SERVICELISTFRAGMENT_TAG);
        if(mServicelistFragment == null) {
            Log.d(TAG, "Adding Fragment to layout...");
            mServicelistFragment = new RadioServiceListFragment();
```

```
            transaction.add(R.id.servicelist_fragment_frame, mServicelistFragment,
SERVICELISTFRAGMENT_TAG);
        } else {
            Log.d(TAG, "ServicelistFragment already attached!");
        }
        transaction.commit();

        /* Getting the scan button and attaching a onClickListener to do something when it's clicked
*/
        mScanButton = (Button)findViewById(R.id.radio_service_scan_button);
        mScanButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                clearMetadataViews();
                Radio.getInstance().startRadioServiceScan();
            }
        });

        /*
         * Ok, here is the real entry point for the RadioAPI
         * Ask the API for its status and handle it.
         * It will be in the state 'STATUS_RADIO_SUSPENDED' when it is not initialized. You have to
initialize it, optionally with the Android App Context,
         * allowing you e.g. to persist the scanned services to the private App data directory
without explicitly asking the WRITE_INTERNAL_ or WRITE_EXTERNAL_STORAGE
         * permission in your app manifest.
         * Be aware that this is a blocking call. If your Radio takes a long time to initialize you
should do this in a separate thread or AsyncTask.
         */
        RadioStatus stat = Radio.getInstance().getRadioStatus();
        switch (stat) {
            case STATUS_RADIO_SUSPENDED: {
                RadioErrorCode initCode = Radio.getInstance().initialize(this);
                if(initCode == RadioErrorCode.ERROR_INIT_OK) {
                    Log.d(TAG, "Radio successfully initialized!");
                }
                break;
            }
            case STATUS_RADIO_RUNNING: {
                Log.d(TAG, "Great, the Radio is already running.");
                for(Tuner tuner : Radio.getInstance().getAvailableTuners()) {
                    if(tuner.getCurrentRunningRadioService() != null) {
                        tuner.getCurrentRunningRadioService().subscribe(mServiceSlideshowListener);
                        tuner.getCurrentRunningRadioService().subscribe(mServiceDynamicLabelListener
);
                    }
                }
                break;
            }
            default: {
                break;
            }
        }

        mServiceList.clear();
        mServiceList.addAll(Radio.getInstance().getRadioServices());
        mServicelistFragment.updateServiceList(mServiceList);

        /* The Radio is running, now we are getting the available Tuners.
         * Before we initialize a Tuner we should subscribe a TunerListener to it to get callbacks
on Tuner state changes.
         */
        List<Tuner> tunerList = Radio.getInstance().getAvailableTuners();
        Log.d(TAG, "Found " + tunerList.size() + " Tuners!");
        for(Tuner tuner : tunerList) {
            Log.d(TAG, "TunerType: " + tuner.getTunerType().toString());
            Log.d(TAG, "TunerStatus: " + tuner.getTunerStatus().toString());

            tuner.subscribe(mTunerListener);

            if(tuner.getTunerStatus() == TunerStatus.STATUS_TUNER_NOT_INITIALIZED) {
                tuner.initializeTuner();
            }
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy was called!");

        /*
         * Unsubscribe your listeners
         * You may also save your last SLS and DLS and display it again after screen orientation
change.
         */
        for(Tuner tuner : Radio.getInstance().getAvailableTuners()) {
            tuner.unsubscribe(mTunerListener);

            RadioService radioService = tuner.getCurrentRunningRadioService();

            if (radioService != null) {
                radioService.unsubscribe(mServiceSlideshowListener);
                radioService.unsubscribe(mServiceDynamicLabelListener);
            }
        }
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        Log.d(TAG, "onSaveInstanceState was called!");
    }

    private void clearMetadataViews() {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                mDlsView.setText("");
                mSlsView.setImageBitmap(null);
            }
        });
    }

    /*
     * ServicelistListener to handle onClick events from our servicelist Fragment.
     * When a servicelist entry was clicked this listener is called with the RadioService which was
clicked.
     * We first clear our SLS and DLS views and tell the Radio to start clicked service.
     */
    @Override
    public void onServiceSelected(RadioService service) {
        Log.d(TAG, "Service Selected!");
        clearMetadataViews();
        Radio.getInstance().startRadioService(service);
    }
}
```

C.3 RadioServiceArrayAdapter.java

```
package irt.de.universalradiosampleapp;
```

```
import android.content.Context;
import android.graphics.drawable.Drawable;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import org.omri.radioservice.RadioService;
import org.omri.radioservice.RadioServiceDab;
import org.omri.radioservice.RadioServiceType;

import java.util.List;

public class RadioServiceArrayAdapter extends ArrayAdapter<RadioService> {

    private final List<RadioService> mServiceList;
    private final Context mContext;

    public RadioServiceArrayAdapter(Context context, List<RadioService> radioServices) {
        super(context, R.layout.radioservice_adapter_row_layout, radioServices);

        this.mContext = context;
        this.mServiceList = radioServices;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflater inflater =
(LayoutInflater)mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        View srvView = inflater.inflate(R.layout.radioservice_adapter_row_layout, parent, false);
        RadioService service = mServiceList.get(position);
        srvView.setTag(service);

        //Set your station logo here
        ImageView srvLogoView = (ImageView)srvView.findViewById(R.id.radioservice_adapter_icon);

        TextView srvLabelView =
(TextView)srvView.findViewById(R.id.radioservice_adapter_servicelabel);
        RadioServiceType srvType = mServiceList.get(position).getRadioServiceType();
        String srvLabel = "";
        switch (srvType) {
            case RADIOSERVICE_TYPE_DAB: {
                srvLabel = ((RadioServiceDab)service).getServiceLabel();
                break;
            }
            //switch for other service types and specify the servicelabel
            default: {
                break;
            }
        }

        srvLabelView.setText(srvLabel);

        return srvView;
    }
}
```

C.4 RadioServiceListFragment.java

```
package irt.de.universalradiosampleapp;

import android.app.Activity;
import android.os.Bundle;
import android.app.ListFragment;
import android.util.Log;
import android.view.View;
import android.widget.ListView;

import org.omri.radioservice.RadioService;

import java.util.ArrayList;
import java.util.List;
```

```
public class RadioServiceListFragment extends ListFragment {

    private final String TAG = "UniversalRadioList";

    private ServicelistListener mListener;

    private List<RadioService> mServiceList = new ArrayList<RadioService>();
    private RadioServiceArrayAdapter mServiceAdapter;

    public RadioServiceListFragment() {

    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mServiceAdapter = new RadioServiceArrayAdapter(getActivity().getApplicationContext(),
mServiceList);
        setListAdapter(mServiceAdapter);
    }

    /*
     * Be sure that the Activity which attaches this list implements 'ServicelistListener'
interface.
     */
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);

        try {
            mListener = (ServicelistListener)activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement
ServicelistListener");
        }
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mListener = null;
    }

    public void updateServiceList(final List<RadioService> radioServices) {
        Log.d(TAG, "Updating servicelist with " + radioServices.size() + " Services");

        if(mServiceList != null) {
            mServiceList.clear();
            mServiceList.addAll(radioServices);
        } else {
            mServiceList = new ArrayList<RadioService>();
        }

        if(mServiceAdapter != null) {
            mServiceAdapter.notifyDataSetChanged();
        }
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);

        if (null != mListener) {
            mListener.onServiceSelected(mServiceList.get(position));
        }
    }

    public interface ServicelistListener {

        void onServiceSelected(RadioService service);
    }
}
```

History

Document history		
V1.1.1	October 2018	Publication