



Curso Multimedia Home Platform 1.1.2

MHP Stored Services & Cached Apps

Stored Services,

Apps Cache &

APIs

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

- Una de las mayores ventajas que trajo la versión 1.1.x (**Profile 2**, las 1.0.x eran profile 1) fue la **posibilidad de almacenar aplicaciones en el STB**.
- Vamos a ver dos tipos de aplicaciones desde el punto de vista de almacenaje: aquellas que están relacionadas con los **Broadcasted Services**, y que pueden estar cacheadas total o parcialmente y otras que están cacheadas completamente y que pertenecen a los **Stored Services**.
- Veamos el detalle....pero antes ¿ cómo sé si mi STB soporta Stored Apps ? ¿ nos acordamos ?

```
String supportsStored = System.getProperty("mhp.stored.services");  
return supportsStored!=null && supportsStored.equals("SUPPORTED")
```

En nuestro caso OK (Ejercicio ciclo6): [2#1:1] mhp.stored.services=[System]**SUPPORTED**



Stored Services

- ¿ **Qué es un Stored Service ?** Una agrupación de una o más **Stored Apps**
- ¿ **Cómo se crea un Stored Service ?** Fácil: desde un Xlet usando el API **org.dvb.application.storage** que veremos en detalle más adelante.
- ¿ **Cómo es el ciclo de vida de un Storage Service ?** El ciclo de vida de un Stored Service sigue las mismas pautas que el ciclo de vida de uno Broadcasted, y lo mismo ocurre con las aplicaciones que contiene, por ejemplo, si la Stored Service App está en ejecución y el nuevo Service la tiene signalled pues seguirá, y de igual modo cualquier aplicación de un Broadcast Service que esté en ejecución será parada si no está signalled en la AIT del Stored Service.
- **Además:** Si una Stored Service App es Signalled por un Broadcast Service se podrá lanzar y funcionará como una aplicación normal, con la ventaja de que ha sido cargada desde Caché.
- ¿ **Cómo se añade una aplicación a un Stored Service ?** Fácil: desde un Xlet usando el API **org.dvb.application.storage** que veremos en detalle más adelante.

Stored Services

- Un Stored Service tiene una AIT al igual que un Service Broadcasted y se inspecciona de igual manera.
- Sin entrar en el detalle del API aún: **una aplicación se añade al Stored Service desde el Broadcast**, y lo que se almacena en el STB es lo siguiente:
 - El valor de la **storage_property** y la **versión** provenientes del **application_storage_descriptor** de la app (lo veremos a continuación)
 - Información suficiente para reconstruir el **application_descriptor_loop**.
 - El tipo y la identificación de la aplicación.
 - Al menos todos los ficheros marcados como **Critical** en el **ADF, Application Description File** que veremos después. Habrán de ser los que necesitamos para ejecutar la app de forma autónoma.
 - Información necesaria para poder darle a la aplicación los permisos que requiera cuando se ejecute.
- Cualquier información que necesite la App para funcionar y que no esté será su responsabilidad el obtenerla.

Cached Apps

- ¿ **Qué es una Cached APP ?** Aquella que es Signalled por el Broadcast pero se encuentra almacenada total o parcialmente en el STB, con lo que su carga se produce más rápidamente. Es la única diferencia con respecto a una app normal.
- ¿ **Cómo se cachea una App ?** Fácil: desde un Xlet usando el API **org.dvb.application.storage** que veremos en detalle más adelante.
- ¿ **Cómo sabe el STB que la App signalled en el Broadcast igual puede cargarse desde el deco ?** Porque se le ha incluido un descriptor especial en la AIT: **application_storage_descriptor**.
- ¿ **Qué cacheo de una APP y como lo indico ?** Se indica de forma similar a la de una Stored APP: mediante el **Application Descriptor File, ADF**.

Cached Apps. Proactive Caching

- Si el STB lo soporta, mediante la publicación del **ADF** y el **application_storage_descriptor**, este puede **proactivamente** cachear los ficheros marcados como Critical en el ADF.
- La gestión posterior de esta caché es suya igualmente.

Cached APPs

- Es importante indicar que el STB sólo procederá a usar ficheros cacheados cuando coincidan el **organisation ID**, **application ID**, y **version number** de la aplicación signalled con los que contienen los ficheros almacenados, aplicando las excepciones en cuanto a versiones que veremos a continuación.

application_storage_descriptor

- Veamos el descriptor **application_storage_descriptor** de la AIT que ayuda a la gestión de las Apps con respecto al Storage/Cache (descriptor_tag = 0x10)

Table 110: Syntax of application storage descriptor

	No.of Bits	Identifier
application_storage_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
storage_property	8	uimsbf
not_launchable_from_broadcast	1	bslbf
launchable_completely_from_cache	1	bslbf
is_launchable_with_older_version	1	bslbf
reserved	5	bslbf
reserved	1	bslbf
version	31	uimsbf
priority	8	uimsbf
}		

A0068r1

application_storage_descriptor

- Para el caso de Cached APPs su existencia nos viene a indicar que la aplicación podría cargarse desde el STB.
- Tanto para **Cached** como para **Stored Apps**, si la intención es almacenar una APP entonces habrá de existir un **ADF** donde tengamos el detalle de qué, y además es el **application_storage_descriptor** el que nos dirá la Versión de la APP que vamos a almacenar.

Veamos los campos del descriptor.

- **descriptor_tag**: 0x10
- **launchable_completely_from_cache**: 0, 1
 - **1**: La app **se puede (que no quiere decir que deba)** ejecutar completamente desde caché sin necesidad de conectarse al Carousel, **siempre que todos los ficheros críticos estén cacheados.**
 - **0**: Hay que conectarse al Carousel pues es seguro que va a necesitar ficheros del mismo.
 - **Este Flag sólo se tiene en cuenta para aplicaciones relacionadas con el Broadcasted Service**, NO se tendrá en cuenta para aquellas que forman parte de un **Stored Service**, pues en este caso siempre se cargan completamente de caché sin necesidad de conectarse a un Carousel.

application_storage_descriptor

- **storage_property**: ver tabla abajo
 - 0: **broadcast related**. El ciclo de vida de la aplicación está asociado al del Service, puede tener ficheros cacheados pero no pueden ser ejecutadas como “Stand alone”.
 - 1: **stand alone**. En esta versión de MHP se refiere a aplicaciones de **Stored Services** y a **Interaction Channels Downloaded APPS**: aquellas cuyo signalling se hace a través del canal de retorno obteniendo la AIT con una URL HTTP/HTTPS.

Table 111: Semantics of storage property values

storage_property	property
0	broadcast related
1	stand alone
2 to 255	reserved

application_storage_descriptor

- **is_launchable_with_older_version: 0, 1**
 - **1:** El STB lanzará la app cacheada con independencia de la versión. En este caso es la propia app la encargada de mantener la coherencia entre las versiones.
 - **0:** El STB no lanzará la app cacheada si la versión no es la misma que la indicada
- **not_launchable_from_broadcast: 0, 1**
 - **1:** indica que la app debe de cargarse desde cache. Sólo se podrá si se dispone en caché de todos los ficheros definidos en el “Application Descriptor File” como critical.
 - **0:** indica que puede cargarse de Broadcast y que puede tener elementos cacheados.
- **version:** nos da la versión de la app. Comienza en 0 y se incrementa de 1 en 1. Cambiará cuando o bien los ficheros indicados en el Application Descriptor File varíen o bien cambie algo del contenido del Application Descriptor File.
- **Importante:** en un STB se pueden almacenar diferentes versiones de una APP. Lo que identifica unitariamente a una Stored/Cached APP es: **OrgID+AppID+Version**. Por ejemplo, dos cadenas de TV que usen la misma APP, puede que no utilicen la misma versión de la misma.
 - **priority:** sólo usado por STB que pueden aplicar Proactive Caching para decidir cual cachean

Application Descriptor File, ADF

- Proporciona la lista de ficheros que han de instalarse así como otra información.
- Para aquellas aplicaciones que pueden ser almacenadas habrá de residir en el directorio Base de las mismas en el mismo Carousel que ellas.
- Se deberá denominar de la forma:
 - 'dvb.storage.00000000.aaaa' donde 00000000 y aaaa son el **organization_id** y el **app_id** respectivamente, en hexadecimal en minúsculas y con leading 0 hasta completar el tamaño del campo. Por ejemplo:

dvb.storage.0000000a.00b1

Ver ejemplo en : /logs/dvb.storage.00000014.0029

Application Descriptor File, ADF

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//DVB//DTD Application Description File 1.0//EN"
"http://www.dvb.org/mhp/dtd/applicationdescriptionfile-1-0.dtd">
<!ENTITY % object "(dir|file)">
<!ELEMENT applicationdescription (%object;)+>
    <!ATTLIST applicationdescription
        version NMTOKEN #REQUIRED>
<!ELEMENT dir (%object;)*>
    <!ATTLIST dir
        name CDATA #REQUIRED
        priority NMTOKEN #IMPLIED >
<!ELEMENT file EMPTY>
    <!ATTLIST file
        name CDATA #REQUIRED
        priority NMTOKEN #IMPLIED
        size NMTOKEN #REQUIRED >
```

Application Descriptor File, ADF

- Descripción de campos del DTD
 - **Version:** version number sin leading 0. **Debe coincidir con el indicado en el application_stored_descriptor, si no, el ADF será inválido.**
 - **Name:** Nombre del Fichero o Directorio. NO INCLUYE PATH info: ni “/” ni “.” Como véis el DTD define una estructura de directorios.
 - Importante: **SOLO para el caso de Files** se pueden utilizar * al final del nombre como comodín de referencia de ficheros dentro del directorio.
 - **Priority:** entre 0 y 255. 0 = Critical. Valor por defecto =0. **OJO:** un elemento hereda la prioridad del padre!!!
 - **Size:** Tamaño del fichero o ficheros cuando el nombre contiene *.

Ejemplo ADF

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//DVB//DTD Application Description File 1.0//EN"
  "http://www.dvb.org/mhp/dtd/applicationdescriptionfile-1-0.dtd">
<applicationdescription version="1">
  <dir name="code4tv" priority="0">
    <dir name="mhp112" priority="0">
      <dir name="exercise_cach2" priority="0">
        <file name="Exercise_cach2" priority="0" size="9399"/>
      </dir>
    </dir>
  </dir>
</applicationdescription>
```

Seguridad

- Las unsigned apps **NO** tienen permiso para almacenar apps.
- Las signed apps pueden usar el API siempre que se solicite en el PRF de la forma:

```
<!ELEMENT applicationstorage (applicationstorageorg)*>
```

```
<!ATTLIST applicationstorage  
manageservice: (true|false) "false"  
createservice: (true|false) "false"  
deleteservice: (true|false) "false"  
managecache: (true|false) "false" >
```

```
<!ELEMENT applicationstorageorg EMPTY>
```

```
<!ATTLIST applicationstorageorg  
orgid CDATA #REQUIRED  
storeservice (true|false) "false"  
removeservice (true|false) "false"  
storecache (true|false) "false"  
removecache (true|false) "false" >
```

Seguridad

Veamos el significado en detalle.

- **manageservice:** si true solicita permiso para manejar un Stored Service con el organization ID de la aplicación actual.
- **createservice:** si true solicita permiso para crear un Stored Service con el organization ID de la aplicación actual.
- **deleteservice:** si true solicita permiso para eliminar un Stored Service con el organization ID de la aplicación actual.
- **managecache:** si true solicita permiso para manejar la cache de una app con el organization ID de la aplicación actual.

- **orgid:** es el organization_id respecto al cual se refieren los tokens de este Permission Request. En hexadecimal sin leading 0. Puede valer "*" para aplicarlo a TODOS los organization_ids.
- **storeservice:** si true solicita permiso para almacenar aplicaciones de la organization_id en un stored service.
- **removeservice:** si true solicita permiso para eliminar aplicaciones de la organization_id de un stored service
- **storecache:** si true solicita permiso para almacenar aplicaciones de la organization_id en cache.
- **removecache:** si true solicita permiso para eliminar aplicaciones de la organization_id de cache.

¿ De cuanto espacio dispongo ?

- En las MHP 1.1.2 Specs dice:

Memory for stored services as defined in clause 9.7.2

Either 0 or greater than or equal to **65 536 bytes**. This memory shall be non-volatile. If greater than zero, there shall be a means to empty this memory, at least for the purposes of running conformance tests. (note 3).

EL API. Cache org.dvb.application.storage.*

- El procedimiento que sigue se usa para almacenar en Caché Broadcast Signalled Apps.
- El procedimiento que sigue se usa para **ALMACENAR** Aplicaciones en Caché en un momento **DADO**, **NO** es que te ofrezca un API de lo que hay en la Caché global.
- Para almacenar APPs en Caché disponemos de la clase **ApplicationCache** cuyo API resulta bastante sencillo, no obstante apuntemos un par de cosas:
 - El org.dvb.application.AppID (orgid+appid) de una App es único en el contexto de un Service: sólo se puede transmitir una versión de una APP.
 - Cuando se hace signalling de una APP, el dato de la versión es el que viaja en el **application_storage_descriptor**, que recordemos es el descriptor que nos indica la posibilidad de cargar la App de caché y qué versión, y también nos da la posibilidad de almacenarla.
 - Recordad que las Cached APPs sólo se pueden lanzar si son Signalled en el Broadcast!
- El ApplicationCache **es un objeto de tipo Singleton** el cual **SOLO** puede almacenar una versión de cada Aplicación, es decir, la clave única de sus repositorio es el APPID.

EL API. Cache org.dvb.application.storage.*

- En un STB se pueden almacenar diferentes versiones de una APP, de hecho, pueden haber diferentes ApplicationCache funcionando, pero todos comparten un repositorio común, el cual será capaz de manejar todas las APPs y sus versiones, ya se hayan almacenado vía distintos ApplicationCache o sean parte de StoredServices.
- Todas aquellas aplicaciones que tengan el mismo OrganizationID accederán al mismo ApplicationCache.
- Cuando se almacena en ApplicationCache una Aplicación cuya versión no coincide con la de una versión previa existente en ese ApplicationCache entonces la que haya en ApplicationCache será reemplazada por la nueva.
- El repositorio que hay por debajo es “listo” compartiendo las mismas clases y no duplicando código...y es el mismo repositorio usado para los StoredServices.
- **Es responsabilidad del Broadcaster el gestionar el versionado y cacheado de aplicaciones con coherencia.**

EL API. Cache org.dvb.application.storage.*

- ApplicationCache API:
 - public static org.dvb.application.storage.ApplicationCache **getDefaultCache()**
 - Singleton para acceder a la ApplicationCache
 - **OJO: el objeto devuelto será el mismo para todas aquellas aplicaciones que tengan el mismo OrganizationID**
 - public org.dvb.application.AppID[] **getStoredAppIDs()**
 - Apps en este ApplicationCache.
 - public int **getVersionNumber**(org.dvb.application.AppID appld);
 - Version de la app cacheada en este ApplicationCache indicada. -1 si no está cached.
 - public void **remove**(org.dvb.application.AppID appld);
 - Elimina la app de este ApplicationCache, **pero no de otros en los que pueda estar, ni tampoco si está en un StoredService**. No tiene que preguntarse al usuario.

EL API. Cache org.dvb.application.storage.*

- ApplicationCache API:
 - public void **store**(org.dvb.application.AppProxy app, boolean canPrompt)
 - Almacenamos la APP. Cuando esto se produce el STB puede querer preguntarle al usuario. Esto ocurre si y solo si todas las condiciones siguientes se dan:
 - canPrompt = true
 - se dispone de los permisos
 - la app es válida
 - el ADF es válido
 - no hay suficientes recursos para almacenarla y no puede liberar espacio "silenciosamente" y el deco entiende que podría haber espacio si le dan permiso para liberar recursos.
 - Como ya hemos dicho: Cuando se almacena en ApplicationCache una Aplicación cuya versión no coincide con la de una versión previa de la misma en ese ApplicationCache entonces la que exista en ApplicationCache será reemplazada por la nueva.

EL API. Stored Services org.dvb.application.storage.*

- Para crear Stored Services tenemos nuestro **StoredApplicationServiceFactory** cuyo API ofrece dos métodos: uno para acceder al Singleton y otro para crear los **StoredApplicationService** al cual se le pasan el orgid, serviceid y la descripción que queremos que aparezca cuando se muestre.
 - public static org.dvb.application.storage.**StoredApplicationServiceFactory** getInstance()
 - public org.dvb.application.storage.**StoredApplicationService** createStoredApplicationService(int organisation_id, int service_id, java.lang.String serviceName)

...pero ¿ como accedemos a ellos una vez creados ? ¿ cómo los eliminamos ?

- Los **StoredApplicationService** que creamos heredan de **javax.tv.service.Service** y serán accesibles mediante los APIs que nos den acceso, por ejemplo a través de **javax.tv.service.SIManager**.

EL API. Stored Services org.dvb.application.storage.*

- Para leer los Services
 - recordemos que había que crear un filtro:

```
private void readServices(){
    try{
        SIManager sm = javax.tv.service.SIManager.createInstance();
        ServiceList lista = sm.filterServices(new ALLServicesFilter());
        ServiceIterator si = lista.createServiceIterator();
        javax.tv.service.Service aux = null;
        while(si.hasNext()){
            aux = si.nextService();

            ...

        }
    }catch(Throwable err){
        err.printStackTrace();
    }
}
```

EL API. Stored Services org.dvb.application.storage.*

- Para leer los Services

- Nuestro filtro puede ser :

```
filtro = new javax.tv.service.navigation.ServiceTypeFilter(  
    org.dvb.application.storage.StoredApplicationServiceType.STORED_APPLICATION_SERVICE);
```

- Aunque es sencillo hacer otro a nuestro modo heredando directamente de ServiceFilter y preguntando por ejemplo con **instanceof StoredApplicationService**

```
public boolean accept(Service service) {  
    return service instanceof StoredApplicationService;  
}
```

EL API. Stored Services org.dvb.application.storage.*

- Veamos de qué disponemos una vez que accedemos al **Service** (ved el Java API). Fundamentalmente nos permitirá **añadir/actualizar/eliminar/listar** las aplicaciones que contiene.
- **OJO:** muchos métodos provienen de ApplicationStorageController pues: StoredApplicationService extends ApplicationStorageController.
- StoredApplicationService API
 - public void **store**(org.dvb.application.AppProxy app, boolean autoStart, java.lang.String[] args)
 - Almacenamos una App en el StoredService. Le indicamos si es autoStart o present y los parámetros a los que podrá acceder con la Xlet property "dvb.installer.parameters".
 - Si ya existía la actualiza. **Es síncrono.**
 - Ved API Java en detalle para el comportamiento en Ejecución.
- **Importante: si había una versión anterior distinta (no menor o mayor) a la de la nueva se elimina del Stored Service. Es decir, en un StoredService de cada aplicación sólo puede haber una versión.**
 - public void **store**(org.dvb.application.AppProxy[] apps, boolean[] autoStart, java.lang.String[][] args)
 - Ídem para varias a la vez.

EL API. Stored Services org.dvb.application.storage.* (y 2)

- StoredApplicationService API

- public void **remove**(org.dvb.application.AppID[] applds)
 - Elimina varias a la vez.
- public void **remove**(org.dvb.application.AppID appld)
 - Elimina una app. Ved API Java para ver comportamiento si se está ejecutando...etc
- public int **getVersionNumber**(org.dvb.application.AppID appld);
 - Version de la app indicada (-1 si no pertenece al Stored Service)
- public org.dvb.application.AppID[] **getStoredAppIDs**();
 - Apps que contiene
- public javax.tv.service.SIRequest **retrieveDetails**(javax.tv.service.SIRequestor requestor);
 - Para obtener información adicional del un Service
- public void **removeService**()
 - Elimina el StoredService

EL API. Stored Services org.dvb.application.storage.* (y 3)

- StoredApplicationService API
 - public boolean **hasMultipleInstances()**
 - Siempre devuelve false. El método devolvería true si el service estuviera disponible en otros TS.
 - public javax.tv.service.ServiceType **getServiceType()**
 - Siempre: org.dvb.application.storage.StoredApplicationServiceType.STORED_APPLICATION_SERVICE
 - public javax.tv.service.ServiceInformationType **getServiceInformationType()**
 - Siempre: javax.tv.service.ServiceInformationType.UNKNOWN
 - OJO: no es lo mismo que getServiceType().
 - public String **getName()**
 - Nombre del StoredService
 - public javax.tv.locator.Locator **getLocator()**
 - No tiene porqué ser un org.davic.net.dvb.locator

EL API. Stored Services org.dvb.application.storage.* (y 3)

- StoredApplicationService API
 - public int **getServiceId()**
 - Service ID
 - public int **getOrganisationId()**
 - Organization a la que pertenece el StoredService
- Recordemos que para los StoredServices también es necesario publicar el **application_descriptor_file** si queremos almacenar APPS.

EL API. ExtendedAppAttributes

- **Solo cuando el STB soporta Stored Services y Cached apps** en el API **org.dvb.application.AppsDatabase** (recordemos APPListing & Launching) cuando se solicita información acerca de las características de las apps en lugar de devolver un objeto del tipo :

`org.dvb.application.AppAttributes`

se devuelve otro que hereda de este del tipo:

`org.dvb.application.storage.ExtendedAppAttributes`

y que nos ofrece información relativa a la gestión de almacenaje de las Apps.

EL API. ExtendedAppAttributes

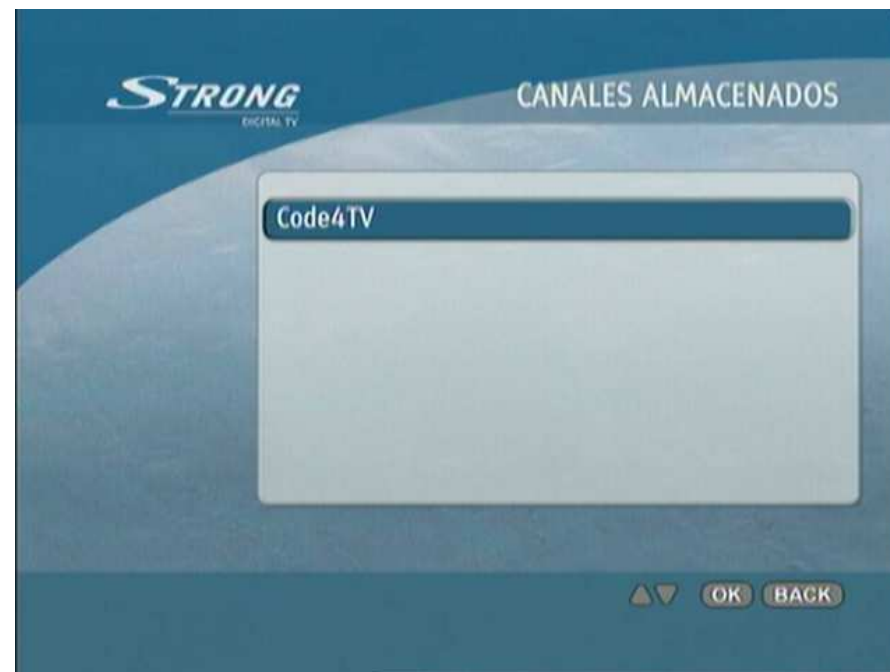
- Veamos el API. ExtendedAppAttributes API
 - public boolean **isStored()**
 - True si stored o cached
 - public boolean **isStorageRequired()**
 - Si la app ha sido signalled como “not launchable from broadcast”. Si la app es Stored devuelve true.
 - public boolean **isStartable()**
 - Si se puede arrancar. **NO** lo es si se da alguna de las condiciones siguientes:
 - El que la ejecuta no tiene permisos.
 - Si no es ni autostart ni present
 - La app se ha transmitido por una remote conn y o no tiene application_storage_descriptor o no está cached o no ha sido signalled como “launchable completely from cache”
 - Se han transmitido con un application_storage_descriptor con not_launchable_from_broadcast=1 y no está ni stored ni cached

EL API. **ExtendedAppAttributes** (y 2)

- public boolean **canCache()**
 - Si la app ha sido signalled para que se pueda cachear (ADF, application_storage_descriptor...)
- public boolean **canAddToStoredService()**
 - True si ha sido signalled para que se pueda almacenar.
- public int **getCurrentVersionNumber()**
 - Devuelve la versión de la app signalled, si está disponible el **application_storage_descriptor**. -1 si no lo está.

Ejercicios Bloque STOSER-1

- Podemos crear el Service, aunque no la App al no disponer de un App Signalling real



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