



# Curso Multimedia Home Platform 1.1.2

## Service Context - JavaTV Selection

Service Contexts

Cambiando de Service usando JavaTV

## 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

- Unidad básica y central: el Service (el canal, vamos).
- Un canal se puede seleccionar bien por el STB Navigator o por una aplicación MHP ofreciendo EPG.
- Un **service** consiste en un conjunto de piezas como señal de audio, video, aplicaciones, información de servicio...etc.
- Un servicio se ejecuta en un **contexto del servicio**. Es decir, en un MHP STB un servicio se presenta en un **Service Context**, que viene a ser un entorno de ejecución y presentación del servicio, y que en gran medida viene a definir cómo se ejecutan las aplicaciones MHP. Podría incluso darse el caso de que se presentasen simultáneamente 2 servicios, por ejemplo: PIP. El componente **ServiceContext** permitirá gestionar este tipo de situaciones.

## Introducción

- Importante matiz: en un STB MHP tiene preponderancia el hecho de que sea un entorno MHP, esto lo condiciona todo: es MHP quien describe cómo se presentan los servicios etc... Véase: *“Every service that gets presented by an MHP platform is presented within a service context”*.
- El service context:
  - Es un entorno en el cual el Service se presenta.
  - Define los límites del Servicio, describiendo las piezas que lo componen, e incluso permite manejarlo como una unidad.
  - Permite a la plataforma y aplicaciones identificar cuales de las piezas que se están presentando pertenecen al servicio

## Introducción

- En una aplicación **DVB-J** (DVB-J = the Java platform defined as part of the MHP specification) este Service Context se representa por la clase  
**javax.tv.service.selection.ServiceContext**  
y se obtiene mediante la factoría:  
**javax.tv.service.selection.ServiceContextFactory**
- Los cambios que una aplicación efectúe sobre una instancia de este objeto será visible a otras instancias del mismo obtenidas por otras aplicaciones en ejecución en el mismo ServiceContext.
- Un entorno MHP hará que una aplicación vea que cada Service tiene un ServiceContext propio.

## Introducción

- Pero ¿ qué nos ofrece un ServiceContext desde el punto de vista de aplicación?
  - **Un ServiceContext Permite Cambiar de Canal/Service!**
    - `javax.tv.service.selection.ServiceContext.select(...)`
  - Un ServiceContext **nos informa** cuando un Service **se ha dejado de presentar y ha llegado otro**
  - Podemos **parar la presentación** de un Service...
  - Podemos **acceder a los “componentes”** que configuran el servicio...
  - Permite que Aplicaciones que no van unidas a un Servicio se puedan gestionar...p.e. StoredChannels, Interaction Channels...Abstract Services...

## Ciclo de Vida del ServiceContext

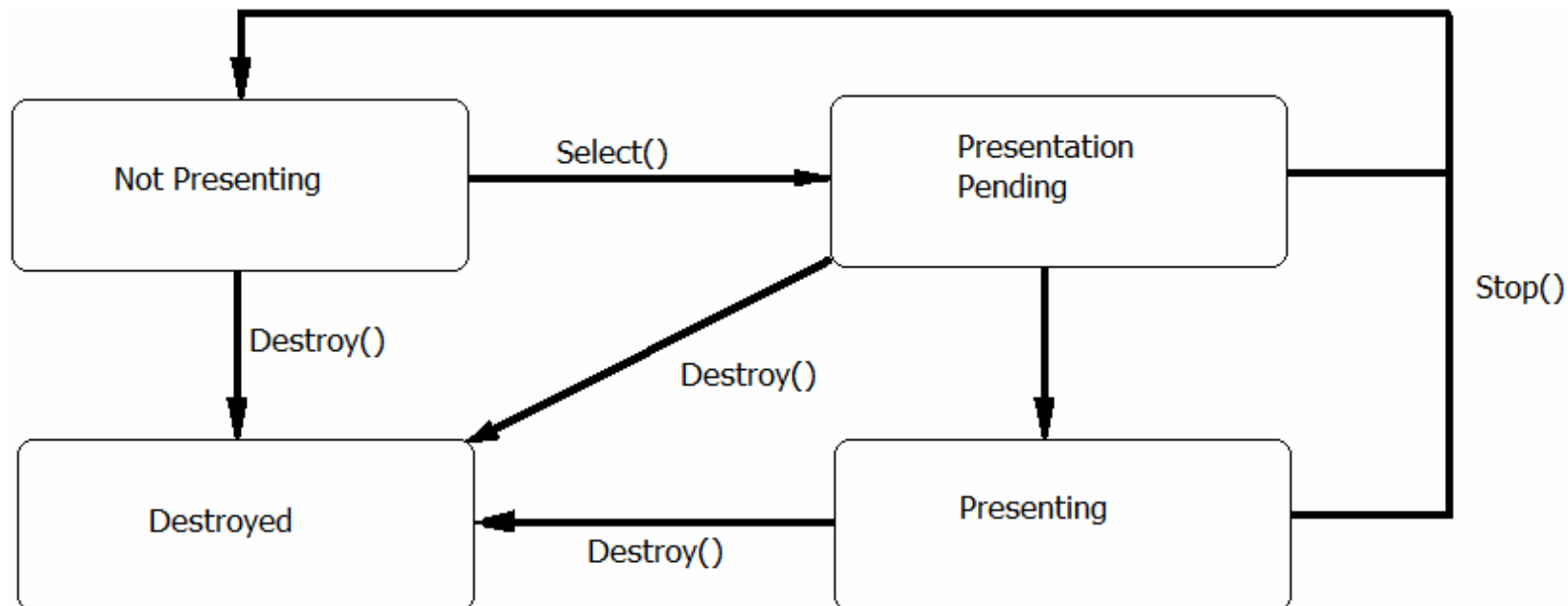
- Un ServiceContext puede encontrarse en 4 estados:
  - **Not Presenting**
    - Estado Inicial de un ServiceContext, o bien al que vamos si se llama a stop() sobre uno en estado Presenting.
  - **Presentation Pending**
    - Cuando se selecciona un Service se entra en Presentation Pending. Una vez que se está presentando se pasa a Presenting
  - **Presenting**
    - Hay un Service seleccionado y presentándose
  - **Destroyed**
    - Cuando se llama a destroy() en el Service

## Ciclo de Vida del ServiceContext

- Todas las llamadas que se realizan que pueden provocar cambios de estado son **asíncronas** y la notificación de los resultados se realiza mediante un Listener: **ServiceContextListener** (lo vemos a continuación)

## Ciclo de Vida del ServiceContext

- Como podemos imaginar cambiar de “Canal” implica numerosas operaciones: arranque y parada de aplicaciones, selección de streams de media y presentación...
- A continuación el diagrama de estados de ServiceContext



## API `javax.tv.service.selection.ServiceContext`

- Resumen del API:
  - public void **select**(`javax.tv.service.Service` selection) throws `SecurityException`;
  - public void **select**(`javax.tv.locator.Locator`[] components)
  - public `javax.tv.service.Service` **getService**();
  
  - public void **stop**() throws `SecurityException`;
  - public void **destroy**() throws `SecurityException`;
  
  - public `javax.tv.service.selection.ServiceContentHandler`[] **getServiceContentHandlers**()
  
  - public void **addListener**(`javax.tv.service.selection.ServiceContextListener` listener);
  - public void **removeListener**(`javax.tv.service.selection.ServiceContextListener` listener);

## API javax.tv.service.selection.ServiceContext

- El API en detalle
  - public void **select**(Service selection) throws SecurityException;
    - Selecciona un Service.
    - El método es asíncrono.
    - Sólo se puede llamar si el ServiceContext NO está en estado **Destroyed**
    - Si va bien la selección entonces se lanzará un evento **NormalContentEvent** o bien **AlternativeContentEvent**.
    - Si en la llamada al método se puede determinar que no va bien la selección se lanza una Exception.
    - Si la selección se detecta que no va bien después se lanzará un **SelectionFailedEvent**. Y si el Service que se estaba presentando no se puede recuperar (problemas del sintonizador...) se lanzará un **PresentationTerminatedEvent** y entrará en estado **not\_presenting**
    - Si la selección ha sido correcta se proporcionarán tantos **ServiceContentHandler** como sean necesarios para presentar los componentes del Service.
    - **Si la aplicación no está signalled en el servicio elegido morirá.**
    - Los componentes que se están presentando en ambos servicios permanecen.

## API javax.tv.service.selection.ServiceContext

- El API
  - public void **select**(Locator[] components)
    - Selección de un servicio mediante la provisión de los componentes del mismo que deseamos que presente. **Sólo se garantiza que funcionen los de tipo Video, Audio y Subtítulos.**
    - Como veremos más adelante en detalle cuando seleccionamos un Service el ServiceContext nos ofrecerá lo que se llaman los ServiceContentHandler y que se encargan de la “presentación” del contenido, pues bien, este elemento nos ofrece un método getLocators() que indica los elementos que gestiona.
- **Importante:** a una aplicación MHP sólo se le permite presentar como contenido de Video y Subtítulos lo que proviene del Broadcast y siempre en el contexto de un mismo ServiceContentHandler, es decir, asociados a un Service!!!
- Mediante el uso de JMF Controls se podrá presentar contenido de Audio desde ficheros o bien desde URLs mediante el componente HSound. Se ve en detalle en el capítulo dedicado a Java Media.
- **En realidad en un ServiceContext no es posible seleccionar mas que Services.**

## API javax.tv.service.selection.ServiceContext

- El API. Qué ocurre cuando se cambia de Service.
  - El Service Context parará cualquier tipo de contenido que no forme parte del Service Destino y comenzará a presentar el nuevo contenido. Aquel que permanezca puede que sufra algún tipo de “alteración” momentánea.
  - El ServiceContext pasará a **PresentationPending**, probablemente pasando por **Not presenting**
  - Sintonizará el nuevo TS si fuera necesario.
  - Usará la información de SI así como las preferencias de usuario para decidir qué Streams presentar.
  - Comenzará a Presentar los Streams en el ServiceContext.
  - El ServiceContext pasará a **Presenting** y generará un **PresentationChangedEvent**
  - Leerá la AIT y procederá como está especificado con respecto a las Apps.
  - Si la selección del Service se hizo con Locators antes de “sintonizar” comprobará que estos pertenecen todos al mismo Service y que pueden presentarse “a la vez”. Efectuará una serie de comprobaciones como si solo hay audio o solo video, o si los eventids son válidos....etc.

## API javax.tv.service.selection.ServiceContext

- El API
  - Toda aplicación se ejecuta en un ServiceContext. Hemos visto que es posible cambiar de service mediante el API select() en ServiceContext. En este caso, el nuevo Service es el activo en el ServiceContext sobre el que se ha efectuado la llamada y todos aquellos servicios arrancados en el anterior se pararán salvo que sean signalled en el de destino.
  - public void **stop()** throws SecurityException;
    - Pasa de **Presentation\_pending** o **Presenting** a **Not\_presenting**.
    - Los recursos serán liberados, los ServiceMediaHandler(más adelante los vemos) no volverán a arrancar.
    - Se lanza un **PresentationTerminatedEvent**.
    - Es una llamada asíncrona
  - public void **destroy()** throws SecurityException;
    - Destruye el Service y provoca que se liberen todos los recursos.
    - Si estaba en **Presenting**: primero se para generando un **PresentationTerminatedEvent**, a continuación del cual se lanza un **ServiceContextDestroyedEvent** cuando ya está **Destroyed**.
    - Después todos los ServiceMediaHandlers habrán sido cerrados o estarán en estado **unrealized**
    - Es una llamada asíncrona

## API javax.tv.service.selection.ServiceContext

- El API
    - `public ServiceContentHandler[] getServiceContentHandlers()`
      - Ofrece la lista de los ServiceContentHandler del servicio: **devolverá uno del tipo ServiceMediaHandler que se encarga de la toda la presentación de tipo Media, si es que se está presentando media, y uno o varios encargados de gestionar los Xlet en ejecución, si es que los hay.**
      - cero si el ServiceContext está en `not_presenting` o `presentation_pending`
    - `public Service getService()`
      - Service presentándose en este ServiceContext
    - `public void addListener(ServiceContextListener listener);`
    - `public void removeListener(ServiceContextListener listener);`
      - Suscripción/De-suscripción del Listener.
- ```
public interface ServiceContextListener extends EventListener {  
    public void receiveServiceContextEvent(ServiceContextEvent e);  
}
```

Veamos en detalle los tipos de eventos que se pueden recibir.

## API `javax.tv.service.selection.ServiceContextListener`

- El API. Los events del Listener pueden ser del tipo:
  - **PresentationChangedEvent**: Indica que o bien hay un nuevo Service presentándose o que el que había ha dejado de hacerlo. El estado del ServiceContext es **Presenting**
    - Este event es el padre de **NormalContentEvent** y **AlternativeContentEvent**, y se dará cuando no se pueda generar ninguno de estos dos.
  - **SelectionFailedEvent**: indica que no se ha podido seleccionar un servicio. El estado en que quedará el ServiceContext será **Presenting** o **Not\_Presenting**, dependiendo de si había Service y se ha podido recuperar o no. Este Event ofrece un método para indicarnos la causa: `getReason()`, y que puede ser (ver API):
    - `public final static int INTERRUPTED`
    - `public final static int CA_REFUSAL`
    - `public final static int CONTENT_NOT_FOUND`
    - `public final static int MISSING_HANDLER`
    - `public final static int TUNING_FAILURE`
    - `public final static int INSUFFICIENT_RESOURCES`

## API `javax.tv.service.selection.ServiceContextListener`

- El API. Los events del Listener pueden ser del tipo:
  - **PresentationTerminatedEvent**: indica que el servicio ha dejado de presentar contenido y por lo tanto está en estado **Not\_Presenting**. Se lanza cuando se ha ejecutado **stop()** o **destroy()** sin pasar por `stop()`. Este Event ofrece un método para indicarnos la causa: `getReason()`, y que puede ser (ver API):
    - `public final static int SERVICE_VANISHED`
    - `public final static int TUNED_AWAY`
    - `public final static int RESOURCES_REMOVED`
    - `public final static int ACCESS_WITHDRAWN`
    - `public final static int USER_STOP`
  - **ServiceContextDestroyedEvent**: El ServiceContext se ha destruido. Se llamó a `destroy()` y el estado es **Destroyed**

## API javax.tv.service.selection.ServiceContextListener

- El API. Los events del Listener pueden ser del tipo:
  - **NormalContentEvent** (hereda de PresentationChangedEvent): Indica que el contenido del Service se está presentando. Se da en dos circunstancias:
    - Cuando se ha seleccionado un nuevo Service, este se presenta correctamente y ninguno de sus componentes es **AlternativeContent**. Si alguno lo es entonces el evento será **AlternativeContentEvent**.
    - Cuando se está presentando **AlternativeContent** pero se reemplaza por contenido que de forma natural pertenece al Service. Por ejemplo cuando salimos de un proceso de Pago de Pay per View
  - **AlternativeContentEvent** (hereda de PresentationChangedEvent): Indica que se está presentando contenido que no forma parte de forma natural del Service, como por ejemplo diálogos con el usuario debido a fallos en un CA. La plataforma es la que lanza el evento. Se da en dos circunstancias, una ya descrita, veamos la segunda:
    - Cuando estando presentando un Service que no tiene AlternativeContent un componente es reemplazado por otro que sí lo es.

## API `javax.tv.service.selection.ServiceContextFactory`

- La obtención de un `ServiceContext` se hace mediante una factory con un singleton.
  - `public static ServiceContextFactory getInstance()`
    - Acceso a la factory mediante un Singleton
    - **Curiosidad:** en código: `theServiceContextFactory = new com.sun.tv.ServiceContextFactoryImpl();`
  - `public ServiceContext createServiceContext()`
    - Crea un `ServiceContext`. No muy usado.
  - `public ServiceContext[] getServiceContexts()`
    - Ofrece los `ServiceContext` a los que se permite acceder. Si tenemos un STB que permite la emisión de dos `Services` a la vez obtendríamos 2.
  - `public ServiceContext getServiceContext(javax.tv.xlet.XletContext ctx)`
    - Ofrece el `ServiceContext` correspondiente al que incluye al Xlet en ejecución
    - **Este método y el anterior serán los más comúnmente usados:** normalmente se desea acceder al `ServiceContext` actual.

## ¿ Qué es un `javax.tv.service.selection.ServiceContentHandler` ?

- Representa un Mecanismo para Presentar, procesar y ejecutar un cierto tipo de contenido parte de un Service
- Un `ServiceContentHandler` puede manejar más de un elemento de un Service. Los elementos que maneja vienen definidos por los Locators que ofrece su API (incluyendo Xlets)

```
public interface ServiceContentHandler {  
    public Locator[] getServiceContentLocators();  
}
```

- Como veremos a continuación, cuando se está presentando un Service **obtendremos un solo `ServiceContentHandler`** del tipo **`ServiceMediaHandler`** para gestionar todo lo que no sean aplicaciones: Video, Audio...etc y cuyos locators describirán todos los elementos que gestiona.
- Aquellos servicios que sólo contengan DVB-J Applications sólo tendrán `ServiceContentHandlers` representando las DVB-J Applications en ejecución.

## Y ¿ Qué es un `javax.tv.service.selection.ServiceMediaHandler`?

- Representa a un Handler (hereda de aquel) de componentes del Service que son de tipo Real Time Media compartiendo el mismo reloj (el mismo PCR), por ejemplo Audio+Video+ Subtítulos de una peli.

```
public interface ServiceMediaHandler extends javax.media.Player, ServiceContentHandler{  
    }  
}
```

- En MHP los ServiceMediaHandler han de ser instancias de **Java Media Framework Player: son players capaces de reproducir todo un Service: audio, video...**
- En definitiva, quienes se encargan de gestionar y presentar las piezas de un Service son los Handlers.
- Si hay aplicaciones estas serán gestionadas por otro tipo de Handler distinto al de JMF

## Abstract Services

- El modelo explicado no funciona muy bien cuando no tenemos aplicaciones asociadas a contenido Broadcast, sino que son aplicaciones que o bien no están asociadas a servicios o bien simplemente son parte del receptor.
- Para solventar las deficiencias se usa el concepto de **Abstract Services**: representan servicios definidos por el Broadcaster y que pueden funcionar de manera agrupada.
- Normalmente **en un STB podrán coexistir a lo sumo tantos ServiceContext como decoders MPEG** disponga el receptor, sin embargo el número puede ser mayor si tenemos en cuenta los servicios “unbound”, los cuales se soportan en algunos decos MHP: por ejemplo, el Strong 5510, ofrece **StoredChannels**.
- En este grupo están contemplados los InternetClients del Profile Internet: mail...de hecho:

```
public interface InternetClient extends javax.tv.service.selection.ServiceContentHandler
```

```
public interface InternetClientService extends javax.tv.service.Service
```

## App Listing & Launching API

- Una referencia a APP Listing & Launching API: Cuando una aplicación lanza a otra se dice que el ServiceContext de la segunda será el mismo que el de aquella que la lanzó.

## Una referencia a la seguridad: Service Selection

- Las Unsigned NO pueden cambiar de Service (Ver /logs/Exercise-sc1b-security.txt por ejemplo). Las signed pueden solicitarlo en el PRF de la forma siguiente:

```
<!ELEMENT servicesel EMPTY>
```

```
<!ATTLIST servicesel value (true|false) "true" >
```

Ejemplo: `<servicesel value="false"></servicesel>`

- **OJO:** Es posible que la selección de un Service implique que debe existir acceso al Return Channel si por ejemplo el nuevo Service necesita de este para bajarse la AIT (Interactive Channels)

## Ejercicios Bloque SERCTX-1

## Problemática en la Selección de Service mediante Locators

- Un apunte acerca de la selección de Services mediante Locators cuando conocemos la url, p.e. `dvb://22d4.a.12c`. Los mecanismos siguientes NO funcionan (al menos en el Strong 5510). El `select()` NO entiende el Locator y lo considera erróneo.

```
Service Context sc...
```

```
sc.select( new Locator[]{new org.davic.net.dvb.DvbLocator("dvb://22d4.a.12c")});  
sc.select( new Locator[]{javax.tv.locator.LocatorFactory.getInstance().createLocator("dvb://22d4.a.12c")});
```

- Sin embargo si obtenemos el Service primero a través de `SIManager` y después seleccionamos el Service SÍ funciona.

```
Service ser = javax.tv.service.SIManager.createInstance().getService(new org.davic.net.dvb.DvbLocator("dvb://22d4.a.12c"));  
Service ser = javax.tv.service.SIManager.createInstance().getService(javax.tv.locator.LocatorFactory.getInstance().createLocator(dvb://22d4.a.12c));  
sc.select(ser);
```

- Curiosidad: ni siquiera usando el mismo Locator de un Service funciona.

```
sc.select(new Locator[]{ser.getLocator()});
```

- Por último: `toExternalForm` de un Locator efectivamente puede ser platform dependent:

```
TEST LOCATOR toString:dvb://22d4.a.64 TEST LOCATOR toExternalForm:dvb://22d4.a.64  
SERVICE LOCATOR toString:dvb://22d4.a.64 SERVICE LOCATOR toExternalForm:osmo:0015dvb://22d4.a.64bound:ptl=
```

- Una de las aplicaciones más habituales en las que podemos usar la selección de canal es una EPG

## Ejercicios Bloque SERCTX-2

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