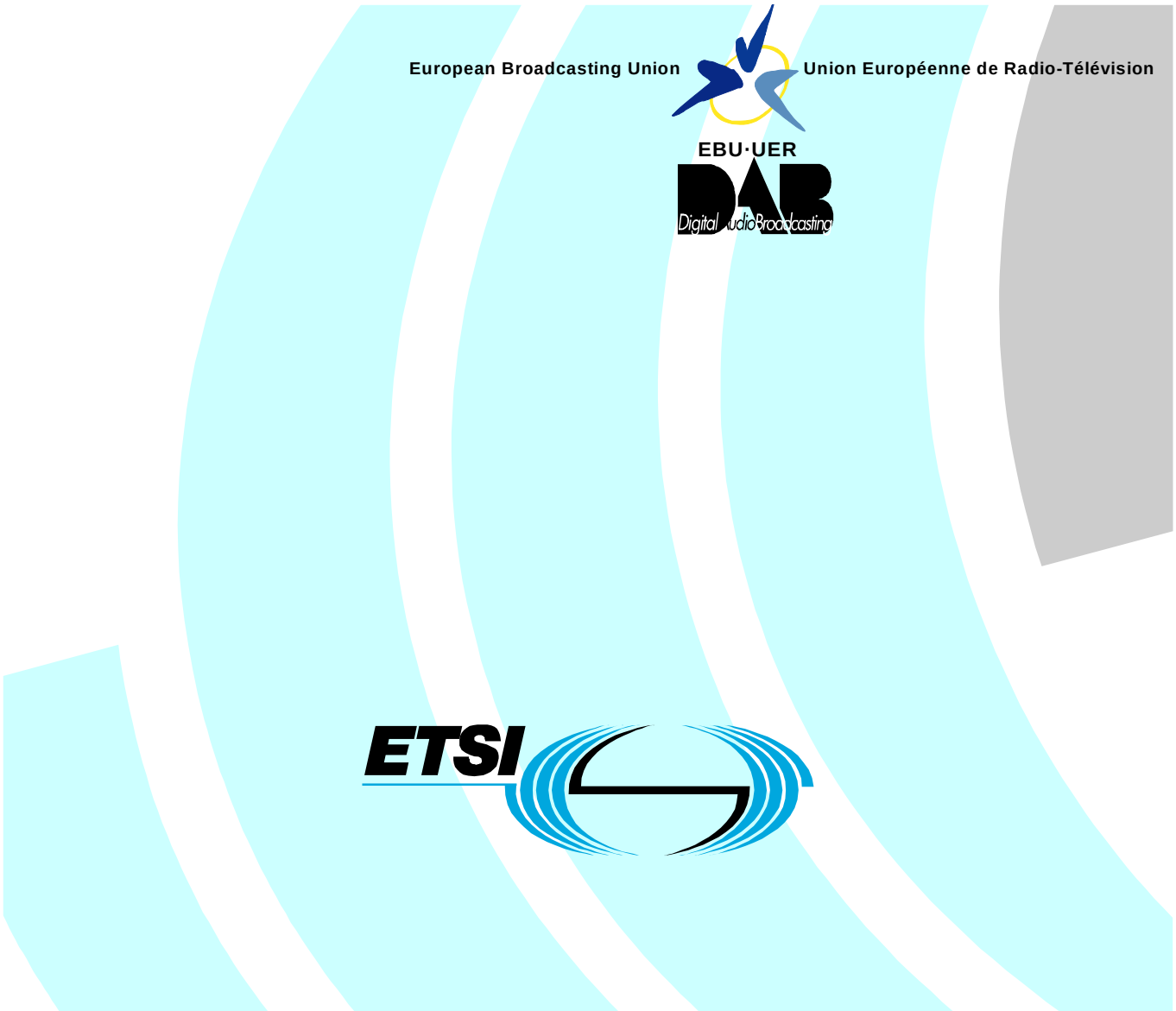
ETSI TS 101 993 V1.1.1 (2002-03)

Technical Specification

Digital Audio Broadcasting (DAB); A Virtual Machine for DAB: DAB Java Specification



Reference
DTS/JTC-DAB-27

Keywords
API, audio, broadcasting, DAB, data, digital

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

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the 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
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, send your comment to:
editor@etsi.fr

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document 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 (http://webapp.etsi.org/IPR/home.asp).

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.

Foreword

This Technical Specification (TS) has been produced by the 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: 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

Introduction

This clause contains an extension of the DAB specifications to provide a Java-based software framework for developing portable DAB data services applications.

A task force (Task Force Virtual Machine) was established inside the EUREKA 147 Consortium to discuss and to specify a Virtual Machine for executing applications. The concept of Virtual Machine is related to the requirements of providing a type of application that can be executed independently from hardware specific configuration.

Due to its strong flexibility and commercial success, Java was chosen as a base technology for solving the requirements of a portable environment and for specifying a set of API designed for the DAB environment.

A DAB extension to the Java API have been designed by the members of the Task Force for virtual Machine: such extension provides the software framework for designing, implementing and executing portable applications specifically targeted to the DAB system.

The DAB Java Framework is divided in three basics module or packages: a DAB specific extension of the Java API, a runtime support for the DAB applications execution environment, and a DAB I/O package for signalling the DAB Java extension over the DAB signal.

1 Scope

The present document specifies a DAB related API for Java. This API enables the download of Java programs via DAB and their control of their execution. Additionally, it provides an interface to the functionality of DAB.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication and/or edition number or version number) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies.

[1] ETSI EN 301 234 (V1.2.1 onwards): "Digital Audio Broadcasting (DAB); Multimedia Object Transfer (MOT) protocol".

[2] ETSI TS 101 812: "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification".

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

3 Definitions and abbreviations

3.1 Definitions

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

application controller: This entity is part of the runtime framework and is responsible for the control of a downloaded application. It acts as an intermediate between the application that initiated the download and the downloaded application.

command: transaction between a DAB client and the DAB package. It consists of a request, which is sent from the client to the package and confirmations and notifications, which are sent from the package to the client.

DAB resource: collection of hardware and software components that reside on a DAB terminal

NOTE: For example, hardware resources are audio, video, input devices, DAB receiver settings and commands; software resources are DAB Platform API access, DAB terminal API access, etc.

package: This is used throughout the text in two ways. First, it designates a Java package. Additionally, it also designates a component in DAB Java.

virtual component: PAD user application, which is treated like a regular (audio or data) component

3.2 Abbreviations

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

API	Application Programming Interface
DAB	Digital Audio Broadcasting
EPG	Electronic Programme Guide
FIG	Fast Information Group
IO	Input/Output
JDK	Java Development Kit
MOT	Multimedia Object Transfer
NPJP	Network-enabled Personal Java Profile
PAD	Programme Associated Data
SPJP	Standard Personal Java Profile
UA	User Apllication
VM	Virtual Machine (not only the Java Virtual Machine, but also in the sense of the whole runtime environment for DABJava)
WIRC	WorldDAB Information and Registration Centre, Wyvil Court, Wyvil Road, LONDON SW8 2TG, England, Tel: +44 171 896 90 51, Fax: +44 171 896 90 55, E-mail: worlddab-irc@worlddab.org

4 The DAB package

4.0 Summary

The DAB package enables applications (and applets) to access DAB resources. In clause 4.1 the basic and high-level communication concept is described. The former is based on the Event-Listener pattern. The latter consists of transactions, so called commands, in which the application sends requests to the package and the package responds with confirmations and notifications. In clause 4.2 the particular commands and resulting patterns are presented. The use of these commands is exemplified in clause 4.3. For each command there is a typical interaction. This is described in clause 4.4. In clause 4.5 the dependencies between the commands are explained. The commands can be only be used if the client is registered. This is shown in clause 4.6. Clause 4.7 consists of a description of the classes, which are contained in the package.

4.1 The communication concept

This clause describes the communication concepts for the use of the DAB package. The DAB package provides high-level access to the services of the Digital Audio Broadcasting (DAB) System. The package uses the Event-Listener pattern (see Bibliography, "Design Pattern, Element of Reusable Object-oriented Software by Erich Gamma") for the communication between the DAB system and the application. On top on this basic communication pattern a transaction concept is defined (e.g. to deal with ongoing events).

4.1.1 The communication between the application and the DAB package

The DAB package provides an asynchronous interface. If an application calls a method of `DABSource` (the basic interface), the result is not passed back as a return value. Instead the method initiates a new transaction and just returns. This transaction will generate events for intermediate or final results that are passed to the application using the `DABListener` interface.

This mechanism for passing back results to the application, follows the Event-Listener pattern. The package acts as a source for events that are distributed to all components which implement the `DABListener` interface and which are registered as listeners.

The following piece of code shows, how to use this in an application:

```
import dab.*;

public class MyApplication extends DABAdapter {
  private DABClient dabClient=new DABClient();

public void start() {
  // Configuration
  dabClient.open();
  // Registration
  dabClient.addDABListener(this);
  …

  // Initiate request for object
  dabClient.selectObjectReq(…);

  …
  dabClient.removeDABListener(this);
}

public void selectObjectCnf(SelectObjectCnfEvent e) {
  …
}
```

The class `MyApplication` is subclassing `DABAdapter` to act as a `DABListener`. `DAPAdapter` is an auxiliary class, which implements the `DABListener` with empty methods. The variable `dabClient` is used to interface to the DAB package. Before the client can be used, the connection to the receiver has to be set up using `open`. The next step is to register the application, so that it gets events. This is achieved by calling `addDABListener`. When we initiate a request like `selectObjectReq` (see table 1), the package will call our `selectObjectCnf` method for events of type `SelectObjectCnfEvent`. In the end the application calls `removeDABListener` to stop any event distribution to itself.

4.2 Commands

The basic transaction model in the DAB package is that a client issues requests and the DAB package responds with confirmations and notifications. Such a transaction is called a command. A command is initiated by a request and is finalized by a confirmation.

If notifications are sent while a command is executed, the notification informs about the progress of the transaction. Notifications are also used in situations when a DAB client requests particular information which cannot be delivered immediately or in situations where updates may occur after the first request for information has been satisfied. In the latter case notifications are delivered after the confirmation until the command is explicitly cancelled.

Table 1: Commands

Command	Request	Notification (in between)	Confirmation	Notification (following)
Pattern	**Req**		**Cnf**	
Tune	tuneReq	-	tuneCnf	-
GetEnsembleInfo	getEnsembleInfoReq	-	getEnsembleInfoCnf	-
GetServiceInfo	getServiceInfoReq	-	getServiceInfoCnf	-
GetComponentInfo	getComponentInfoReq	-	getComponentInfoCnf	-
SelectComponent	selectComponentReq	-	selectComponentCnf	-
SelectApplication	selectApplicationReq	-	selectApplicationCnf	-
SelectComponentStream	selectComponentStreamReq	-	selectComponentStream Cnf	-
Pattern	**Req**	**Ntf**	**Cnf**	
Search	searchReq	searchNtf	searchCnf	-
Scan	ScanReq	scanNtf	scanCnf	
Pattern	**Req**		**Cnf**	**Ntf**
SelectSI	selectSIReq	-	SelectSICnf	siNtf
SelectReceptionInfo	selectReceptionInfoReq	-	selectReceptionInfoCnf	receptionInfoNtf
SelectObject	selectObjectReq	-	selectObjectCnf	objectNtf
GetLocationInfo	getLocationInfoReq	-	getLocationInfoCnf	locationInfoNtf
OperationControl	operationControlReq	-	operationControlCnf	ServiceFollowingNtf, drcModeNtf
SystemFailure				SystemFailureNtf

To summarize this, commands are executed by sending requests, confirmations and notifications. Three different command patterns are used in the DAB package (see figure 1). All commands are listed in table 1 sorted by the pattern type (endings with Req = request, Cnf = confirmation, Ntf = notification). The patterns describe only the message sequence for one command. If commands are interleaved which means two commands running at the same time an arbitrary sequence of message types is possible.

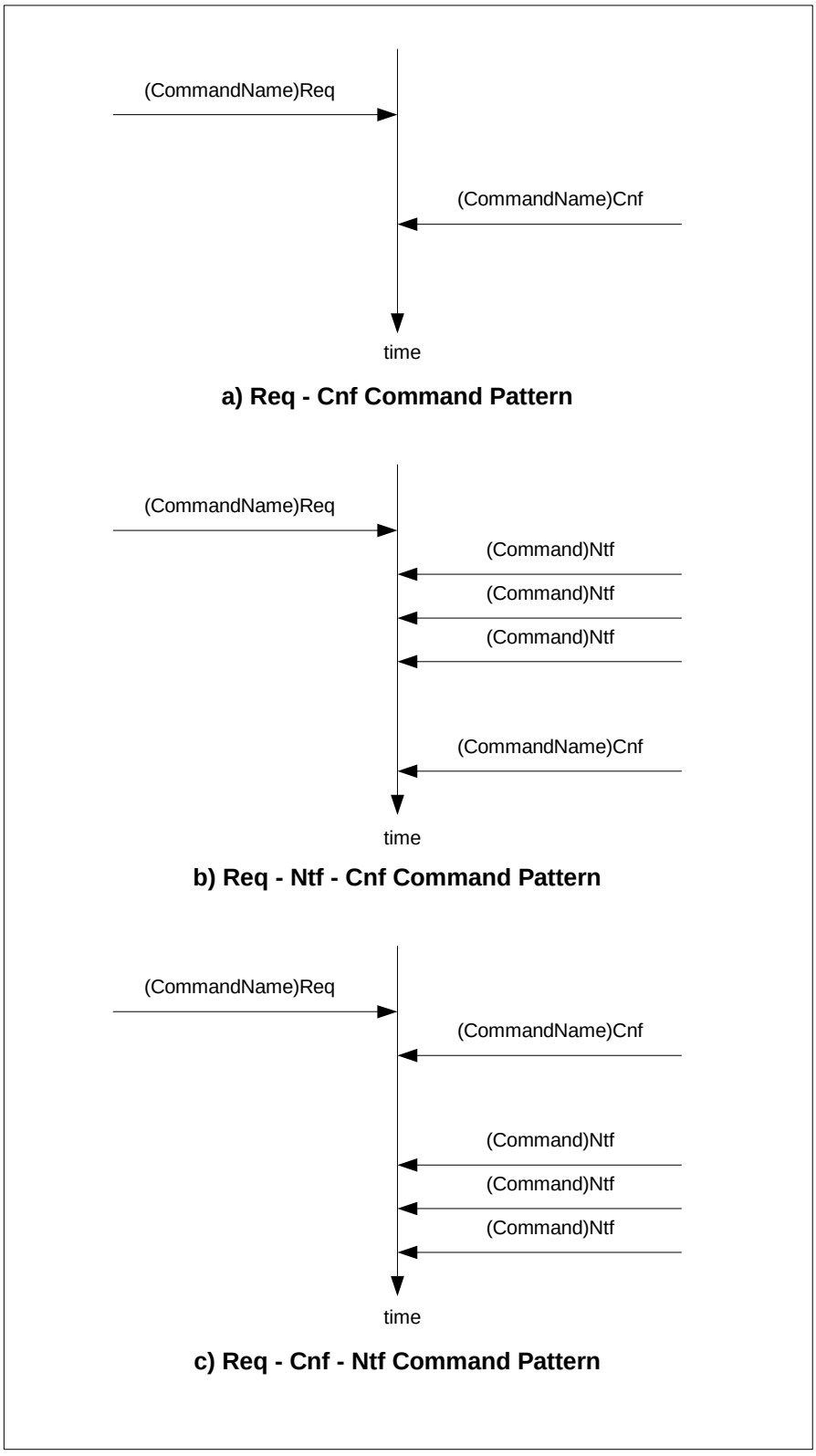


Figure 1: Command patterns

The asynchronous messaging approach fits well with the requirements of a broadcast based information system. In a broadcast system only unidirectional communication from the service provider to the client is possible. Therefore clients have to be prepared that changes might occur at anytime. A DAB ensemble is able to provide several services simultaneously. The number of services or their type can change at anytime. Also the reception conditions of a DAB receiver in a mobile environment might change very often. The DAB package keeps track on all these dynamic aspects of a broadcast system and informs a connected client by sending update notifications.

Interleaved started commands are also processed interleaved as far as the semantic of the commands allows this, e.g. it is possible to change the volume while a SelectObject command is pending. If two commands cannot be processed interleaved the one which has been sent first is also processed first.

4.3 Examples

The following examples demonstrate the use of commands in typical areas like service information access and service presentation. In the first example a simple EPG is described. The second example shows a stock market ticker.

4.3.1 EPG

In the following we will show how to use the Java DAB interface for an EPG application. We will focus on the main steps for initializing and controlling the DAB system. Basically, we need only two main classes: a `DABListener` (usually on the application side, in our case the EPG class) and a `DABClient` (the main entry for controlling the DAB receiver - see figure 2).

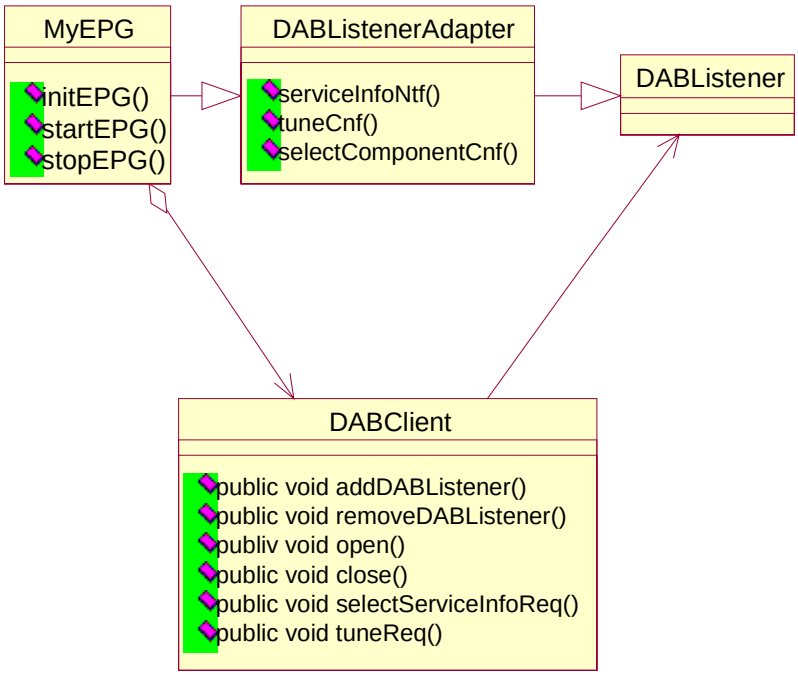


Figure 2: The classes of the EPG

Before making any actions to the DAB receiver we have to initialize it, and register a `DABListener` (or an adapter class) for the incoming messages: in our case the EPG application class implements the listener interface directly.

This leads to the following initialization code:

```
public class MyEPG implements DABListenerAdapter {
      static public void main(String[] args){
            ...
            initEPG();
            startEPG(frequency);
            ...
            pause EPG();
      }
}
```

The implementation of the main MyEPG methods are:

```
public void initEPG(){
      DABClient dab = new DABClient();
      dab.addDABListener(this);

      try {
            System.out.println("Sending Open Request ....");
            dab.open();
            System.out.println("... Open Request sent");
      }catch(DABException e){
            System.out.println("Caught exception during open
            request:"+e.toString());
      }
} // end initEPG() method

public void stopEPG(){
   try {
     dab.close();
     dab.removeDABListener(this);
```

```
   }catch(DABException e){
            System.out.println("Caught exception during open
            request:"+e.toString());
   }
   finally{
     System.exit(0);
   }
} // end stopEPG() method
```

NOTE 1: The `open()` method is synchronous, i.e. the application is blocked until the method returns.

NOTE 2: The registration procedure (`dab.addDABListener(…)`) should be done in relation to an `open()`, but absolutely before using any asynchronous methods (see the EPG example).

NOTE 3: The closing procedure is symmetric to the opening: first we close the connection to the DAB system and then we annul the registration as a `DABListener`.

After the initialization steps we implement the basic exchange of messages for controlling the audio services in the DAB ensemble.

We will focus our attention on the following set of asynchronous methods:

on the DAB side:

- `dab.tuneReq(frequency, mode);`
- `dab.selectServiceInfoReq(true,true,true,true);`

on the `DABListener` side:

- `public void selectServiceInfoCnf(SelectServiceInfoCnfEvent e)`
- `public void serviceInfoNtf(ServiceInfoNtfEvent e)`
- `public void tuneCnf(TuneCnfEvent e)`

with the events:

- `TuneCnfEvent`
- `SelectServiceInfoCnfEvent`
- `ServiceInfoNtfEvent`

After tuning to a specific frequency, we register `MyEPG` for receiving notification messages about the available services on the DAB ensemble (`EnsembleInfo`, `ServiceInfo`, `ComponentInfo`). The usage of the information received by the DABClient is a task that is specific to the application. Here, we demonstrate it selecting an audio component.

```
public void startEPG(int frequency){
   try {
            dab.tuneReq(
                  frequency, DABConstants.transmissionModeAutomatic);
   }catch(DABException e){
   }
```

```
   // to be continued
}


// DABListener interface
public void tuneCnf(TuneCnfEvent e){


   int result = e.getResult();
   int tunedFrequency = e.getTuneFrequency();


   System.out.println(
             "Tune cnf received,
             result ="+Integer.toString(result)+"+
             Frequency = "+Integer.toString(tunedFrequency));
   return;
}
```

A tune request message is sent for tuning to a specific frequency (in Hz), using a specific mode (see note 4). If it is successful, a tune confirmation message is delivered and the receiver is tuned to the requested ensemble (see note 5).

NOTE 4: In the DAB specification several modes are specified for the transmission of a DAB ensemble: some DAB receivers can automatically detect the specific transmission mode of an ensemble, in others such a parameter has to be done explicitly (see DAB specification).

NOTE 5: Other more sophisticated tuning actions can be done using the scanReq() method (see DAB Java API).

```
public void startEPG(){
   // continued
   try {
             dab.selectServiceInfoReq(true,true,true,true);
   }catch(DABException e){
   }
}


// DABListener interface
public void serviceInfoNtf(ServiceInfoNtfEvent e){
   int notificationType;
   EnsembleInfo ensembleInfo;
   ServiceInfo  serviceInfo;
   ComponentInfo componentInfo;
```

```
   notificationType = e.getNotification();

  switch(notificationType){
   case DABConstants.notificationEnsembleAdded:
   case DABConstants.notificationEnsembleRemoved:
   case DABConstants.notificationEnsembleChanged:
      ensembleInfo = e.getEnsembleInfo();
      // notify the Application of an Emsemble info
     break;

   case DABConstants.notificationServiceAdded:
   case DABConstants.notificationServiceRemoved:
   case DABConstants.notificationServiceChanged:
      serviceInfo = e.getServiceInfo();
      // notify the Application of a Service info
     break;

   case DABConstants.notificationComponentAdded:
   case DABConstants.notificationComponentRemoved:
   case DABConstants.notificationComponentChanged:
      componentInfo = e.getComponentInfo();
      // notify the Application of a Component info
     break;

   }
  return;
}
```

The application uses `selectServiceInfoReq` for receiving information about all available DAB components (ensemble, services, components). After receiving a confirmation of the request, the `DABClient` will notify every change in the DAB signal information (addition, changing, and removing of ensemble, services, and components). Specifically in this case we ask to receive with the notification the information related to the particular info object (see DAB Java specification for details).

The information is delivered to the application using a special event; the usage of the carried information depends on the application strategy.

The final step for our simple EPG is to select a particular audio component, supposed that we have collected all the information about the services and services components available for the selected ensemble. We assume that the user has selected a service component somehow and that the EPG has identified the selected component.

```
public void selectAudio(ComponentInfo componentInfo){
      if(componentInfo.getType() ==
         DABConstants.componentTypeForegroundSound)
      {
      try {
       dab.selectComponentReq(
          componentInfo.getId(),
             DABConstants.selectionModeReplace);
             }catch(Exception _e){
             }
      }
}

// DABListener interface
public  void selectComponentCnf(SelectComponentCnfEvent e){
      System.out.println(
      "Result"+Integer.toString(e.getResult()));

      return;
}
```

4.3.2 Ticker

The next example, a stock market ticker, demonstrates how DAB Java applications can access data broadcast on DAB. The ticker consists of two classes - as it is displayed in figure 3. The `Ticker` class is the application's envelope, which it is responsible for the application lifecycle and the presentation of the information. The `Decoder` uses the DAB interface to receive the information and retrieve the content from the delivered data objects. `Ticker` communicates with `Decoder` uses the event-listener pattern. Additionally, it controls the lifecycle of the decoder. In the remaining part of this clause we will only show, how the decoder is implemented - as the Ticker class is not dependent on the DAB interface.

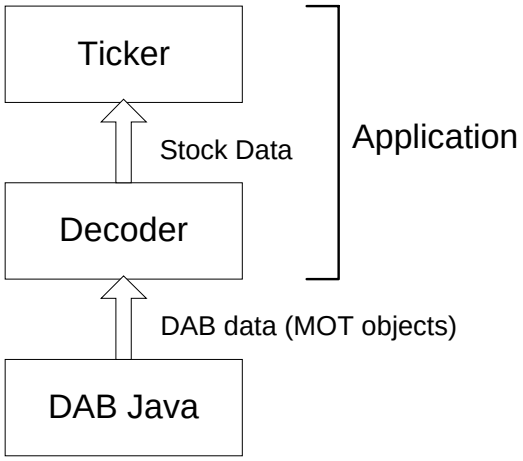


Figure 3: The architecture of the ticker

The decoder is implemented as follows:

```
import dab.*;

public class Decoder extends DABAdapter {
    private DABClient dabClient;
    private StockListener listener=null;
    private ServiceInfoId serviceId=null;
```

The variable `dabClient` contains the interface to DAB. `listener` contains the receiver of stock events (i.e. a `Ticker` object). The `serviceId` specifies the identifier of the DAB service which contains the stock data.

```
    public Decoder() {
        dabClient = new DABClient();
    }
```

When the decoder is created, we also create a DAB client. Note, that the client is not yet configured. This means the decoder is ready for decoding, but no actual action is taken.

```
    public void startDecoding(ServiceInfoId serviceId) throws Exception{
        this.serviceId = serviceId;
        dabClient.open();
        dabClient.addDABListener(this);
```

```
        dabClient.selectComponentReq(serviceId,
                                     DABConstants.selectionModeAdd);
    }
```

The decoding is started with `startDecoding`. First the DAB client is configured. The `open` call sets up the connection to the receiver. After that the decoder registers itself as a listener of DAB events (it is a subclass of `DABAdapter`, which is an adapter class of the `DABListener` interface). Then, the service is selected using `selectComponentReq`. We will assume that the ticker service will follow the slideshow user application model. This means objects will be delivered after the confirmation `selectComponentCnf` is sent. Here, the confirmation is ignored (the method needs not to be implemented as the default implementation in `DABAdapter` exactly behaves like that).

```
    public void stopDecoding() throws Exception {
        dabClient.selectComponentReq(serviceId,
                                     DABConstants.selectionModeRemove);
        dabClient.removeDABListener(this);
        dabClient.close();
    }
```

In `stopDecoding` we do the reverse setup. First the service is stopped and then the client is shut down.

```
    public void addStockListener(StockListener listener) {
        this.listener = listener;
    }

    public void removeStockListener(StockListener listener) {
        this.listener = null;
    }

    private void notifyStockEvent(StockEvent e) {
        if (listener != null)
            listener.stockEvent(e);
    }
```

The communication with the `Ticker` object uses the event-listener model. Thus, the `Ticker` object has to register itself as a listener calling `addStockListener`. With removeStockListener the object will no longer receive events. Stock events are distributed using notifyStockEvent.

```
    public void objectNtf(ObjectNtfEvent e) {
      synchronized (this) {
       Stock[] stocks;
```

```
      try {
        stocks = decode(((MOTObject)e.getObject()).getBody());
      } catch (Exception _) {
        return;
      }

      for (int i = 0; i < stocks.length; i++)
        notifyStockEvent(new StockEvent(this, stocks[i]));
    }
  }
```

When `startDecoding` was called, the decoder will receive object notifications, which means that the method `objectNtf` is called from the DAB VM. First, we will decode the delivered DAB object. Note, that it needs to be cast to a `MOTObject`, because we assume the stock data is transported using the MOT protocol [1]. The result is a list of stocks that are delivered as StockEvents one by one.

4.4 Command Types

The commands supported by the DAB package can be categorized as follows.

- Selecting an Ensemble:
  - Tune: Tune directly to a specified frequency.
  - Search: Search for an Ensemble.
- Accessing Service Directory:
  - SelectSI: Subscribe to Service Directory information.
  - GetEnsembleInfo: Get information about a specified ensemble.
  - GetServiceInfo: Get information about a specified service.
  - GetComponentInfo: Get information about a specified component.
- Monitoring Reception Conditions:
  - SelectReceptionInfo: Subscribe to Reception Condition information.
- Selecting Services:
  - SelectComponent: Start or stop a service. In case of an audio service decoding of audio samples is started automatically. In case of a data service, the service can be accessed with the SelectObject command.
  - SelectApplication: Launch a Java application.
  - SelectComponentStream: Get access to the packet stream of the component.

- Selecting Objects:
  - SelectObject: Request data objects for delivery with or without automatic updating.
- Scanning for DAB Services:
  - Scan: Scan a specified frequency range for DAB Ensembles and update the Service Directory.
- Miscellaneous:
  - OperationControl: access and modify parameters of the receiver.
  - GetLocationInfo: retrieve location information from the receiver.

In the following clauses the typical use of these command types is presented (the launch of Java applications is explained in the runtime package). Note, that in the message sequence charts the arguments of the calls do not represent actual parameters. Only qualitative information is shown to simplify the charts.

4.4.1 Tuning

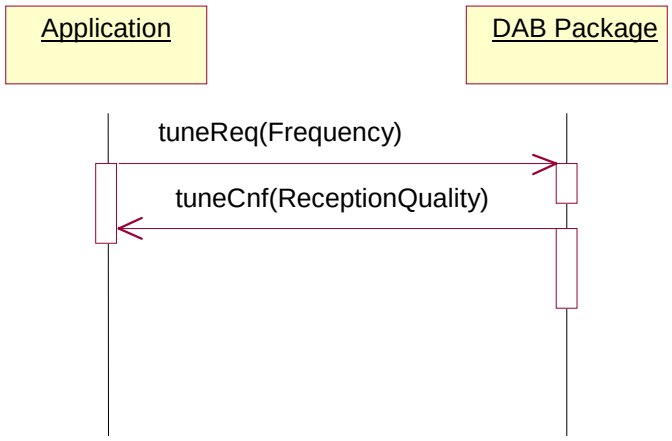


Figure 4: Tuning to an ensemble

The receiver is tuned by calling `tuneReq`. The receiver will tune to the requested frequency and respond afterwards with `tuneCnf` confirmation. The confirmation contains information about the reception quality.

4.4.2 Searching

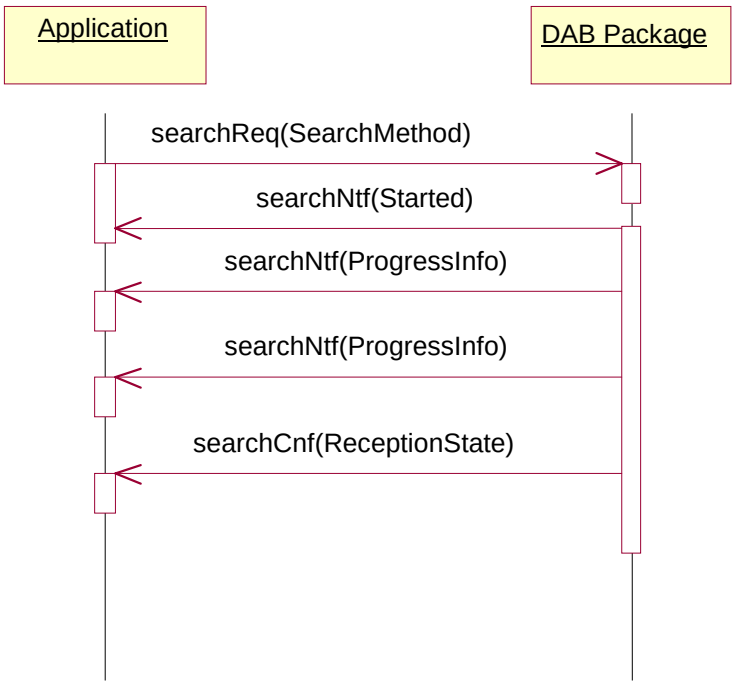


Figure 5: Searching for an ensemble

To search for some ensemble, the application calls `searchReq`. The package will respond with a notification that the search has started. Other notifications are sent in between depending on the search method (e.g. a 16 kHz step was made). The transaction ends with a `searchCnf` confirmation containing the resulting state of the search process.

4.4.3 Scanning

Scanning means looking for ensembles on a specified range. Essentially, it is like searching except that the scanning process looks for all ensembles in the range. When the command has been issued, notification will be sent, after the scanning has been started. Further notifications are sent during the scan, which inform about the progress. When the scan is terminated, a confirmation is sent, which contains information about the scan and the state of the receiver.

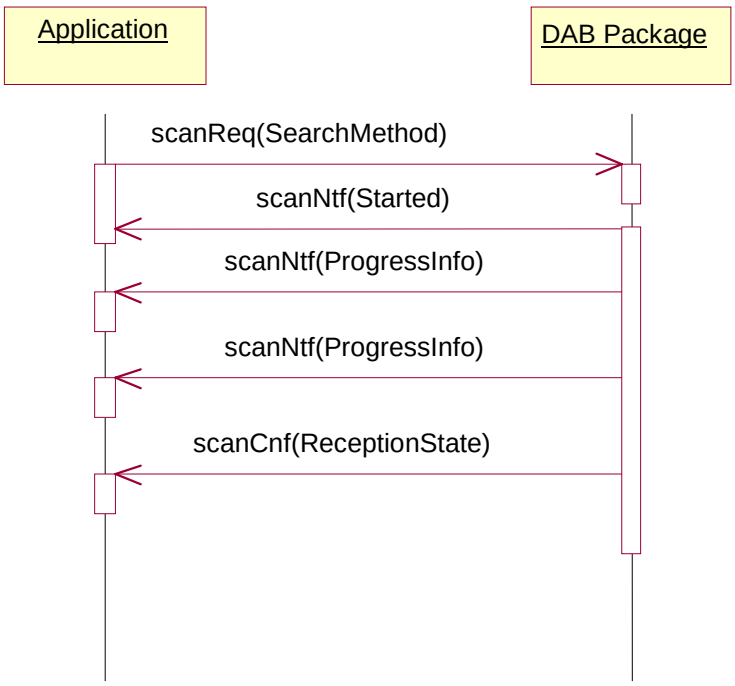


Figure 6: Scanning

4.4.4 Accessing service directory information

Figure 7: Accessing service directory information

The application, which likes to subscribe to information about the service directory of the tuned ensemble, calls `selectSIReq` specifying the service element type.

After the confirmation is sent, the package will transmit notifications back to the application. New service elements are indicated in the notification with the flag "`Added`", for elements that have changed "`Changed`" is set and for elements that are removed "`Removed`" is set.

The application finishes the subscription calling `selectSIReq`.

Note, that application can determine, whether it likes to get the respective objects of the service directory (e.g. the ensemble information) directly using this mechanism or indirectly using the other mechanism which is shown in the next clause.

4.4.5 Accessing service information

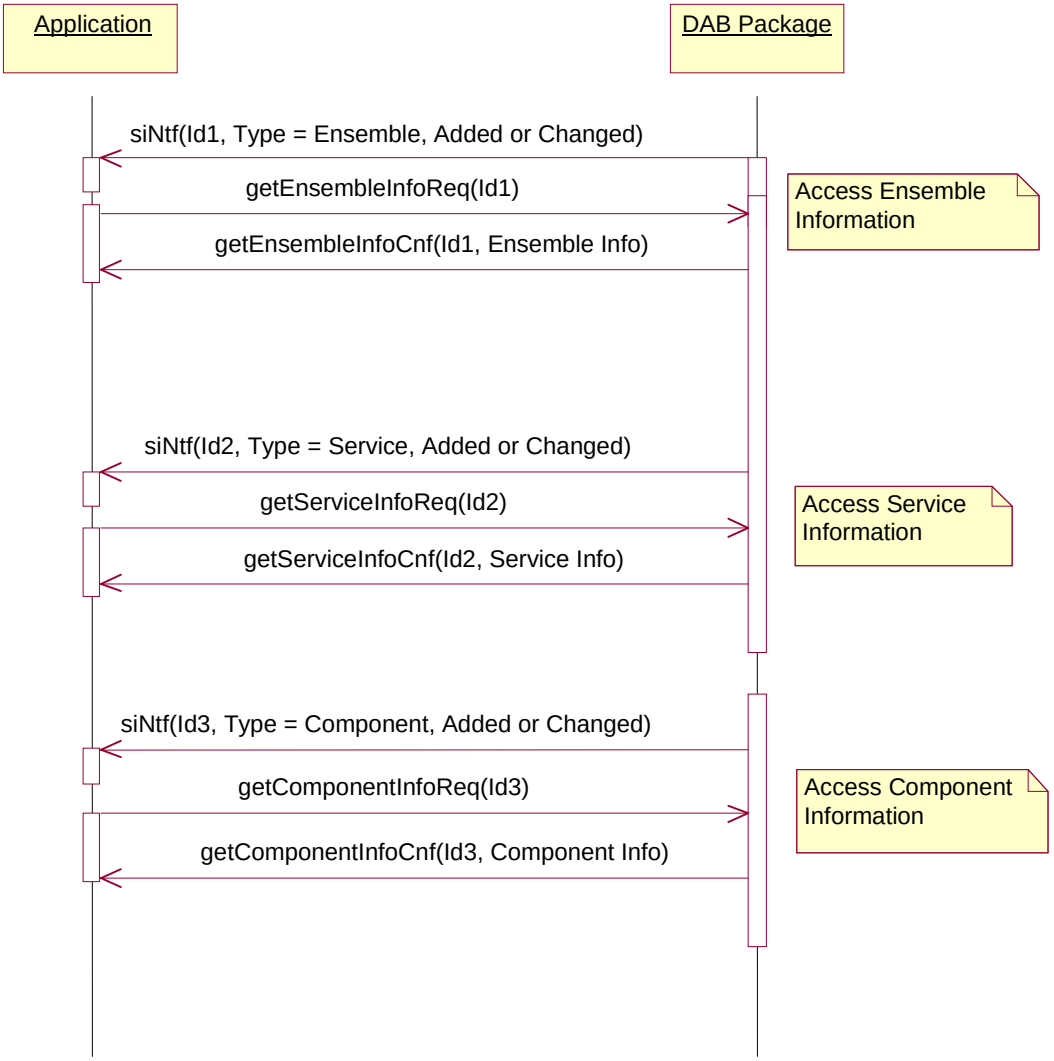


Figure 8: Accessing service information

Apart from getting service information directly (see the previous clause), the application can also use the ServiceInfo command to retrieve the respective service information objects. It has to specify the service identifier in the `siReq` request. The confirmation will then contain the requested object.

4.4.6 Monitoring reception quality

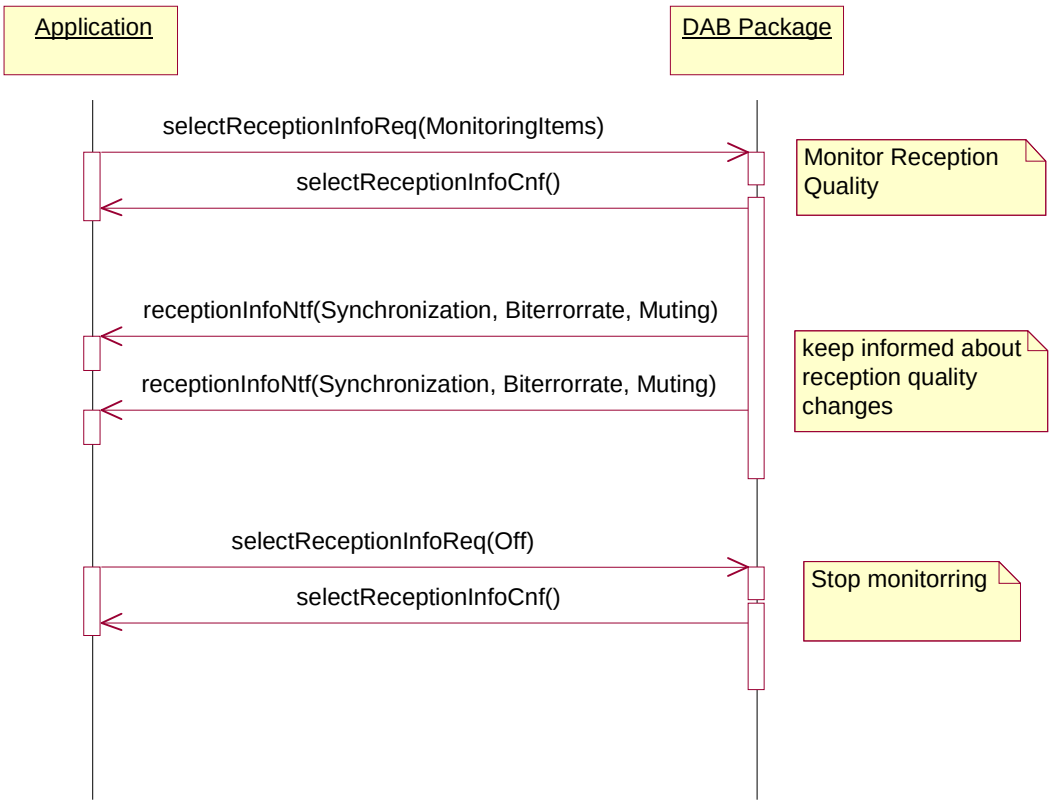


Figure 9: Monitoring reception quality

The reception quality can be monitored using the SelectReceptionInfo command. The application has to make a `selectReceptionInfoReq` request specifying what parameters are monitored. Then it will receive `receptionInfoNtf` notifications as long as the monitoring is not stopped (`selectionReceptionInfoReq(Off)`).

4.4.7 Selecting an audio service

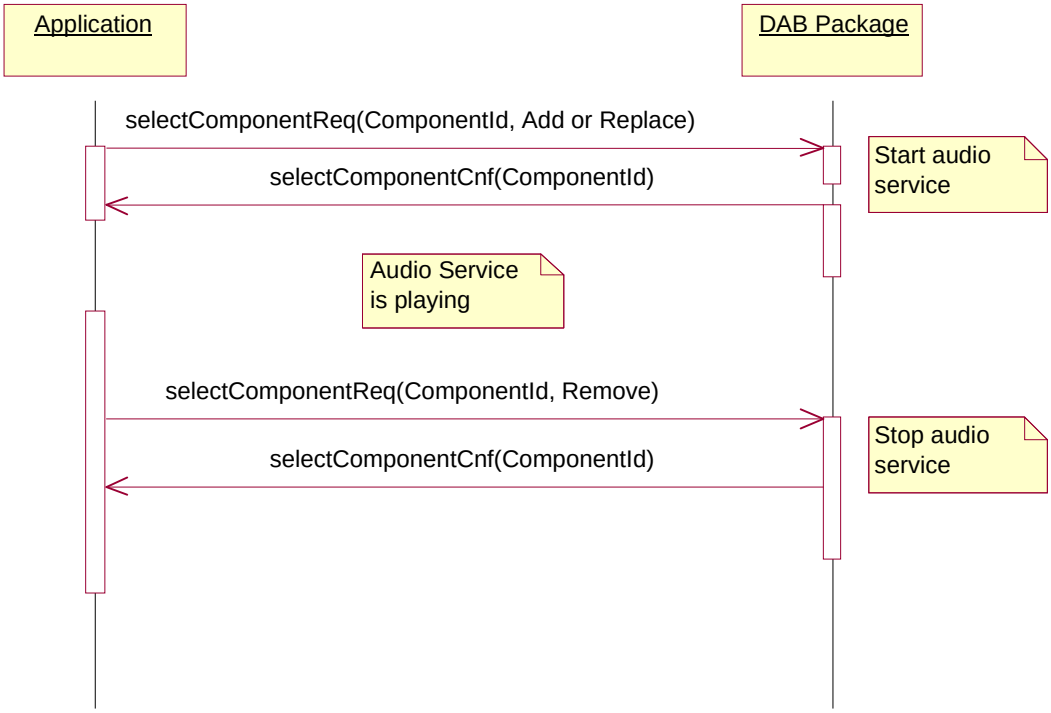


Figure 10: Selecting audio service

An audio service is started with the SelectComponent command. The application calls selectComponentReq passing the identifier of the audio component. The package will start the audio service and sends back a confirmation.

To stop this audio service, the application calls selectComponentReq again now specifying that the component has to be removed. When the package responds with a confirmation, the audio service was stopped.

4.4.8 Selecting a slideshow or a dynamic label service

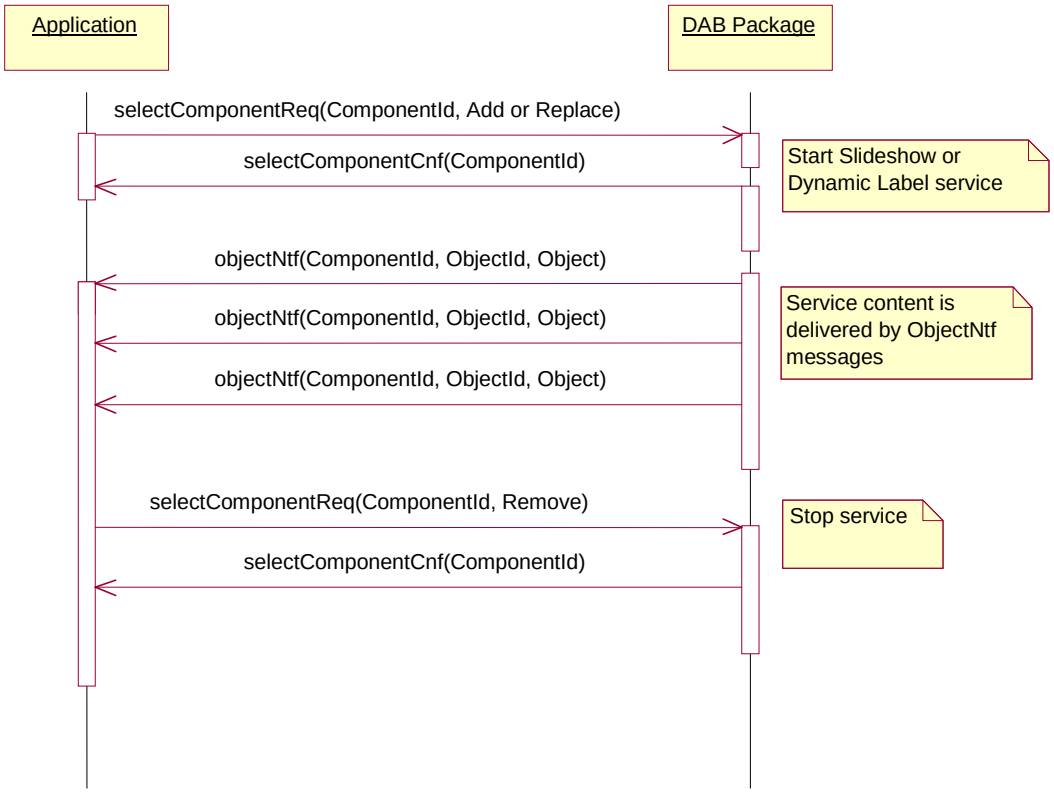


Figure 11: Selecting a slideshow or a dynamic label service

An application selects a slideshow or a dynamic label service with the SelectComponent command. When the request `selectComponentReq` with the respective service identifier is issued, the service will be started and a confirmation is sent back. The application will then receive `objectNtf` notifications containing objects of the service. To stop the service, `selectComponentReq` is called again setting `selectionMode` to `selectionModeRemove`. The removal of the service will be confirmed.

4.4.9 Selecting a broadcast website service

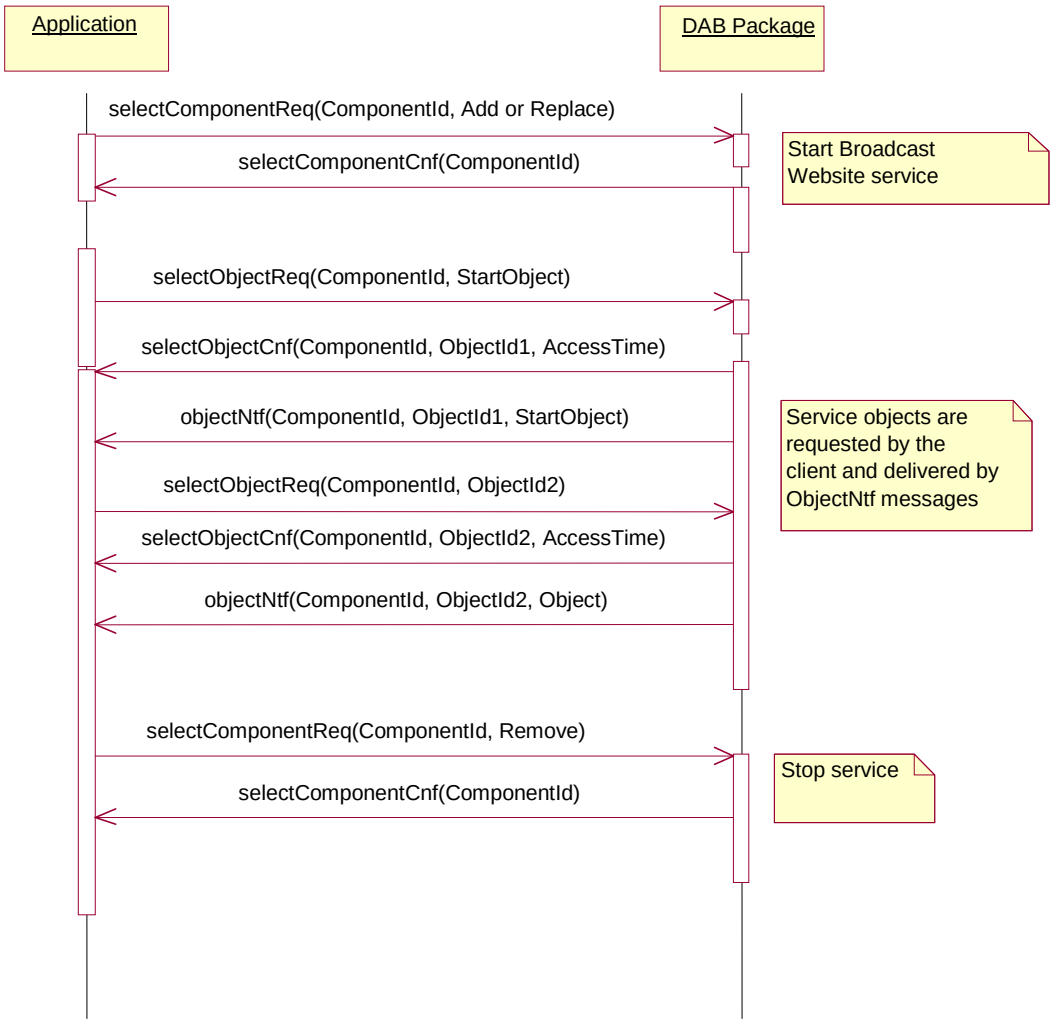


Figure 12: Selecting a broadcast website service

For running a Broadcast Website service the component has to be selected. This is accomplished calling selectComponentReq with the respective service identifier. The start of the service will be confirmed by the package.

The actual objects of the service are retrieved with the selectObject command. Usually, the start object is demanded first. For that, a `selectObjectReq` request is issued with the service identifier of the component and the object identifier of the start object. The DAB package will send back a confirmation including the likely access time. The actual object is received with an `objectNtf` notification. All other objects of the service are requested and delivered similarly.

The service is stopped calling `selectComponentReq` specifying the removal of the service.

Note, that the SelectComponent command can be used to improve the access time of the requested time (e.g. especially caching the objects of the service).

4.4.10 Selecting an object

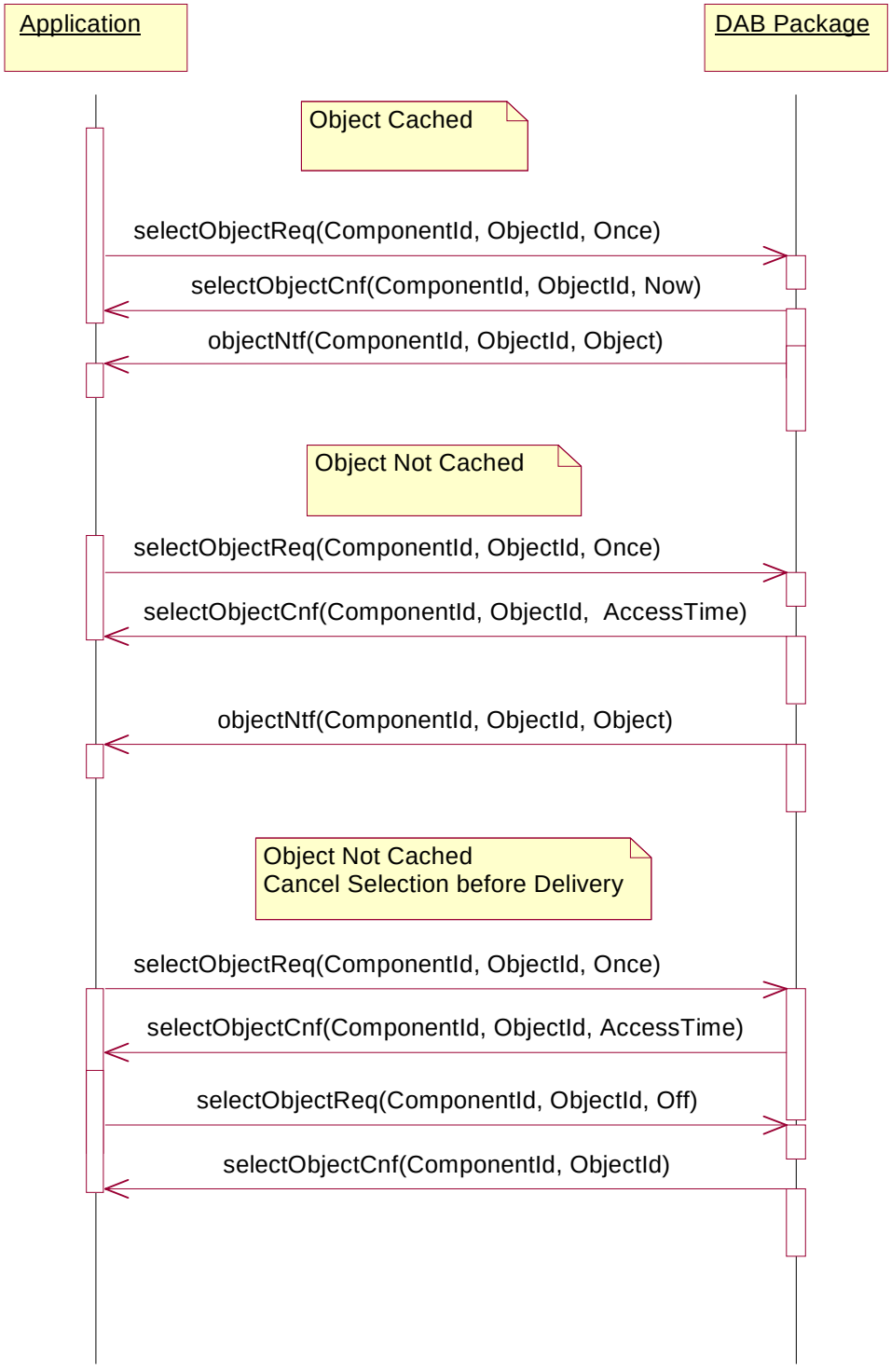


Figure 13: Selecting an object with selection mode Once

There are several cases for object selection depending on the selection mode, on the caching and on changing state between the selection modes. In figure 13 the simplest case is shown, in which an object is selected only one time.

The reaction of the package for the selection depends on whether the object is cached or not. If the object is cached, a confirmation is sent back indicating that the object is directly available. The actual object is delivered with the `objectNtf` notification. When the object is not cached, the confirmation will indicate the access time. The application may also cancel the selection of an object in between. For that, a SelectObject command is issued specifying "`Off`" for the selection.

The application may wish to get updates from an object which was selected with `SelectionMode=Once`. This behaviour is shown in the upper half of figure 14. Because of the change of the selection mode the application has to switch off the delivery of the object.

In the lower half of figure 14, it is demonstrated, that when the application requests an object another time just once, that was not delivered yet, the object will be delivered only one time.

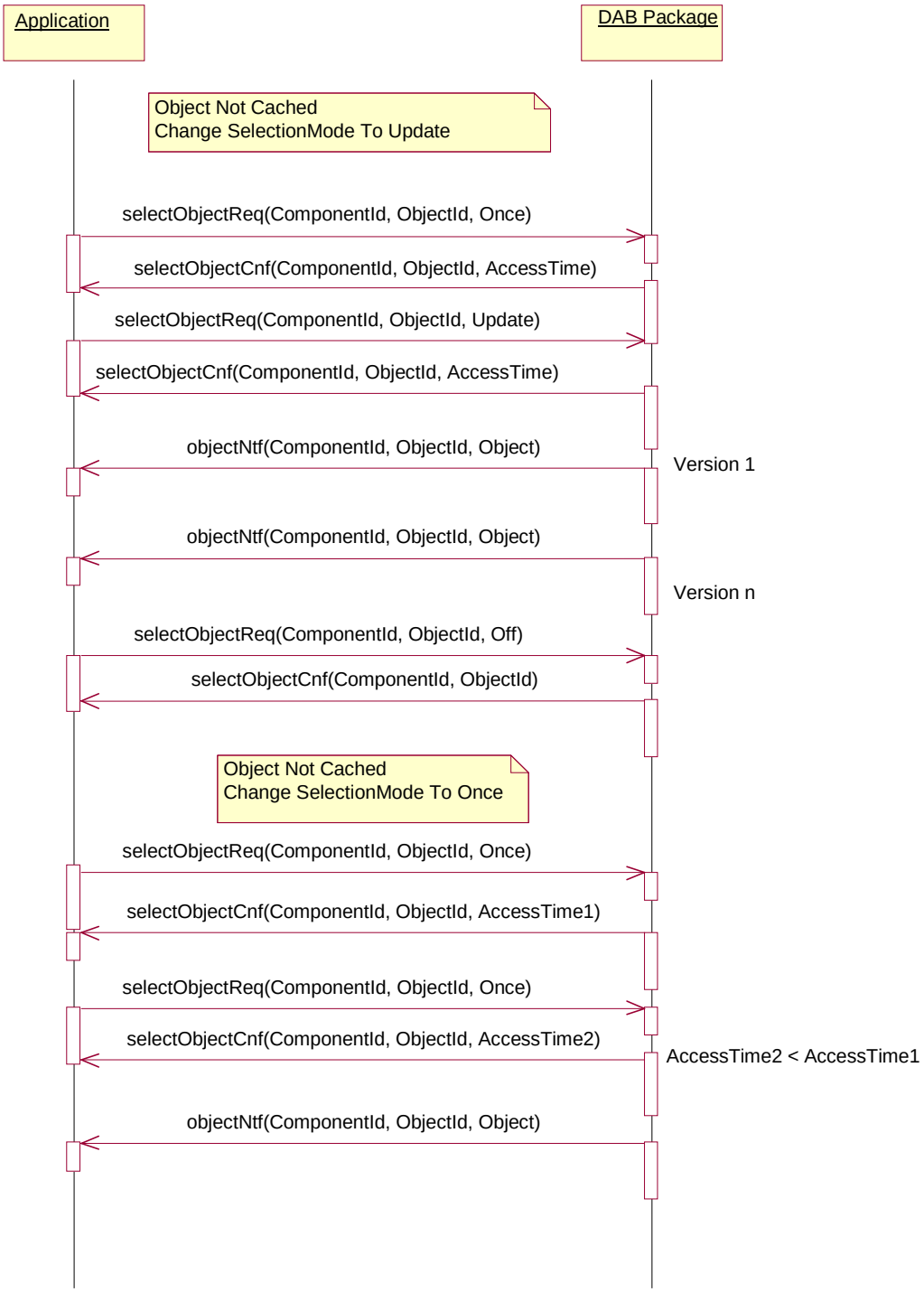


Figure 14: Selecting an object and changing selection mode from Once

Figure 15 shows the behaviour of the package, when the update selection mode was chosen. Note, that only one confirmation is sent to indicate the access time for the first version of the requested object.

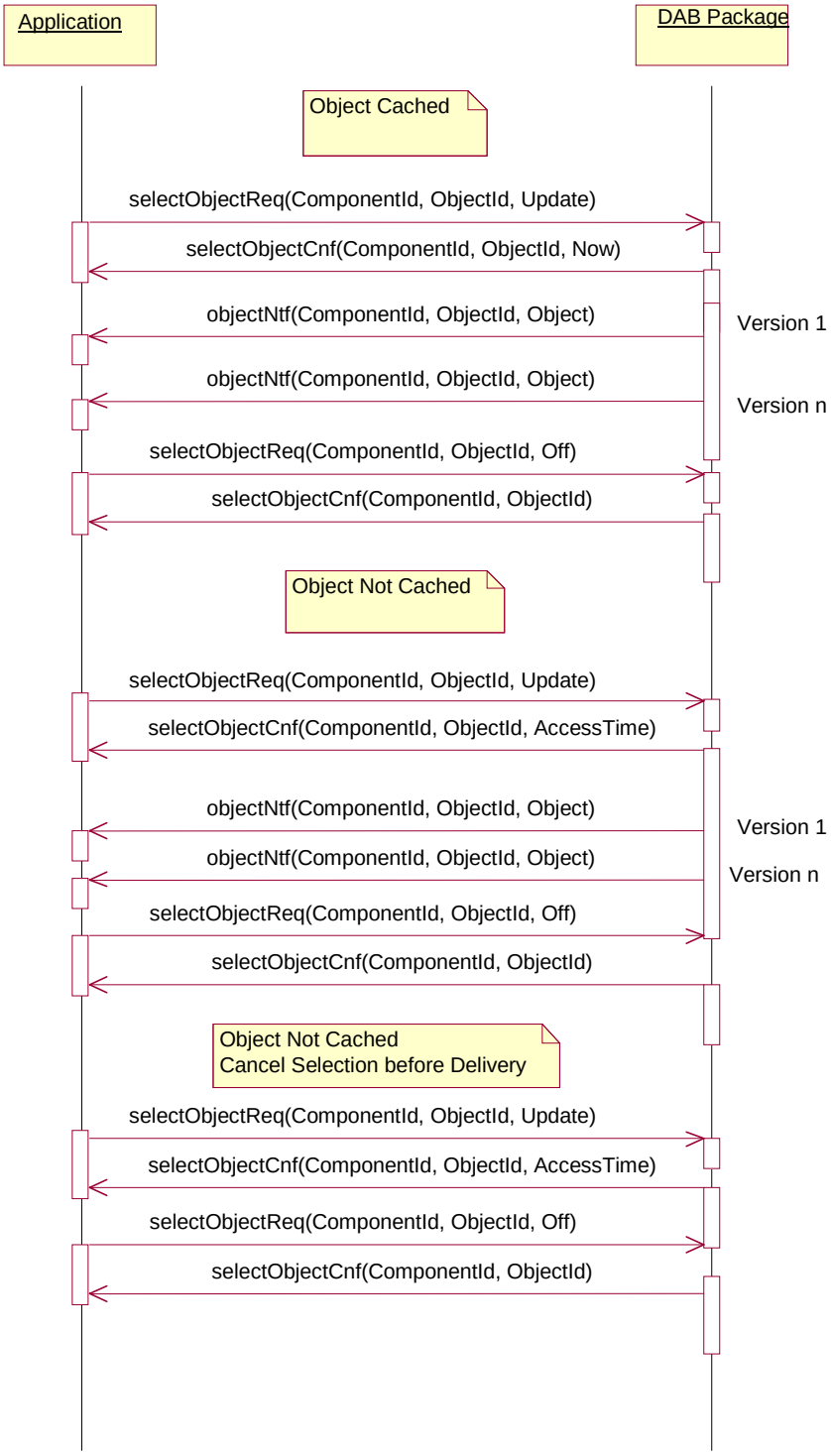


Figure 15: Selecting an object with selection mode Update

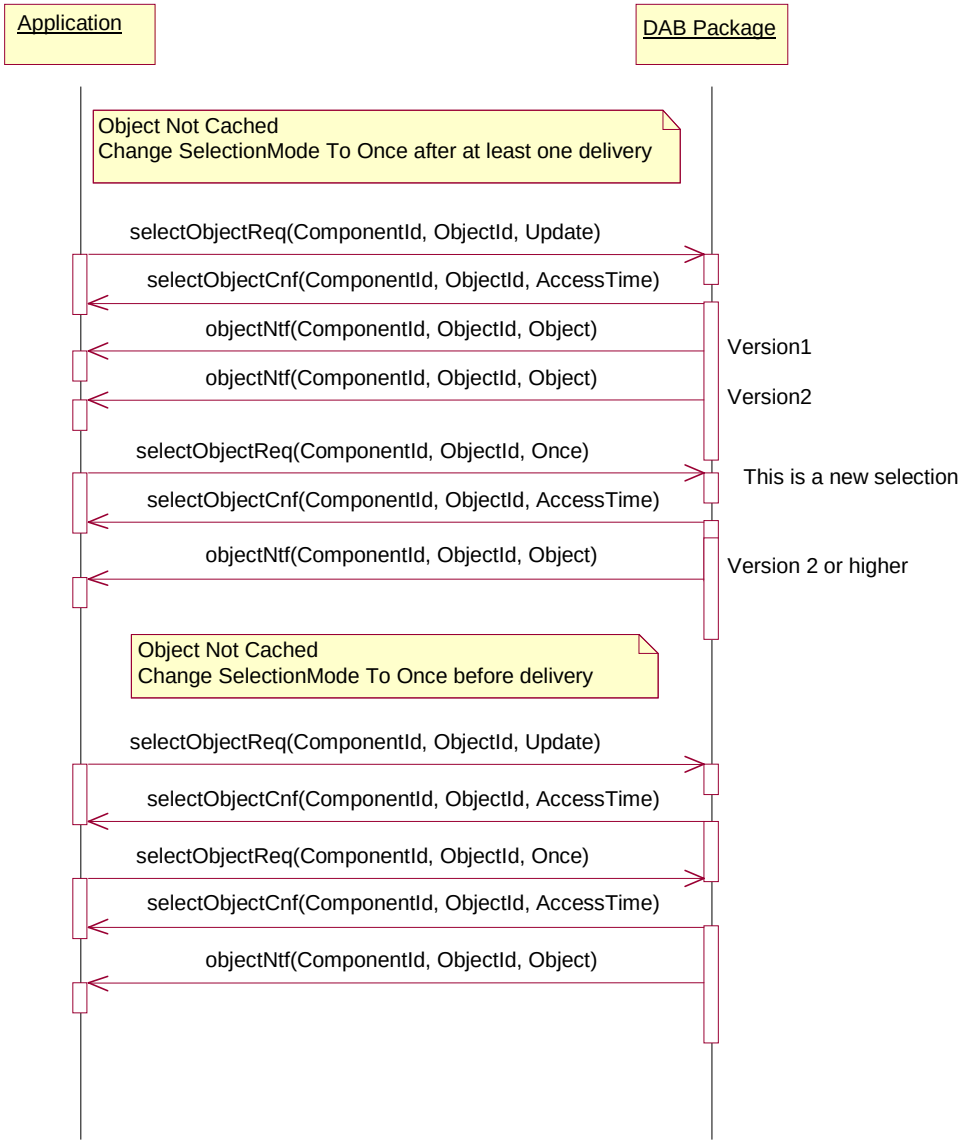


Figure 16: Selecting an object and changing selection mode from Update

In figure 16 the state change from selection mode `Update` to `Once` is shown. Note, that after the transition the object will be sent again.

4.4.11 Selecting a component stream

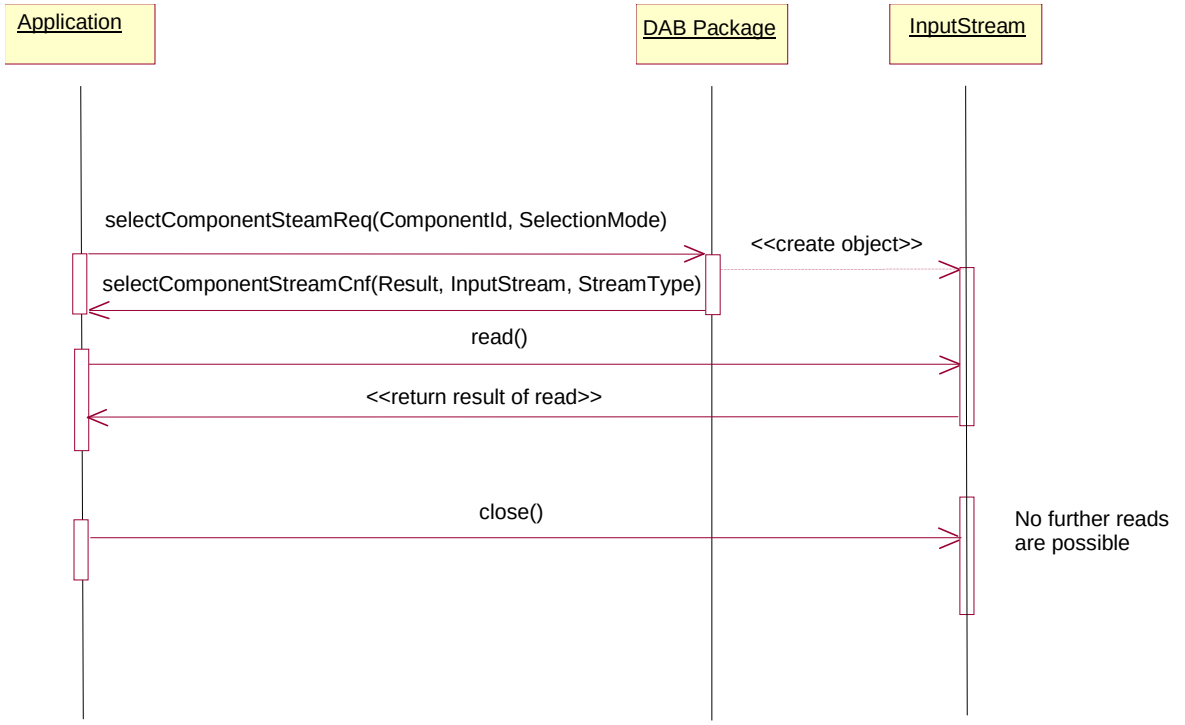


Figure 17: Selecting a component stream

Stream data can be accessed using the selectComponentStreamReq request. The confirmation that is sent back to the client carries an input stream object. That object provides stream data until `close` is called.

4.4.12 Operation control

Figure 18: Setting and getting operation attributes

In figure 18, it is displayed, how operation attributes are retrieved or are modified. The XXX is a placeholder for some attribute, e.g. Volume with the respective GetVolume and SetVolume operations.

In a get operation the given attribute value in the request is not considered. With regards to a set operation the given value will replace the current one, which is delivered back in the confirmation.

4.4.13 Retrieving location information

Figure 19: Retrieving location information

Location information can be retrieved in different modes. In the pull mode, LocationInfoOnce, only a single notification is sent. In the push modes, LocationInfoByTime and LocationInfoByDistance, are sent in regular intervals until the subscription is stopped.

4.5 Dependencies between the commands

Certain commands have influence on the outcome of other commands - mainly because the DAB receiver has a state. In particular:

- Current ensemble: If the current ensemble is modified (e.g. by the means of tune, search, or scan), all transactions which depend on this information are terminated. The exception from this rule is service information, which still can be received - even when the ensemble is changed. But as the information is not updated, only the old data is delivered (using the internal service directory).

- Selection of components: It is possible to select more than one data or stream component simultaneously. But only one audio component can be selected. A PAD component is only available, when the associated audio component was selected. Objects can only be requested when the respective component was selected.

4.6 Client registration

Clients have a state with respect to the used DAB receiver. They are either disconnected or connected. Transactions can only be processed in the connected state (otherwise an exception is thrown).

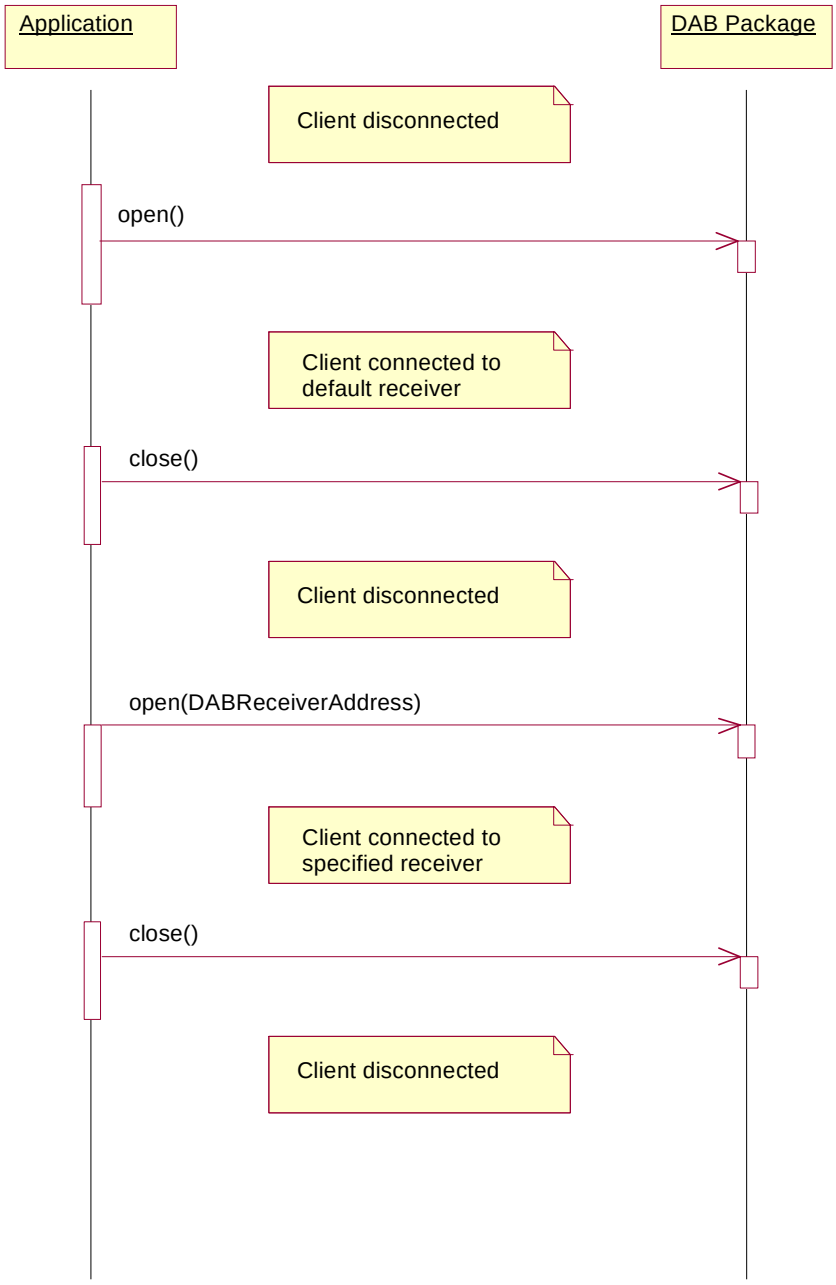


Figure 20: Client registration

A newly generated instance of `DABClient` is in the disconnected state. It is set to the connected state calling `open` in `DABClient`. It will be disconnected if `close` is called. This behaviour is depicted in figure 20. Note, that the second version only needs to be supported if multiple receivers are available.

4.7 The package structure

The classes of the `dab` package are shown in figure 21 (the details are given in the javadoc specification of the DAB Java package). Note, in contrast to the specification in the annex this and the following figures do not contain any classes or methods which are related to runtime issues (see the following clause).

`DABSource` and `DABListener` are the main classes. The `DABSource` interface contains the requests and the `DABListener` interface contains the notifications and confirmations.

Usually, these two classes will not be used directly. The `DABClient` class which implements `DABSource` is the standard interface to the DAB system. It also includes an interface to register new DAB listeners. The `DABAdapter` class provides an implementation of `DABListener` based on dummy methods, so that applets, which use only a part of the functionality of the DAB package, need not to implement all methods of `DABListener`. `DABReceiverAddress` is an abstract class that can be used to support multiple DAB tuners.

`DABException` is the superclass of all exceptions used inside the package. It is particularly used to generalize from all exceptions in `DABSource`. The `DABNotAvailableException` is thrown in the data classes when optional attributes are currently not available or not available at all. `DABConnectionException` is thrown, if there are problems with the connection to the receiver.

`DABConstants` contains all constants that are used in the rest of the package.

The package has three sub-packages: `si`, `data` and `events`. The package si contains classes describing service information; `data` contains various data classes that are not directly related to service information; `events` contains all event classes.

The structure of `si` is shown in figure 22. The class `SIId` is an abstract class for referencing service information. `EnsembleId`, `ServiceId` and `ComponentId`, which are all derived from `SIId`, represent the identifier for the respective service information type. The classes `EnsembleInfo`, `ServiceInfo`, and `ComponentInfo` reflect the respective levels in the hierarchy of service information.

In the `data` package (see figure 23) the other data classes are collected. `DABObject` is a generic class to represent all kind of data transported via DAB. `MOTObject` is a subclass of `DABObject` which models only data objects transmitted in the MOT protocol [1]. Additionally, there is derived class, `DLSObject`, for data of a DLS data service. `MOTObject` has two subclasses `MOTDirectoryObject` and `BWSObject`, which model MOTDirectories (in the data carousal) and objects of a BWS data service respectively. The interface MOTObjectHeader is used to specify information, which is provided both in `MOTObject` and in an entry in a MOT directory.

There is one more specialization, `BWSDirectoryObject`, a subclass of `MOTDirectoryObject`. This class specifies additional information in a directory, which is available only for BWS services. `BWSDirectoryIndex` is a helper class for that.

For the purpose of referencing the different kinds data objects (of type `DABObject`), `ObjectId` may be used.

The classes `Label`, `AnnouncementSupport`, `ProgrammeNumber`, `SubscriberType` and `ProgrammeType` are auxiliary classes used to model attributes inside the service information classes and `DABObject`. `LocationInfo` models location information.

Finally, we have the `events` package (see figure 24). `DABEvent` is the superclass of all events used by the package. The particular events are divided into confirmation and notification events related to the respective method in `DABListener`. Note, that there is no direct reference of `DABEvent` in `DABListener`, the associative link was only used to simplify the diagram.

Note, that there are no direct associations between `EnsembleInfo`, `ServiceInfo`, `ComponentInfo` and `DABObject`. For this the service and object identifiers are used. This means, that the service identifiers inside `EnsembleInfo` refer to services, the service identifiers inside `ServiceInfo` refer to components and so on.

To reduce the complexity of the class diagrams only the associations inside the data classes are indicated.

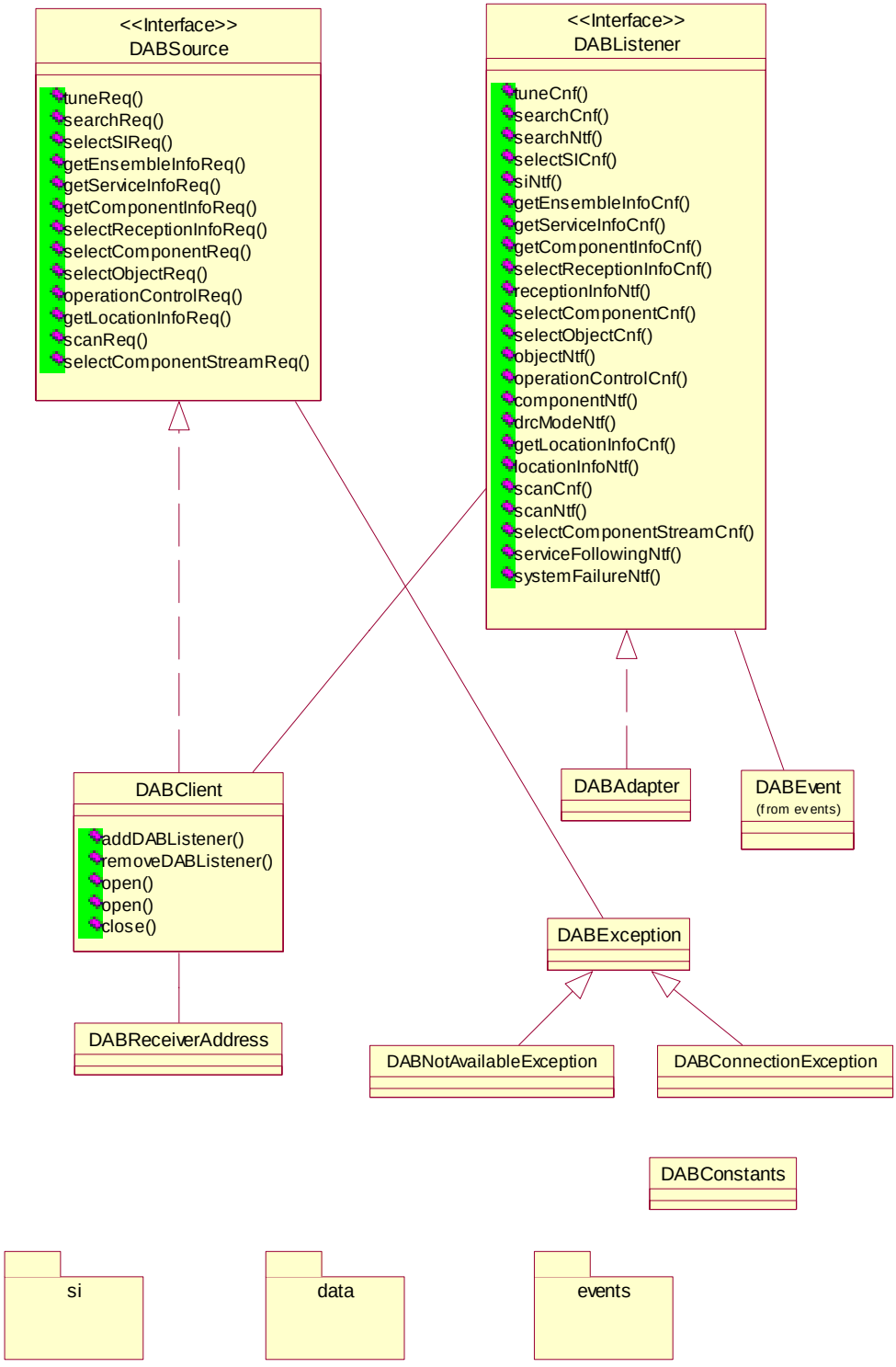


Figure 21: The classes of the dab (main) package

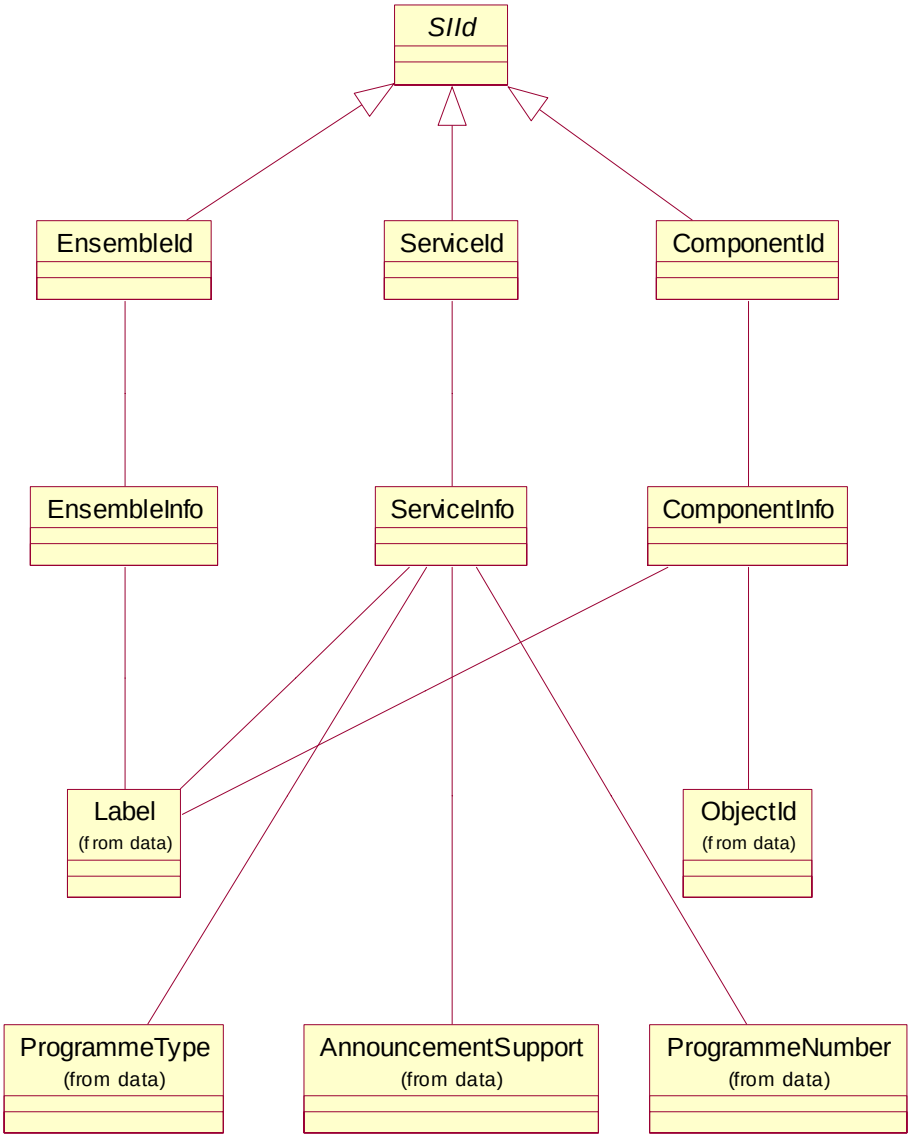


Figure 22: The Classes of the si package

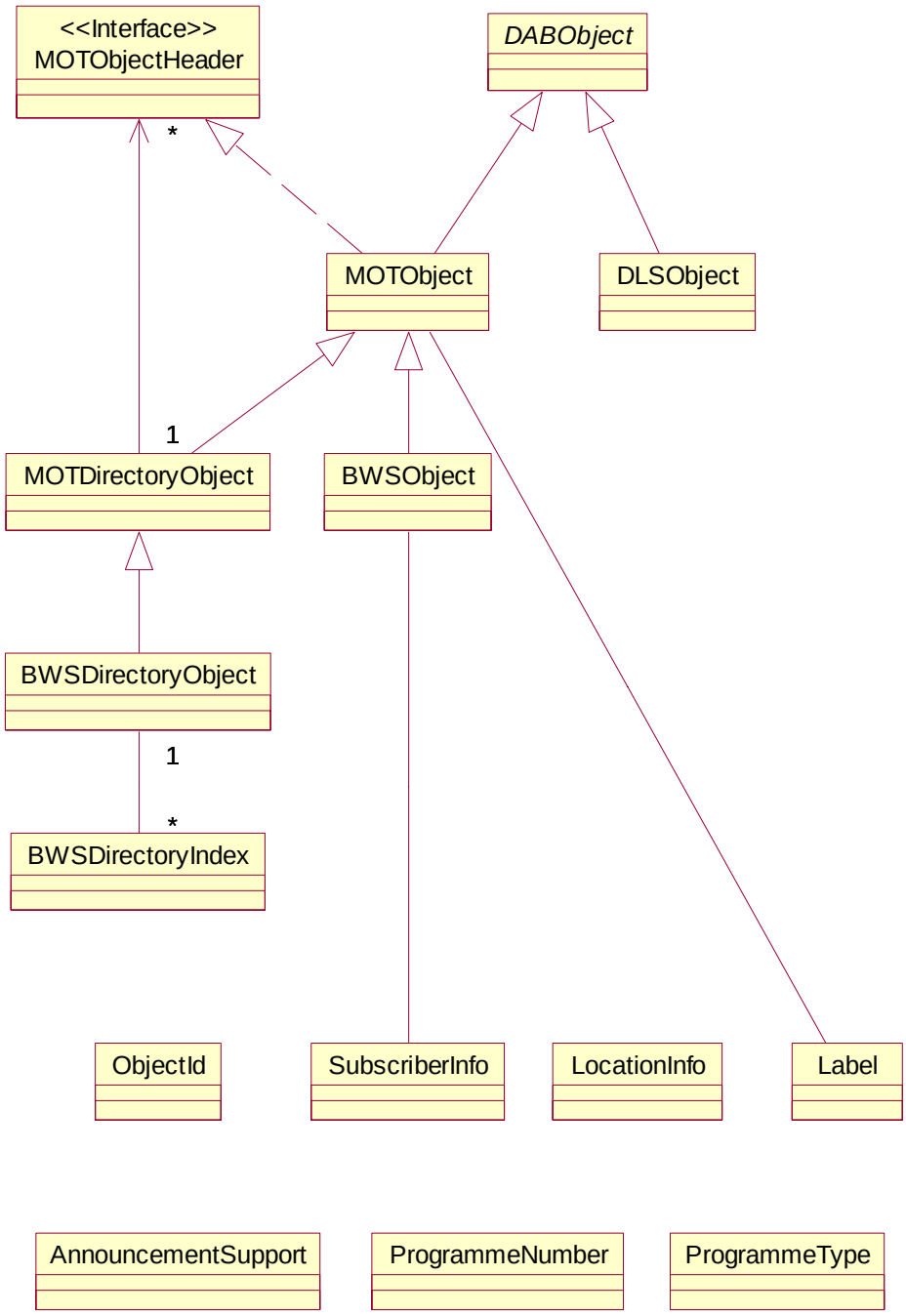


Figure 23: The classes of the data package

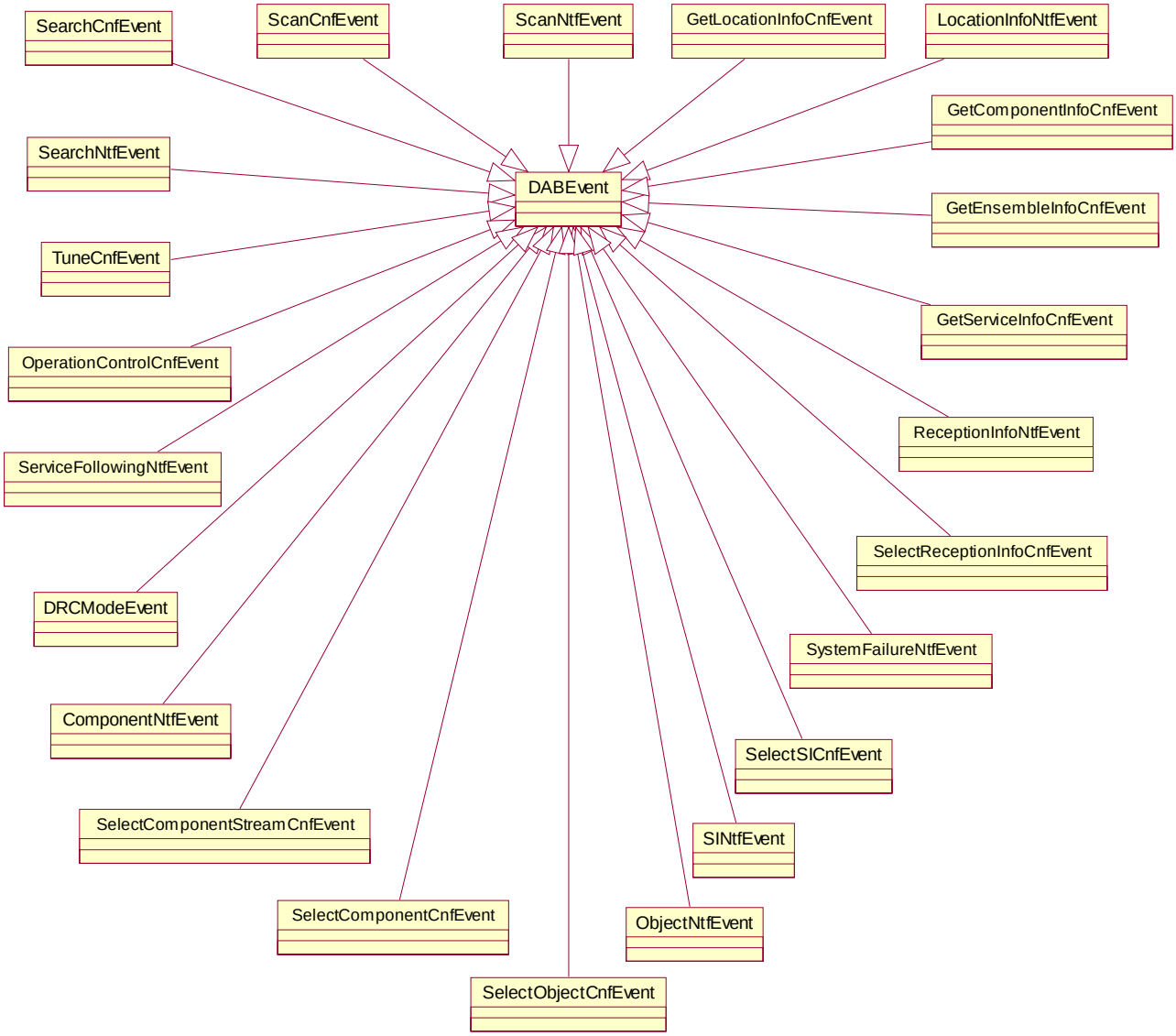


Figure 24: The classes of the events package

5 The runtime package

5.0 Summary

The DAB runtime package deals with the basic runtime components that support the execution environment for a DAB terminal. It consists of the following parts:

- **DAB application model.** This part defines a lifecycle for DAB Java applications based on the Xlet model of JavaTV.
- **Control of Java Applications.** Here, we specify how an application is launched and how its state can be controlled.
- **Security Management.** This part handles the security issues with regard to DAB Java applications.
- **Resource Management.** The resource management provides mechanisms for sharing resources between different DAB Java applications.
- **Configuration Management.** This part deals with handling of the internal profile (i.e. the profile information that is available to the application).

5.1 The DAB Application Model

The Xlet model from Java TV (see Bibliography, "DVB Java Specification") is used for all applications, which are downloaded via DAB. The state machine of the Xlet is described in the following state model (see figure 25). The application controller can control the Xlet component calling specific methods from one side and the Xlet can notify the application controller about the changes of his internal state (see the class diagram in figure 26). After the Xlet code is loaded, it can be instantiated (calling the new operator). When `initXlet()` is called, the method should initialize all the resources needed by the Xlet and put the Xlet in the PAUSED state. The life of the Xlet then is controlled calling respectively the `startXlet()` and `pauseXlet()`. Additionally, the Xlet can notify the host application about the internal state changes using the XletContext (see figure 26): for example, if no further computation is possible, the Xlet can go in the PAUSED state and can notify with the paused methods the host application.

The final step of the lifecycle of the Xlet is reached, when `destroyXlet()` method is called. The Xlet then has to deallocate all used resources.

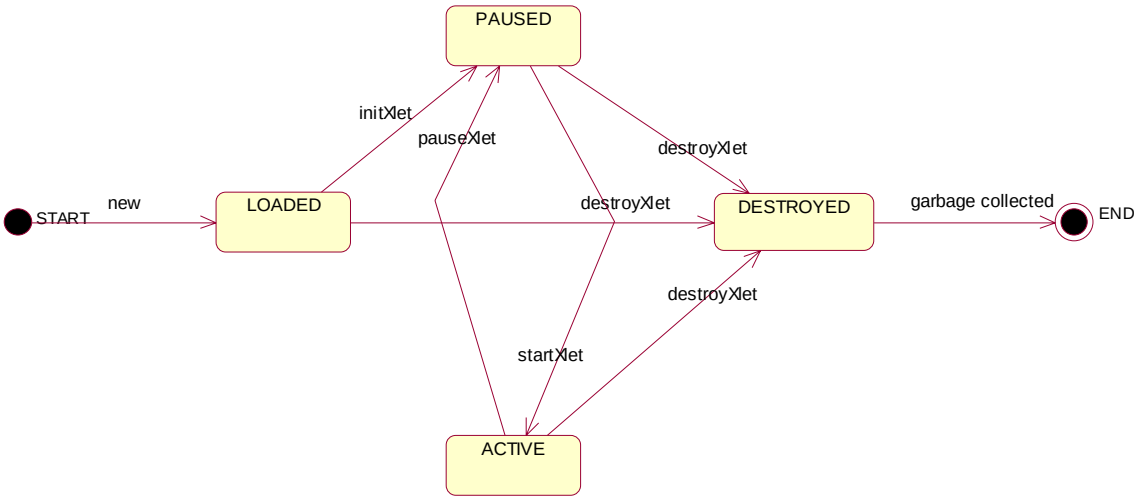


Figure 25: The Xlet Model

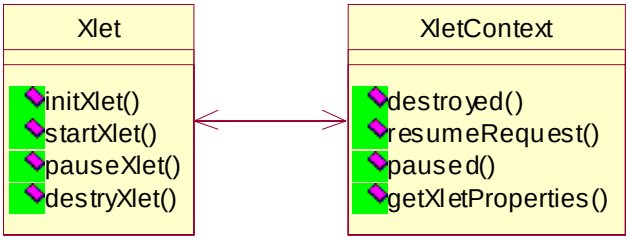


Figure 26: Context for the Xlet model

Moreover application designers are supposed to check the host environment to control if the needed resources are accessible: graphical interfaces, display area, DAB specific commands, etc. In the Xlet implementation this action is performed passing an array of strings to the loaded component using the XletContext `getXletProperties(String)`, where all the necessary variables are parameterized (see clause 5.5).

The following example demonstrates the some of Xlet methods. It is the main class of the stock market ticker (see above):

```
public void initXlet(XletContainer cxt){
   buffer = "";
   decoder=new Decoder((ServiceInfoId)cxt.getXletProperty("dab.xlet.componentId"));
   started = false;
   containerPanel =
   (java.awt.Panel) ctxt.getXletProperty("dab.xlet.panelContainer");
   containerPanel.add(this);
}


public void startXlet()
throws XletStateChangeException{
   if (!started) {
        requestForStop = false;
        thread = new Thread(this);
        thread.start();
        decoder.addStockListener(this);
        decoder.startDecoding();
        started = true;
   }
   else {
        thread.resume();
   }
     }
};
```

5.2 Control of Java applications

Two types of Java-based user applications are supported as it shown in figure 27. One choice are Java applications, which are embedded in HTML pages. The other choice are applications running in some sort of framework or EPG. The Java application can be distributed as a set of packages, which are transported using the MOT protocol [1]. Such an application is described by a profile, which specifies the required platform, the application model and the application type. If an application is chosen for execution, the launch procedure is initiated: the procedure consists in loading the classes (archive), in setting up a runtime context to the application and in initializing the application itself.

Figure 27: Environments for DAB Java applications

In the following clauses these different aspects are described.

5.2.1 Packaging

A Java-based user application is distributed as a sequence of MOT Objects, which contain collection of classes in the JAR format (see Bibliography, "JAR Archive Documentation"). Apart from object code such a jar file can also contain arbitrary data files. Additionally, a so-called Manifest file can be used for configuration issues (see clause 5.5).

The Manifest file inside a Jar archive contains meta-information about the content of the archive (see Bibliography, "Manifest"). With regard to the launching procedure the following attributes are used:

- `Class-Path`: This attribute specifies a sequence of relative URLs, which refer to other distribution objects. These paths will be used to resolve dependencies when loading classes from the archive. Each URL has to correspond to an identifier of a MOT object.
- `Main-Class`: If this attribute is set, the application launcher will execute the designated class (the name has to be a relative URL). If this attribute is not set, then the MOT object identifier (obtained by the `getID()` method on the ObjectId class) is used to identify the main class name (the object identifier has to have the form <relative path-name>/<classname>.class)

The following information can also be included inside the Manifest file for describing the content and for repeating attributes available at the FIC level (these values should not be used for system settings):

- `Platform` : the supported platform (see clause 6.3)
- `Access:` the access pattern `(see clause 5.3)`

The respective type for the MOT object is 6 ="system" and the subtype is 1="Java" (see [1]).

5.2.2 Loading classes

The class loader used in the runtime package has to support the loading of classes transported via DAB (see clause 5.2.1). When a non-system class is referenced, its name (including the package path) is mapped to a relative URL (e.g. MPEG.Decoder is mapped to MPEG/Decoder.class). Then, the respective class file is searched first in the current distribution object. If it is not found, the search is continued in the list of distribution objects designated by the `Class-Path` attribute in the Manifest file (continuing from left to right). The search will stop, when the class file is found or all distribution objects were tried.

Additionally, the class loader has to take care for updates. In general, the class loader will only consider the newest version of a distribution object. If an update of such a distribution object happens during the loading of classes, the class loader shall stop the loading of the current class and restart the process for all classes of the application using the newest version.

5.2.3 Control of applications

If an application is chosen for execution, the controller will use its `DABClient` object to load the application. The result of the operation is a proxy object, which has methods for controlling the application state.

5.2.3.1 Application context

The class `DABClient` provides a method for loading applications (`selectApplicationReq`). This method uses the given user application context and id for the start object to load the application. When the transaction is completed, a `SelectApplicationCnfEvent` is sent to the listener. When the loading was successful, a proxy object is returned in the event.

5.2.3.2 Proxy

Applications are controlled (indirectly) using a simplified version of the `org.dvb.application.AppProxy` interface (from the MHP specification [2]). The concept is an indirect control of applications. This means if for example an application likes to stop another application, it calls the respective method in the AppProxy interface. The application controller, which implements the interface, will then call the method in the application.

The `DABAppProxy` interface offers methods to change the state of the controlled application. Each time it is checked, whether the application is permitted to issue the request for a state change. Additionally, there are methods to be notified about state changes.

For the notification of state changes, the classes `dab.events.AppStateChangeEvent` and `dab.AppStateChangeEventListener` are used.

The `AppStateChangeEventListener` class was modified to comply with the convention to indicate the event source. For that an additional interface `AppStateChangeEventSource` was defined. Finally, there is no application identifier (the reference to the proxy object can be used for that purpose).

There are six states defined for the application proxy (as it is shown in figure 28). The proxy is in the `NotLoaded` state when it is created and returned by `selectApplicationCnf` to the application. This means that the requested application was not yet (down-)loaded. When `load` is called the application will be loaded and the state of the proxy is now `Loaded`. The proxy move on to `Inited` when `init` is called. As part of the state change `initXlet` is invoked in the loaded application.

The `pause` action is possible for two states. If the proxy is in the `Inited` state, the proxy is just moved to the `Paused` state. If it was in the `Started` state, additionally `pauseXlet` is called.

The start action is valid for the first four states in figure 4. It is effectively a composite action, whose effect depends on the state of the proxy:

1) NotLoaded: load, init, pause, resume
2) Loaded: init, pause, resume
3) Inited: pause, resume
4) Paused: resume

The `resume` action can only be invoked, when the proxy is in `Paused`. It invokes `startXlet` and sets the proxy to `Started`. The proxy can be set to the `Destroyed` state from any other state using `stop`. During the state change `stopXlet` is called. Each state change is signalled to all `AppStateChangeEvent` listeners accordingly.

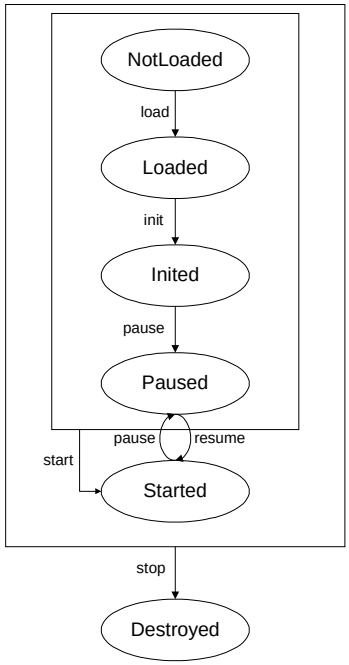


Figure 28: Proxy states

5.2.3.3 Example

In figure 28, it is shown how some application is loaded and how it gets controlled. First, the controller (here an EPG) has to select the component. In the second step the application is loaded. In the third step the application is started (an example for control). In the fourth step the EPG will register for state changes. In the fifth step it receives a state change event. Note, that the messages that are sent to and from the application container are labelled with stereotypes, as they depend on the implementation.

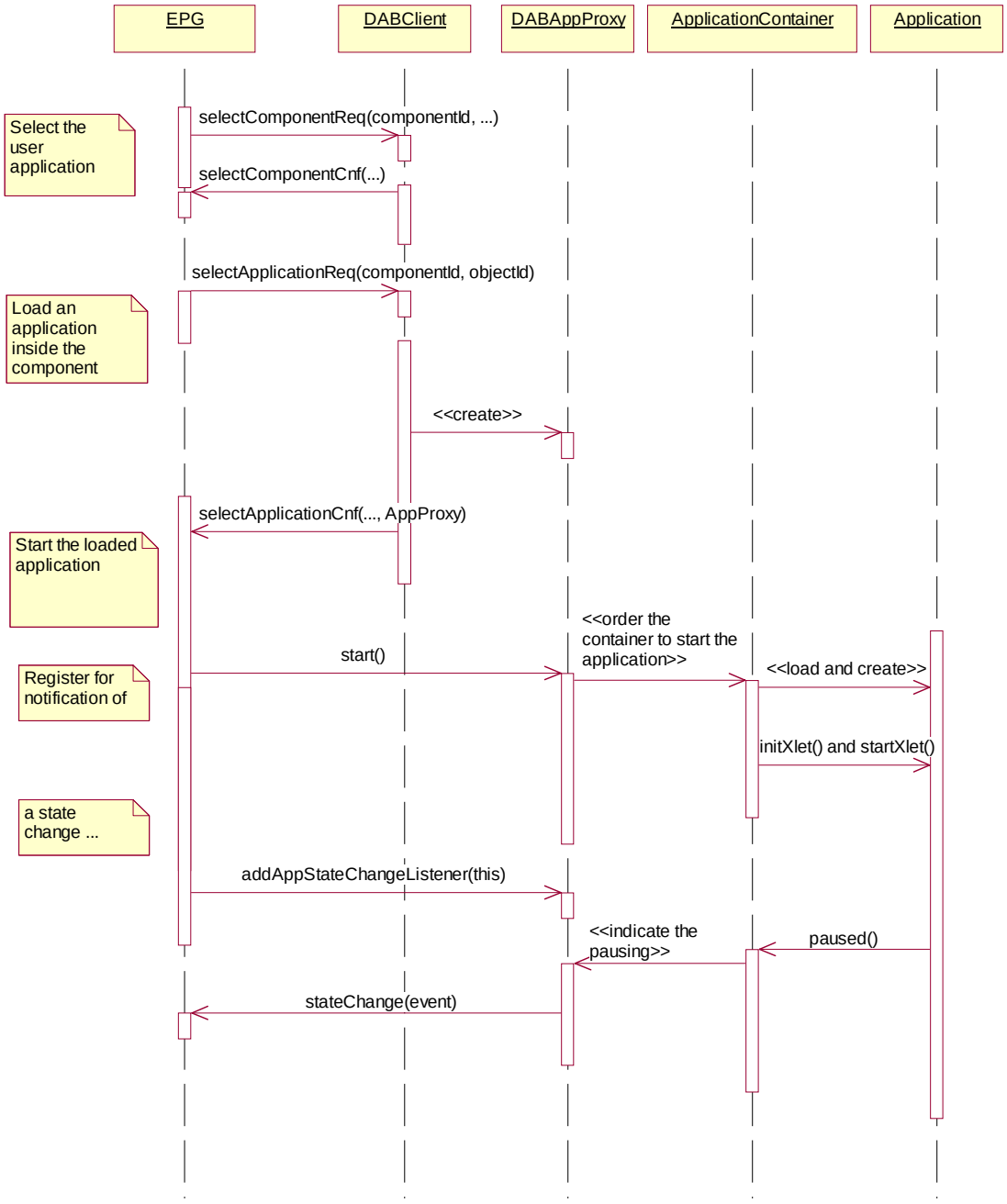


Figure 29: Launching an application

Example of selection of a Java object from an EPG application:

```
private void selectJava(ComponentInfo _component){
   try {
   …
   dab.selectApplicationReq(_component.getId(),jarObject);
   } catch (DABException _e) {
   }
return;
}
…
```

the confirmation event carries the "Proxy" object for controlling the loaded application

```
public void selectApplicationCnf(SelectApplicationCnfEvent _e){
   if(_e.getResult() == 0){
   // Xlet container (implementation specific)
         javaXletAppViewer = new XletAppViewer();
         javaAppProxy = _e.getApplicationProxy();
     if(javaAppProxy == null){
     return;
     }
         javaXletAppViewer.setXletAppProxy(javaAppProxy);
         javaXletAppViewer.show();
   }
return;
}
….
```

5.3 Security management

The DAB security model (see figure 30) that we present here is independent from the different security architectures presented in the different JDK releases (see Bibliography, "Implementing Protection Domains in the Java Development Kit 1.2" and "Java Security Architecture"). This means, the developer of the DAB VM may choose the security architecture that fits best to his/her own implementation requirements.

The security framework presented here is based on three main subsystems: policy profiling, resource access and security implementation subsystem.

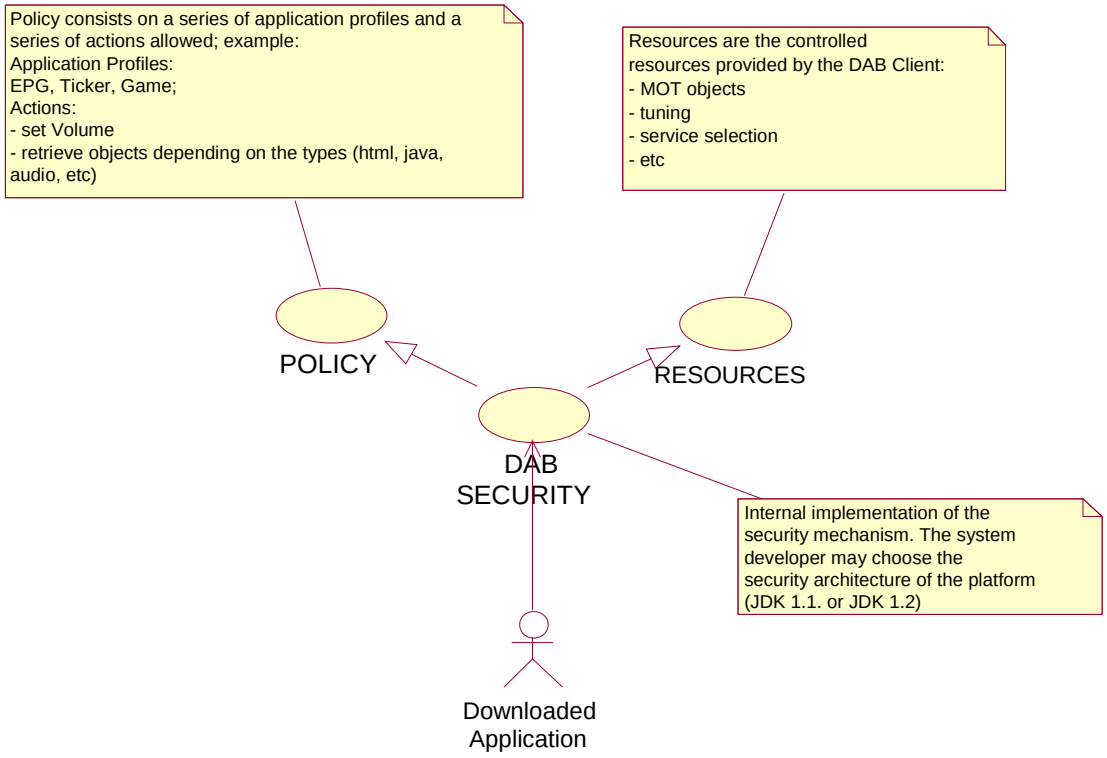


Figure 30: The DAB security model

The only access to DAB resources allowed to downloaded applications from the DAB channel is through the DABClient class; the security subsystem checks this accesses: the security system can return silently or throws a Java.lang.SecurityException (see clause 5.5 for other possible way to access system properties).

For simplifying the management of the security settings security profiles may be used. Three profiles are defined: EPG, MediaPresenter, and NODABAccess (see clause 6.3 for the coding). The EPG defines the class of applications that are allowed maximum access to the DAB terminal. The `MediaPresenter` profile refers to the applications that are allowed to access limited resources (for example only data file transmitted in the channel from where they are downloaded). Ultimately the `NoDABAccess` profile: such profile can only use few terminal resources (Video, audio) but no DAB access is allowed.

In table 2 it is demonstrated, how these security profiles can be used to differentiate between the DAB commands considering the caller.

Table 2: Example of access to DAB resources vs. security profiles

Method name	EPG	MediaPresenter	NoDABAccess
`void getComponentInfoReq(ServiceInfoId serviceId)`	**totally**	no	no
`void setEnsembleInfoReq(ServiceInfoId serviceId)`	totally	no	no
`void getServiceInfoReq(ServiceInfoId serviceId)`	totally	partially	no
`void scanReq(int searchMode, int tables, int startFrequency, int stopFrequency, int tansmissionModes, int notifications)`	totally	no	no
`void searchReq(int searchMode, int tables, int startFrequency, int stopFrequency, int transmissionModes, int notifications)`	totally	no	no
`void selectComponentReq(ServiceInfoId serviceId, int selectionMode)`	totally	partially	no
`void selectObjectReq(ServiceInfoId serviceId, ObjectId objectId, int requestMode, boolean replaceSelections, int deliveryMode, int cacheHint)`	totally	partially	partially
`void selectReceptionInfoReq(boolean synchronizationNotification, boolean bitErrorRateNotifcations, boolean muteStateNotifications, boolean requestOnce)`	totally	partially	no
`void selectServiceInfoReq(boolean ensembleInfo, boolean serviceInfo, boolean componentInfo, boolean autoDelivery)`	totally	partially	no
`void setVolumeReq(int volume)`	totally	partially	partially
`void tuneReq(int tuneFrequency, byte transmissionMode)`	totally	**no**	no
`String System.getProperties(String)`	totally	**partially**	no
`String System.setProperties(String,String)`	totally	**no**	no
`void selectApplicationReq(ServiceInfoId serviceId, ObjectId objectId)`	totally	**no**	no
`org.dvb.aplication.AppProxy methods`	totally	**no**	no

Imagine the following scenario. An EPG application is downloaded from some channel (XXX Radio); the application can control the volume of the platform because is specified in the profile. The implementation of the YYY company provides a fine-grained security mechanism: the user can set min, max, normal volume for every different application profile and for every different content provider. The implementation of the ZZZ Company is more profile-oriented and it controls only the profile of the downloaded application. The internal mechanisms used in the two implementations are hidden to the loaded application.

For instance, we have the following code fragment in an application:

```
try {
   dab.open();
}catch(SecurityException e){
   // …
}
```

```
try{
     dab.setVolumeReq(10);
}catch(SecurityException e){
   //…
}
```

In the VM it is implemented as follows:

```
public void open() throws DABException, SecurityException
    {
  // perform a DAB security check using the AccessController
  AccessController.checkPermission(new DABRuntimePermission("open"));
  // the AC throws a securityException if the policy settings do not allow the permission
      receiver = DABSystem.getReceiver();
  receiver.addReceiverListener(this);
    }
```

5.4 Resource management

Resource management is used to share exclusive resources among different concurrently active clients. In the case of the DAB VM such kinds of resources are mostly related to the constraint set by the DAB receiver: e.g. only one ensemble can be accessed at the same time. We will present a model for the resource management and describe how resource conflicts are resolved within this model.

5.4.1 Model

The resource management is opaque. But resource conflicts are signalled. This means that the allocation of resources during a transaction is not visible outside the API. The VM has to care for all issues of resource management like allocation of resources, their release and deadlock avoidance. If the VM cannot allocate the required resources, the conflict resolution is initiated (see clause 5.4.2). If the conflict resolution fails, the failure of the transaction is indicated to the requesting client.

For the indication of failures because of resource conflicts a dedicated exception is used:

```
class ResourceConflictException extends DABException {
      ResourceConflictException() {}
      ResourceConflictException(String message) {}
}
```

It is expected, when a request is issued (e.g. calling selectReceptionInfoReq) and the request fails due to resource conflicts, the `ResourceConflictException` is thrown.

5.4.2 Conflict Resolution

The VM handles the conflict resolution and negotiates with the involved parties. The conflict resolution consists of four turns (i.e. phases).

1) **Proceed**: Ask the requesting client for commitment despite resource conflicts.

2) **Probe**: If the client insists on his request, all other clients which own needed resources are asked whether they like to release their resources. If there is at least one who does not like to release a needed resource, the conflict resolution fails.

3) **Stop**: If all clients agreed to the request, they are actually asked to release the resources. When all clients have released the resources, the transaction is restarted.

4) **Preempt**: If there are some clients, which did not release the resources in the last turn, then these are preempted from the resources.

NOTE 1: In case a client does not respond to such requests, it is up to the VM to handle such behaviour. Typically, a timeout needs to be introduced and a default response has to be defined (e.g. it is assumed that the client agreed to release the resource). Like security management that should be configurable by the user.

NOTE 2: When clients are asked to release their resources (also for the willingness to do so), not the actual resources are specified but rather the related operation. That means the VM has to track the relationship between resources and operations.

The API supports the conflict resolution by the following notification:

```
interface DABListener {
 …
      conflictResolutionNtf (ConflictResolutionNtfEvent event);
}

class ConflictResolutionNtfEvent {
      int getTransaction() {…}
      int getTurn() {…}
      int getOperation() {…}
      int getSuboperation() {…}
}

class  DABConstants {
 …
      public static final int conflictResolutionTurnProceed=0;
      public static final int conflictResolutionTurnProbe=1;
      public static final int conflictResolutionTurnStop=2;
      public static final int conflictResolutionTurnPreempt=3;
```

```
        public static final int conflictResolutionOperationNone=0;
        … // plus constants for all
          // available operations (see DABSource)
    }
```

The related event, `ConflictResolutionNtfEvent`, contains the information about the action:

1) `getTransaction` delivers the transaction number. This can be used to provide a transaction context.
2) `getTurn` returns a code for the turn of the resource conflict resolution protocol:
   - `conflictResolutionTurnProceed`: This is sent to the client which requested the operation. It indicates that there is a resource conflict. The client is asked whether he likes to proceed.
   - `conflictResolutionTurnProbe`: This notification is sent to all clients in order to probe for their willingness to release the needed resources.
   - `conflictResolutionTurnStop`: The client is asked to stop the indicated operation in order to release the resources.
   - `conflictResolutionTurnPreempt`: The client is informed that the indicated operation was stopped. This action shall normally only be taken, when the client failed to do a stop in the previous turn.
3) `getOperation` gives back a code of the involved operation (see extensions to DABConstants).
4) `getSuboperation` returns a code for the suboperation, which depends on the operation code. This is useful for "aggregated" operations like operationControl.

The response to the notifications described above are given using a request in DABSource:

```
    interface DABSource {
     …
        respondConflictResolutionReq(int transaction,
                                     int turn,
                                     int operation,
                                     int suboperation,
                                     int answer);
}

    class  DABConstants {
     …
        public static final int conflictResolutionAnswerNo=0;
        public static final int conflictResolutionAnswerYes=1;
    }
```

The argument `answer` refers to the answer with respect to the notification. The following values are possible:

- Turn Proceed
  - `conflictResolutionAnswerYes`: The client likes to have his request continued so that the negotiations for resources may start.
  - `conflictResolutionAnswerNo`: The client agrees to stop the request, which will result in a `ResourceConflictException` (see above).
- Turn Probe
  - `conflictResolutionAnswerYes`: The client is willing to stop the operation.
  - `conflictResolutionAnswerNo`: The client does not agree to stop the operation.
- Turn Stop
  - The request confirms that the operation was stopped by the client. No special value has to be specified for answer.

The other arguments of this request correspond to the attributes in the `ConflictResolutionNtfEvent`.

A confirmation will be sent for this request:

```
interface DABListener {
  …
        respondConflictResolutionCnf(conflictResolutionCnfEvent);
}

class conflictResolutionCnfEvent {
        int getResult();
}
```

The method getResult returns the result of the request.

The following example demonstrates these extensions. For this, we assume that we have selected an audio programme in a resident EPG and started a stock market ticker. Additionally, we have just launched a new EPG (which a provided by a broadcaster), which likes to do scanning.

In figure 31 it is indicated, how the emerging resource conflict is resolved. The VM signals the downloaded EPG the resource conflict (other applications have selected components inside the current ensemble).

The downloaded EPG decides to proceed with the conflict resolution. The VM then sends notifications to the resident EPG and to the ticker to probe for their agreement to stop the selections. When the agreement is given, they are actually asked to stop the selections. After that is done, the transaction for the scan is started and the `scanReq` call returns to the client.

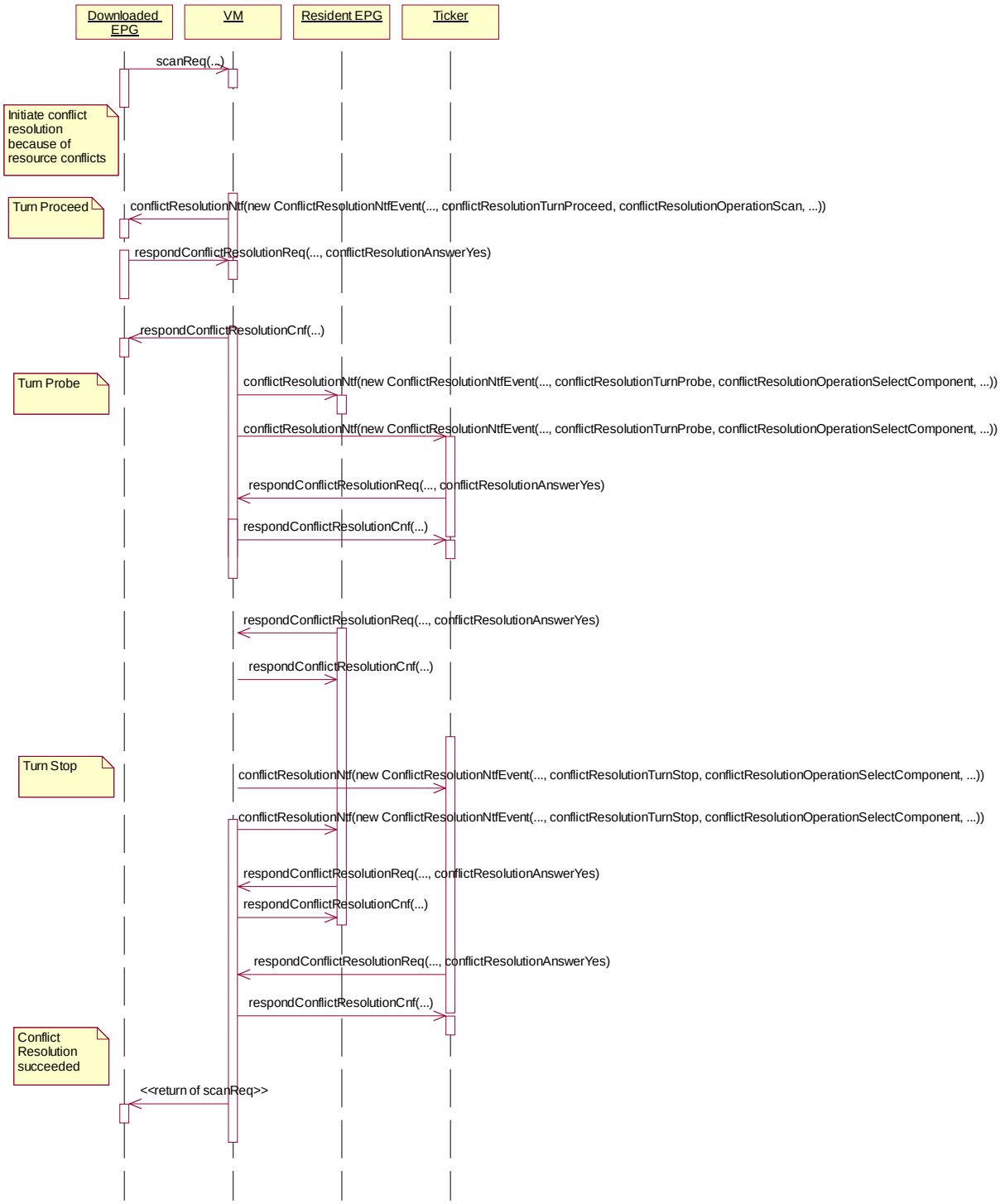


Figure 31: Example of conflict resolution

5.5 Configuration management

Configuration Management is related to the capability to profile the characteristics of the host terminal in a fine-grained manner.

Downloaded applications have to use a standard mechanism for determining the environment of the terminal for adapting their UI or their input routines. Additionally, the hosting environment has to give the application hooks to the runtime environment.

This leads to two classes of attributes, which the Xlet is interested in: platform scoped profile attributes and Xlet scoped profile attributes.

There is a very simple mechanism inside the Java runtime environment that permits to specify a set of properties for the Runtime system. Such properties are pairs of string values (key,value): for example:

- Java.home=c:\jdk1_2
- os.name=NT

These properties are set using methods located in the Java.lang.System class: System.setProperty(String key, String value) and String System.getProperty(String value).

Table 3: Predefined Xlet properties

dab.xlet.componentId	The component identifier from which the Xlet was loaded (of type dab.ServiceInfoId)
dab.xlet.objectId	The object identifier of the main archive of the Xlet (of type dab.ObjectId)
dab.xlet.panelContainer	A reference to the GUI of the Xlet (of type Java.awt.Panel)
dab.xlet.datastream.in	The standard input stream (of type Java.io.DataInputStream)
dab.xlet.datastream.out	The standard output stream (of type Java.io.DataOutputStream)
dab.xlet.datastream.err	The standard error stream (of type Java.io.DataOutputStream)
dab.xlet.platform	The platform (of type Integer)
dab.xlet.accessType	The access type (of type Integer)
Dab.xlet.content	The content type (of type Integer)

For all the other attributes that are Xlet dependent we use the getXletProperties method of the XletContext. This method differs slightly from the System method: it returns objects (not strings). Using this method we can give to the Xlet objects needed for its runtime execution (see table: panelContainer, datastream.in, etc). In table 3, the predefined properties are listed.

The following example demonstrates the different attribute mechanisms:

Inside the application framework before activating the Xlet:

```
Panel xletPanel = new Panel();

appProxy.setAppProperty("dab.xlet.panelContainer",xletPanel);

appProxy.setAppProperty("dab.xlet.datastream.out",System.out);
```

Inside the Xlet:

```
public void initXlet(XletContext ctx) throws XletStateChangeException {

        xPanel = (Java.awt.Panel) ctx.getXletProperty("dab.xlet.panelContainer");
        xOut = (Java.io.DataStream) ctx.getXletProperty("dab.xlet.datastream.out");
};
```

Note, that the code contains no error checks.

6 The User I/O Package

The DAB User IO package specifies the Java platform that should be supported for DABJava. It also defines the profiles for DABJava and the method for signalling the profile using the FIG0/13 User Application Type (see EN 300 401 [3]).

The DAB Java User IO package consists of the following parts:

- **Signalling.** This part defines the signalling in DAB of the DABJava profiles.
- **DABJava platform.** This part defines the current defined platforms and profiles that are used in DABJava devices. In the future other profiles can be added to the stadard following the rules defined in clause 6.1.

6.1 Signalling

6.1.1 DAB Java User Application Profile (DJUAP)

DABJava is defined as a User Application (UA) type within FIG 0/13 (see EN 300 401 [3]). The DABJava UA parameters "Platform" and "Version" are used to signal the DABJava profile or application environment. They are carried in the User Application specific part of the FIG 0/13.

6.1.2 Platform

The DABJava UA parameter "Platform" signals the major version of the application environment (e.g. SPJP or NPJP). It should be a number between 0 and 255. New platform numbers must be registered at the WIRC by registration of a new DJUAP with the parameter "Version" set to 0.

DJUA platforms are not necessarily compatible with each other. This means that although platform A may partly cover platform B, an application for platform A will not necessarily run in the application environment of platform B.

6.1.3 Version

The UA parameter "Version" signals the minor version of the application environment. It should be a number between 0 and 255. New version numbers are obtained from the WIRC by updating an existing platform specification. The new version number shall be the previous version number increased by 1.

DABJava UA versions must be backwards compatible. That means that for a certain Platform where version A is older than version B, then an application written for version A must run in the application environment defined in version B.

6.1.4 Content

The UA parameter "Content" signals the application type. It should be a number between 0 and 255. New version numbers are obtained from the WIRC by updating an existing platform specification. The new version number shall be the previous version number increased by 1.

6.1.5 Access

The UA parameter "Access" signals the access type. It should be a number between 0 and 255. New version numbers are obtained from the WIRC by updating an existing platform specification. The new version number shall be the previous version number increased by 1.

The following values are predefined:

0) EPG

1) MediaPresenter

2) NoDABAccess

6.1.6 Defined profiles

The currently defined profiles are described in details in clause 6.2, and they are signalled as follows:

6.1.6.1 Standard Personal Java Profile (SPJP)

The SPJP supports the APIs and the application environment of PersonalJava without the package "Java.net".

The UA parameters for this profile are:

Name	Value
Platform	0
Version	0
Content	0 to 255
Access	0 to 2

6.1.6.2 Network enabled Personal Java Profile (NPJP)

The NPJP supports the APIs and the application environment of PersonalJava together with the class "org.dvb.net.DatagramSocketBufferControl" and the interface "Javax.tv.net.InterfaceMap" defined in the Java TV API Specification (see Bibliography).

The UA parameters for this profile are:

Name	Value
Platform	1
Version	0
Content	0 to 255
Access	0 to 2

6.2 DABJava platforms

6.2.1 PersonalJava 1.1

Two profiles have been defined for DAB Java based on Personal Java. They modify the packages from PersonalJava 1.1 (see Bibliography) as shown in the following.

6.2.1.1 Core Packages

The packages used in PersonalJava 1.1 and common to the 2 profiles (6.16.1 and 6.1.6.2) are shown below, the supported packages are slightly modified for DAB Java profiles as described in clause 6.2.3:

- Java.applet
- Java.awt
- Java.awt.datatransfer
- Java.awt.event
- Java.awt.image
- Java.awt.peer
- Java.beans
- Java.io *(file support is optional)*

- Java.lang
- Java.lang.reflect
- Java.math *(package is optional)*
- Java.net *(some protocols are optional)*
- Java.rmi *(package is optional)*
- Java.rmi.dgc *(package is optional)*
- Java.rmi.registry *(package is optional)*
- Java.rmi.server *(package is optional)*
- Java.security *(package is optional)*
- Java.security.acl *(package is unsupported)*
- Java.security.interfaces *(package is optional)*
- Java.sql *(package is optional)*
- Java.text
- Java.text.resources *(modified and optional)*
- Java.util
- Java.util.zip

6.2.1.2 DABJava profiles: specific packages.

6.2.1.2.1 Standard Personal Java Profile (SPJP)

- Java.applet (package is optional)
- Java.beans (package is optional)
- Java.net (package is unsupported)

6.2.1.2.2 Network-enabled Personal Java Profile (NPJP)

- Java.applet (package is optional)
- Java.beans (package is optional)
- org.dvb.net.DatagramSocketBufferControl is added (see Bibliography, "DVB Java Specification")
- Javax.tv.net.InterfaceMap is added (see Bibliography, "DVB Java Specification")

Annex A (normative): The DAB Java Classes

A.1 Package dab

public class **dab.DABAdapter** implements dab.DABListener

The DABAdapter class provides default methods for the implementation of a DABListener.

The default behaviour is that incoming events are ignored.

See Also

dab.DABListener DABListener

Version

1.01

Constructors **public DABAdapter()**

Methods **public void tuneCnf(TuneCnfEvent e)**

public void searchCnf(SearchCnfEvent e)

public void searchNtf(SearchNtfEvent e)

public void scanCnf(ScanCnfEvent e)

public void scanNtf(ScanNtfEvent e)

public void selectSICnf(SelectSICnfEvent e)

public void siNtf(SINtfEvent e)

public void getEnsembleInfoCnf(GetEnsembleInfoCnfEvent e)

public void getServiceInfoCnf(GetServiceInfoCnfEvent e)

public void getComponentInfoCnf(GetComponentInfoCnfEvent e)

public void selectReceptionInfoCnf(SelectReceptionInfoCnfEvent e)

public void receptionInfoNtf(ReceptionInfoNtfEvent e)

public void selectComponentCnf(SelectComponentCnfEvent e)

public void selectComponentStreamCnf(SelectComponentStreamCnfEvent e)

public void componentNtf(ComponentNtfEvent e)

public void selectObjectCnf(SelectObjectCnfEvent e)

public void selectApplicationCnf(SelectApplicationCnfEvent e)

public void objectNtf(ObjectNtfEvent e)

public void getLocationInfoCnf(GetLocationInfoCnfEvent e)

public void locationInfoNtf(LocationInfoNtfEvent e)

public void conflictResolutionNtf(ConflictResolutionNtfEvent e)

public void respondConflictResolutionCnf(RespondConflictResolutionCnfEvent e)

public void operationControlCnf(OperationControlCnfEvent e)

public void serviceFollowingNtf(ServiceFollowingNtfEvent e)

public void drcModeNtf(DRCModeNtfEvent e)

public void systemFailureNtf(SystemFailureNtfEvent e)

public interface **dab.DABSource**

DABSource defines the interface of a DAB resource (usually a DAB receiver). The interface is asynchronous. When an application issues a requests, it gets back the results as confirmation and notification events. Look at the particular methods for more details.

See Also

dab.DABListener DABListener

Version

1.07

Methods **public void tuneReq(**

int tuneFrequency,

int transmissionMode)

The tuneReq request initiates the Tune command. The Tune command sets directly a specified DAB frequency. A DAB receiver must be tuned to a DAB frequency and synchronized in order to get access to DAB services. A tuned DAB receiver tries automatically to synchronize on a DAB Ensemble.

The Tune command is used to select a specified DAB frequency. The tuneReq request initiates the Tune command. Depending on the specification for the Transmissionmode it is tested if a DAB Ensemble can be detected. If the connected DAB receiver supports automatic detection the default setting for transmissionMode (=DABConstants.transmissionModeAutomatic) can be used. Otherwise it has to be specified which transmission modes should be tested. The result of the command is delivered by the tuneCnf confirmation. All currently existing selections of audio and data services or selections of data objects are automatically stopped before tuning is performed by the DAB receiver.

Parameters

- **tuneFrequency** - This parameter specifies the frequency the DAB receiver will be tuned to in Hertz.
- **transmissionModes** - This parameter specifies the transmission modes a DAB receiver tests for DAB Ensembles. The default value is DABConstants.transmissionModeAutomatic which means that the receiver is automatically detecting the transmission mode. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.transmissionModeAutomatic: The transmission mode is automatically detected. All other flags are ignored in this case.
  - DABConstants.transmissionMode1: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 1.
  - DABConstants.transmissionMode2: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 2.
  - DABConstants.transmissionMode3: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 3.

- DABConstants.transmissionMode4: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 4.

See Also

dab.DABListener#tuneCnf tuneCnf

dab.DABSource#searchReq searchReq

dab.DABListener#searchCnf searchCnf

dab.DABListener#searchNtf searchNtf

public void searchReq(

int searchMode,

int tables,

int startFrequency,

int stopFrequency,

int transmissionModes,

int notifications)

The searchReq request initiates a Search command. The Search command searches for a DAB Ensemble according to a specified search mode. After a successful execution of the Search command a DAB Ensemble has been found, the state Tuned is entered and the DAB receiver tries to synchronize automatically to the found DAB Ensemble.

The Search command is used to search for a DAB Ensemble. The searchReq request initiates the search and specifies the frequencies and transmission modes to test. Additionally the notifications can be specified which the DAB client gets while the command is executed. Searching for an ensemble may require a substantial amount of time from only a second up to several minutes. This depends also on the search mode specified. If the reception conditions are bad it is possible that no DAB Ensemble at all is detected. In order to stop searching for a DAB Ensemble the Tune command can be used which tunes the DAB Receiver to a certain frequency independent from the reception conditions. The start of searching is indicated by a SearchNtf event with a status code 'Started'. In this case the state machine of Tune State enters the state Searching (see Figure 4). In case that the previous state was Tuned all currently existing selections of services or objects are stopped automatically. While searching is performed several notifications delivering information about the current status are sent to the client. The command ends with a SearchCnf event.

Parameters

- **searchMode** - This parameter specifies the way the DAB receiver is searching for a DAB Ensemble. The default value is searchSearchAutomatic which means it is searching according to a default method. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.SearchModeAutomatic: default method
  - DABConstants.SearchMode16kHzSteps: The frequency range is searched in 16 kHz steps. This is a very intensive search which means that command execution can take a large amount of time.
  - DABConstants.SearchModeUp: The search direction is from low to high frequencies.
  - DABConstants.SearchModeDown: The search direction is from high to low frequencies.
  - DABConstants.SearchModeUseTables: The search is based on the specified frequency tables.

- DABConstants.SearchModeUseFrequencyRange: The search is based on the specified frequency range.
- DABConstants.SearchModeContinuous: The search is looping over the specified frequency range until a DAB Ensemble has been found. The default is to stop after the specified frequency range has been checked once.

- **tables** - This parameter specifies frequency tables the receiver uses in order to search for DAB Ensembles. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.searchCEPTFrequencyTableBandIII: The frequencies according to the CEPT frequency table for Band III are tested for DAB Ensembles.
  - DABConstants.SearchCEPTFrequencyTableLBand: The frequencies according to the CEPT L-Band table are tested for DAB Ensembles.
  - DABConstants.SearchCanadaFrequencyTableLBand: The frequencies according to the Canadian L-Band table are tested for DAB Ensembles.
- **transmissionModes** - This parameter specifies the transmission modes a DAB receiver tests for DAB Ensembles. The default value is DABConstants.transmissionModeAutomatic which means that the receiver is automatically detecting the transmission mode. The parameter is a flag field supporting the following flags which can be specified together:
  - transmissionModeAutomatic: The transmission mode is automatically detected. All other flags are ignored.
  - transmissionMode1: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 1.
  - transmissionMode2: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 2.
  - transmissionMode3: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 3.
  - transmissionMode4: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 4.
- **notifications** - This parameter specifies the type of notifications the client wants to get while the Seek command is performed. The parameter is a flag field supporting the following flags which can be specified together:
  - notificationsOff: No intermediate notifications are sent. Only a SearchNtf notification which informs about the start of searching is sent.
  - notifications16kHzSteps: With each 16 kHz step a notification is sent. This is only possible if 16 kHz step searching is specified as search mode.
  - notificationsTableEntry: With each table entry frequency a notification is sent. This is the default value.

See Also

dab.DABListener#searchCnf searchCnf

dab.DABListener#searchNtf searchNtf

dab.DABSource#tuneReq tuneReq

dab.DABListener#tuneCnf tuneCnf

```
public void scanReq(
    int searchMode,
```

int tables,

int startFrequency,

int stopFrequency,

int tansmissionModes,

int notifications)

The ScanReq request initiates a Scan command. The Scan command is used in order to perform a search for all available DAB Ensembles in a specified frequency range. Depending on the frequency range and the search mode this operation may require a substantial amount of time from only a second up to several minutes. The command is started by the ScanReq request and is finished with the ScanCnf confirmation. In between ScanNtf notification are sent in order to inform about the current status of scanning if notifications are requested.

In case of searching from lower to higher frequencies (searchMode=DABConstants.searchModeUp) the value of startFrequency is not allowed to be larger than the value of stopFrequency. In case of searching from higher to lower frequencies (searchMode=DABConstants.searchModeDown) the value of startFrequency is not allowed to be smaller than the value of stopFrequency.

Parameters

- **searchMode** - This parameter specifies the way the DAB Receiver is searching for a DAB Ensemble. The default value is DABConstants.searchModeAutomatic which means it is searching according to a default method. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.searchModeAutomatic: default method
  - DABConstants.searchMode16kHzSteps: The frequency range is searched in 16 kHz steps.
  - DABConstants.searchModeUp: The search direction is from low to high frequencies.
  - DABConstants.searchModeDown: The search direction is from high to low frequencies.
  - DABConstants.searchModeUseTables: The search is based on the specified frequency tables.
  - DABConstants.searchModeUseFrequencyRange: The search is based on the specified frequency range.
  - DABConstants.searchModeContinuous: The search is looping over the specified frequency range until a DAB Ensemble has been found. The default is to stop after the specified frequency range has been checked once.
- **tables** - This parameter specifies frequency tables the receiver uses in order to search for DAB Ensembles. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.searchCEPTFrequencyTableBandIII: The frequencies according to the CEPT frequency table for Band III are tested for DAB Ensembles.
  - DABConstants.searchCEPTFrequencyTableLBand: The frequencies according to the CEPT L-Band table are tested for DAB Ensembles.
  - DABConstants.searchCanadaFrequencyTableLBand: The frequencies according to the Canadian L-Band table are tested for DAB Ensembles.
- **startFrequency** - This parameter specifies the start frequency at which the DAB Receiver starts its search for DAB Ensembles.
- **stopFrequency** - This parameter specifies the stop frequency at which the DAB Receiver stops its search for DAB Ensembles.

- **transmissionModes** - This parameter specifies the transmission modes a DAB Receiver should look for DAB Ensembles. The default value is DABConstants.transmissionModeAutomatic which means that the receiver is automatically detecting the transmission mode. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.transmissionModeAutomatic: The transmission mode is automatically detected.
  - DABConstants.transmissionMode1: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 1.
  - DABConstants.transmissionMode2: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 2.
  - DABConstants.transmissionMode3: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 3.
  - DABConstants.transmissionMode4: At the specified frequency it is tested if a DAB Ensemble is sent in transmission mode 4.
  - notifications - This parameter specifies the type of notifications wanted by the application while the Seek command is performed. The parameter is a flag field supporting the following flags which can be specified together:
  - DABConstants.notificationsOff: No notifications are sent.
  - DABConstants.notifications16kHzSteps: With each 16 kHz step a notification is sent.
  - DABConstants.notificationsTableEntry: With each table entry frequency a notification is sent. This is the default value.

See Also

dab.DABListener#scanCnf scanCnf

dab.DABListener#scanNtf scanNtf

public void selectSIReq(

boolean ensembleInfo,

boolean serviceInfo,

boolean componentInfo,

boolean autoDelivery)

The selectSIReq method initiates a SelectSI command. The SelectSI command starts, stops or changes subscription to Service Directory Information.

The SelectSI command allows a DAB client to subscribe for Service Directory Information. The Service Directory contains all available ensembles, services, components and related information. The subscription is requested by the selectSIReq request and is confirmed with the selectSICnf confirmation. The subscription level can be changed by another SelectSI command. This includes the termination of subscription.

After a successful subscription a connected client receives SINtf notifications. Just after the subscription has been activated the complete content of the Service Directory is mapped on SINtf notifications. This means for each stored instance of an service element (ensemble, service and component) is a SINtf notification sent which indicates that this service element is available (Added). As time goes on SINtf notifications are sent which indicate that a new service element is available (Added), that an existing is no longer available (Removed) or that its attributes have changed (Changed).

By use of autoDelivery it can be specified if SINtf sends only a notification or a notification together with the related information object. If subscription is terminated by setting ensembleInfo, serviceInfo and componentInfo to false, then autoDelivery has no meaning.

By selecting a certain subscription level the client is informed about all currently known service elements by sending related SINtf notifications. This means if a client subscribes for service-specific notifications and seven services exist at this time, then seven SINtf(DABConstants.serviceAdded) notifications are generated. The client is not informed about known ensembles or components. As time goes on the client is informed when new services are added, known services are removed or changed. If a currently selected subscription level is increased meaning that more notification types are subscribed then the client is informed about all currently known service elements that are related to the new subscribed notification type. This means if a subscription is changed from service-specific to service-specific and component-specific change notifications, then for each currently known Component a SINtf(DABConstants.componentAdded) notification is generated.

As time goes on the client is informed when new services or components are added, known services or components are removed or changed. If a currently selected subscription level is decreased meaning that less notification types are subscribed then the client is informed only about notifications related to the remaining subscribed notification types. This means if a subscription is changed from service-specific and component-specific to service-specific notifications, then the client is informed when new services are added, known services are removed or changed. But the client is no longer notified about changes related to components.

Parameters

- **ensembleInfo** - This parameter specifies if ensemble-specific notifications will be sent to the DAB client. The following values are supported:
  - true: The DAB client is notified about DABConstants.ensembleAdded, DABConstants.ensembleChanged and DABConstants.ensembleRemoved events. This is the default setting.
  - false: The DAB client is not notified about DABConstants.ensembleAdded, DABConstants.ensembleChanged and DABConstants.ensembleRemoved events.
- **serviceInfo** - this parameter specifies if service-specific notifications will be sent to the DAB client. The following values are supported:
  - true: The DAB client is notified about DABConstants.serviceAdded, DABConstants.serviceChanged and DABConstants.serviceRemoved events. This is the default setting.
  - false: The DAB client is not notified about DABConstants.serviceAdded, DABConstants.serviceChanged and DABConstants.serviceRemoved events.
- **componentInfo** - This parameter specifies if component-specific notifications will be sent to the DAB client. The following values are supported:
  - true: The DAB client is notified about DABConstants.componentAdded, DABConstants.componentChanged and DABConstants.componentRemoved events. This is the default setting.
  - false: The DAB client is not notified about DABConstants.componentAdded, DABConstants.componentChanged and DABConstants.componentRemoved events.
- **autoDelivery** - This parameter specifies if the information related to the notification is sent together with the notification (SINtf) or not. The following values are supported:
  - true: The SINtf notification delivers the notification together with the information object. The information object is sent together with the notification if the notification type is -Added or -Changed. In case of -Removed no information object is sent because it is no longer existing. This is the default setting.

- false: The SINtf notification delivers only the notification. The information object (EnsembleInfo, ServiceInfo or ComponentInfo) itself can be obtained by use of getEnsembleInfoReq, getServiceInfoReq or getComponentInfoReq.

See Also

dab.DABListener#selectSICnf selectSICnf

dab.DABListener#siNtf siNtf

public void getEnsembleInfoReq(EnsembleId id)

The getEnsembleInfoReq method initiates a GetEnsembleInfo command. The GetEnsembleInfo command requests information about the specified DAB Ensemble.

The GetEnsembleInfo command provides a DAB client with information about a specified DAB Ensemble, e.g. Label, No of Services, and so on. The command is initiated by a getEnsembleInfoReq request and is finished by a getEnsembleInfoCnf confirmation.

Parameters

- **id** - This parameter is a handle identifying the DAB Ensemble.

See Also

dab.DABListener#getEnsembleInfoCnf getEnsembleInfoCnf

public void getServiceInfoReq(ServiceId id)

The getServiceInfoReq requests initiates a GetServiceInfo command. The GetServiceInfo command requests information about a specified DAB Service.

The GetServiceInfo command provides a DAB client with information about a specified DAB Service, e.g. Label, No of Components, and so on. The command is initiated by a getServiceInfoReq request and is finished by a getServiceInfoCnf confirmation.

Parameters

- **id** - This parameter is a handle identifying the DAB Service.

See Also

dab.DABListener#getServiceInfoCnf getServiceInfoCnf

public void getComponentInfoReq(ComponentId componentId)

The getComponentInfoReq request initiates a GetComponentInfo command. The GetComponentInfo command requests information about a specified DAB Component.

The GetComponentInfo command provides a DAB client with information about a specified DAB Component, e.g. Label, Language and so on. The command is initiated calling getComponentInfoReq and is finished by a call to getComponentInfoCnf.

Parameters

- **serviceId** - This parameter is a handle identifying the DAB Component.

See Also

dab.DABListener#getComponentInfoCnf getComponentInfoCnf

public void selectReceptionInfoReq(

boolean synchronizationNotification,

boolean bitErrorRateNotifcations,

boolean muteStateNotifications,

boolean requestOnce)

The selectReceptionInfoReq request initiates the SelectReceptionInfo command. The SelectReceptionInfo command starts, stops or changes subscription to state change notifications concerning reception conditions. It is possible to monitor synchronization, biterrorrate and audio decoder muting.

The SelectReceptionInfo command allows a DAB client to subscribe for state change notifications concerning reception conditions in terms of synchronization, biterrorrate and audio decoder muting. The subscription is requested by the selectReceptionInfoReq request and is confirmed with the selectReceptionInfoCnf confirmation. The subscription level can be changed by another SelectReceptionInfo command. This includes stopping of subscription. After a successful subscription the calling DAB client receives ReceptionInfoNtf notifications when state changes occur.

Parameters

- **synchronizationNotification** - This parameter specifies if the calling client is notified about state changes concerning DAB signal synchronization. If the parameter is set to true (default) notifications are provided, if it is set to false no notifications are provided.
- **bitErrorRateNotifications** - This parameter specifies if the calling client is notified about state changes concerning the biterrorrate. If the parameter is set to true (default) notifications are provided, if it is set to false no notifications are provided.
- **muteStateNotifications** - This parameter specifies if the calling client is notified about state changes concerning the mute state of the audio decoder. If the parameter is set to true (default) notifications are provided, if it is set to false no notifications are provided.
- **requestOnce** - This parameter specifies if the reception condition information is wanted only once. In this case the reception condition is once detected and the DAB client informed by one and only one receptionInfoNtf call.

See Also

dab.DABListener#selectReceptionInfoCnf selectReceptionInfoCnf

dab.DABListener#receptionInfoNtf receptionInfoNtf

public void selectComponentReq(

ComponentId id,

int selectionMode)

The selectComponentReq request initiates the SelectComponent command. The SelectComponent command starts or stops an application delivered in a DAB Component.

The SelectComponent command allows to start or stop applications delivered in DAB components. In general more than one component of the same DAB Ensemble can be selected simultaneously. It is possible to select one audio component, all programme-associated data components of the selected audio component and more than one independent data component at the same time. The selection of a component is requested by the selectComponentReq request and is confirmed by the selectComponentCnf confirmation. It is possible that a component is removed from a DAB Ensemble which means it is no longer broadcast and therefore no longer available. This is indicated by a SINtf call and means that the selection is removed automatically. If the selection of a component is removed also all existing object selections belonging to the component are removed.

If the user application is a slide show or a dynamic label, its objects are delivered automatically (using objectNtf notifications) after the SelectComponent confirmation was sent.

If the selected component is an audio service, its PAD data services become available as . This means service information is generated for all PAD services and they can be selected. If the selection of the audio service is stopped, also all PAD services are stopped and they are not available anymore.

If the component is not in the current ensemble, it depends on the implementation whether it is selected nevertheless.

Parameters

- **id** - This parameter is a pointer to the identifier of the DAB Component which is to be selected. If all component selections should be removed (set selectionMode to DABConstants.selectionModeRemoveAll) this parameter is ignored and should be set to null.
- **selectionMode** - This parameter specifies the selection mode for the component. The following flags are supported:
  - DABConstants.selectionModeReplace: All currently selected components of the same type are stopped and the specified component is to be started. The same type means an audio component replaces any other selected audio component, a data component replaces all other selected independent data components and a programme-associated data component replaces all other selected programme-associated data components.
  - DABConstants.selectionModeAdd: The application delivered by the specified component is to be started. Other selected components are not affected.
  - DABConstants.selectionModeRemove: The selection of the specified component is stopped.
  - DABConstants.selectionModeRemoveAll: All existing component selections are removed. Set serviceId to null in this case.

See Also

dab.DABListener#selectComponentCnf selectComponentCnf

dab.DABListener#siNtf siNtf

dab.DABSource#selectObjectReq serviceObjectReq

public void selectComponentStreamReq(ComponentId componentId)

The following request provides access to the DAB transport streams. The requested stream is delivered back in the confirmation, which ends the command.

Parameters

- **componentId** - the service identifier of the component which carries the stream

See Also

dab.DABListener#selectComponentStreamCnf selectComponentStreamCnf

public void selectApplicationReq(

ComponentId serviceId,

ObjectId objectId)

The selectApplicationReq selects applications

The SelectApplication command enables a client to load and control an application. The request is confirmed with the selectApplicationCnf confirmation.

Parameters

- **serviceId** - the component in which the application is located
- **objectId** - the id of the start object

See Also

dab.DABListener#selectApplicationCnf selectApplicationCnf

public void selectObjectReq(

ComponentId id,

ObjectId objectId,

int requestMode,

boolean replaceSelections,

int deliveryMode,

int cacheHint)

The selectObjectReq request initiates the SelectObject command. The SelectObject command selects an object from a selected DAB Component. This includes requesting an object from a data component, delivery after reception and notification of updates as long as the object is selected.

The SelectObject command selects an object from a selected component. Selection means it is requested for delivery and if wanted also updates of the object are delivered. Additionally it is possible to give some hints for caching. More than one object and also from more than one component can be selected simultaneously. The selection of an object is requested by the selectObjectReq request and is confirmed by the selectObjectCnf confirmation. The object is delivered using the objectNtf method. This includes first-time delivery and all updates. Beyond starting or stopping a selection it is possible to remove all other selections belonging to the same component by setting parameter replaceSelections to true. It is possible that a component is removed from a DAB Ensemble. This is indicated by a serviceInfoNtf call. In this case also the selected objects of the service are no longer selected.

It is possible that an object is removed from current on-air service. This is indicated by an objectNtf call. In this case the selections for this object are automatically disabled.

Currently object selection makes only sense with applications of type BroadcastWebSite. Objects of applications like Slideshows or Dynamic Label are delivered automatically by objectNtf calls.

Parameters

- **id** - This parameter identifies the selected component the object is belonging to.
- **objectId** - This parameter identifies the object which is to be selected.
- **selectionMode** - This parameter specifies the selection mode of the object. The following values are supported:
  - DABConstants.requestModeOff: This is used in order to stop the selection of objects which are requested with the request mode DABConstants.requestModeUpdate. It is not needed for objects which are requested with the DABConstants.requestModeOnce flag except for the case that a SelectObjectReq is pending and the delivery is no longer wanted.
  - DABConstants.requestModeOnce: The object is requested for one-time delivery. After the first reception from the broadcast channel the object is delivered to the connected DAB client. The client is not notified about new versions.
  - DABConstants.requestModeUpdate: The object is requested for update delivery. After the first reception from the broadcast channel the object is delivered to the connected client. Additionally each new version of the object is delivered.
- r**eplaceSelections** - This parameter specifies if all current object selections belonging to the component identified by serviceId are replaced with this selection. If this parameter is set to true, then all selections are removed. If this parameter is set to false, then existing selections remain unchanged.
- **deliveryMode** - This parameter specifies the delivery mode of the object. The following values are supported:
  - DABConstants.deliveryModeComplete: Only the complete object is delivered to the DAB client.
  - DABConstants.deliveryModePartial: The object may be delivered in parts.
- **cacheHint** - This parameter specifies a hint for caching of the selected object.

See Also

dab.DABListener#selectObjectCnf selectObjectCnf

dab.DABListener#objectNtf objectNtf

public void getLocationInfoReq(

int type,

int mode,

int desiredDelta,

int desiredAccuracy)

The getLocationInfoReq initiates the GetLocationInfoCommand.

Parameters

- **type** - This parameter indicates the type of location information, that is requested. Supported flags are DABConstants.LocationInfoPosition and DABConstants.LocationInfoRegionId.

- **mode** - The information is delivered according to the values of this parameter:
  - DABConstants.LocationInfoOnce: the information is delivered only one time. The parameter desiredDelta is not considered.
  - DABConstants.LocationInfoPeriodByTime: The information is delivered in intervals given by the value of desiredDelta (in milliseconds)
  - DABConstants.LocationInfoPeriodByDistance: The information is delivered after the distance has passed given by the value of desiredDelta (in meters)
  - DABConstants.LocationInfoStop: The delivery of information is stopped. The parameter desiredDelta is not considered.

desiredDelta - see description of the mode parameter

desiredAccuracy - This parameter indicates the desired accuracy in meters. The value is only considered if type&DABConstants.LocationInfoPosition!=0 <P> This command is optional and may only partially be supported (e.g. only mode=DABConstants.LocationInfoOnce and mode=DABConstants.LocationInfoStop) or may not be supported at all.

See Also

dab.DABListener#getLocationInfoCnf getLocationInfoCnf

dab.DABListener#locationInfoNtf locationInfoNtf

public void respondConflictResolutionReq(

int transaction,

int turn,

int operation,

int suboperation,

int answer)

The respondConflictResolutionReq is used to respond to a resource conflict notification.

Parameters

- **transaction** - the identifier of the transaction of the resource conflict
- **turn** - the code of the turn (see DABConstants.conflictResolutionTurn*)
- **operation** - the code of the operation (see DABConstants.conflictResolutionOperation*)
- **suboperation** - the code of the suboperation (see DABConstants.conflictResolutionSuboperation*)
- **answer** - the actual answer (see DABConstants.conflictResolutionAnswer*)

See Also

dab.DABListener#conflictResolutionNtf conflictResolutionNtf

dab.DABListener#respondConflictResolutionCnf respondConflictResolutionCnf

public void operationControlReq(

int attribute,

Object value)

The OperationControl command enables the DAB client to change or read receiver parameters. The command is initiated by operationControlReq and is finalized by the confirmation operationControlReq.

Parameters

- **attribute** - The parameter can be set as follows:
  - DABConstants.operationControlSetVolume: The volume of the receiver is set. The parameter value has to be of type Integer in the range from 0 to 100 (percent).
  - DABConstants.operationControlGetVolume: The volume of the receiver is read. The parameter value is not considered.
  - DABConstants.operationControlSetServiceFollowing: The service following feature is changed. Value has to be of type Boolean. If it is set to true and the receiver supports service following, then service following for services is switched on. If it is set to false, service following is switched off.
  - DABConstants.operationControlGetServiceFollowing: Read the state of the service following. The parameter value is not considered.
  - DABConstants.operationControlGetServiceFollowingNotifications: Instruct the package to send service following notifications. If the parameter value (of type Boolean) is set to true, the notifications are sent. If it is set to false, then no further notifications are sent.
  - DABConstants.operationControlSetDRCMode: Sets the DRC (Dynamic range control) mode. The DAB concept provides the option of Dynamic Range Control (DRC). The information is generated from the broadcaster's side (transported inside PAD, Programme Associated Data) to influence the audio output signal's dynamic range. The audio output signal will be modified if the option is activated with this call. <P> Value has to be of type Boolean. If it is set to true and the receiver supports DRC, then the DRC mode for audio services is switched on. If it is set to false, the DRC mode is switched off.
  - DABConstants.operationControlGetDRCMode: Read the state of the DRC mode. The parameter value is not considered.
  - DABConstants.operationControlGetDRCModeNotifications: Instruct the package to send DRC mode notifications. If the parameter value (of type Boolean) is set to true, the notifications are sent. If it is set to false, then no further notifications are sent.
- **value** - see description of attribute

See Also

dab.DABListener#operationControlCnf operationControlCnf

dab.DABListener#serviceFollowingNtf serviceFollowingNtf

dab.DABListener#drcModeNtf drcModeNtf

public interface **dab.DABAppProxy** implements dab.AppStateChangeEventSource

This interface can be used to control applications that were launched using selectApplicationReq in DABSource.

See Also

"Digital Video Broadcasting (DVB) Multimedia Home Platform (MHP), TS 101 812"

Version

0.2

Methods

public void addAppStateChangeEventListener(AppStateChangeEventListener listener)

adds a listener for application state changes

public void removeAppStateChangeEventListener(AppStateChangeEventListener listener)

removes a listener for application state changes

public int getState()

returns the current state of the application (see the defined constants)

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to retrieve the application state

public void load()

loads the classes of the application. The state of the application changes to LOADED.

This action is only successful, if the application was not loaded before.

A state change event is signalled in any case.

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to load the application

public void init()

initialises the application. The routine initXlet in the related application will be called. The application is afterwards in the INITED state.

This action is only successful, if the application was not initialized before. If the application was not loaded, the application will first be loaded and and then initialized.

A state change event is signalled in any case. An additional state change is signalled if the application also has to be loaded. In this case first the state change to LOADED is signalled and afterwards that one to INITED.

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to initialize the application

See Also

dab.xlet.Xlet Xlet

public void pause()

pauses the application. The routine pauseXlet in the related application will be called. The application is afterwards in the PAUSED state.

This action is only successful, if the application is either in the INITED state or in the STARTED state.

A state change event is signalled in any case.

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to pause the application

See Also

dab.xlet.Xlet Xlet

public void resume()

resumes the application. The routine startXlet in the related application will be called. The application is afterwards in the STARTED state.

This action is only successful, if the application was in the PAUSED state.

A state change event is signalled in any case.

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to resume the application

See Also

dab.xlet.Xlet Xlet

public void start()

starts the application. The routine startXlet in the related application will be called. The application is afterwards in the STARTED state.

This action is only successful, if the application was not paused or destroyed. If the application was not loaded, the application will first be loaded, then initialized and finally be started. If the application was not initialized, the application will be initialized and then started.

A state change event is signalled in any case. Additional state changes will also be signalled (e.g. NOT_LOADED - LOADED - INITED - STARTED).

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to start the application

See Also

dab.xlet.Xlet Xlet

public void stop(boolean forced)

requests to stop the application. The routine destroyXlet in the related application will be called. The application is afterwards in the DESTROYED state.

This action is only successful, if the application was not destroyed.

A state change event is signalled in any case.

Parameters

- **forced** - if set to true the application is asked to stop and may refuse. if set to false, the application is stopped in any case.

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to stop the application

See Also

dab.xlet.Xlet Xlet

public void setAppProperty(

String key,

Object value)

sets a property of the application.

Parameters

- **key** - the name of the property
- **value** - the new value of the property

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to set an property of the application

public Java.lang.Object getAppProperty(String key)

gets a property of the application. It returns the value of the property or NULL if the property is not defined.

Parameters

- **key** - the name of the property

Throws

Java.lang.SecurityException - the exception is thrown if the caller is not permitted to retrieve an property of the application

Fields

public static final DESTROYED

the final state of the application - no further actions are possible

public static final NOT_LOADED

the application was selected, but is not yet loaded

public static final LOADED

the application is loaded

public static final INITED

the application is loaded and initialized, but there is no activity yet

public static final PAUSED

the application is paused, which means it is not active

public static final STARTED

the application is active

public class **dab.DABException** extends Java.lang.Exception

DABException is the superclass for exceptions inside the DAB package.

Version

1.0

Constructors

public DABException()

public DABException(String msg)

public class **dab.DABConnectionException** extends dab.DABException

The DABConnectionException is thrown when there are problems with the connection between the DAB client and the receiver.

Version

1.0

Constructors **public DABConnectionException()**

public class **dab.DABReceiverAddress**

DABReceiverAddress is used to specify the location of DAB receivers.

Version

1.0

Constructors **public DABReceiverAddress(String address)**

Generates a DABReceiverAddress object from a textual representation of the address. Note, that the actual format of address depends on the implementation

See Also

DABReceiverAddress#getAddress getAddress

Methods **public Java.lang.String toString()**

Generates a textual representation of the object.

public Java.lang.String getAddress()

Generates a textual representation of the object, that can be used to construct a DABReceiverAdress object

public interface **dab.AppStateChangeEventSource**

An interface that can be used to identify an application by its related controller.

Version

0.2

public class **dab.DABClient** implements dab.DABSource

The DABClient class is used to access a DAB resource. Usually the DAB resource might be a receiver that resides on the same host or is at least directly connected to it. But, it could also be a network device.

Note, that the actual interface is defined in DABSource.

See Also

dab.DABSource DABSource

Version

1.05

Constructors **public DABClient()**

Create a DABClient object

Methods

public synchronized void addDABListener(DABListener listener)

Register a DAB listener. DAB events, that relate to this client, are distributed to all registered listeners.

See Also

dab.events.DABEvent DABEvent

DABListener DABListener

dab.DABClient#removeDABListener removeDABListener

public synchronized void removeDABListener(DABListener listener)

Removes the given listener from the list of DAB Listeners.

See Also

DABListener DABListener

dab.DABClient#addDABListener addDABListener

public void open()

A connection to the default receiver is opened.

Throws

DABException - when the client could not be registered

SecurityException - when the application controlling the DABClient does not have the permission to call the open method

public void open(DABReceiverAddress receiverAddress)

A connection to the given receiver is opened. This method is only supported in configurations with multiple receivers.

Parameters

- **receiverAddress** - This parameter specifies the address of the receiver to be used

Throws

DABException - when the client could not be registered

SecurityException - when the application controlling the DABClient does not have the permission to call the open method

public void close()

The connection to the current receiver is closed. All ongoing transactions of the client are cancelled.

Throws

DABException - when no connection was opened

public void tuneReq(

int tuneFrequency,

```
    int transmissionMode)

public void searchReq(
    int searchMode,
    int tables,
    int startFrequency,
    int stopFrequency,
    int transmissionModes,
    int notifications)

public void scanReq(
    int searchMode,
    int tables,
    int startFrequency,
    int stopFrequency,
    int transmissionModes,
    int notifications)

public void selectSIReq(
    boolean ensembleInfo,
    boolean serviceInfo,
    boolean componentInfo,
    boolean autoDelivery)

public void getEnsembleInfoReq(EnsembleId id)

public void getServiceInfoReq(ServiceId id)

public void getComponentInfoReq(ComponentId id)

public void selectReceptionInfoReq(
    boolean synchronizationNotification,
    boolean bitErrorRateNotifications,
    boolean muteStateNotifications,
    boolean requestOnce)
```

```
public void selectComponentReq(
    ComponentId id,
    int selectionMode)

public void selectComponentStreamReq(ComponentId componentId)

public void selectApplicationReq(
    ComponentId componentId,
    ObjectId objectId)

public void selectObjectReq(
    ComponentId id,
    ObjectId objectId,
    int requestMode,
    boolean replaceSelections,
    int deliveryMode,
    int cacheHint)

public void getLocationInfoReq(
    int type,
    int mode,
    int desiredDelta,
    int desiredAccuracy)

public void respondConflictResolutionReq(
    int transaction,
    int turn,
    int operation,
    int suboperation,
    int answer)

public void operationControlReq(
    int attribute,
    Object value)
```

public class **dab.ResourceConflictException** extends dab.DABException

The exception indicates unsolved resource conflicts.

Version

0.2

public interface **dab.DABListener** implements Java.util.EventListener

DABListener defines the interface for DAB listeners.

Version

1.06

Methods

public void tuneCnf(TuneCnfEvent e)

The TuneCnf method finalizes a Tune command and is sent as a response to a TuneReq message. It provides information about the currently selected DAB frequency and reception conditions.

The Tune command is used to select a specified DAB frequency. The tuneReq request initiates the Tune command. tuneCnf finalizes the Tune command and provides information about the reception state. This includes the selected frequency, the detected transmission mode and the synchronization state of the receiver.

See Also

dab.DABSource#tuneReq tuneReq

dab.DABSource#searchReq searchReq

dab.DABListener#searchCnf searchCnf

public void searchCnf(SearchCnfEvent e)

The searchCnf method finalizes a Search command and provides information about the command status, currently selected DAB frequency and current reception conditions.

The Search command is used in order to search for a DAB Ensemble according to a specified search mode. Searching for a DAB Ensemble can take a large amount of time. The start of searching is indicated by a 'Started' searchNtf message. Other searchNtf messages inform a DAB client about search progress. It is finalized by delivery of the searchCnf message. It informs about the command status, the selected frequency and the synchronization state. No further searchNtf messages will be delivered after the delivery of the searchCnf message.

See Also

dab.DABSource#searchReq searchReq

dab.DABListener#searchNtf searchNtf

public void searchNtf(SearchNtfEvent e)

A SearchNtf event is sent after a search for a DAB Ensemble was started searchReq. It informs about the start of searching and about the progress of searching. A SearchCnf event finalizes a Search command. No more SearchNtf events are sent after a SearchCnf event was sent.

The SearchNtf event is sent after the searching for a DAB Ensemble has been started and while searching is in progress in order to provide information about the current status of searching. The 'Started' notification is sent in any case. Progress notifications are only sent if notifications have been requested with the related SearchReq message. No further notifications will be sent after a SearchCnf message is delivered.

See Also

dab.DABSource#searchReq searchReq

dab.DABListener#searchCnf searchCnf

public void scanCnf(ScanCnfEvent e)

The ScanCnf message finalizes a Scan command. It informs about the result of scanning and the current tune state.

The Scan command is used in order to perform a search for all available DAB Ensembles in a specified frequency range. Depending on the frequency range and the search mode this operation may require a substantial amount of time from only a second up to several minutes. The command is started by the ScanReq message and is finished with the ScanCnf message. In between ScanNtf messages are sent in order to inform about the current status of searching if notifications are requested.

The ScanCnf message indicates that the scan command is finished and informs about the current tune state. As a result of performing the scan command the service information database is filled with information. If a SI subscription is running several SINtf messages are delivered to the connected application.

See Also

dab.DABSource#scanReq scanReq

dab.DABListener#scanNtf scanNtf

public void scanNtf(ScanNtfEvent e)

The ScanNtf message is sent after a search for all available DAB Ensembles in a specified frequency range is started by the ScanReq message. The ScanNtf message provides information about the current status of searching for all available DAB Ensembles in a specified frequency range. It is delivered to the connected application after the search has been started by the ScanReq message and notifications have been requested. No further notifications will be sent after a ScanCnf message is delivered.

See Also

dab.DABSource#scanReq scanReq

dab.DABListener#scanCnf scanCnf

public void selectSICnf(SelectSICnfEvent e)

The selectSICnf method finalizes a SelectSI command and indicates current settings. The SelectSI command starts, stops or changes subscription to Service Directory Information.

The SelectSI command allows a DAB client to subscribe for Service Directory Information. The subscription is requested by the selectSIReq method and is confirmed with the SelectSICnf method. The subscription level can be changed by another SelectSI command. This includes the termination of subscription. After a successful subscription a connected client receives siNtf calls when the Service Directory changes.

See Also

dab.DABSource#selectSIReq selectSIReq

dab.DABListener#siNtf siNtf

public void siNtf(SINtfEvent e)

The siNtf notification is sent as a consequence of subscribing to Service Directory Information.

A call to siNtf indicates that the Service Directory has changed. The type of change is signalled and a handle to the changed service element is provided. If AutoDelivery is activated the changed information object itself is delivered together with the notification. Otherwise it can be requested with getEnsembleInfo, getServiceInfo or getComponentInfo. siNtf message is called as a result of the subscription to Service Directory Information.

See Also

dab.DABSource#selectSIReq selectSIReq

dab.DABListener#selectSICnf selectSICnf

public void getEnsembleInfoCnf(GetEnsembleInfoCnfEvent e)

The GetEnsembleInfoCnf method finalizes the GetEnsembleInfo command and delivers information about a requested DAB Ensemble to a DAB client.

The GetEnsembleInfo command provides a DAB client with information about a specified DAB Ensemble, e.g. Label, No of Services, and so on. The command is initiated by a getEnsembleInfoReq request and is finished by a getEnsembleInfoCnf call.

See Also

dab.DABSource#getEnsembleInfoReq getEnsembleInfoReq

dab.si.EnsembleInfo EnsembleInfo

public void getServiceInfoCnf(GetServiceInfoCnfEvent e)

A call to the getServiceInfoCnf method finalizes the GetServiceInfo command and delivers information about a requested DAB Service to a DAB client.

The GetServiceInfo command provides a DAB client with information about a specified DAB Service, e.g. Label, No of Services, and so on. The command is initiated by a getServiceInfoReq message and is finished by a getServiceInfoCnf message.

See Also

dab.DABSource#getServiceInfoReq getServiceInfoReq

public void getComponentInfoCnf(GetComponentInfoCnfEvent e)

The GetComponentInfoCnf message finalizes the GetComponentInfo command and delivers information about a requested DAB Component to a DAB client.

The GetComponentInfo command provides a DAB client with information about a specified DAB Component, e.g. Label, Language and so on. The command is initiated by a GetComponentInfoReq request and is finished by a call to getComponentInfoCnf message.

See Also

dab.DABSource#getComponentInfoReq getComponentInfoReq

dab.si.ComponentInfo ComponentInfo

public void selectReceptionInfoCnf(SelectReceptionInfoCnfEvent e)

The selectReceptionInfoCnf method finalizes the SelectReceptionInfo command. It informs about the command status and the current subscription level.

The selectReceptionInfo method allows a DAB client to subscribe for state change notifications concerning reception conditions in terms of synchronization, biterrorrate and audio decoder muting. The subscription is requested by selectReceptionInfoReq and is confirmed with selectReceptionInfoCnf. The subscription level can be changed by another SelectReceptionInfo command. This includes stopping of subscription. After a successful subscription the calling DAB client receives receptionInfoNtf calls when state changes occur.

See Also

dab.DABSource#selectReceptionInfoReq selectReceptionInfoReq

dab.DABListener#receptionInfoNtf receptionInfoNtf

public void receptionInfoNtf(ReceptionInfoNtfEvent e)

The receptionInfoNtf method is called as a consequence of subscribing to state changes in synchronization, biterrorrate and audio decoder muting.

receptionInfoNtf indicates that the synchronization state, biterrorrate or mute state has changed (see ReceptionInfoNtfEvent). The ReceptionInfoNtf message is provided to a connected client as a result of subscription to state change notifications concerning reception conditions (selectReceptionInfoReq and selectReceptionInfoCnf messages).

See Also

dab.DABSource#selectReceptionInfoReq selectReceptionInfoReq

dab.DABListener#selectReceptionInfoCnf selectReceptionInfoCnf

public void selectComponentCnf(SelectComponentCnfEvent e)

The SelectComponentCnf confirmation finalizes the SelectComponent command. It informs about the command status and the selection status of the specified component.

The SelectComponent command allows to start or stop applications delivered in DAB components. In general more than one component of the same DAB Ensemble can be selected simultaneously. It is possible to select one audio component, all programme-associated data components of the selected audio component and more than one independent data component at the same time. The selection of a component is requested by the selectComponentReq message and is confirmed by a selectComponentCnf call. It is possible that a component is removed from a DAB Ensemble which means it is no longer broadcast and therefore no longer available. This is indicated by a SINtf call and means that the selection is removed automatically.

See Also

dab.DABSource#selectComponentReq selectComponentReq

dab.DABListener#siNtf siNtf

public void selectComponentStreamCnf(SelectComponentStreamCnfEvent e)

The selectComponentStreamCnf method returns the requested stream (if the command was successful) and informs about the result of the command.

See Also

dab.DABSource#selectComponentStreamReq selectComponentStreamReq

public void componentNtf(ComponentNtfEvent e)

The componentNtf method is called if there are changes to the selection mode of a component. This typically happens, when the selection of a component is stopped.

Note, that this notification will be produced due to internal reasons (e.g. after tuning to another ensemble) and not in response to a selectComponentReq request (that is handled by selectComponentCnf).

public void selectObjectCnf(SelectObjectCnfEvent e)

The SelectObjectCnf method finalizes the SelectObject command. The SelectObject command selects an object from a selected DAB Component. This includes requesting an object from a data component, delivery after reception and notification of updates as long as the object is selected.

The SelectObject command selects an object from a selected component. Selection means it is requested for delivery and if wanted also updates of the object are delivered. Additionally it is possible to give some hints for caching. More than one object and also from more than one component can be selected simultaneously. The selection of an object is requested by selectObjectReq and is confirmed by calling selectObjectCnf. The object is delivered using objectNtf. This includes first-time delivery and all updates. Beyond starting or stopping a selection it is possible to remove all other selections belonging to the same component by setting parameter replaceSelections to true. It is possible that a component is removed from a DAB Ensemble. This is indicated by a call to siNtf. In this case also the selected objects of the service are no longer selected. It is possible that an object is removed from current on-air service. This is indicated by calling objectNtf. In this case the selections for this object are automatically disabled. Currently object selection makes only sense with applications of type BroadcastWebSite. Objects of applications like Slideshows or Dynamic Label are delivered automatically using objectNtf.

See Also

dab.DABSource#selectObjectReq selectObjectReq

dab.DABListener#objectNtf objectNtf

public void selectApplicationCnf(SelectApplicationCnfEvent e)

The method is called as a consequence of selecting an application from a data component by use of the SelectApplication command. It delivers a proxy to control the application.

See Also

dab.DABSource#selectApplicationReq selectApplicationReq

public void objectNtf(ObjectNtfEvent e)

The objectNtf method is called as a consequence of selecting objects from a data component by use of the SelectObject command. It delivers a selected object partially or complete to a DAB client.

objectNtf is used to deliver a selected object to the connected DAB client. Depending on the request mode the object is delivered only once or more than once in case of updates. If the object cannot be delivered in-time as indicated by a call to selectObjectCnf, then objectNtf informs about the delay. If transmission of a selected object is stopped, objectNtf informs about the termination of the object transmission and the object selection. It is possible that a DAB Component is removed from a DAB Ensemble. This is indicated by a call to siNtf. In this case also the selected objects of the component are no longer selected. No termination messages are sent for terminated object selections resulting from termination of a component.

See Also

dab.DABSource#selectObjectReq selectObjectReq

dab.DABListener#selectObjectCnf selectObjectCnf

dab.data.DABObject DABObject

public void getLocationInfoCnf(GetLocationInfoCnfEvent e)

getLocationInfoCnf confirms the getLocationInfo command. This means the delivery of location information will start from now on.

See Also

dab.DABSource#getLocationInfoReq getLocationInfoReq

dab.DABListener#locationInfoNtf locationInfoNtf

public void locationInfoNtf(LocationInfoNtfEvent e)

locationInfoNtf notifies about location information.

See Also

dab.DABSource#getLocationInfoReq getLocationInfoReq

dab.DABListener#getLocationInfoCnf getLocationInfoCnf

public void conflictResolutionNtf(ConflictResolutionNtfEvent e)

The method is called for notifying the listener of resource conflicts. The listener can react to this event using the request respondConflictResolutionReq.

See Also

dab.DABSource#respondConflictResolutionReq respondConflictResolutionReq

dab.DABListener#respondConflictResolutionCnf respondConflictResolutionCnf

public void respondConflictResolutionCnf(RespondConflictResolutionCnfEvent e)

The method is called for confirming a reaction to a resource conflict.

See Also

dab.DABSource#respondConflictResolutionReq respondConflictResolutionReq

dab.DABListener#conflictResolutionNtf conflictResolutionNtf

public void operationControlCnf(OperationControlCnfEvent e)

The confirmation indicates the result of the operationControl command.

See Also

dab.DABSource#operationControlReq operationControlReq

public void serviceFollowingNtf(ServiceFollowingNtfEvent e)

The notification informs about service following actions.

See Also

dab.DABSource#operationControlReq operationControlReq

public void drcModeNtf(DRCModeNtfEvent e)

The notification informs about DRC mode changes.

See Also

dab.DABSource#operationControlReq operationControlReq

public void systemFailureNtf(SystemFailureNtfEvent e)

SystemFailureNtf notifies about severe problems with the hardware or the middleware (e.g. breakdown of the communication to the DAB receiver). This should not be confused with the indication of errors for a particular command, which relates only to the command itself.

Typically, after the notification is sent, the package can no longer be used or needs to be reinitialized.

public class **dab.DABNotAvailableException** extends dab.DABException

The DABNotAvailableException is thrown when particular data is currently not available or even not at all available. This usually happens with respect to so-called optional attributes.

Version

1.0

Constructors **public DABNotAvailableException()**

public interface **dab.AppStateChangeEventListener** implements Java.util.EventListener

AppStateChangeEventListener defines the interface for events originating from the DABAppProxy.

See Also

"Digital Video Broadcasting (DVB) Multimedia Home Platform (MHP), TS 101 812"

DABAppProxy DABAppProxy

Version

0.2

Methods **public void stateChange(AppStateChangeEvent event)**

This method is used to signal a state change for the related application.

public class **dab.DABConstants**

DABConstants contains the constants that are used inside the whole package (including the subpackages).

If you would like to add any new constants, please contact WorldDAB Information and Registration Centre, Wyvil Court, Wyvil Road, LONDON SW8 2TG, England, Tel: +44 171 896 90 51, Fax: +44 171 896 90 55, E-mail: worlddab-irc@worlddab.org.

Version

1.07

Constructors **public DABConstants()**

Methods **public static Java.lang.String result2String(int result)**

The method returns a string for the given result code. It is a textual explanation of the result.

Fields **public static final resultOK**

no problems occurred

public static final resultNotSupported

the requested operation is not supported

public static final resultFatalError

a system error occurred (either related to hardware or to the operating system)

public static final resultInternalError

an internal error occurred in the DAB VM (e.g. an implementation error)

public static final resultInvalidParameter

the value of some parameter is not correct

public static final resultOutOfMemory

the system ran out of memory

public static final resultNonApplicableFunction

the operation is not applicable in the current context

public static final resultEnsembleNotFound

the requested/indicated ensemble was not found

public static final resultServiceNotFound

the requested/indicated service was not found

public static final resultComponentNotFound

the requested/indicated component was not found

public static final resultObjectNotSelected

the indicated object was not be selected (in advance)

public static final resultApplicationNotFound

the requested/indicated application was not found

public static final transmissionModeAutomatic

public static final transmissionMode1

public static final transmissionMode2

public static final transmissionMode3

public static final transmissionMode4

public static final transmissionModeUnknown

public static final searchModeAutomatic

public static final searchMode16 kHzSteps

public static final searchModeUp

public static final searchModeDown

public static final searchModeUseTables

public static final searchModeUseFrequencyRange

public static final searchModeContinuous

public static final searchCEPTFrequencyTableBandIII

public static final searchCEPTFrequencyTableLBand

public static final searchCanadaFrequencyTableLBand

public static final notificationOff

public static final notificationFrequencyStep

public static final notification16kHzStep

public static final notificationTableEntry

public static final notificationEnsembleFound

public static final notificationSearchStarted

public static final notificationNone

public static final notificationEnsembleAdded

public static final notificationEnsembleRemoved

public static final notificationEnsembleChanged

public static final notificationServiceAdded

public static final notificationServiceRemoved

public static final notificationServiceChanged

public static final notificationComponentAdded

public static final notificationComponentRemoved

public static final notificationComponentChanged

public static final selectionModeReplace

public static final selectionModeAdd

public static final selectionModeRemove

public static final selectionModeRemoveAll

public static final requestModeOff

public static final requestModeOnce

public static final requestModeUpdate

public static final deliveryModeComplete

public static final deliveryModePartial

public static final syncStateSynchronizationStateUnknown

public static final syncStateNotSynchronized

public static final syncStateDABSignalDetected

public static final syncStateTimeAndFrequencySynchronized

public static final syncStateFICReadable

public static final tuneStateUnknown

public static final tuneStateNotTuned

public static final tuneStateTuning

public static final tuneStateSearching

public static final tuneStateTuned

public static final updatedNone

public static final updatedLabel

public static final updatedCountry

public static final updatedFrequency

public static final updatedDate

public static final updatedTime

public static final updatedTimeOffset

public static final updatedRegion

public static final updatedStaticProgrammeType

public static final updatedDynamicProgrammeType

public static final updatedAnnouncement

public static final updatedLanguage

public static final updatedRegionId

public static final updatedRegionLabel

public static final updatedAnnouncementSupport

public static final updatedStartObject

public static final updatedObjectDirectory

public static final updatedProgrammeNumber

public static final updatedAudioComponent

public static final updatedBitrate

public static final syncUpdateSynchronizationState

public static final syncUpdateBitErrorRateState

public static final syncUpdateMuteState

public static final bitErrorRateLevelUnknown

public static final bitErrorRateLevel1

public static final bitErrorRateLevel2

public static final bitErrorRateLevel3

public static final bitErrorRateLevel4

public static final bitErrorRateLevel5

public static final muteStateUnknown

public static final muteStateMuting

public static final muteStatePartialMuting

public static final muteStateNotMuting

public static final selectionStateOk

public static final selectionStateDelayed

public static final selectionStateTerminated

public static final serviceSelectorNone

public static final serviceSelectorLabel

public static final serviceSelectorCountry

public static final serviceSelectorFrequency

public static final serviceSelectorDate

public static final serviceSelectorTime

public static final serviceSelectorTimeOffset

public static final serviceSelectorRegion

public static final serviceSelectorStaticProgrammeType

public static final serviceSelectorDynamicProgrammeType

public static final serviceSelectorAnnouncement

public static final serviceSelectorLanguage

public static final serviceSelectorRegionId

public static final serviceSelectorRegionLabel

public static final serviceSelectorAnnouncementSupport

public static final serviceSelectorStartObject

public static final serviceTypeAudioService

public static final serviceTypeDataService

public static final announcementAlarm

public static final announcementRoadTrafficFlash

public static final announcementTransportFlash

public static final announcementWarning_Service

public static final announcementNewsFlash

public static final announcementAreaWeatherFlash

public static final announcementEventAnnouncement

public static final announcementSpecialEvent

public static final announcementReserved1

public static final announcementReserved2

public static final announcementReserved3

public static final announcementReserved4

public static final announcementReserved5

public static final announcementReserved6

public static final announcementReserved7

public static final announcementReserved8

public static final countryAlbania

public static final countryAlgeria

public static final countryAndorra

public static final countryAustria

public static final countryAzores_Portugal

public static final countryBelgium

public static final countryBelarus

public static final countryBosniaHerzegovina

public static final countryBulgaria

public static final countryCanaries_Spain

public static final countryCroatia

public static final countryCyprus

public static final countryCzechRepublic

public static final countryDenmark

public static final countryEgypt

public static final countryEstonia

public static final countryFaroe_Denmark

public static final countryFinland

public static final countryFrance

public static final countryGermany1

public static final countryGermany2

public static final countryGibraltar_UnitedKingdom

public static final countryGreece

public static final countryHungary

public static final countryIceland

public static final countryIraq

public static final countryIreland

public static final countryIsrael

public static final countryItaly

public static final countryJordan

public static final countryLatvia

public static final countryLebanon

public static final countryLibya

public static final countryLiechtenstein

public static final countryLithuania

public static final countryLuxembourg

public static final countryMacedonia

public static final countryMadeira_Portugal

public static final countryMalta

public static final countryMoldova

public static final countryMonaco

public static final countryMorocco

public static final countryNetherlands

public static final countryNorways

public static final countryPalestine

public static final countryPoland

public static final countryPortugal

public static final countryRomania

public static final countryRussianFederation

public static final countrySanMarino

public static final countrySlovakia

public static final countrySlovenia

public static final countrySpain

public static final countrySweden

public static final countrySwitzerland

public static final countrySyrianArabRepublic

public static final countryTunisia

public static final countryTurkey

public static final countryUkraine

public static final countryUnitedKingdom

public static final countryVaticanCityState

public static final countryYugoslavia

public static final acsNone

public static final acsNR_MSK

public static final acsEuroCryptEN50094

public static final acsReserverd1

public static final acsReserverd2

public static final acsReserverd3

public static final acsReserverd4

public static final acsReserverd5

public static final componentTypeUnspecified

public static final componentTypeForegroundSound

public static final componentTypeBackgroundSound

public static final componentTypeMultichannelAudio

public static final componentTypeTrafficMessageChannel

public static final componentTypeEmergencyWarningSystem

public static final componentTypeInteractiveTextTransmissionSystem

public static final componentTypePaging

public static final componentTypeDynamicLabel

public static final componentTypeSlideshow

public static final componentTypeBroadcastWebSite

public static final componentTypeJava

public static final componentTypeIPTunneling

public static final languageUnkown

public static final languageAlbanian

public static final languageBreton

public static final languageCatalan

public static final languageCroatian

public static final languageWelsh

public static final languageCzech

public static final languageDanish

public static final languageGerman

public static final languageEnglish

public static final languageSpanish

public static final languageEsperanto

public static final languageEstonian

public static final languageBasque

public static final languageFaroese

public static final languageFrench

public static final languageFrisian

public static final languageIrish

public static final languageGaelic

public static final languageGalician

public static final languageIcelandic

public static final languageItalian

public static final languageLappish

public static final languageLatin

public static final languageLatvian

public static final languageLuxembourgian

public static final languageLithuanian

public static final languageHungarian

public static final languageMaltese

public static final languageDutch

public static final languageNorwegian

public static final languageOccitan

public static final languagePolish

public static final languagePortuguese

public static final languageRomanian

public static final languageRomansh

public static final languageSerbian

public static final languageSlovak

public static final languageSlovene

public static final languageFinnish

public static final languageSwedish

public static final languageTurkish

public static final languageFlemish

public static final languageWalloon

public static final language2C

public static final language2D

public static final language2E

public static final language2F

public static final languageNational30

public static final languageNational31

public static final languageNational32

public static final languageNational33

public static final languageNational34

public static final languageNational35

public static final languageNational36

public static final languageNational37

public static final languageNational38

public static final languageNational39

public static final languageNational3A

public static final languageNational3B

public static final languageNational3C

public static final languageNational3D

public static final languageNational3E

public static final languageNational3F

public static final languageAmharic

public static final languageArabic

public static final languageArmenian

public static final languageAssamese

public static final languageAzerbijani

public static final languageBambora

public static final languageBelorussian

public static final languageBengali

public static final languageBulgarian

public static final languageBurmese

public static final languageChinese

public static final languageChurash

public static final languageDari

public static final languageFulani

public static final languageGeorgian

public static final languageGreek

public static final languageGujurati

public static final languageGurani

public static final languageHausa

public static final languageHebrew

public static final languageHindi

public static final languageIndonesian

public static final languageJapanese

public static final languageKannada

public static final languageKazakh

public static final languageKhmer

public static final languageKorean

public static final languageLaotian

public static final languageMacedonian

public static final languageMalagasay

public static final languageMalaysian

public static final languageMoldavian

public static final languageMarathi

public static final languageNdebele

public static final languageNepali

public static final languageOriya

public static final languagePapamiento

public static final languagePersian

public static final languagePunjabi

public static final languagePushtu

public static final languageQuechua

public static final languageRussian

public static final languageRuthenian

public static final languageSerbo_Croat

public static final languageShona

public static final languageSinhalese

public static final languageSomali

public static final languageSranan_Tongo

public static final languageSwahili

public static final languageTadzhik

public static final languageTamil

public static final languageTatar

public static final languageTelugu

public static final languageThai

public static final languageUkrainian

public static final languageUrdu

public static final languageUzbek

public static final languageVietnamese

public static final languageZulu

public static final language44

public static final language43

public static final language42

public static final language41

public static final languageBackgroundSoundCleanFeed

public static final charsetCompleteEBULatin

public static final charsetEBUCyrillicGreek

public static final charsetEBUArabic_HebrewETC

public static final charsetISOLatinAlphabetNo2

public static final serviceElementTypeUndefined

public static final serviceElementTypeEnsemble

public static final serviceElementTypeService

public static final serviceElementTypeComponent

public static final locationInfoOnce

public static final locationInfoPeriodByTime

public static final locationInfoPeriodByDistance

public static final locationInfoStop

public static final locationInfoPosition

public static final locationInfoRegionId

public static final operationControlSetVolume

public static final operationControlGetVolume

public static final operationControlSetServiceFollowing

public static final operationControlGetServiceFollowing

public static final operationControlGetServiceFollowingNotifications

public static final operationControlSetDRCMode

public static final operationControlGetDRCMode

public static final operationControlGetDRCModeNotifications

public static final serviceFollowingLeavingService

public static final serviceFollowingTryingAlternativeService

public static final serviceFollowingSelectingService

public static final streamTypeAudio

public static final streamTypePacket

public static final streamTypeStream

public static final streamTypeXPAD

public static final streamTypeFIDC

public static final subscriberInfoNoCA

public static final subscriberInfoNoAlgorithm

public static final subscriberInfoNoSubscription

public static final subscriberInfoExpiredSubscription

public static final conflictResolutionTurnProceed

public static final conflictResolutionTurnProbe

public static final conflictResolutionTurnStop

public static final conflictResolutionTurnPreempt

public static final conflictResolutionOperationNone

public static final conflictResolutionOperationTuneReq

public static final conflictResolutionOperationSearchReq

public static final conflictResolutionOperationScanReq

public static final conflictResolutionOperationSelectSIReq

public static final conflictResolutionOperationGetEnsembleInfoReq

public static final conflictResolutionOperationGetServiceInfoReq

public static final conflictResolutionOperationGetComponentInfoReq

public static final conflictResolutionOperationSelectReceptionInfoReq

public static final conflictResolutionOperationSelectComponentReq

public static final conflictResolutionOperationSelectComponentStreamReq

public static final conflictResolutionOperationSelectObjectReq

public static final conflictResolutionOperationGetLocationInfoReq

public static final conflictResolutionOperationOperationControlReq

public static final conflictResolutionOperationSelectApplicationReq

public static final conflictResolutionSuboperationNone

public static final conflictResolutionAnswerNo

public static final conflictResolutionAnswerYes

A.2 Package dab.si

public abstract class **dab.si.SIId**

The SIId is an identifier for a DAB Ensemble, a DAB Service or a DAB Service Component. It defines a handle to one of these service elements and is used to start and stop services or to query service information.

The identifier for each entity is globally unique. This means an identifier for a component or service includes information about the service context as for instance two services are considered different even if they have the same (DAB) service identifier.

Version

1.07

Constructors **public SIId()**

Methods **public int compareTo(Object object)**

This method compares the object with the given object. The behaviour is the same as it is specified in the compareTo method of the Java.lang.Comparable interface.

public int compareTo(SIId siid)

This method compares the object with the given object. The behaviour is the same as it is specified in the compareTo method of the Java.lang.Comparable interface.

public Java.lang.String getId()

Returns an external representation of the identifier in a textual format. The returned value can be used to construct a service identifier.

public class **dab.si.EnsembleId** extends dab.si.SIId

The EnsembleId is an identifier for a DAB ensemble.

Version

1.03

Constructors **public EnsembleId(String Id)**

Constructs an EnsembleId object from the given string.

See Also

SIId#getId getId

public EnsembleId(EnsembleId id)

Constructs a copy of the given EnsembleId object.

public class **dab.si.ServiceInfo**

ServiceInfo is used to represent a service.

Version

1.04

Constructors **protected ServiceInfo(**

ServiceId id,

int type,

EnsembleId parent,

ComponentId[] componentIds,

boolean isLocal,

int accessControlSystem,

boolean hasLabel,

Label label,

boolean hasLanguage,

int language,

boolean hasIsPrimary,

boolean isPrimary,

boolean hasRegionId,

int regionId,

boolean hasRegionLabel,

Label regionLabel,

boolean hasStaticProgrammeType,

ProgrammeType staticProgrammeType,

boolean hasDynamicProgrammeType,

ProgrammeType dynamicProgrammeType,

boolean hasProgrammeNumber,

ProgrammeNumber programmeNumber,

boolean hasTimeOffset,

int timeOffset,

boolean hasAnnouncementSupport,

AnnouncementSupport announcementSupport,

boolean hasCountry,

int country)

Methods

public dab.si.ServiceId getId()

Returns the id of the service

public int getType()

Returns the service type (see DABConstants.serviceType*)

public dab.si.EnsembleId getParent()

Returns the parent ensemble

public dab.si.ComponentId[] getComponentIds()

Returns a reference to ids of the components of the service

public boolean isLocalService()

Indicates whether the service is local or not

public int getAccessControlSystem()

Returns the access control system (see DABConstants.acs*)

public dab.data.Label getLabel()

Returns the label of the service

Throws

DABNotAvailableException - when the label is not available

public int getLanguage()

Returns the language of the service (see DABConstants.language*)

Throws

DABNotAvailableException - when the language is not available

public boolean isPrimaryComponentLanguage()

Indicates, whether the language of the service is the language of the primary component

Throws

DABNotAvailableException - when the information is not available

public int getRegionId()

Returns the region id of the service

Throws

DABNotAvailableException - when the id is not available

public dab.data.Label getRegionLabel()

Returns the region label of the service

Throws

DABNotAvailableException - when the label is not available

public dab.data.ProgrammeType getStaticProgrammeType()

Returns the static programme type

Throws

DABNotAvailableException - when the programme type is not available

public dab.data.ProgrammeType getDynamicProgrammeType()

Returns the dynamic programme type

Throws

DABNotAvailableException - when the programme type is not available

public dab.data.ProgrammeNumber getProgrammeNumber()

Returns the programme number

Throws

DABNotAvailableException - when not available

public int getTimeOffset()

Returns the time offset of the service (with respect to the time of the ensemble). The result is returned in minutes. It ranges from -12 hours to 12 hours.

Throws

DABNotAvailableException - when the offset is not available

See Also

EnsembleInfo#getDate getDate

public dab.data.AnnouncementSupport getAnnouncementSupport()

Returns the information about announcement support

Throws

DABNotAvailableException - when the announcement support is not available

public int getCountry()

Returns the country information of the service (see DABConstants.country*)

Throws

DABNotAvailableException - when the country information is not available

public class **dab.si.ComponentInfo**

ComponentInfo is used to represent components.

Version

1.06

Constructors **protected ComponentInfo(**

ComponentId id,

int type,

byte[] data,

boolean isPrimary,

ServiceId[] parentIds,

int accessControlSystem,

boolean hasLabel,

Label label,

boolean hasLanguage,

int language,

boolean hasStartObjectId,

ObjectId startObjectId,

boolean hasObjectDirectoryId,

ObjectId objectDirectoryId,

boolean hasAudioComponent,

ComponentId audioComponent,

boolean hasBitrate,

int bitrate)

Methods

public dab.si.ComponentId getId()

Returns the id of the component

public int getType()

Returns the type of the component. This is essentially the user application type (see DABConstants.componentType*)

public byte[] getData()

Returns the application specific data of the component (i.e. the user application data).

public boolean isPrimary()

Indicates whether the component is primary or not

public int getAccessControlSystem()

Returns the access control system of the component (see DABConstants.acs*)

public dab.si.ServiceId[] getParentIds()

Returns a reference to the ids of the parents

public dab.data.Label getLabel()

Returns the label of the component

Throws

DABNotAvailableException - when the label is not available

public int getLanguage()

Returns the language information (see DABConstants.language*)

Throws

DABNotAvailableException - when the information is not available

public dab.data.ObjectId getStartObjectId()

Returns the id of the start object

Throws

DABNotAvailableException - when the start object is not available

public dab.data.ObjectId getObjectDirectoryId()

Returns the id of the object directory

Throws

DABNotAvailableException - when the object directory is not available

public int getBitrate()

Returns the maximum bitrate of the component in bits per second.

Throws

DABNotAvailableException - when the bitrate is not available

public dab.si.ComponentId getAudioComponent()

Returns the SIId of the related audio component. Note, that the object has to be a PAD component; otherwise null is returned.

Throws

DABNotAvailableException - when not available

public class **dab.si.ComponentId** extends dab.si.SIId

The ComponentId is an identifier for a DAB component.

Version

1.03

Constructors **public ComponentId(String Id)**

Constructs a ComponentId object from the given string.

See Also

SIId#getId getId

public ComponentId(ComponentId id)

Constructs a copy of the given ComponentId object.

public class **dab.si.ServiceId** extends dab.si.SIId

The ServiceId is an identifier for a DAB service.

Version

1.05

Constructors **public ServiceId(String Id)**

Constructs a ServiceId object from the given string.

See Also

SIId#getId getId

public ServiceId(ServiceId id)

Constructs a copy of the given ServiceId object.

See Also

SIId#getId getId

Methods **public boolean sameService(ServiceId id)**

returns true, if id and the called object belong to the same service; otherwise false is returned.

public class **dab.si.EnsembleInfo**

EnsembleInfo represents information about a particular ensemble.

Version

1.03

Constructors **protected EnsembleInfo(**

EnsembleId id,

ServiceId[] serviceIds,

int frequency,

int transmissionMode,

boolean hasDate,

Date date,

boolean hasLabel,

Label label,

boolean hasCountry,

int country)

Methods **public dab.si.ServiceId[] getServiceIds()**

Returns a reference to the ids for the services that are contained in the ensemble

public dab.si.EnsembleId getId()

Returns the id of the ensemble

public int getFrequency()

Returns the frequency of the ensemble in Hz

public int getTransmissionMode()

Returns the transmission Mode (see DABConstants.transmissionMode*)

public dab.data.Label getLabel()

Returns the label of the ensemble

Throws

DABNotAvailableException - if the label is not available

public int getCountry()

Returns country information about the ensemble (see DABConstants.country*)

Throws

DABNotAvailableException - if the country information is not available

public Java.util.Date getDate()

Returns date and time associated with the ensemble (given as local time)

Throws

DABNotAvailableException - if the date is not available

A.3 Package dab.events

public class **dab.events.DABEvent** extends Java.util.EventObject

DABEvent is the superclass for all events used inside the DAB package.

Version

1.01

Constructors **protected DABEvent(DABSource source)**

public class **dab.events.ComponentNtfEvent** extends dab.events.DABEvent

Version

1.02

Constructors **protected ComponentNtfEvent(**

DABSource source,

int reason,

ComponentId componentId,

int selectionMode)

Methods **public int getReason()**

Returns the reason for change of the selectionMode (the code is compatible with DABConstants.result*).

public dab.si.ComponentId getComponentId()

Returns the component which is involved

public int getSelectionMode()

Returns the new selection mode for the component.

See Also

dab.events.SelectComponentCnfEvent#getSelectionMode getSelectionMode

public class **dab.events.OperationControlCnfEvent** extends dab.events.DABEvent

OperationControlCnfEvent is generated in response to an operationControlReq request.

See Also

dab.DABListener#operationControlCnf operationControlCnf

Version

1.01

Constructors **protected OperationControlCnfEvent(**

DABSource source,

int result,

int attribute,

Object value)

Create an OperationControlCnfEvent object.

Methods **public int getResult()**

Returns the status of the OperationControl command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public int getAttribute()

Returns the attribute of the receiver that was involved (see DABConstants.operationControl*)

public Java.lang.Object getValue()

Returns a copy of the attribute's value. This is either the actual value, when a read request was issued, or the former value when a change request was issued.

- DABConstants.operationControlSetVolume: The former volume of the receiver is returned. It is of type Integer in the range from 0 to 100 (percent).

- DABConstants.operationControlGetVolume: The current volume of the receiver is returned. It is of type Integer in the range from 0 to 100 (percent).
- DABConstants.operationControlSetServiceFollowing: The former state of the service following feature is returned. It is of type Boolean: true indicates that the service following was switched on, false indicates that it was switched off.
- DABConstants.operationControlGetServiceFollowing: The current state of the service following feature is returned. It is of type Boolean: true indicates that the service following is switched on, false indicates that it is switched off.
- DABConstants.operationControlGetServiceFollowingNotifications: null is returned.
- DABConstants.operationControlSetDRCMode: The former state of the DRCMode following feature is returned. It is of type Boolean: true indicates that the DRCMode was switched on, false indicates that it was switched off.
- DABConstants.operationControlGetDRCMode: The current state of DRC mode is returned. It is of type Boolean: true indicates that the DRC mode is switched on, false indicates that it is switched off.
- DABConstants.operationControlGetDRCModeNotifications: null is returned.

See Also

dab.DABSource#operationControlReq operationControlReq

public class **dab.events.RespondConflictResolutionCnfEvent** extends dab.events.DABEvent

Constructors **protected RespondConflictResolutionCnfEvent(**

DABSource source,

int _result)

Methods **public int getResult()**

Returns the status of the respondConflictResolution command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public class **dab.events.GetComponentInfoCnfEvent** extends dab.events.DABEvent

The GetComponentInfoCnfEvent is generated in response to a GetComponentInfoReq request.

See Also

dab.DABListener#getComponentInfoCnf getComponentInfoCnf

Version

1.01

Constructors **protected GetComponentInfoCnfEvent(**

DABSource source,

int result,

ComponentInfo componentInfo)

Create a GetComponentInfoCnfEvent object.

Methods **public int getResult()**

Returns the status of the GetComponentInfo command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public dab.si.ComponentInfo getComponentInfo()

Returns information about the subscribed DAB Component.

public class **dab.events.SelectSICnfEvent** extends dab.events.DABEvent

The SelectSICnfEvent is generated in response to a selectSIReq request.

See Also

dab.DABListener#selectSICnf selectSICnf

Version

1.02

Constructors **protected SelectSICnfEvent(**

DABSource source,

int result,

boolean ensembleInfo,

boolean serviceInfo,

boolean componentInfo,

boolean autoDelivery)

Creates a SelectSICnfEvent.

Methods **public int getResult()**

Returns the status of the SelectSI command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public boolean getEnsembleInfo()

Returns the ensemble info. This value specifies if the DAB client is subscribed to ensemble-specific notifications. The following values are supported:

- true: The client is notified about ensembleAdded, ensembleChanged and ensembleRemoved events.
- false: The client is not notified about ensembleAdded, ensembleChanged and ensembleRemoved events.

public boolean getServiceInfo()

Returns service info. This value specifies if the DAB client is subscribed to service-specific notifications. The following values are supported:

- true: The client is notified about serviceAdded, serviceChanged and serviceRemoved events.
- false: The client is not notified about serviceAdded, serviceChanged and serviceRemoved events.

public boolean getComponentInfo()

Returns component info. This value specifies if the client is subscribed to component-specific notifications. The following values are supported:

- true: The client is notified about componentAdded, componentChanged and componentRemoved events.
- false: The client is not notified about componentAdded, componentChanged and componentRemoved events.

public boolean getAutoDelivery()

Returns auto delivery. This value specifies if the information related to the notification is sent together with the notification (serviceInfoNtf) or not. The following values are supported:

- true: The serviceInfoNtf method delivers the notification together with the information object. This is only possible for -Added and -Changed notifications but not for -Removed because in the latter case the service element is no longer existing.
- false: The serviceInfoNtf method delivers only the notification. The information object (EnsembleInfo, ServiceInfo or ComponentInfo) itself can be obtained by use of getEnsembleInfoReq, getServiceInfoReq or getComponentInfoReq.

public class **dab.events.SelectApplicationCnfEvent** extends dab.events.DABEvent

SelectApplicationCnfEvent is generated in response to a selectApplicationReq request.

See Also

dab.DABSource#selectApplicationReq selectApplicationReq

Version

0.2

Constructors **protected SelectApplicationCnfEvent(**

DABSource source,

int result,

DABAppProxy proxy)

Creates a SelectApplicationCnfEvent object.

Methods **public int getResult()**

Returns the status of the SelectApplication command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public dab.DABAppProxy getApplicationProxy()

Returns the proxy for the loaded application. The value is null, when result != DABConstants.resultOK

public class **dab.events.SelectComponentStreamCnfEvent** extends dab.events.DABEvent

SelectComponentStreamCnfEvent is generated in response to a selectComponentStreamReq.

Version

1.02

Constructors **protected SelectComponentStreamCnfEvent(**

DABSource source,

int result,

int streamType,

InputStream stream)

Methods **public int getResult()**

Returns the status of the SelectComponentStream command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public int getStreamType()

Returns the type of the stream (see DABConstants.streamType*).

public Java.io.InputStream getStream()

Returns the stream.

public class **dab.events.TuneCnfEvent** extends dab.events.DABEvent

The TuneCnfEvent is generated in response to a tuneReq request.

See Also

dab.DABListener#tuneCnf tuneCnf

Version

1.01

Constructors **protected TuneCnfEvent(**

DABSource source,

int result,

int tuneState,

int tuneFrequency,

int transmissionMode,

int synchronizationState)

Creates a TuneCNfEvent.

Methods

public int getResult()

Returns the status of the Tune command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public int getTuneState()

Returns the current tune state independent from the command result indicated by result. The following values are supported:

- tuneStateNotTuned: The DAB receiver is not tuned to a known frequency. An error has occurred in this case and the following parameters are undefined.
- tuneStateTuned: The DAB receiver is tuned to a frequency specified by tuneFrequency and the following parameters are defined.

public int getTuneFrequency()

Return the frequency currently in use.

public int getTransmissionMode()

Returns the DAB transmission mode the DAB receiver has detected. The following values are supported:

- DABConstants.transmissionMode1: The found DAB Ensemble is sent in Transmissionmode 1.
- DABConstants.transmissionMode2: The found DAB Ensemble is sent in Transmissionmode 2.
- DABConstants.transmissionMode3: The found DAB Ensemble is sent in Transmissionmode 3.
- DABConstants.transmissionMode4: The found DAB Ensemble is sent in Transmissionmode 4.
- DABConstants.transmissionModeUnknown: The transmission mode is unknown.

public int getSynchronizationState()

Returns the current synchronization state of the DAB Receiver. The following values are supported:

- DABConstants.stateNotSynchronized: The DAB Receiver is not synchronized. This is the lowest level of synchronization.
- DABConstants.stateDABSignaldetected: The DAB Receiver has detected a DAB Signal.
- DABConstants.stateTimeAndFrequencySynchronized: The DAB Receiver is time and frequency synchronized

- DABConstants.stateFICReadable: The Service Information channel is readable. This is the highest level of synchronization.

public class **dab.events.GetServiceInfoCnfEvent** extends dab.events.DABEvent

The GetServiceInfoCnfEvent is generated in response to a GetServiceInfoReq request.

See Also

dab.DABListener#getServiceInfoCnf getServiceInfoCnf

Version

1.01

Constructors **protected GetServiceInfoCnfEvent(**

DABSource source,

int result,

ServiceInfo serviceInfo)

Creates a GetServiceInfoCnfEvent object.

Methods **public int getResult()**

Returns the status of the GetServiceInfo command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public dab.si.ServiceInfo getServiceInfo()

Returns a reference to an object which provides information about the DAB Service.

public class **dab.events.GetLocationInfoCnfEvent** extends dab.events.DABEvent

The GetLocationInfoCnfEvent is generated in response to a getLocationInfoReq request.

See Also

dab.DABListener#getLocationInfoCnf getLocationInfoCnf

Version

1.01

Constructors **protected GetLocationInfoCnfEvent(**

DABSource source,

int result,

int mode,

int deliveredDelta,

int deliveredAccuracy)

Creates a GetLocationInfoCnfEvent object.

Methods **public int getResult()**

Returns the status of the GetLocationInfo command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public int getMode()

Returns the mode of the GetLocationInfo command.

See Also

dab.DABSource#getLocationInfoReq getLocationInfoReq

public int getDeliveredDelta()

Returns the delivered delta of the GetLocationInfo command.

See Also

dab.DABSource#getLocationInfoReq getLocationInfoReq

public int getDeliveredAccuracy()

Returns the delivered accuracy of the GetLocationInfo command.

See Also

dab.DABSource#getLocationInfoReq getLocationInfoReq

public class **dab.events.ScanCnfEvent** extends dab.events.DABEvent

ScanCnfEvent is generated in response to a scanReq request.

See Also

dab.DABSource#scanReq scanReq

Version

1.01

Constructors **protected ScanCnfEvent(**

DABSource source,

int result,

int tuneState,

int tuneFrequency,

int transmissionModes,

int noOfEnsemblesFound)

Creates a ScanCnfEvent object.

Methods **public int getResult()**

Returns the status of the Scan command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public int getTuneState()

Returns the current tune state. The following values are supported:

- DABConstants.tuneStateNotTuned: The DAB Receiver is not tuned to a known frequency.
- DABConstants.tuneStateTuned: The DAB Receiver is tuned to a frequency specified by tuneFrequency.

public int getTuneFrequency()

Returns the currently tuned frequency.

public int getTransmissionModes()

Returns the transmission modes a DAB Receiver should look for DAB Ensembles. The default value is DABConstants.transmissionModeAutomatic which means that the receiver is automatically detecting the Transmissionmode. The returned value is a flag field supporting the following flags which can be specified together:

- DABConstants.transmissionModeAutomatic: The Transmissionmode is automatically detected
- DABConstants.transmissionMode1: At the specified frequency it is tested if a DAB Ensemble is sent in Transmissionmode 1.
- DABConstants.transmissionMode2: At the specified frequency it is tested if a DAB Ensemble is sent in Transmissionmode 2.
- DABConstants.transmissionMode3: At the specified frequency it is tested if a DAB Ensemble is sent in Transmissionmode 3.
- DABConstants.transmissionMode4: At the specified frequency it is tested if a DAB Ensemble is sent in Transmissionmode 4.

public int getNoOfEnsemblesFound()

Returns the number of DAB Ensembles that have been found during the execution of the scan command.

public class **dab.events.ConflictResolutionNtfEvent** extends dab.events.DABEvent

Constructors **protected ConflictResolutionNtfEvent(**

DABSource source,

int _transaction,

int _turn,

int _operation,

int _suboperation)

Methods **public int getTransaction()**

Delivers the transaction number. This can be used to provide a transaction context.

public int getTurn()

Returns a code for the turn of the resource conflict resolution protocol:

- DABConstants.conflictResolutionTurnProceed: This is sent to the client which requested the operation. It indicates that there is a resource conflict. The client is asked whether he likes to proceed.
- DABConstants.conflictResolutionTurnProbe: This notification is sent to all clients in order to probe for their willingness to release the needed resources.
- conflictResolutionTurnStop: The client is asked to stop the indicated operation in order to release the resources.
- DABConstants.conflictResolutionTurnPreempt: The client is informed that the indicated operation was stopped. This action shall normally only be taken, when the client failed to do a stop in the previous turn.

public int getOperation()

Gives back a code of the involved operation (see DABConstants.conflictResolutionOperation*)

public int getSuboperation()

Gives back a code of the involved suboperation (see DABConstants.conflictResolutionSuboperation*)

public class **dab.events.GetEnsembleInfoCnfEvent** extends dab.events.DABEvent

The GetEnsembleInfoCnfEvent is generated in response to a GetEnsembleInfoReq request.

See Also

dab.DABListener#getEnsembleInfoCnf getEnsembleInfoCnf

Version

1.01

Constructors **protected GetEnsembleInfoCnfEvent(**

DABSource source,

int result,

EnsembleInfo ensembleInfo)

Creates a GetEnsembleInfoCnfEvent object.

Methods **public int getResult()**

Returns the status of the GetEnsembleInfo command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public dab.si.EnsembleInfo getEnsembleInfo()

Returns a reference to an object which provides information about a DAB Ensemble.

public class **dab.events.AppStateChangeEvent** extends Java.util.EventObject

AppStateChangeEvent reflects state changes in an application.

See Also

dab.AppStateChangeEventListener AppStateChangeEventListener

"Digital Video Broadcasting (DVB) Multimedia Home Platform (MHP), TS 101 812"

Version

0.2

Constructors **protected AppStateChangeEvent(**

AppStateChangeEventSource source,

int fromState,

int toState,

boolean failed)

Methods **public int getFromState()**

Returns the state from which the application was switching

public int getToState()

Returns the state to which the application switched

public boolean hasFailed()

Indicates whether the switching failed (=true) or not (=false)

public class **dab.events.SINtfEvent** extends dab.events.DABEvent

The SINtfEvent is generated in response to a selectSIReq request.

See Also

dab.DABListener#siNtf siNtf

Version

1.04

Constructors **protected SINtfEvent(**

DABSource source,

int notification,

int updateFlags,

SIId serviceInfoId,

EnsembleInfo ensembleInfo,

ServiceInfo serviceInfo,

ComponentInfo componentInfo)

Creates a SINtfEvent object.

Methods **public int getNotification()**

Returns the notification type. The following values are supported:

Ensemble-related notification:

- ensembleAdded: A new DAB Ensemble is available.
- ensembleRemoved: A known DAB Ensemble is no longer available. All dependent services and components are also no longer available.
- ensembleChanged: A known DAB Ensemble has changed which means its attributes have changed. This case signals changes to the ensemble itself and not changes in linking to child services. This means if a child service is added or removed this is not indicated by an ensembleChanged notification.

Service-related notification:

- serviceAdded: A new DAB Service is available.
- serviceRemoved: A known DAB Service is no longer available. All dependent components are also no longer available.
- serviceChanged: A known DAB Service has changed which means its attributes have changed. This case signals changes to the service itself and not changes in linking to child components. This means if a child component is added or removed this is not indicated by a DABConstants.serviceChanged notification.

Component-related notification:

- componentAdded: A new DAB Component is available.
- componentRemoved: A known DAB Component is no longer available.
- componentChanged: A known DAB Component has changed which means its attributes have changed.

public int getUpdateFlags()

Returns more detailed information about which part of the Service Directory has changed. The value is a flag field and supports the following flags depending on the service element type.

In case of a DABConstants.ensembleAdded or DABConstants.ensembleChanged notification the following values are defined:

- DABConstants.updatedLabel: The Ensemble label has changed.
- DABConstants.updatedCountry: The Country information which specifies which area is covered by the Ensemble has changed.

In case of a DABConstants.serviceAdded or DABConstants.serviceChanged notification the following values are defined:

- DABConstants.updatedLabel: The Service label has changed.
- DABConstants.updatedCountry: The Country information which specifies which area is covered by the Service has changed.
- DABConstants.updatedTimeOffset: The time offset for the specified Service has changed.
- DABConstants.updatedRegion: The region has changed.
- DABConstants.updatedStaticProgrammeType: The static programme type information of the specified audio service has changed.
- DABConstants.updatedDynamicProgrammeType: The static programme type information of the specified audio service has changed.
- DABConstants.updatedAnnouncement: The announcement information of the specified audio service has changed.
- DABConstants.updatedLanguage: The language information of the specified audio service has changed.
- DABConstants.updatedRegionId: The region identifier has changed.
- DABConstants.updatedRegionLabel: The region label has changed.
- DABConstants.updatedAnnouncementSupport: The announcement support information of the specified audio service has changed.
- DABConstants.updatedProgrammeNumber: The programme number has changed

In case of a DABConstants.componentAdded or DABConstants.componentChanged notification the following values are defined:

- DABConstants.updatedLabel: The component label has changed.
- DABConstants.updatedLanguage: The language information of the specified audio component has changed.
- DABConstants.updatedStartObject: In case of a BroadcastWebSite application carried in the related component this indicates that the start object (homepage) is known.
- DABConstants.updatedObjectDirectory: The MOT object directory has changed
- DABConstants.updatedAudioComponent: The link to the audio component has changed
- DABConstants.updatedBitrate: The bitrate has changed.

public dab.si.SIId getServiceInfoId()

Returns the instance of the service element (Ensemble, Service, Component) that has changed. It can be used in order to request the related information object with the getEnsembleInfo, getServiceInfo or getComponentInfo command.

public dab.si.EnsembleInfo getEnsembleInfo()

If notification signals an ensemble-related notification of type DABConstants.ensembleAdded or DABConstants.ensembleChanged and AutoDelivery has been activated with the subscription, then the returned value refers to an ensemble information object. If AutoDelivery is not activated or this is a service-related or component-related notification then null is returned.

public dab.si.ServiceInfo getServiceInfo()

If notification signals a service-related notification of type DABConstants.serviceAdded or DABConstants.serviceChanged and AutoDelivery has been activated with the subscription, then the returned value refers to a service information object. If AutoDelivery is not activated or this is an ensemble-related or component-related notification then null is returned.

public dab.si.ComponentInfo getComponentInfo()

If notification signals a component-related notification of type DABConstants.componentAdded or DABConstants.componentChanged and AutoDelivery has been activated with the subscription, then the value refers to a component information object. If AutoDelivery is not activated or this is an ensemble-related or service-related notification then null is returned.

public class **dab.events.SearchNtfEvent** extends dab.events.DABEvent

SearchNtfEvent is generated in response to a searchReq request.

See Also

dab.DABListener#searchNtf searchNtf

Version

1.01

Constructors **protected SearchNtfEvent(**

DABSource source,

int tuneFrequency,

int notifications)

Creates a SearchNtfEvent.

Methods **public int getTuneFrequency()**

Returns the currently tuned frequency in Hertz.

public int getNotifications()

Returns the notification type. The value is a flag field supporting the following flags which can be specified together:

- notifications16kHzSteps: A 16 kHz step has been made.
- notificationsTableEntry: A frequency of the specified frequency table has been reached.
- notificationsSearchStarted: Searching for a DAB Ensemble has been started.

public class **dab.events.SearchCnfEvent** extends dab.events.DABEvent

SearchCnfEvent is generated in response to a searchReq request.

See Also

dab.DABListener#searchCnf searchCnf

Version

1.01

Constructors

protected SearchCnfEvent(

DABSource source,

int result,

int tuneState,

int tuneFrequency,

int transmissionMode,

int synchronizationState)

Create a SearchCnfEvent

Methods

public int getResult()

Returns the result. This value indicates the status of the Tune command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public int getTuneState()

Return the tune state. This value indicates the current tune state independent from the command result indicated by result. The following values are supported:

- DABConstants.stateNotTuned: The DAB receiver is not tuned to a known frequency. An error has ccurred in this case and the following parameters are undefined.
- DABConstants.stateTuned: The DAB receiver is tuned to a frequency specified by tuneFrequency and the following parameters are defined.

public int getTuneFrequency()

Return the tune frequency in use.

public int getTransmissionMode()

Returns the transmission mode. This value specifies the DAB transmission mode the DAB receiver has detected. The following values are supported:

- DABConstants.transmissionMode1: The found DAB Ensemble is sent in Transmissionmode 1.
- DABConstants.transmissionMode2: The found DAB Ensemble is sent in Transmissionmode 2.
- DABConstants.transmissionMode3: The found DAB Ensemble is sent in Transmissionmode 3.
- DABConstants.transmissionMode4: The found DAB Ensemble is sent in Transmissionmode 4.
- DABConstants.transmissionModeUnknown: The transmission mode is unknown.

public int getSynchronizationState()

Returns the synchronization state. This value specifies the current synchronization state of the DAB Receiver. The following values are supported:

- DABConstants.stateNotSynchronized: The DAB Receiver is not synchronized. This is the lowest level of synchronization.
- DABConstants.stateDABSignaldetected: The DAB Receiver has detected a DAB Signal.
- DABConstants.stateTimeAndFrequencySynchronized: The DAB Receiver is time and frequency synchronized
- DABConstants.stateFICReadable: The Service Information channel is readable. This is the highest level of synchronization.

public class **dab.events.SelectReceptionInfoCnfEvent** extends dab.events.DABEvent

The SelectReceptionInfoCnfEvent is generated in response to a selectReceptionInfoReq request.

See Also

dab.DABSource#selectReceptionInfoReq selectReceptionInfoReq

Version

1.01

Constructors

protected SelectReceptionInfoCnfEvent(

DABSource source,

int result,

boolean synchronizationNotifications,

boolean bitErrorRateNotifications,

boolean muteStateNotifications)

Creates a SelectReceptionInfoCnfEvent object.

Methods

public int getResult()

Returns the status of the SelectReceptionInfo command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public boolean getSynchronizationNotifications()

Returns synchronization notifications. The value specifies if the client is notified about state changes concerning DAB signal synchronization. If the returned value is true notifications are provided, if it is false no notifications are provided.

public boolean getBitErrorRateNotifications()

Returns bit error rate notifications. The value specifies if the client is notified about state changes concerning the biterrorrate. If the returned is true notifications are provided, if it is false no notifications are provided.

public boolean getMuteStateNotifications()

Returns mute state notifications. This value specifies if the client is notified about state changes concerning the mute state of the audio decoder. If the returned value is true notifications are provided, if it is false no notifications are provided.

public class **dab.events.SelectObjectCnfEvent** extends dab.events.DABEvent

The SelectObjectCnfEvent is generated in response to a selectObjectReq request.

See Also

dab.DABSource#selectObjectReq selectObjectReq

Version

1.02

Constructors

protected SelectObjectCnfEvent(

DABSource source,

int result,

ComponentId componentId,

ObjectId objectId,

int requestMode,

boolean replaceSelections,

Date accessTime)

Creates a SelectObjectCnfEvent object.

Methods

public int getResult()

Returns the status of the SelectObject command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public dab.si.ComponentId getComponentId()

Returns the component the object is belonging to.

public dab.data.ObjectId getObjectId()

Returns the selected object

public int getRequestMode()

Returns the current selection mode for the specified object.

- DABConstants.requestModeOff: The object selection is removed.
- DABConstants.requestModeOnce: The object is requested for one-time delivery. After the first reception from the broadcast channel the object is delivered to the connected DAB client. The client is not notified about new versions.

- DABConstants.requestModeUpdate: The object is requested for update delivery. After the first reception from the broadcast channel the object is delivered to the connected client. Additionally each new version of the object is delivered.

public boolean getReplaceSelections()

Returns all current object selections belonging to the component identified by serviceInfoId are replaced with this selection. If the returned value is true, then all selections are removed. If the returned value is false, then existing selections remain unchanged.

public Java.util.Date getAccessTime()

Returns the expected relative access time for delivery of the object.

public class **dab.events.ServiceFollowingNtfEvent** extends dab.events.DABEvent

The ServiceFollowingNtfEvent is generated when a service following action is taken by the receiver.

See Also

dab.DABListener#serviceFollowingNtf serviceFollowingNtf

Version

1.02

Constructors **protected ServiceFollowingNtfEvent(**

DABSource source,

int action,

EnsembleId ensembleId,

ComponentId componentId)

Creates a ServiceFollowingNtfEvent event.

Methods **public int getAction()**

Returns the service following action code. The following codes are possible:

- DABConstants.ServiceFollowingLeavingService: The service following has started. The current ensemble and service are left.

- DABConstants.ServiceFollowingTryingAlternativeService: An alternative frequency is tried.

- DABConstants.ServiceFollowingSelectingService: The service following is finished. The receiver is tuned to a new ensemble and audio service.

public dab.si.EnsembleId getEnsembleId()

Returns the service identifier of the ensemble that is involved in the action

public dab.si.ComponentId getComponentId()

Returns the service identifier of the component that is involved in the action

public class **dab.events.SystemFailureNtfEvent** extends dab.events.DABEvent

The System-Failure event is generated when a fatal error has happened in the system.

Version

1.02

Constructors **protected SystemFailureNtfEvent(**

DABSource source,

int reason)

Methods **public int getReason()**

Returns the reason for the system failure (the codes are compatible with DABConstants.result*)

public class **dab.events.SelectComponentCnfEvent** extends dab.events.DABEvent

The SelectComponentCnfEvent is generated in response to a selectComponentReq request.

See Also

dab.DABListener#selectComponentCnf selectComponentCnf

Version

1.02

Constructors **protected SelectComponentCnfEvent(**

DABSource source,

int result,

ComponentId componentId,

int selectionMode)

Create a SelectComponentCnfEvent object.

Methods **public int getResult()**

Returns the status of the SelectComponent command. If it is equal to DABConstants.resultOK, the command was successful. Otherwise an error has occurred.

public dab.si.ComponentId getComponentId()

Returns the component which delivers the started or stopped application.

public int getSelectionMode()

Return the selection mode for the component. The following flags are supported:

- DABConstants.selectionModeReplace: All former selected components of the same type are stopped and the specified component is started. The same type means an audio component replaces any other selected audio component, a data component replaces all other selected i ndependent data components and a programme-associated data component replaces all other selected programme-associated data components.
- DABConstants.selectionModeAdd: The application delivered by the specified component is started. Other selected components are not affected.
- DABConstants.selectionModeRemove: The selection of the specified component is removed.
- DABConstants.selectionModeRemoveAll: All existing component selections are removed. The parameter serviceId is set to null in this case.

public class **dab.events.DRCModeNtfEvent** extends dab.events.DABEvent

The DRCModeNtfEvent is generated when a DRC mode change is taken by the receiver.

See Also

dab.DABListener#serviceFollowingNtf serviceFollowingNtf

Version

1.02

Constructors **protected DRCModeNtfEvent(**

DABSource source,

boolean currentState)

Creates a DRCModeNtfEvent event.

Methods **public boolean getMode()**

Returns the DRC mode. The return value is true, if it is now switched on; it is false when it is now switched off.

public class **dab.events.ReceptionInfoNtfEvent** extends dab.events.DABEvent

ReceptionInfoNtfEvent is generated in response to a selectReceptionInfoReq request.

See Also

dab.DABListener#receptionInfoNtf receptionInfoNtf

Version

1.01

Constructors **protected ReceptionInfoNtfEvent(**

DABSource source,

int updateFlags,

int synchronizationState,

int bitErrorRateState,

int muteState)

Creates a ReceptionInfoNtfEvent object.

Methods

public int getUpdateFlags()

Returns the updateFlags. This value is a flag field which indicates if synchronization, biterrorrate and/or mute state has changed. The following values are supported:

- DABConstants.syncUpdateSynchronizationState: The synchronization state has changed. The new state is specified by synchronizationState.
- DABConstants.syncUpdateBitErrorRateState: The biterrorrate state has changed. The new state is specified by bitErrorRateState.
- DABConstants.syncUpdateMuteState: The mute state has changed. The new state is specified by muteState.

public int getSynchronizationState()

Returns the synchronization state. This value specifies the current synchronization state of the DAB Receiver. The following values are supported:

- DABConstants.stateSynchronizationStateUnknown: The synchronization state is not known.
- DABConstants.stateNotSynchronized: The DAB Receiver is not synchronized. This is the lowest level of synchronization.
- DABConstants.stateDABSignaldetected: The DAB Receiver has detected a DAB Signal.
- DABConstants.stateTimeAndFrequencySynchronized: The DAB Receiver is time and frequency synchronized
- DABConstants.stateFICReadable: The Service Information channel is readable. This is the highest level of synchronization.

public int getBitErrorRateState()

Returns the bit error rate state. This value specifies the current biterrorrate state. The following values are supported:

- DABConstants.bitErrorRateLevelUnknown: The current biterrrorrate is unknown.
- DABConstants.bitErrorRateLevel1: The biterrorrate is smaller than 5e-4.
- DABConstants.bitErrorRateLevel2: The biterrorrate is smaller than 5e-3.
- DABConstants.bitErrorRateLevel3: The biterrorrate is smaller than 5e-2.
- DABConstants.bitErrorRateLevel4: The biterrorrate is smaller than 1e-1.
- DABConstants.bitErrorRateLevel5: The biterrorrate is equal or larger than 1e-1.

public int getMuteState()

Returns the mute state. This value specifies the current mute state. The following values are supported:

- DABConstants.muteStateUnknown: The current mute state is unknown.
- DABConstants.muteStateMuting: The DAB Receiver is permanently muting.
- DABConstants.muteStatePartialMuting: Some audio frames were muted.
- DABConstants.muteStateNotMuting: No frame was muted.

public class **dab.events.ScanNtfEvent** extends dab.events.DABEvent

ScanNtfEvent is generated in response to a scanReq request.

See Also

dab.DABSource#scanReq scanReq

Version

1.01

Constructors

protected ScanNtfEvent(

DABSource source,

int tuneFrequency,

int notifications)

Creates a ScanNtfEvent object.

Methods

public int getTuneFrequency()

Returns the currently tuned frequency in Hertz.

public int getNotifications()

Returns the notification type. The returned value is a flag field supporting the following flags which can be specified together:

- DABConstants.notifications16kHzSteps: A 16 kHz step has been made.
- DABConstants.notificationsTableEntry: A frequency of the specified frequency table has been reached.
- DABConstants.notificationsEnsembleFound: A DAB Ensemble has been found.

public class **dab.events.ObjectNtfEvent** extends dab.events.DABEvent

The ObjectNtfEvent is generated in response to an ObjectNtfReq request.

See Also

dab.DABListener#objectNtf objectNtf

Version

1.02

Constructors **protected ObjectNtfEvent(**

DABSource source,

ComponentId componentId,

ObjectId objectId,

int selectionState,

DABObject object)

Creates an ObjectNtfEvent event.

Methods **public dab.si.ComponentId getComponentId()**

Returns the component the object is belonging to.

public dab.data.ObjectId getObjectId()

Returns the id of the selected object.

public int getSelectionState()

Returns the current selection state. The following values are supported:

- DABConstants.selectionStateOK: This message delivers a selected object to the connected DAB client. The object is available by parameter object.
- DABConstants.selectionStateDelayed: Delivery of the selected object is delayed.
- DABConstants.selectionStateTerminated: Transmission of the selected object is terminated. The object selection is removed.

public dab.data.DABObject getObject()

Returns a reference to a DAB object. As the DABObject class is just an abstraction of data objects, you have to check the actual type of the returned object (e.g. instance of MOTObject) to know which kind of object is delivered.

public class **dab.events.LocationInfoNtfEvent** extends dab.events.DABEvent

The LocationInfoNtfEvent represents notifications related to the GetLocationInfo command.

See Also

dab.DABListener#locationInfoNtf locationInfoNtf

Version

1.02

Constructors **protected LocationInfoNtfEvent(**

DABSource source,

Date timestamp,

int[] regionIds,

LocationInfo info)

Creates a LocationInfoNtfEvent object.

Methods **public Java.util.Date getTimestamp()**

Returns the timestamp

public int[] getRegionIds()

Returns the list of region identifiers. When no region ids are available or are not requested, the result is an empty array.

public dab.data.LocationInfo getLocationInfo()

Returns the location info. When the location info was not requested, the result is null.

A.4 Package dab.data

public class **dab.data.BWSDirectoryIndex**

The BWSDirectoryIndex class represents profile information in a BWS directory

Version

1.01

Constructors **public BWSDirectoryIndex()**

Methods **public int getProfileId()**

returns the profile id that this index is for

public Java.lang.String getIndexName()

returns the index page name

public class **dab.data.ProgrammeNumber**

ProgrammeNumber represents a programme number that can be used for "programming" a service.

Version

1.00

Constructors **public ProgrammeNumber()**

Methods **public Java.util.Date getTransmissionTime()**

returns the transmission time

public boolean isInterrupted()

signals, whether the programme is interrupted by later continued

public boolean isRedirected()

signals, whether the programme is redirected to a different service and time

See Also

dab.data.ProgrammeNumber#getNewService getNewService

public dab.si.ServiceId getNewService()

returns the ServiceId of the new service when the programme is redirected

See Also

dab.data.ProgrammeNumber#isRedirected isRedirected

public abstract class **dab.data.DABObject**

The DABObject class represents all kind of data that is transported via DAB.

Version

1.01

Constructors **public DABObject()**

public class **dab.data.MOTObject** extends dab.data.DABObject implements dab.data.MOTObjectHeader

The MOTObject represents data that is transported via the MOT protocol.

Version

1.06

Constructors **public MOTObject()**

Methods **public int getContentType()**

Returns the content type (the main category)

public int getContentSubtype()

Returns the content subtype (the exact type)

public byte[] getBody()

Returns the body of the object (the actual content)

public Java.lang.String getContentDescription()

Returns the content description

Throws

DABNotAvailableException - when the content description is not available

public int getContentDescriptionCharset()

Returns the charset of the content description (see DABConstants.charset*)

Throws

DABNotAvailableException - when the charset is not available

public Java.lang.String getContentName()

Returns the content name

Throws

DABNotAvailableException - when the content name is not available

public int getContentNameCharset()

Returns the charset of the content name (see DABConstants.charset*)

Throws

DABNotAvailableException - when the charset is not available

public dab.data.Label getLabel()

Returns the label

Throws

DABNotAvailableException - when the priority is not available

public int getPriority()

Returns the priority (0=lowest priority; 255=highest priority)

Throws

DABNotAvailableException - when the priority is not available

public int getRepetitionDistance()

Returns the repetition distance (in ms)

Throws

DABNotAvailableException - when the repetition distance is not available

public int getVersionNumber()

Returns the version number of the object

Throws

DABNotAvailableException - when the version is not available

public boolean getValidity()

Returns false, if validity is now; otherwise true. Note, if the validity is set to false the referred time routines have to be ignored.

See Also

dab.data.MOTObject#getCreationTime getCreationTime

dab.data.MOTObject#getStartValidity getStartValidity

dab.data.MOTObject#getExpireTime getExpireTime

dab.data.MOTObject#getTriggerTime getTriggerTime

public Java.util.Date getCreationTime()

Returns the authoring date of the object

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObject#getValidity getValidity

public Java.util.Date getStartValidity()

Returns the date after which the object is valid

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObject#getValidity getValidity

public Java.util.Date getExpireTime()

Returns the date after which the object is not valid anymore

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObject#getValidity getValidity

public Java.util.Date getTriggerTime()

Returns the date for presenting the object

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObject#getValidity getValidity

public Java.lang.String toString()

Returns a textual representation of the object

public Java.lang.String getMimeType()

Returns the MIME type of the object

public int getCompressionType()

Returns the compression type of the object

Throws

DABNotAvailableException - when the content description is not available

public class **dab.data.MOTDirectoryObject** extends dab.data.MOTObject

The MOTDirectoryObject class represents a MOT carousel directory of a component

Version

1.01

Constructors **public MOTDirectoryObject()**

Methods **public int getNumberOfObjects()**

Returns number of objects described by the directory

public int getCarouselPeriod()

Returns maximum time (in tenths of second) for the carousel to cycle

public dab.data.MOTObjectHeader[] getContents()

Returns MOT Headers for objects described by the directory

public class **dab.data.ProgrammeType**

ProgrammeType represents provided programme types of a certain service. It consists of an international code, an optional coarse code and two optional fine codes.

Version

1.01

Constructors **public ProgrammeType(**

int internationalCode,

byte[] fineCode,

boolean hasCoarseCode,

int coarseCode)

Methods **public int getInternationalCode()**

Returns the international code

public int getCoarseCode()

Returns the coarse code

Throws

DABNotAvailableException - when the code is not available

public byte[] getFineCode()

Returns a reference to the fine codes

public class **dab.data.BWSObject** extends dab.data.MOTObject

The BWSObject class represents data that is part of the BWS service

Version

1.01

Constructors **public BWSObject()**

Methods **public Java.lang.String getAdditionalHeader()**

Returns the additional header (the HTTP header field)

Throws

DABNotAvailableException - when the content description is not available

public byte[] getProfileSubset()

Returns the list of profiles for which the object is relevant

Throws

DABNotAvailableException - when the content description is not available

public int getCryptoAlgorithm()

Returns the crypto algorithm for the object

public int getScramblingMode()

Returns the scrambling mode for the object

public dab.data.SubscriberInfo getSubscriberInfo()

Returns information about how to subscribe to the service

Throws

DABNotAvailableException - when the content description is not available

public class **dab.data.Label**

Label models a textual string which is used in the DAB System for service labels, object labels and so on. It contains a text with max. 16 characters. Additionally the character set is indicated and it is specified how the label is to be displayed on a display with less than 16 characters.

Version

1.01

Constructors

public Label(

int charSet,

String label,

int characterFlagField)

Methods

public int getCharset()

Returns the charset (see DABConstants.charset*)

public int getCharacterFlagField()

Returns the character flag field

public Java.lang.String getLabel()

Returns the content of the label

public class **dab.data.AnnouncementSupport**

AnnouncementSupport represents supported announcement types of a certain DAB service, e.g. News, Traffic and so on.

Version

1.01

Constructors **public AnnouncementSupport(int announcementSupportFlags)**

Methods **public boolean equals(int announcementSupportFlags)**

Returns true when this object supports all the given flags; otherwise false

public boolean support(int announcement)

Returns true when the announcement is supported; otherwise false

public class **dab.data.BWSDirectoryObject** extends dab.data.MOTDirectoryObject

The BWSDirectoryObject class represents the carousel directory of a BWS user application.

Version

1.01

Constructors **public BWSDirectoryObject()**

Methods **public dab.data.BWSDirectoryIndex[] getDirectoryIndex()**

returns a list of profile index pages

public class **dab.data.SubscriberInfo**

SubscriberInfo contains information how to subscribe to a service.

Version

1.00

Constructors **public SubscriberInfo()**

Methods **public int getReason()**

returns a flag field (see DABConstants.subscriberInfo*) that explains why the related BWS object could not be descrambled

public int getEncryptionSpecificFlags()

returns a flag field that can be used for redirection purposes

public Java.lang.String getContentName()

returns the content name of the alternative object

public interface **dab.data.MOTObjectHeader**

The MOTObjectHeader represents the header information of an MOT object

Version

1.02

Methods

public int getContentType()

Returns the content type (the main category)

public int getContentSubtype()

Returns the content subtype (the exact type)

public Java.lang.String getContentDescription()

Returns the content description

Throws

DABNotAvailableException - when the content description is not available

public int getContentDescriptionCharset()

Returns the charset of the content description (see DABConstants.charset*)

Throws

DABNotAvailableException - when the charset is not available

public Java.lang.String getContentName()

Returns the content name

Throws

DABNotAvailableException - when the content name is not available

public int getContentNameCharset()

Returns the charset of the content name (see DABConstants.charset*)

Throws

DABNotAvailableException - when the charset is not available

public dab.data.Label getLabel()

Returns the label

Throws

DABNotAvailableException - when the priority is not available

public int getPriority()

Returns the priority (0=lowest priority; 255=highest priority)

Throws

DABNotAvailableException - when the priority is not available

public int getRepetitionDistance()

Returns the repetition distance (in ms)

Throws

DABNotAvailableException - when the repetition distance is not available

public int getVersionNumber()

Returns the version number of the object

Throws

DABNotAvailableException - when the version is not available

public boolean getValidity()

returns false, if validity is now; otherwise true. Note, if the validity is set to false the referred time routines have to be ignored.

See Also

dab.data.MOTObjectHeader#getCreationTime getCreationTime

dab.data.MOTObjectHeader#getStartValidity getStartValidity

dab.data.MOTObjectHeader#getExpireTime getExpireTime

dab.data.MOTObjectHeader#getTriggerTime getTriggerTime

public Java.util.Date getCreationTime()

returns the authoring date of the object

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObjectHeader#getValidity getValidity

public Java.util.Date getStartValidity()

returns the date after which the object is valid

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObjectHeader#getValidity getValidity

public Java.util.Date getExpireTime()

returns the date after which the object is not valid anymore

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObjectHeader#getValidity getValidity

public Java.util.Date getTriggerTime()

returns the date for presenting the object

Throws

DABNotAvailableException - when not available

See Also

dab.data.MOTObjectHeader#getValidity getValidity

public Java.lang.String getMimeType()

Returns the MIME type of the object

public int getCompressionType()

Returns the compression type of the object

Throws

DABNotAvailableException - when the content description is not available

public class **dab.data.LocationInfo**

LocationInfo represents location data this is returned by the GetLocationInfo command.

Note, if the quality is below zero, than all other attributes are invalid.

The used coordinates have the same reference system as GPS.

Version

1.01

Constructors **public LocationInfo()**

Methods **public int getLongitude()**

Returns the longitude in 100 000ths of a degree (from +180 degrees for easterly longitudes to -180 degrees for westerly longitudes).

public int getLatitude()

Returns the latitude in 100 000ths of a degree (from +90 degrees for northerly latitudes to -90 degrees for southerly latitudes).

public int getAltitude()

Returns the altitude in meters above ground.

public int getVelocity()

Returns the velocity in 100 000ths of a meter per second.

public int getDirection()

Returns the direction in 100 000ths of a degree (range: [0,360[in degrees; 0 degrees points to north).

public int getQuality()

Returns the overall quality of the data. The range is from +100 (best) to -100 (worst). Negative values indicate invalid data.

public class **dab.data.ObjectId**

The ObjectId is an identifier for objects carried in a data service channel. It is used to request objects and for identification of delivered objects to the application.

Version

1.04

Constructors **public ObjectId()**

Constructs an ObjectId object

public ObjectId(ObjectId objectId)

Constructs a copy of the given ObjectId.

See Also

ObjectId#getId getId

public ObjectId(String stringId)

Constructs an ObjectId object from the given string.

See Also

ObjectId#getId getId

Methods **public int compareTo(Object objectId)**

This method compares the object with the given object. The behavior is the same as it is specified in the compareTo method of the Java.lang.Comparable interface.

public int compareTo(ObjectId objectId)

This method compares the object with the given object. The behaviour is the same as it is specified in the compareTo method of the Java.lang.Comparable interface.

public Java.lang.String getId()

Returns an external representation of the identifier in a textual format. The returned value can be used to construct an object id.

public class **dab.data.DLSObject** extends dab.data.DABObject

The DLSObject represents data of the Dynamic Label Service.

Version

1.02

Constructors **public DLSObject()**

Methods **public byte[] getRawDynamicLabelSegment()**

Returns an array of bytes containing the DLS as it is.

Remark : The CRC check for the DLS must be successfully passed

public Java.lang.String getDynamicLabelSegment()

Returns the DLS converted to Unicode and without control characters

Remark : Not all codetables for Unicode may be implemented on the receiver

Throws

DABNotAvailableException - when the information is not available

public int getCharSet()

Returns the charSet of the DLS (see DABConstants.charset*)

Throws

DABNotAvailableException - when the information is not available

public int getCharacterFlagField()

Returns the CharacterFlagField for the DLS

Throws

DABNotAvailableException - when the information is not available

public int getEndofHeadlinePosition()

Returns the position of the last character belonging to the Headline inside the DLS

Throws

DABNotAvailableException - when the information is not available

public int[] getPreferedLineBreakPositions()

Returns the positions of the last character before a line break suggested by the broadcaster

Throws

DABNotAvailableException - when the information is not available

public int[] getPreferedWordBreakPositions()

Returns the positions of the last character before a word break suggested by the broadcaster

Throws

DABNotAvailableException - when the information is not available

public int getSegmentNumber()

Returns the SegmentNumber

Throws

DABNotAvailableException - when the information is not available

public boolean isToggle()

Returns the Toggle Flag

Throws

DABNotAvailableException - when the information is not available

public boolean isCommand()

Returns the Command Flag

Throws

DABNotAvailableException - when the information is not available

Annex B (informative): Bibliography

Implementing Protection Domains in the Java Development Kit 1.2 *(Li Gong and Roland Schemers)* - Proceeding of Internet Society …

Java Security Architecture (http://Java.sun.com/…/security-spec.html)

Security reference Model for JDK 1.0.2 by M. Erdos, B. Hartman, M. Mueller (13 November 1996) - Sun specification

Java TV API Specification (http://java.sun.com/products/javatv/)

DVB Java specification (http://www.dvb.org/dvb_technology/framesets/standspec-fr.html)

Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface (GSM 11.14 version 7.2.0 Release 1998)

Connected, Limited Device Configuration (March 8, 2000)
(http://Java.sun.com/aboutJava/communityprocess /jsr/jsr_030_j2melc.html)

PDA Profile for J2ME (http://Java.sun.com/aboutJava/communityprocess/jsr/jsr_075_pda.html)

J2ME Connected Device Configuration (http://Java.sun.com/aboutJava/communityprocess/ jsr/jsr_036_j2mecd.html)

Requirements for Runtime Package (TF VM 44 - GNM - 17.2.2000)

JAR Archive documentation (http://www.Javasoft.com/j2se/1.3/docs/guide/jar/index.html)

Manifest (http://www.Javasoft.com/j2se/1.3/docs/guide/jar/jar.html#The META-INF directory)

The HAVi Specification. (http://www.havi.org)

PersonalJava 1.1 http://Java.sun.com/products/personalJava/

PersonalJava datasheet (http://Java.sun.com/products/personalJava/pJava_ds.html)

DAB Java: The Runtime Package, WorldDAB TF-VM (Antonio Barletta)

DAB Java User Application Signalling, WorldDAB TF-VM

PersonalJava and J2ME http://Java.sun.com/products/personalJava/faq.html#A11

Design Pattern, Element of Reusable Object-oriented Software by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (Addison-Wesley - ISBN0201633612)

History

Document history		
V1.1.1	March 2002	Publication