



Curso Multimedia Home Platform 1.1.2

APP Signalling

¿ Cómo nos llega la App ?

Transport Protocols File System Implemented only Via de Interaction Channel

Transport Protocols Hybrid between Broadcast Stream e Interaction Channel

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.

Documentos a tener a mano

- ETSI EN 300 468
Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems
- ISO/IEC 13818 1
Information technology - Generic coding of moving pictures and associated audio information: Systems
- MHP 1.1.2 Specs A068r1

Empecemos retomando conceptos....

- Program Association table (**PAT**)
 - Describe para cada Servicio que PID (identificador de ES) contiene su tabla PMT (Program Map Table)
 - También contiene el PID del ES donde va la NIT, la Network Information Table
 - Ejemplo:
 - Telecinco PID1
 - Telemadrid PID2
 - Canal 6 PID3
 - NETWORK PID4

- Formato de la **PAT**:
Program Association
Table (table_id= 0x0000)

ISO-13818-1

Table 2-25 – Program association section

Syntax	No. of bits	Mnemonic
program_association_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
transport_stream_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
for (i = 0; i < N; i++) {		
program_number	16	uimsbf
reserved	3	bslbf
if (program_number == '0') {		
network_PID	13	uimsbf
}		
else {		
program_map_PID	13	uimsbf
}		
}		
CRC_32	32	rpchof
}		

- Program Map Table (**PMT**)
 - Describe para un servicio/canal todos los ES que lo componen indicando **el tipo de cada uno**. Además indica cual de estos ES contiene el **MPEG Program Clock Reference: PCR** (**sirve para sincronizar el reloj con el deco!**)
- **PCR**: Program Clock Reference
 - Muy importante para mantener la sincronización del programa y el receptor.
 - Se guarda en una zona opcional de los paquetes del PES.

- Formato de la **PMT**: Program Map Table, (table_id= 0x02)

ISO-13818-1

Table 2-28 – Transport Stream program map section

Syntax	No. of bits	Mnemonic
TS_program_map_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved	3	bslbf
PCR_PID	13	uimsbf
reserved	4	bslbf
program_info_length	12	uimsbf
for (i = 0; i < N; i++) {		
descriptor()		
}		
for (i = 0; i < N1; i++) {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID	13	uimsbf
reserved	4	bslbf
ES_info_length	12	uimsbf
for (i = 0; i < N2; i++) {		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

- Como veis, en el caso de la PMT hay un bucle de descriptores Globales y otro bucle en el que se describen los ES de que consta el Service. (ved ISO 13818-1).

Veamos los descriptores más relevantes en cuanto a App Signalling se refiere.

- **Tagging: stream_identifier_descriptor.** Cuando se identifica un Stream en la **PMT** (Program Map Table) puede incluirse en el bucle de descriptores uno del tipo **stream_identifier_descriptor (0x52)** dentro del cual se define un valor único al que se puede hacer referencia para localizar el Stream desde otros lugares. Ved EN 300 468

Table 87: Stream identifier descriptor

Syntax	Number of bits	Identifier
<code>stream_identifier_descriptor() {</code>		
<code> descriptor_tag</code>	8	uimsbf
<code> descriptor_length</code>	8	uimsbf
<code> component_tag</code>	8	uimsbf
<code>}</code>		

EN 300 468

- ¿ por qué este stream_identifier_descriptor ? **Los PID pueden variar. Tus tag NO**
- Desde fuera normalmente el campo se denomina también **component_tag**. Veremos un ejemplo en el transport_protocol_descriptor para Object Carousels



- **application signalling descriptor:** Un ES de tipo 0x05 (MPEG-2 Private Sections) puede tener un descriptor del tipo: **application signalling descriptor** (descriptor_tag = 0x6F), cuyo formato es:

Table 80: application signalling descriptor syntax

	No.of Bits	Identifier
<code>application_signalling_descriptor() {</code>		
<code>descriptor_tag</code>	8	uimsbf
<code>descriptor_length</code>	8	uimsbf
<code>for(i=0; i<N; i++){</code>		
<code>reserved_future_use</code>	1	
<code>application_type</code>	15	uimsbf
<code>reserved_future_use</code>	3	bslbf
<code>AIT_version_number</code>	5	uimsbf
<code>}</code>		
<code>}</code>		

MHP 1.1.2 A068r1

- Esto nos indica que ese ES contendrá la **AIT** (Application Information Table) donde se describirán las Aplicaciones del tipo **application_type**: 0x0001 DVB-J, 0x0002 DVB-HTML
- El **AIT_version_number** nos ofrece la versión de dicha tabla. El Deco no necesita leer la AIT para detectar cambio de versión.

- En el ES indicado por el `application_signalling_descriptor` nos encontraremos con el descriptor de la AIT cuyo formato apuntamos (`table_id = 0x74`)
- Dentro de la AIT tenemos información que aplica a todas las Apps: hay un **Global/Common descriptors loop**, e información relativa a cada App en particular: **application_loop**, dentro de la cual a su vez tenemos otro bucle denominado **inner descriptors loop**, que nos aportará información para cada APP.

Veamos el detalle.

MHP 1.1.2 A068r1

Table 74: Application Information Section syntax

	No.of Bits	Identifier
<code>application_information_section() {</code>		
<code>table_id</code>	8	uimsbf
<code>section_syntax_indicator</code>	1	bslbf
<code>reserved_future_use</code>	1	bslbf
<code>reserved</code>	2	bslbf
<code>section_length</code>	12	uimsbf
<code>test_application_flag</code>	1	bslbf
<code>application_type</code>	15	uimsbf
<code>reserved</code>	2	bslbf
<code>version_number</code>	5	uimsbf
<code>current_next_indicator</code>	1	bslbf
<code>section_number</code>	8	uimsbf
<code>last_section_number</code>	8	uimsbf
<code>reserved_future_use</code>	4	bslbf
<code>common_descriptors_length</code>	12	uimsbf
<code>for(i=0;i<N;i++){</code>		
<code>descriptor()</code>		
<code>}</code>		
<code>reserved_future_use</code>	4	bslbf
<code>application_loop_length</code>	12	uimsbf
<code>for(i=0;i<N;i++){</code>		
<code>application_identfier()</code>		
<code>application_control_code</code>	8	uimsbf
<code>reserved_future_use</code>	4	bslbf
<code>application_descriptors_loop_length</code>	12	uimsbf
<code>for(j=0;j<N;j++){</code>		
<code>descriptor()</code>		
<code>}</code>		
<code>}</code>		
<code>CRC_32</code>	32	rpchof
<code>}</code>		

Campos “de fuera” del application loop

- **test_application_flag:** usado para que las apps no se vean por los Apis. Para testing del deco.
- **application_type:** tipos de aplicaciones definidas en esta AIT.

application_type	description
0x0000	reserved_future_use
0x0001	DVB-J application
0x0002	DVB-HTML application
0x0003 to 0x7FFF	subject to registration with DVB

- **Global / Common descriptor() Loop:** los iremos viendo después.

MHP 1.1.2 A068r1

Table 74: Application Information Section syntax

	No.of Bits	Identifier
application_information_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
test_application_flag	1	bslbf
application_type	15	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved_future_use	4	bslbf
common_descriptors_length	12	uimsbf
for(i=0;i<N;i++){		
descriptor()		
}		
reserved_future_use	4	bslbf
application_loop_length	12	uimsbf
for(i=0;i<N;i++){		
application_identfier()		
application_control_code	8	uimsbf
reserved_future_use	4	bslbf
application_descriptors_loop_length	12	uimsbf
for(j=0;j<N;j++){		
descriptor()		
}		
}		
}		
CRC_32	32	rpchof
}		

Campos de dentro del Applications Loop

- **Application_identifier():**

MHP 1.1.2 A068r1

- Como ya hemos comentado resulta interesante la norma de los appid **para ahorrar trabajo y tiempo** a la capa de seguridad:

Table 76: Application identifier syntax

	No.of Bits	Identifier
<code>application_identifier {</code>		
<code>organisation_id</code>	32	bslbf
<code>application_id</code>	16	bslbf
<code>}</code>		

Table 77: Value ranges for application_id

application_id values	Use
0x0000 to 0x3fff	Application_ids for unsigned applications
0x4000 to 0x7fff	Application_ids for signed applications
0x8000 to 0xffff	Reserved for future use by DVB
0xfffe	Special wildcard value for signed applications of an organization
0xffff	Special wildcard value for all applications of an organization

MHP 1.1.2 A068r1

- Veremos después en el descriptor **External Authorization Applications Descriptor** como pueden usarse los valores **0xfffe y 0xffff**

- Campos de dentro del Applications Loop
- **Application_control_code:** nos dice como debe comportarse el ciclo de vida de la APP.
- **Inner loop descriptors():** los vemos a continuación.

Table 78: DVB-J application control code values

code	identifier	semantics
0x00		reserved_future_use
0x01	AUTOSTART	The file system element(s) (e.g. an Object Carousel module) containing the class implementing the Xlet interface is loaded. The class implementing the Xlet is loaded into the VM and an Xlet object is instantiated, and the application is started subject to usual restrictions, etc.
0x02	PRESENT	Indicates that the application is present in the service, but is not autostarted.
0x03	DESTROY	When the control code changes from AUTOSTART or PRESENT to DESTROY, the destroy method of the Xlet is called (with the unconditional parameter set to false) by the application manager and the application is allowed to destroy itself gracefully.
0x04	KILL	When the control code changes from AUTOSTART or PRESENT to KILL, the destroy method of the Xlet is called (with the unconditional parameter set to true) by the application manager.
0x05		reserved_future_use
0x06	REMOTE	This identifies a remote application that is only launchable after service selection.
0x07 to 0xFF		reserved_future_use

- Inner. Application Descriptor, `descriptor_tag = 0x00`
- Lo más importante de este descriptor es que nos dirá **qué versiones y perfiles de MHP soporta** y las **posibles ubicaciones de la App**.
- Cada APP tendrá exactamente un `application_descriptor`

Table 83: application descriptor syntax

	No.of Bits	Identifier
<code>application_descriptor() {</code>		
<code>descriptor_tag</code>	8	uimsbf
<code>descriptor_length</code>	8	uimsbf
<code>application_profiles_length</code>	8	uimsbf
<code>for(i=0; i<N; i++) {</code>		
<code>application_profile</code>	16	uimsbf
<code>version.major</code>	8	uimsbf
<code>version.minor</code>	8	uimsbf
<code>version.micro</code>	8	uimsbf
<code>}</code>		
<code>service_bound_flag</code>	1	bslbf
<code>visibility</code>	2	bslbf
<code>reserved_future_use</code>	5	bslbf
<code>application_priority</code>	8	uimsbf
<code>for(i=0; i<N; i++) {</code>		
<code>transport_protocol_label</code>	8	uimsbf
<code>}</code>		
<code>}</code>		

MHP 1.1.2 A068r1

- **Inner. Application Descriptor (y 2) descriptor_tag = 0x00**
- **Bucle de Profiles /versiones:** Indican los Profiles y para cada uno la versión de MHP que soporta la App.
- **service_bound_flag:** si “1” es una aplicación asociada al Servicio que la emite de manera que al principio del cambio de Service se matará. Si “0” la vida de la App puede trascender al cambio de canal (lo veremos)
- **transport_protocol_label:** este código nos lleva a un descriptor de protocolo de transporte (ahora lo vemos) **que nos dice de donde bajarnos las clases!!!** Puede haber más de una posibilidad, no es que las clases estén repartidas. La primera posibilidad es la mejor opción.

Table 83: application descriptor syntax

	No.of Bits	Identifier
<code>application_descriptor() {</code>		
<code>descriptor_tag</code>	8	uimbsf
<code>descriptor_length</code>	8	uimbsf
<code>application_profiles_length</code>	8	uimbsf
<code>for(i=0; i<N; i++) {</code>		
<code>application_profile</code>	16	uimbsf
<code>version.major</code>	8	uimbsf
<code>version.minor</code>	8	uimbsf
<code>version.micro</code>	8	uimbsf
<code>}</code>		
<code>service_bound_flag</code>	1	bslbf
<code>visibility</code>	2	bslbf
<code>reserved_future_use</code>	5	bslbf
<code>application_priority</code>	8	uimbsf
<code>for(i=0; i<N; i++) {</code>		
<code>transport_protocol_label</code>	8	uimbsf
<code>}</code>		
<code>}</code>		

Inner. Application Descriptor (y 3), descriptor_tag = 0x00

- **Visibility:** Vemos en la tabla siguiente los posibles valores.
- **application_priority:** mayor cuanto más grande. Recursos limitados...

Table 84: Definition of visibility states for applications

visibility	description
00	This application shall not be visible either to applications via an application listing API or to users via the navigator with the exception of any error reporting or logging facility, etc.
01	This application shall not be visible to users but shall be visible to applications via an application listing API.
10	reserved_future_use
11	This application can be visible to users and shall be visible to applications via the application listing API.

MHP 1.1.2 A068r1

- Un ejemplo interesante de uso de la visibilidad:
 - 00 - Aplicación Autostart que ofrece al usuario que Juego Lanzar
 - 11 - Los juegos que puede lanzar
 - 01 - Aplicación puente con el servidor de juegos con la que todos los Juegos se comunican.

Inner. Application name descriptor `descriptor_tag = 0x01`

- Nos permite describir el nombre de la App en diferentes idiomas.
- Cada aplicación tendrá un `application_name_descriptor` asociado en su inner loop.
- **ISO_639_language_code**: código de idioma de 3 char definido en ISO 639-2

Table 85: application name descriptor syntax

	No.of Bits	Identifier
<code>application_name_descriptor() {</code>		
<code>descriptor_tag</code>	8	uimsbf
<code>descriptor_length</code>	8	uimsbf
<code>for (i=0; i<N; i++) {</code>		
<code>ISO_639_language_code</code>	24	bslbf
<code>application_name_length</code>	8	uimsbf
<code>for (i=0; i<N; i++) {</code>		
<code>application_name_char</code>	8	uimsbf
<code>}</code>		
<code>}</code>		
<code>}</code>		

MHP 1.1.2 A068r1

Inner/Global. Transport Protocol Descriptor, descriptor_tag = 0x02

- Nos ofrece unos **datos de transporte** que se pueden usar para proporcionar la APP. Básicamente consisten en ubicaciones físicas en distintos sistemas de almacenamiento.
- Este descriptor puede encontrarse **en el loop principal de la AIT** o bien en el de una **APP** en concreto. Cuando se encuentra en el loop principal aplica a todas las APPs y si hay otro asociado a una de ellas se considera una opción más además del existente en el loop principal.

Table 90: transport protocol descriptor syntax

	No.of Bits	Identifier	Value
transport_protocol_descriptor() {			
descriptor_tag	8	uimsbf	
descriptor_length	8	uimsbf	
protocol_id	16	uimsbf	
transport_protocol_label	8	uimsbf	
for(i=0; i<N; i++) {			
selector_byte	8	uimsbf	N1
}			
}			

Inner/Global. Transport Protocol Descriptor descriptor_tag = 0x02 (y 2)

- **descriptor_tag**: es el valor fijo que identifica el tipo de descriptor: **0x02**
- **protocol_id**: código que identifica el tipo de protocolo.

Table 91: Protocol_id

protocol_id	description
0x0000	reserved_future_use
0x0001	MHP Object Carousel as defined in annex B.
0x0002	IP via DVB multiprotocol encapsulation as defined in EN 301 192 [5], TR 101 202 [49]
0x0003	Transport via HTTP over the interaction channel as described in clause 10.8.1.3.
0x0004 to 0x00FF	Reserved for use by DVB
0x0100 to 0xFFFF	Subject to registration in TR 101 162 [10]

MHP 1.1.2 A068r1

Inner/Global. Transport Protocol Descriptor descriptor_tag = 0x02 (y 3)

- Hemos de recordar que este descriptor puede usarse no solo para indicar el mecanismo de transporte de Apps.
- Como vemos el **0x0001** se usaría para transmitir la App por el Broadcast y el **0x0003** si se baja por el canal de retorno!!! El **0x0002** se refiere a **DSMCC MPE (Multiprotocol)**

Table 91: Protocol_id

protocol_id	description
0x0000	reserved_future_use
0x0001	MHP Object Carousel as defined in annex B.
0x0002	IP via DVB multiprotocol encapsulation as defined in EN 301 192 [5], TR 101 202 [49]
0x0003	Transport via HTTP over the interaction channel as described in clause 10.8.1.3.
0x0004 to 0x00FF	Reserved for use by DVB
0x0100 to 0xFFFF	Subject to registration in TR 101 162 [10]

Inner/Global. Transport Protocol Descriptor `descriptor_tag = 0x02` (y 4)

- El caso `0x0003` es el usado para definir el protocolo:

File System Implemented only Via de Interaction Channel

Inner/Global. Transport Protocol Descriptor `descriptor_tag = 0x02` (y 5)

- **`transport_protocol_label`**: Contiene un identificador único de 8 bits que identifica a este **`transport_protocol_descriptor` de forma única en la AIT**. El **`application_descriptor`** usa este valor para referirse a este `transport_protocol_descriptor` para identificar las conexiones que pueden llevar la aplicación, en orden de preferencia; recordemos su definición en el `application_descriptor`:
 - **`transport_protocol_label`**: este código nos lleva a un descriptor de protocolo de transporte (ahora lo vemos) **que nos dice de donde bajarnos las clases!!!** Puede haber más de una posibilidad, no es que las clases estén repartidas. La primera posibilidad es la mejor opción.
- Como veremos para el caso 0x003: canal de retorno, es posible que haya varios descriptors de este tipo **con un mismo `transport_protocol_label`**. Esto se ofrece para poder describir sin limitaciones las ubicaciones de las clases. En realidad se considera como uno solo.
- El valor de **`selector_byte`** depende del tipo de transporte, es decir, nos han dicho **el tipo de transmisión**, y ahora nos van a indicar los **lugares donde buscar las clases**.
- **OJO: AÚN NO NOS HAN DICHO QUÉ CLASES!!!!**

Veamos el detalle del `selector_byte` para cada tipo de transmisión.

Transport Protocol Descriptor descriptor_tag = 0x02 (y 6)

selector_byte para **0x0001 Object Carousel** (Broadcast):

Table 93: Syntax of selector bytes for OC transport

Syntax	Bits	Mnemonic
<code>remote_connection</code>	1	bslbf
<code>reserved_future_use</code>	7	bslbf
<code>if(remote_connection == "1") {</code>		
<code>original_network_id</code>	16	uimsbf
<code>transport_stream_id</code>	16	uimsbf
<code>service_id</code>	16	uimsbf
<code>}</code>		
<code>component_tag</code>	8	uimsbf

MHP 1.1.2 A068r1

- **remote_connection:** Cuando vale “1” se está indicando que el Object Carousel no lo proporciona el mismo Service donde se está publicando la AIT. Por eso en el IF se da la información necesaria: ¿ recordamos como identifica un service de forma única ?
- **component_tag:** es el tag con el que ha sido “taggeado” (stream_identifier_descriptor) el stream donde reside el Object Carousel, el cual habrá de publicar un DSMCC-DSI Message donde se describe el ServiceGateway: el directorio raíz del Object Carousel.(lo veremos)

Transport Protocol Descriptor descriptor_tag = 0x02 (y 7)

selector_byte para **0x0002 IP Multiprotocol Encapsulation** (IP Broadcasted):

- **remote_connection:** ídem a 0x0001
- **El resto de datos nos ofrece las URLs de las que obtener las clases/datos necesarios. Estas han de refererirse con IPs no con nombres al no disponer de DNS en modo Broadcasting.**

Table 94: Syntax of selector bytes for IP transport

Syntax	Bits	Mnemonic
<code>remote_connection</code>	1	bslbf
<code>reserved_future_use</code>	7	bslbf
<code>if(remote_connection == "1") {</code>		
<code>original_network_id</code>	16	uimsbf
<code>transport_stream_id</code>	16	uimsbf
<code>service_id</code>	16	uimsbf
<code>}</code>		
<code>alignment_indicator</code>	1	bslbf
<code>reserved_future_use</code>	7	bslbf
<code>for(i=0; i<N; i++){</code>		
<code>URL_length</code>	8	uimsbf
<code>for(j=0; j<URL_length; j++){</code>		
<code>URL_byte</code>	8	uimsbf
<code>}</code>		
<code>}</code>		

Transport Protocol Descriptor descriptor_tag = 0x02 (y 8)

selector_byte para 0x0003 Via de Interaction Channel

- Nos vamos a bajar las clases por el canal de Retorno...(Xapplets?)

Table 95: Syntax of selector bytes for interaction transport

Syntax	Bits	Mnemonic
for(i=0; i<N; i++){ URL_base_length	8	uimsbf
for(j=0; j<N; j++){ URL_base_byte	8	uimsbf
}		
URL_extension_count	8	uimsbf
for(j=0; j<URL_extension_count; j++){ URL_extension_length	8	uimsbf
for(k=0; k<URL_length; k++){ URL_extension_byte	8	uimsbf
}		
}		
}		

MHP 1.1.2 A068r1

Transport Protocol Descriptor `descriptor_tag = 0x02` (y 9)

- `selector_byte` para **0x0003 Via de Interaction Channel**.
- Este mecanismo se denomina en las Specs: **Transport Protocols File System Implemented only Via de Interaction Channel**
- Por eficiencia las URLs se determinan usando la URL Base y a continuación todas las que “cuelgan” de esta.
- Las URLs para HTTP deben ser conformes a HTTP 1.0 (ver RFC 1945), y a HTTPS (RFC 2818)
- Las URLs definen directorios o **ficheros ZIP!!!!**
- **Importante:** en este caso (0x0003) puede haber más de un `protocol_transport_descriptor` de manera que pueda definirse un gran número de directorios distintos: la condición es que todos han de tener el mismo `transport_protocol_label`
- **Aún no sabemos qué clases son, pero ya sí conocemos de donde las obtendremos**

Inner. DVB-J Application Descriptor, `descriptor_tag = 0x03`

- Nos permite establecer para la App los parámetros de arranque, es decir, aquellos que se obtienen del contexto del Xlet.
- Habr  0 o 1 asociado a la aplicaci n concreta.

Table 99: DVB-J application descriptor syntax

	No.of Bits	Identifier
<code>dvb_j_application_descriptor(){</code>		
<code>descriptor_tag</code>	8	uimsbf
<code>descriptor_length</code>	8	uimsbf
<code>for(i=0; i<N; i++) {</code>		
<code>parameter_length</code>	8	uimsbf
<code>for(j=0; j<parameter_length; j++) {</code>		
<code>parameter_byte</code>	8	uimsbf
<code>}</code>		
<code>}</code>		
<code>}</code>		

MHP 1.1.2 A068r1

Inner. DVB-J Application Location Descriptor, descriptor_tag = 0x04

- **POR FIN: Descriptor que nos dirá los componentes de la aplicación!!!!: El classpath y la clase que implementa el Xlet**
- Habrá sólo 1 asociado a la aplicación concreta.

Table 100: DVB-J application location descriptor syntax

	No.of Bits	Identifier
<code>dvb_j_application_location_descriptor {</code>		
<code>descriptor_tag</code>	8	uimsbf
<code>descriptor_length</code>	8	uimsbf
<code>base_directory_length</code>	8	uimsbf
for(i=0; i<N; i++) {		
<code>base_directory_byte</code>	8	uimsbf
}		
<code>classpath_extension_length</code>	8	uimsbf
for(i=0; i<N; i++) {		
<code>classpath_extension_byte</code>	8	uimsbf
}		
for(i=0; i<N; i++) {		
<code>initial_class_byte</code>	8	uimsbf
}		
}		

Inner. DVB-J Application Location Descriptor, descriptor_tag = 0x04

- **descriptor_tag**: 0x04 identifica a este descriptor.
- **base_directory_length**: bytes del base_directory. Al menos 1 byte.
- **base_directory_byte**: String especificando un directorio. Los separadores de directorios son el “/”. El nombre del directorio es **relativo al top-level directory del file System** que contendrá a la aplicación. Este nombre de directorio es usado como un directorio base para los paths relativos. **Este directorio es considerado el primer directorio del classpath** después de los del sistema. Si el directorio base es la raíz será “/”.
- En realidad el top-level directory no necesitamos conocerlo, es una ubicación física interna del sistema, por ejemplo: /hdd/disc1/applications. **El base_directory SERÁ COMO NUESTRO DIRECTORIO RAÍZ AL QUE NOS REFERIREMOS PARA DESCRIBIR PATHS ABSOLUTOS.**
- **classpath_extension_byte**: contiene el Classpath. Separados por “;” (o depende de la property path.separator). Si los directorios son absolutos comienzan por “/” (seguido del base_directory). Si son relativos al base no. Los directorios se delimitan con “/” y acaban en “/”.
- **initial_class_byte**: clase que implementa el interface Xlet. Por ejemplo "com.broadcaster.appA.MainClass"
 - Nota: El classpath contiene sólo directorios. Un fichero jar **no** puede ser referenciado desde un **DVB-J application location descriptor**.

Inner. Application icons descriptor `descriptor_tag = 0x0B`

- Nos permite definir un icono en formato **PNG** para la App.
- Cada aplicación tendrá 0 o 1 `application_icons_descriptor` asociado en su inner loop.

Table 86: application icons descriptor syntax

	No.of Bits	Identifier
<code>application_icons_descriptor() {</code>		
<code>descriptor_tag</code>	8	uimsbf
<code>descriptor_length</code>	8	uimsbf
<code>icon_locator_length</code>	8	uimsbf
for (i=0; i<N; i++) {		
<code>icon_locator_byte</code>	8	uimsbf
}		
<code>icon_flags</code>	16	bslbf
for (i=0; i<N; i++) {		
<code>reserved_future_use</code>	8	bslbf
}		
}		

Inner. Application icons descriptor descriptor_tag = 0x0B (y 2)

- icon_locator_byte: en función del tipo de aplicación tendrá un valor u otro. En el caso de DVB-J Applications es un **path** relativo al “Base Directory” de la aplicación y no acaba en “/”
- icon_flags: mediante la activación de los flags nos dice los iconos que se proporcionan, los cuales se encuentran en el directorio indicado y se llamarán: **dvb.icon.<HEXVALUE>**, donde HEXVALUE es el valor de icon_flags si solo estuviera activado el icono en cuestión. A lo sumo hay 12: dvb.icon.0001, dvb.icon.0002....dvb.icon.000C

Table 88: Definition of different icon flags

Icon flag bits	Description of icon size and pixel aspect ratio
0000 0000 0000 0001	32 x 32 for square pixel display
0000 0000 0000 0010	32 x 32 for broadcast pixels on 4:3 display (note)
0000 0000 0000 0100	24 x 32 for broadcast pixels on 16:9 display
0000 0000 0000 1000	64 x 64 for square pixel display
0000 0000 0001 0000	64 x 64 for broadcast pixels on 4:3 display
0000 0000 0010 0000	48 x 64 for broadcast pixels on 16:9 display
0000 0000 0100 0000	128 x 128 for square pixel display
0000 0000 1000 0000	128 x 128 for broadcast pixels on 4:3 display
0000 0001 0000 0000	96 x 128 for broadcast pixels on 16:9 display
0000 0010 0000 0000	256 x 256 for square pixel display
0000 0100 0000 0000	256 x 256 for broadcast pixels on 4:3 display
0000 1000 0000 0000	192 x 256 for broadcast pixels on 16:9 display
xxxx 0000 0000 0000	reserved_future_use

Ejercicios Bloque APPSIG-1

Global. External application authorization descriptor, descriptor_tag=0x05

- En el Loop Global de la AIT puede haber 0-n ocurrencias de **external_application_authorization_descriptor** que lo que hace es permitir a aplicaciones que ya estaban ejecutándose en el Service anterior seguir haciéndolo en este pero NO se pueden arrancar desde este Service.
- En el application_identifier() se pueden usar los wildcard (**0xffff** o **0xfffe**, ¿ recordamos ?) para dar paso a todas las unsigned o signed apps de una determinada organización.

Table 89: external application authorisation descriptor syntax

	No.of Bits	Identifier
external_application_authorisation_descriptor() {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
for(i=0; i<N; i++) {		
application_identifier()		
application_priority	8	uimsbf
}		
}		

Inner. Prefetch Descriptor descriptor_tag = 0x0C

- Es opcional, puede aparecer n veces y se usa cuando el protocolo es Broadcast (0x0001) para ayudar al STB a ir conectando con el Object Carousel y de esta forma reducir los tiempos de latencia.

Table 97: Syntax of the pre-fetch descriptor

Syntax	Bits	Mnemonic
prefetch_descriptor () {		
descriptor_tag	8	uimsbf
descriptor_length	8	uimsbf
transport_protocol_label	8	uimsbf
for(i=0; i<N; i++) {		
label_length		
for(j=0; j<label_length; j++) {		
label_byte	8	uimsbf
}		
prefetch_priority	8	uimsbf
}		
}		

MHP 1.1.2 A068r1

Inner. Prefetch Descriptor descriptor_tag = 0x0C (y 2)

- El **transport_protocol_label** nos indica el transport_protocol_descriptor al que se refiere.
- **label_byte** nos indica los “módulos” que debe ir bajándose del Object Carousel (Data Carousel) y que están “labelled” con “**Label Descriptors**” con este valor. Algo parecido al stream_identifier_descriptor pero para hacer referencia a bloques del Object Carousel.
- Un “**Label Descriptor**” tiene el siguiente formato y va incluido en el campo userInfo de los modules del Data Carousel. El descriptor_tag es **0x70**:

Table B.8: Label descriptor syntax

Syntax	bits	Type	Value
label_descriptor() {			
descriptor_tag	8	uimsbf	0x70
descriptor_length	8	uimsbf	N1
for (n=0; n<N1; n++) {			
label_byte	8	uimsbf	
}			
}			

- Estos no son los únicos descriptors que afectan a las Aplicaciones, existen otros descriptors relacionados con estas, que se verán en cada tema concreto: Stored apps, Plug-ins...

Transport Protocol: Hybrid between Broadcast Stream e Interaction Channel

- Este es el caso en el que la aplicación se baja por los dos caminos de la siguiente forma: Toda la información de directorios y nombres de ficheros es igual que en el Broadcast, **lo que varía es donde reside el contenido del fichero**: este se obtiene mediante una URL que se especifica dentro del mensaje BIOP:File del DSCMM Object Carousel usando lo que se denomina un HTTPProfileBody dentro del mensaje. Las URLs tendrán la forma:
http://host:port/path_segments

Table 2: HTTP Profile Body syntax

Syntax	bits	Type	Value	Comment
HTTPProfileBody {				
profileId_tag	32	uimsbf	0x44564200	
profile_data_length	32	uimsbf	*	
profile_data_byte_order	8	uimsbf	0x00	big endian byte order
version.major	8	uimsbf	0x01	protocol major version 1
version.minor	8	uimsbf	0x00	protocol minor version 0
host_data_length	8	uimsbf	N1	
for (k=0; k<N1; k++) {				
host_data	8	uimsbf	+	
}				
port	16	uimsbf		
objectKey_length	16	uimsbf	N2	
for (k=0; k<N2; k++) {				
objectKey_data	8	uimsbf	+	
}				
}				
NOTE:	The tag values 0x44564201 to 0x44564201 have been reserved for future use by the DVB. 0x44 is ASCII for "D", 0x56 is "V" and 0x42 is "B".			

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