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

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

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

Copyright Notification

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.

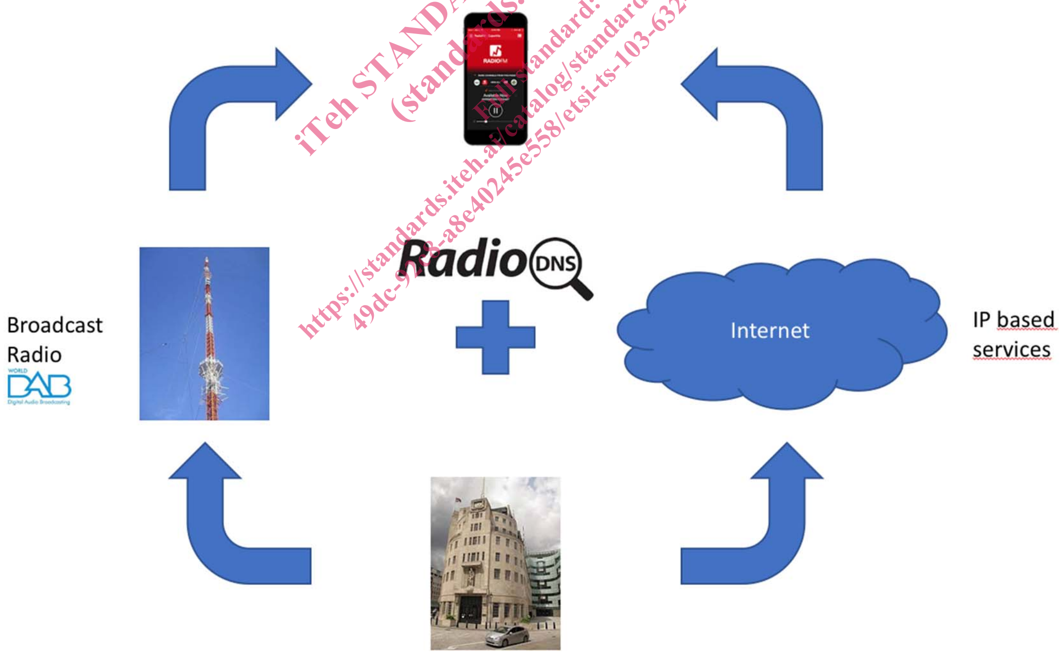

Figure 1: System overview

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.

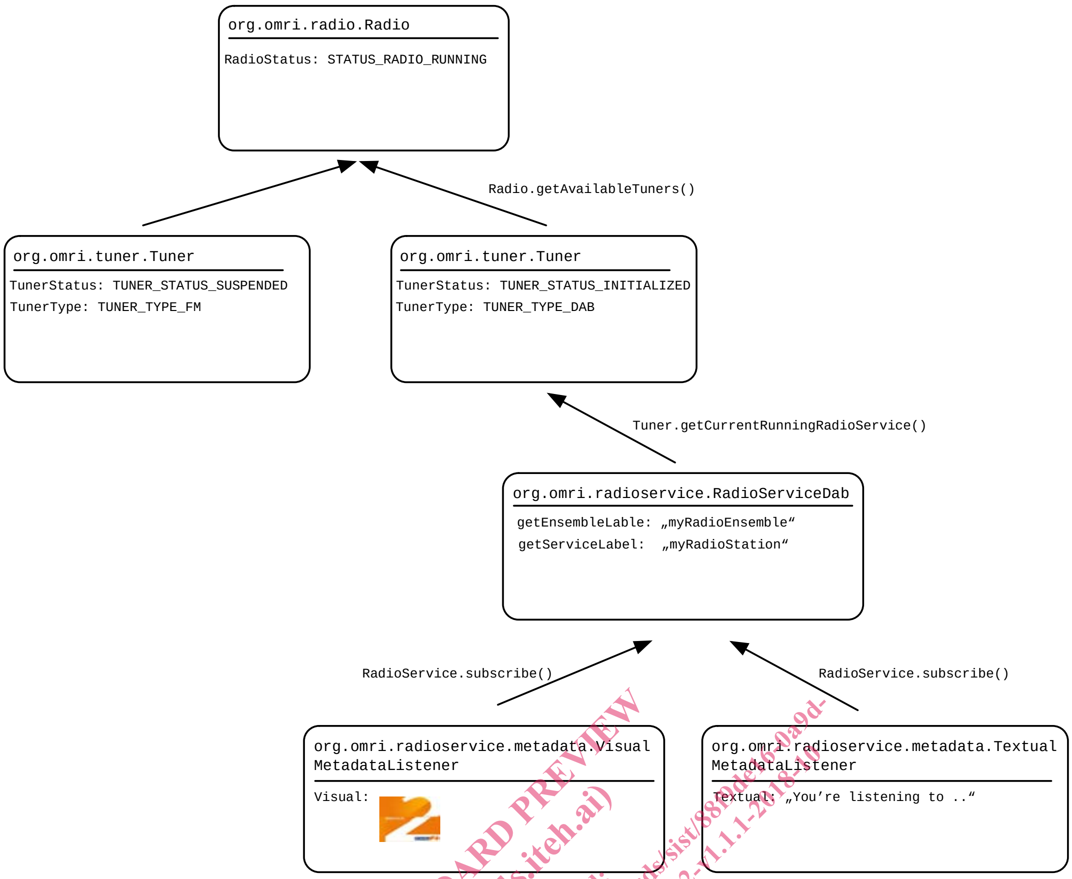

Figure 2: OMRI instance

4.3.3 Radio state model

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

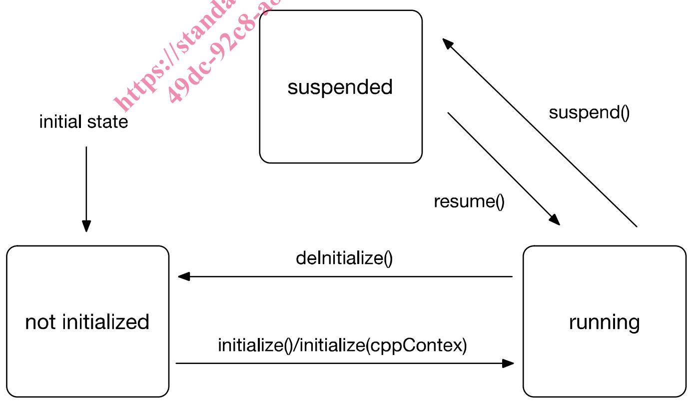

Figure 3: Radio class state model

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.

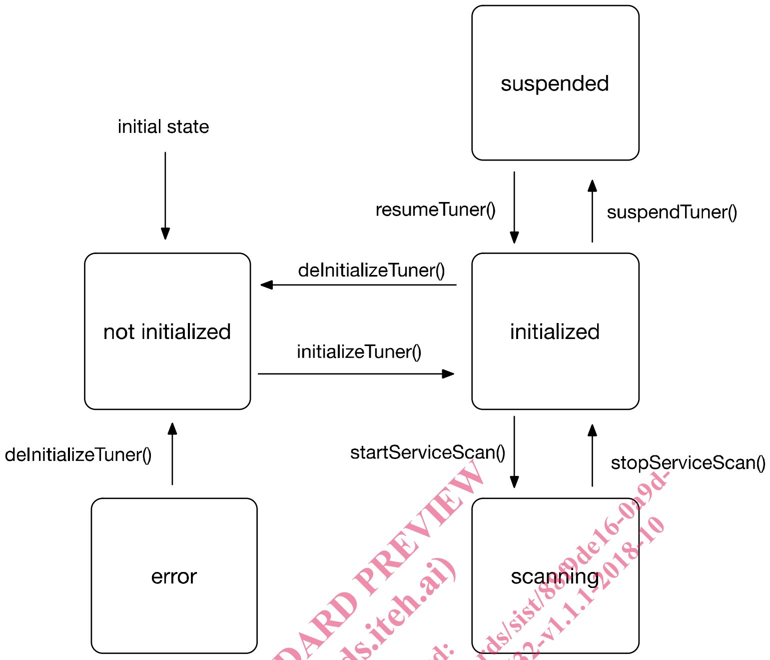

Figure 4: Tuner object state model

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);
```