



Curso Multimedia Home Platform 1.1.2

Application Listing & Launching API

Sepamos qué aplicaciones existen

Ejecutemos aplicaciones

Curso Multimedia Home Platform 1.1.2

Copyright 2008 © Enrique Pérez Gil

Licensed under the ***Creative Commons Attribution-Non-Commercial-No Derivative Works 3.0 Unported License***. You may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode>

This is a human-readable summary of the License applied:

(<http://creativecommons.org/licenses/by-nc-nd/3.0/>)

You are free to Share, to copy, distribute and transmit the work **Under the following conditions:**

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Noncommercial.** You may not use this work for commercial purposes.
- **No Derivative Works.** You may not alter, transform, or build upon this work.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.

Introducción

- Por un lado se trata de tener acceso a las aplicaciones que se encuentran disponibles, de ver sus propiedades, de saber en qué estado se encuentran y cuando cambian de un estado a otro...en definitiva de **disponer de información**, y por otro de ser capaces de **Lanzar Aplicaciones** y de controlar el estado de estas cuando lo hacemos
- El API se encuentra en **org.dvb.application**

Información

- ¿ Cómo sabemos qué aplicaciones existen ? Bien, disponemos de una clase bastante representativa: **org.dvb.application.AppsDatabase**
- **AppsDatabase** nos ofrecerá información (con bloqueo) de lo que en el momento de solicitarla existe en el sistema. Esto quiere decir que puede que esta información se vea actualizada en un futuro.... ¿ y como sabremos que esto ha ocurrido ? Veamos:
 - AppsDatabase ofrece información que proviene de la **AIT Signalling**
 - Cuando cambiamos de canal la BBDD se actualiza para mostrar las nuevas apps disponibles y eliminar las NO disponibles. Para aquellas que se mantienen el sistema actualizará sus **properties**, **no siendo necesario re-pedir estas a la BBDD**. Todo esto sucederá antes de lanzarse el evento de notificación de newDatabase (luego lo vemos).
 - La información de una App que sigue signalled en el nuevo canal también se actualizará con la info de la nueva AIT.
 - Cuando una APP ha sido autorizada externamente sus características serán las que se tuvo en el servicio en el que se arrancó. Este tipo de apps sólo aparecen en la BBDD cuando se arrancan!

¿ Cómo me ofrece AppsDatabase qué Apps existen y sus propiedades ?

- AppsDatabase ofrece los siguientes métodos para obtener los objetos de tipo **org.dvb.application.AppAttributes** que contienen la información de cada App devuelta.
- AppsDatabase API
 - AppsDatabase static **getAppsDatabase()**
 - AppsDatabase es un Singleton (casi como siempre)
 - public AppAttributes **getAppAttributes** (AppID key)
 - le pasamos el AppID de la APP que nos interesa y nos devuelve su AppAttributes
 - **org.dvb.application.AppID** se contruye con el orgId y el appId (en enteros):
AppID apid = new AppID(23,34);
 - AppID es un dato que ofrece **AppAttributes**

¿ Cómo me ofrece AppsDatabase qué Apps existen y sus propiedades ? (y 2)

- AppsDatabase API
 - public Enumeration **getAppAttributes**(AppsDatabaseFilter filter)
 - En esta ocasión se nos devuelve un conjunto de **AppAttributes**: los correspondientes a aquellas aplicaciones que cumplen la condición del filtro pasado.
¿ qué es un AppsDatabaseFilter ?
 - **org.dvb.application.AppsDatabaseFilter** es en realidad una clase abstracta de la que heredan dos clases incluidas en el paquete, las cuales simplemente mediante su instanciación proporcionan dos filtros muy simples:
 - **org.dvb.application.CurrentServiceFilter**: le pedimos los AppAttributes de las APPs signalled en el Service seleccionado actualmente.
 - **org.dvb.application.RunningApplicationsFilter**: le pedimos los AppAttributes de las APPs que están en ejecución.
 - NOTA: El filtro es muy sencillo (boolean accept(AppID) , de manera que si queréis haceros uno heredando de cualquiera de los 2 filtros podéis hacerlo sin problema.

Ejemplo: AppsDatabase.getAppDatabase().getAppAttributes(new RunningApplicationsFilter());

¿ Cómo me ofrece AppsDatabase qué Apps existen y sus propiedades ? (y 3)

- AppsDatabase API
 - public Enumeration **getAppIDs**(AppsDatabaseFilter filter)
 - En esta ocasión se nos devuelve un conjunto de **AppID**: los correspondientes a aquellas aplicaciones que cumplen la condición del filtro pasado.
 - **Gestión de Eventos de AppsDatabase**: Mediante la gestión de eventos siguiente AppsDatabase nos informará de cuando la BBDD es nueva (cambio de canal), de cuando ha aparecido una nueva app, de cuando ha desaparecido otra, y de cuando una existente ha sufrido alguna modificación:

```
public void addListener(AppsDatabaseEventListener listener)
public void removeListener(AppsDatabaseEventListener listener)
public interface AppsDatabaseEventListener extends EventListener{
    public void newDatabase(AppsDatabaseEvent evt) ;
    public void entryAdded(AppsDatabaseEvent evt) ;
    public void entryRemoved(AppsDatabaseEvent evt) ;
    public void entryChanged(AppsDatabaseEvent evt) ;
}
```

¿ Cómo me ofrece AppsDatabase qué Apps existen y sus propiedades ? (y 4)

- AppsDatabase API
 - **Gestión de Eventos de AppsDatabase:** La clase AppsDatabaseEvent nos ofrece mediante [AppID getAppId()] el id de la APP que ha cambiado

```
public AppID getAppID()
public int getEventId() (NEW_DATABASE, APP_CHANGED, APP_ADDED, APP_DELETED)
```
- Antes de proseguir con la gestión de **Launching** veamos qué información nos ofrece **AppAttributes**
- AppAttributes API
 - public boolean **isVisible()**;
 - Indica si la aplicación es visible a los usuarios. Sólo las que sí deberán mostrarse por un lanzador genérico.
 - public org.davic.net.Locator **getServiceLocator()**
 - Nos devuelve un Locator del Service en el que la app es Signalled.

org.dvb.application.AppAttributes

- AppAttributes API
 - public int **getPriority()**
 - Prioridad de la APP
 - public **AppIcon** **getAppIcon ()**
 - Icono de la APP o null si no tiene. (ved el API y **APP Signalling**)
 - public AppID **getIdentifier()**
 - El AppID de la APP
 - public boolean **getIsServiceBound()**
 - Nos dice si está asociada al Service.
 - public String **getName()**
 - Nombre de la app

org.dvb.application.AppAttributes

- AppAttributes API

- public boolean **isStartable()**

Una APP se puede arrancar si **NO** se da ninguna de las siguientes condiciones:

- La app es transmitida sobre una Remote Connection y o no tiene un application storage descriptor, o no está cached, o no está signalled como “Launchable completely from cache”
- La app ha sido signalled como not_launchable_from_broadcast y no está almacenada.
- Quien llama al método NO tiene permisos para arrancarla.
- Si la app ha sido signalled de otra forma distinta a AUTOSTART o PRESENT (kill o destroy)

- public String[] **getProfiles()**

- Nos devuelve los mínimos Profiles de la APP para que se pueda ejecutar (enhanced, interactive, internet...)

- public int[] **getVersions(String profile)**

- Un profile tiene a su vez versiones. Devuelve version.major, version.minor y version.micro

org.dvb.application.AppAttributes

- AppAttributes API
 - public int **getType()**
 - org.dvb.application.AppAttributes.DVB_J_application o DVB_HTML_application
 - public Object **getProperty** (String index).
 - Este método proporciona información respecto a las siguientes propiedades de la APP

Propiedad	Tipo.Valor
dvb.j.location.base	String. Se corresponde con el valor base_directory_bytes del Descriptor 10.9.2 DVB-J application location descriptor. Punto de anclaje respecto al raíz del sistema de archivos broadcasted del que cuelga la APP.
dvb.j.location.cpath.extension	String[]. Se corresponde con el valor classpath_extension_byte del Descriptor 10.9.2 DVB-J application location descriptor. Classpath extra donde buscar clases además del base.
dvb.transport.oc.component.tag	Int. Cuando el protocolo de envío de la APP es un Object Carousel, el descriptor de transporte tiene un código 0x0001. Cuando esto es así el contenido del Descriptor correspondiente a este código lleva un campo component_tag (10.8.1.1) que en definitiva nos va a decir en qué Stream se encuentra nuestro sistema de ficheros.
dvb.ait.descriptors	En 11.7.8 Plug-in APIS se dice: " The semantics of the method org.dvb.application.AppAttributes.getProperty() are extended such that when it is called with the string "dvb.ait.descriptors", an array of bytes containing the application_descriptors_loop shall be returned."
dvb.transport.oc.locator	org.davic.net.Locator que identifica el Object Carousel. OJO: válido para hacer un ServiceDomain.attach(Locator)

Ejercicios Bloque APPLIS-1

Launching & Controlling.

- AppsDatabase ofrece el método siguiente:

```
public AppProxy getAppProxy(AppID key)
```

- ¿ qué es un **org.dvb.application.AppProxy** ? : Es el objeto mediante el cual podremos actuar sobre la Aplicación elegida bien sea para arrancarla, pararla, etc.
- En realidad cuando solicitemos el objeto y la aplicación sea del tipo DVB-J (Xlet) el tipo devuelto será **org.dvb.application.DVBJProxy** y si la aplicación fuera de tipo DVB-HTML este sería: **org.dvb.application.DVBHTMLProxy**, ambos heredando de AppProxy.

Estudiemos el API de **DVBJProxy** en detalle.

Launching & Controlling

- DVBJProxy API
 - public int **getState** ()
 - Estado en que se encuentra.
 - STARTED
 - DESTROYED
 - NOT_LOADED
 - PAUSED
 - LOADED (aportado por DVBJProxy)
 - INVALID *Ya no aparece en el signalling.*
 - public void **start** (String args[]);
 - Se solicita el arranque de la app pasándole como parámetros los indicados sólo si esta no había sido “inited”, si ya lo había sido se descartan “silently”.
 - Sólo si AUTOSTART o PRESENT y además: si la DVB-J está en NOT Loaded o Paused.
 - Si estaba en Paused pero NUNCA se había arrancado se arrancará.
 - Asíncrono.

Launching & Controlling

- DVBJProxy API

- public void **start** ();

- Se solicita el arranque de la app. Asíncrono. (aplica lo anterior).

- public void **stop**(boolean forced);

- Solicita que la app pase a destroyed. Asíncrono

- public void **pause**();

- Solicita que la app pase a Paused. Ha de estar Active. Se llamará a pauseXlet.

- public void **resume**()

- Solicita que reanude la ejecución de la app. Ha de estar Paused. Se llamará a startXlet. Asíncrono.

- public void **load** ();

- Solicita la carga de al menos la clase Xlet. Asíncrono.

Launching & Controlling

- DVBJProxy API

- public void **init** () throws java.lang.SecurityException;

Solicita la llamada a initXlet. Asíncrono. Ha de estar NOT_LOADED o LOADED.

- public void **init**(java.lang.String[] args);

Solicita la llamada a initXlet pasándole los parámetros especificados. Asíncrono. Ha de estar NOT_LOADED o LOADED.

- **Eventos**: Nos permite saber cuando la app cambia de estado. En todos los métodos asíncronos después se generará un evento y el método **hasFailed** de AppStateChangeEvent nos indicará si ha ido bien o no.

```
public void addAppStateChangeListener (AppStateChangeListener listener)
public void removeAppStateChangeListener(AppStateChangeListener listener)
public interface AppStateChangeListener extends EventListener{
    public void stateChange(AppStateChangeEvent evt);
}
```

Launching & Controlling

- DVBJProxy API

```
public class AppStateChangeEvent{
```

```
    public AppID getAppID ()
```

```
    public int getFromState ()
```

Del estado que partíamos si hasFailed false y en el nos hemos quedado si hasFailed true.

```
    public int getToState ():
```

Al que hemos llegado si hasFailed false y al que íbamos si hasFailed = true.

```
    public boolean hasFailed ():
```

si la transición falló.

```
}
```

Ejercicios Bloque APPLIS-2

ISO/IEC 13818-1	Part 1. Elementary Streams transport definition
ISO/IEC 13818-6	Part 6. Extensions for DSM-CC. Digital Storage Media Command and Control
ETSI EN 300 468	Digital Video Broadcasting (DVB);Specification for Service Information (SI) in DVB systems
ETSI EN 301 192	DVB specification for data broadcasting
ETSI TR 101 202	Implementation Guidelines for Data broadcasting
ETSI TR 101 162	Digital broadcasting systems for television, sound and data services; Allocation of Service Information (SI) codes for Digital Video Broadcasting (DVB) systems
ETSI TR 102 154	Implementation guidelines for the use of MPEG-2 Systems, Video and Audio in Contribution and Primary Dist
ETSI TR 101 211	Guidelines on implementation and usage of Service Information (SI)
ETSI TR 101 200	Digital Video Broadcasting (DVB); A guideline for the use of DVB specifications and standards
DAVIC	Digital Audio Visual Council. davic 1.4.1
HAVI	Specification of the Home Audio/Video Interoperability (HAVi) Architecture
Interactivetvweb	http://www.interactivetvweb.org/
Wikipedia DSMCC	http://en.wikipedia.org/wiki/DSM-CC
MHP 1.1.2	Multimedia Home Platform, A068r1 & tam668r23_11xdraft_20061115
MHP 1.1.3	Multimedia Home Platform, A068r3
CDC 1.1	Connected Device Configuration (CDC) 1.1 (JSR=218).
PBP 1.1	Personal Basis Profile 1.1 (JSR 217)
MHP.org	www.mhp.org
INTRO MHP 1.1.3	tam1032r1-mhp-iptv-presentation