Guía de desarrollo de TV Apps en el STB híbrido

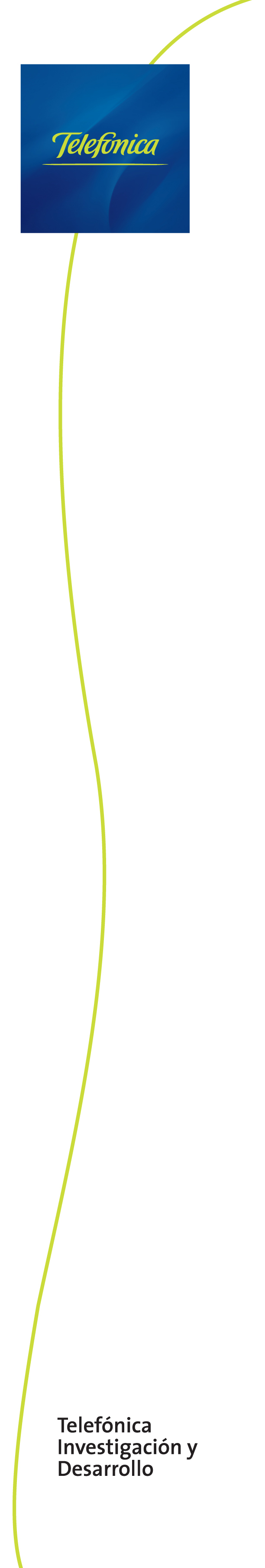
Fecha 23-3-2012

Autor Iniciativa TV Apps

Código TID-100323-000003

Edición 04.01 Rev. 00

USO INTERNO



© 2009 Telefónica Investigación y Desarrollo, S.A. Unipersonal  
Reservados todos los derechos

[1 INTRODUCCIÓN 6](#_Toc329155744)

[1.1 Objetivo 6](#_Toc329155745)

[1.2 Estructura del documento 6](#_Toc329155746)

[1.3 Documentación de referencia 7](#_Toc329155747)

[2 VISIÓN GENERAL DE LA PLATAFORMA DE TVApps 8](#_Toc329155748)

[2.1 Componentes de la Plataforma de TVApps 9](#_Toc329155749)

[2.1.1 Servidor de la Plataforma 9](#_Toc329155750)

[2.1.2 Contenedor 10](#_Toc329155751)

[2.1.3 Emulador del Contenedor 11](#_Toc329155752)

[2.1.4 Consola de Administración 13](#_Toc329155753)

[2.2 TVApps 14](#_Toc329155754)

[2.2.1 Vista funcional 14](#_Toc329155755)

[2.2.2 Vista técnica 14](#_Toc329155756)

[2.2.3 Organización de las TVApps en el servidor 16](#_Toc329155757)

[3 ENTORNO DE DESARROLLO DE LA PLATAFORMA DE TVApps 18](#_Toc329155758)

[3.1 Contenido del SDK 18](#_Toc329155759)

[3.2 Herramientas de desarrollo 18](#_Toc329155760)

[3.3 API del Contenedor TV 20](#_Toc329155761)

[3.3.1 Códigos de teclas 20](#_Toc329155762)

[3.3.1.1 Teclas de control 20](#_Toc329155763)

[3.3.1.2 Teclas de control de reproducción 21](#_Toc329155764)

[3.3.2 Funciones generales de manejo de una TVApp 22](#_Toc329155765)

[3.3.3 Funciones para el manejo de propiedades del TVApp 23](#_Toc329155766)

[3.3.4 Otras funciones 23](#_Toc329155767)

[3.4 Pasos para el desarrollo de TVApps 24](#_Toc329155768)

[3.5 Teclado virtual 26](#_Toc329155769)

[3.5.1 Configuración del teclado virtual 26](#_Toc329155770)

[3.5.2 Ejemplos de configuración y de personalización 28](#_Toc329155771)

[3.5.2.1 Teclado de login con datos rellenados 28](#_Toc329155772)

[3.5.2.2 Teclado de búsqueda 29](#_Toc329155773)

[3.5.2.3 Teclado de comentarios con elemento enfocable a la izquierda 30](#_Toc329155774)

[3.5.2.4 Teclado estándar 30](#_Toc329155775)

[3.5.2.5 Teclado estándar customizado 31](#_Toc329155776)

[3.5.3 Personalización del teclado 32](#_Toc329155777)

[3.5.3.1 Estilos de las teclas 32](#_Toc329155778)

[3.5.3.2 Estilos del formulario de login 33](#_Toc329155779)

[3.5.3.3 Estilos del formulario de búsqueda 34](#_Toc329155780)

[3.5.3.4 Estilos del teclado de comentarios 34](#_Toc329155781)

[3.6 Creación de popups con mensaje 35](#_Toc329155782)

[3.6.1 Personalización de popups 35](#_Toc329155783)

[3.7 Creación de popups de cargando 36](#_Toc329155784)

[3.7.1 Creación de popups de cargando animados 36](#_Toc329155785)

[3.7.2 Personalización 37](#_Toc329155786)

[3.8 Reproducción de vídeo 37](#_Toc329155787)

[3.8.1 Workaround forward y rewind en streaming 38](#_Toc329155788)

[3.8.2 Gestión de teclas 38](#_Toc329155789)

[3.8.3 Funciones de control de reproducción 38](#_Toc329155790)

[3.8.4 Eventos 39](#_Toc329155791)

[3.8.4.1 onSpeedChanged 39](#_Toc329155792)

[3.8.4.2 onSpeedChangeFailed 39](#_Toc329155793)

[3.8.4.3 onSetSpeedOk 40](#_Toc329155794)

[3.8.4.4 onSetSpeedFailed 40](#_Toc329155795)

[3.8.4.5 onStopOk 40](#_Toc329155796)

[3.8.4.6 onStopFailed 40](#_Toc329155797)

[3.8.4.7 onPositionChanged 40](#_Toc329155798)

[3.8.4.8 onPositionChangeFailed 40](#_Toc329155799)

[4 DESARROLLO DE HELPERS 42](#_Toc329155800)

[4.1 Conceptos Generales 42](#_Toc329155801)

[4.2 Gestión de Helpers 42](#_Toc329155802)

[4.2.1 Lista de Helpers 42](#_Toc329155803)

[4.2.2 Consulta de datos de un Helper 43](#_Toc329155804)

[4.2.3 Consulta de log de un Helper 44](#_Toc329155805)

[4.3 Librería de Helpers 45](#_Toc329155806)

[4.3.1 Método parse 47](#_Toc329155807)

[4.3.2 Códigos de Ejecución 47](#_Toc329155808)

[4.3.3 Control de Errores 48](#_Toc329155809)

[4.3.4 Funciones adicionales de la librería 48](#_Toc329155810)

[4.4 Recomendaciones de Desarrollo 51](#_Toc329155811)

[4.5 Ejemplo de desarrollo de Helper 53](#_Toc329155812)

[5 DESARROLLO DE TVAPPS PARA EL DESCODIFICADOR HÍBRIDO 55](#_Toc329155813)

[5.1 Gestión de TVApps 55](#_Toc329155814)

[5.1.1 Lista de TVApps 55](#_Toc329155815)

[5.1.2 Eliminación de una TVApp 56](#_Toc329155816)

[5.1.3 Redespliegue de una TVApp 57](#_Toc329155817)

[5.1.4 Consulta/modificación de datos de una TVApp 57](#_Toc329155818)

[5.1.5 Consulta/modificación de modelos de STB soportados por un TVApp 58](#_Toc329155819)

[5.1.6 Consulta/modificación de plataformas activas para una TVApp 58](#_Toc329155820)

[5.1.7 Consulta/modificación de grupos para una TVApp 59](#_Toc329155821)

[5.1.8 Despliegue de una TVApp nueva 60](#_Toc329155822)

[5.2 Limitaciones técnicas 61](#_Toc329155823)

[5.3 Formato de una TVApp 62](#_Toc329155824)

[5.4 Recomendaciones de estilo 64](#_Toc329155825)

[5.5 Estructura de una TVApp 65](#_Toc329155826)

[5.5.1 manifest.xml 66](#_Toc329155827)

[5.5.2 index.html 68](#_Toc329155828)

[5.5.3 widget.js 70](#_Toc329155829)

[5.6 Acceso a fuentes de datos 71](#_Toc329155830)

[5.6.1 Acceso estático a datos 71](#_Toc329155831)

[5.6.2 Acceso dinámico a datos 72](#_Toc329155832)

[glosario de términos y acrónimos 73](#_Toc329155833)

1. INTRODUCCIÓN
   1. Objetivo

Este documento es una guía detallada de cómo desarrollar aplicaciones para el descodificador híbrido a través de la Plataforma de TVApps desarrollada por Telefónica I+D. A estas aplicaciones las llamamos TVApps.

Las TVApps son pequeñas aplicaciones interactivas que llevan servicios de Internet a la televisión y que ofrecen una experiencia de usuario adaptada al entorno de televisión, facilitando el acceso a la información.

El objetivo de este documento es analizar qué son las TVApps, la estructura que tienen y los pasos a seguir para su completo desarrollo, teniendo en cuenta las herramientas y componentes que se necesitan.

Actualmente la Plataforma de TVApps permite que el front-end de las TVApps para el descodificador híbrido pueda desarrollarse en HTML y Javascript, y el back-end en PHP. El presente documento describe cómo desarrollar el front-end y back-end de TVApps con estas tecnologías.

* 1. Estructura del documento

El documento se divide en cinco capítulos principales:

* **Capitulo 1: Introducción**
* **Capitulo 2: Visión general de la Plataforma de TVApps**

Análisis general de las TVApps y la plataforma en la que se apoya: qué son las TVApps, partes que lo componen, qué es la Plataforma de TVApps y partes que la forman.

* **Capitulo 3: Entorno de Desarrollo de la Plataforma de TVApps**

Pasos a seguir para el desarrollo de nuevas TVApps y descripción de las herramientas y del API necesario.

* **Capitulo 4: Desarrollo de Helpers (back-end de TVApps)**

Análisis de los Helpers y su librería: qué es un Helper, cómo se desarrollan y cómo se gestionan.

* **Capitulo 5: Desarrollo del front-end de TVApps para el descodificador híbrido**

Análisis más detallado de las TVApps: cómo gestionar las TVApps a través de la consola de administración, formato y estilo de las TVApps, visualización y desarrollo de las TVApps en HTML y Javascript.

* 1. Documentación de referencia

Otros documentos relacionados con éste y en donde se puede ampliar información de las TVApps y su desarrollo son:

* Guía de estilo de TVApps
* Guía de referencia del API JS de TVApps para el descodificador híbrido

1. VISIÓN GENERAL DE LA PLATAFORMA DE TVApps

La Plataforma de TVApps es una solución para ofrecer servicios interactivos en la TV, a través del descodificador híbrido, y que permite que los usuarios accedan a información y servicios de Internet de una forma guiada y sencilla, a través de las TVApps.

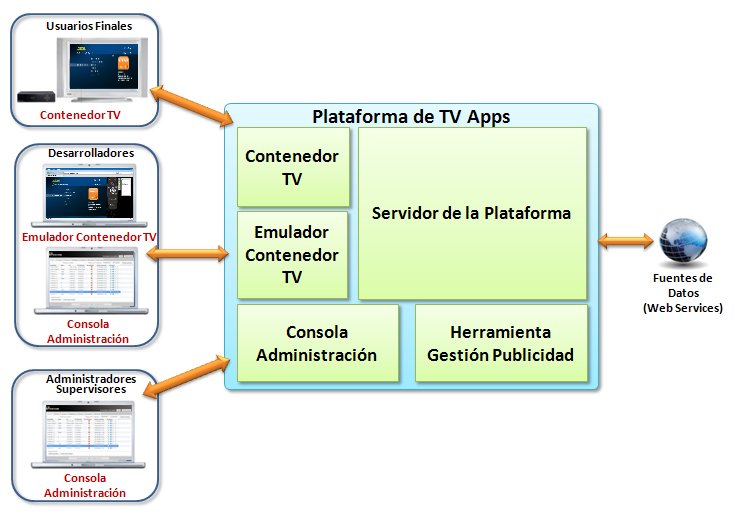
Las TVApps son pequeñas aplicaciones que incorporan funcionalidades de los servicios que hay en la web, y que ofrecen una experiencia de usuario agradable, sencilla y adaptada a la TV.

La Plataforma de TVApps está diseñada para que Telefónica o desarrolladores terceros puedan crear TVApps de una forma rápida y sencilla, y que se puedan ofrecer gran cantidad de servicios interactivos a los usuarios finales a través de la TV.

Las principales características de la Plataforma de TVApps son las siguientes:

* **Arquitectura abierta**:
* Incluye un API, un SDK y un entorno de desarrollo para que terceros puedan crear sus propias TVApps.
* Permite una fácil integración de cualquier servicio de Internet.
* **Permite generar servicios interactivos de gran fiabilidad**
* **Arquitectura escalable.**
* **Basado en estándares web y lenguajes W3C:** html, css, javascript**.**
* **Modular y desacoplada**: es fácilmente ampliable y configurable.
* **Incluye API de administración de la plataforma:** extensible a nuevos módulos e integrable con distintos sistemas.

La siguiente figura muestra una visión general de la Plataforma de TVApps, con sus componentes principales y las entidades externas con las que se relaciona.



1. Visión general de la Plataforma de TVApps
   1. Componentes de la Plataforma de TVApps

A continuación se describe desde el punto de vista funcional cada uno de los componentes de la Plataforma.

* + 1. Servidor de la Plataforma

El servidor es el componente principal de la Plataforma de TVApps.

Es el repositorio central de datos y ficheros y el encargado de ejecutar los distintos procesos necesarios para el correcto funcionamiento del sistema.

Incluye los siguientes elementos:

* **Base de datos**: dónde se almacenan todos los datos de la Plataforma.
* **pinchos-pserver**: procesos que permiten gestionar TVApps, usuarios, terceros, encuestas, estadísticas, etc. También incluye un API REST para que la Consola, el Contenedor TV y el Emulador puedan realizar operaciones dentro de la Plataforma de TVApps.
* **pinchos-stats**: procesos necesarios para la exportación de datos estadísticos en formato Excel y PDF.
* **Repositorio de TVApps** desplegados en la plataforma. Cada TVApps puede incluir un helper para acceder a los datos de las fuentes de datos externas.
  + 1. Contenedor

Este componente se sirve desde el Web Server de la Plataforma de TVApps, pero se ejecuta en el navgador del entorno del descodificador híbrido. Es el encargado de gestionar las TVApps asociadas a los usuarios y de controlar el acceso y funcionamiento de dichas TVApps.

Una vez cargado en el descodificador, al acceder a la opción TVApps, el Contenedor muestra el menú principal (carrusel) con las TVApps asociadas al usuario. Cada usuario podrá tener asociadas distintas TVApps, las cuales serán configuradas desde la consola de TVApps por un administrador de la plataforma (los propios terceros podrán realizar ciertas operaciones de configuración de las TVApps usando la consola).

En el menú principal del contenedor, se muestra el icono de cada una de las TVApps. El usuario interactúa con el contenedor utilizando el mando a distancia del descodificador híbrido. El contenedor realiza las peticiones al Servidor de la Plataforma para obtener los datos que necesita para cargar y ejecutar cada TVApp.

Cuando el usuario selecciona una TVApp, el contenedor la abre y oculta el menú principal. Mediante el mando a distancia el usuario puede navegar dentro de una TVApp. Las TVApps se muestran a pantalla completa.

El contenedor realiza las siguientes acciones:

* Gestiona la información de autenticación del usuario activo.
* Gestiona la información de personalización asociada a cada una de las TVApps del usuario.

El contenedor ofrece un API Javascript que permite a los desarrolladores de TVApps utilizar de forma muy sencilla ciertas capacidades de la Plataforma de TVApps y del descodificador. Algunas de estas capacidades son las siguientes:

* Reproducir vídeo y audio unicast bajo demanda.
* Acceder a recursos web externos a la Plataforma de TVApps.
* Recuperar y almacenar datos de personalización asociados a un usuario y una TVApp.



1. Pantalla principal del descodificador del híbrido



1. Menú principal del contenedor

Durante el desarrollo y prueba de las TVApps, para facilitar el trabajo, los desarrolladores no tendrán que utilizar el Contenedor directamente en la TV, sino que podrán usar en su lugar el Emulador del contenedor, que se describe a continuación.

* + 1. Emulador del Contenedor

Este componente consiste en una máquina virtual

Al igual que el Contenedor TV, este componente se sirve desde el Web Server de la Plataforma de TVApps y se ejecuta en un navegador web de un PC. Es el elemento principal del ecosistema de terceros de la Plataforma de TVApps, ya que permite que los desarrolladores puedan probar el comportamiento de sus TVApps sin necesidad de hacer todas las pruebas directamente en Mediabox. Ofrece las mismas funcionalidades que el Contenedor TV.



1. Emulador Container TV

Se recomienda que mientras se desarrolla la TVApp se realicen las pruebas en el emulador y que cuando se haya completado el desarrollo se pruebe la TVApp en la caja de Mediabox. Puede ocurrir que la apariencia de una TVApp cambie ligeramente entre el emulador y la caja de Mediabox. Por lo tanto, es importante realizar estas pruebas finales en la caja de Mediabox para comprobar que su funcionamiento es el esperado.

Para ejecutar el emulador del Mediabox es necesario tener disponible en el ordenador lo siguiente:

1. El navegador Mozilla Firefox 3.0 o superior (se recomienda utilizar siempre la última versión).
2. La URL del emulador del container para Mediabox del entorno de desarrollo de TVApps.
3. Un monitor con la suficiente resolución para mostrar el emulador. Se recomienda una resolución de 1280x1024.

Adicionalmente se recomienda (aunque no es obligatorio) instalar el plugin Firebug en el Firefox (<http://getfirebug.com>). Con el objetivo de poder depurar las aplicaciones que se estén desarrollando dentro del emulador.

El emulador se maneja usando el teclado del PC. A continuación se muestra la correspondencia entre los botones del mando a distancia y las teclas del PC.

|  |  |
| --- | --- |
| Botón del mando a distancia | Tecla del teclado del PC |
| Izquierda | Flecha izquierda |
| Derecha | Flecha derecha |
| Arriba | Flecha arriba |
| Abajo | Flecha abajo |
| OK | Enter |
| Atrás | Backspace |

* + 1. Consola de Administración

La Consola de Administración de la Plataforma de TVApps es una aplicación Web que permite gestionar distintos aspectos de esta plataforma: usuarios finales y sus instancias de TVApps, TVApps, terceros, usuarios de consola, etc.

Para poder acceder a la Consola de Administración es necesario conocer la URL de esta aplicación web y disponer de un login y una password válido de un usuario de consola de la Plataforma de TVApps.

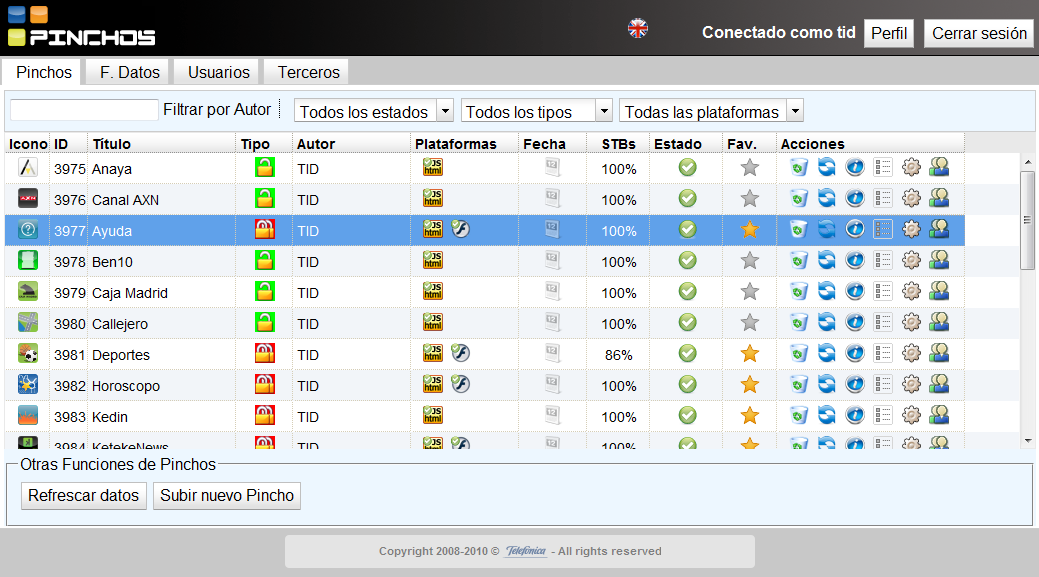
Un administrador de la Plataforma de TVApps dará de alta a cada desarrollador y le enviará por correo electrónico los datos necesarios para acceder a la Consola de Administración y al Emulador del Contenedor.

El acceso a las distintas funcionalidades de la consola está basado en un sistema de perfiles. Cada usuario de consola tiene un único perfil, y cada perfil tiene permiso para realizar ciertas operaciones.

Los desarrolladores de TVApps tendrán el perfil “Tercero” y podrán realizar las siguientes operaciones dentro de la consola:

* Gestionar TVApps propias.
* Consultar fuentes de datos asociadas a TVApps propias.
* Gestionar usuarios finales propios.
* Gestionar los datos del tercero.
* Consulta de datos del usuarios de consola y cambio de password.

La siguiente figura muestra el aspecto que tiene la Consola de Administración para el perfil “Tercero”:



1. Consola de Administración para perfil “Tercero”
   1. TVApps

Como ya se ha dicho anteriormente, las TVApps son pequeñas aplicaciones que traen hasta el televisor servicios de Internet y los adaptan a las peculiaridades de este dispositivo.

El contenedor es el encargado de gestionar las TVApps asignadas a cada usuario y de controlar el acceso a los mismos.

* + 1. Vista funcional

Una TVApp, desde el punto de vista funcional para los usuarios finales, está formada por una **aplicación** **en formato TVApps** que ofrece la funcionalidad completa del servicio. Según se va navegando por una TVApp se va mostrando información, fotos, etc. que generalmente reside en un servicio externo a la Plataforma de TVApps.

* + 1. Vista técnica

Una TVApp, desde el punto de vista técnico, es una aplicación web que reside en el repositorio de TVApps de la Plataforma de TVApps, y que está formada por:

* Un **backend** de la TVApp, que puede ser un **helper** o un **service.**

El **helper** sirve para acceder a las fuentes de datos o interfaces de Internet, cachear los datos obtenidos en el servidor de la Plataforma de TVApps y adaptar dichos datos al formato más apropiado y óptimo para los descodificadores híbridos. Los helpers deben desarrollarse en PHP. Los helpers se ejecutan periódicamente (mediante el cron del servidor), según su configuración, dentro del Servidor de la Plataforma de TVApps. Sólo se permite tener un helper por TVApp.

Si una TVApp requiere que los datos que utiliza se obtengan en tiempo de ejecución, entonces el backend de la TVApp se denomina **service**. La diferencia es que los services se invocarán desde la aplicación en el momento que sea necesario en lugar de hacerse periódicamente. Los services residirán dentro de la Plataforma de TVApps, formado parte de la TVApp correspondiente, y se ejecutarán dentro del Servidor de la Plataforma. Además, los datos obtenidos no se cachearán, ya que serán exclusivos de la instancia de la TVApp que los ha solicitado. Se pueden tener varios services por TVApp.

Un service solo debería crearse en caso de que no haya otras opciones, ya que es un elemento que consume recursos exponencialmente del servidor, a medida que aumenta el número de usuarios por aplicación.

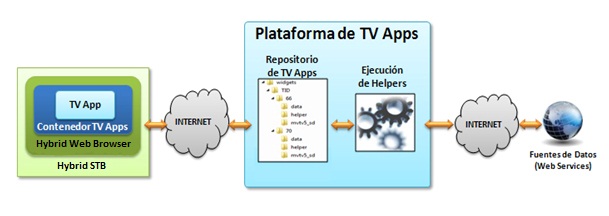
En general, las fuentes de datos de Internet ofrecen datos en formato XML, pero para los descodificadores híbridos, debido a la capacidad de procesamiento reducida que tienen, es recomendable ofrecer los datos en formato JSON. Además, es preferible adaptar al máximo los datos obtenidos a la funcionalidad que ofrece la TVApp.De esta forma también se evitará tener que manipular los datos en la TVApp que se ejecuta en el descodificador. Otra recomendación es reducir el número de peticiones de datos que tienen que hacerse desde la TVApp a las fuentes de datos. Es recomendable realizar pocas peticiones y que cada una de ellas recoja el máximo de datos necesarios.

El backend de la TVApp es opcional, pero en general se recomienda que las TVApps tengan un helper o service, salvo que sean TVApps con datos estáticos que no cambian a lo largo del tiempo o que utilicen datos o interfaces accesibles directamente a través de Internet.

* Una parte de **frontend,** que es la aplicación web que se ejecuta en el navegador HTML del descodificador híbrido, dentro del Contenedor TV. El front-end de las TVApps se desarrollará usando HTML y Javascript.

El Contenedor y las TVApps se ejecutan dentro del navegador del descodificador h.

Mediante un API Javascript el front-end de la TVApp accederá a los datos y recursos gráficos almacenados en el servidor de la Plataforma de TVApps. Algunos de estos datos y recursos serán estáticos y otros se obtendrán dinámicamente a través del helper de la TVApp.

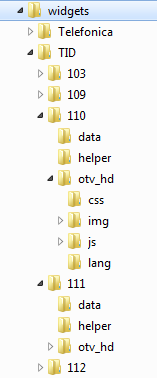


1. Esquema general del entorno de ejecución de TVApps
   * 1. Organización de las TVApps en el servidor

Dentro del repositorio de TVApps, las TVApps se organizan por tercero/desarrollador, es decir, existe una carpeta por cada tercero/desarrollador que contiene todos las TVApps asociadas al mismo.

Cada TVApp está contenida en una carpeta distinta cuyo nombre es el identificador de la TVApp.

La siguiente figura muestra la estructura de directorios del Repositorio de TVApps.



1. Estructura de directorios del repositorio de TVApps.

La carpeta de una TVApp contiene varios subdirectorios:

* **helper**: subdirectorio que contiene el código fuente PHP del helper o services de la TVApp. Si una TVApp no tiene helper o services, este directorio no existe.
* **data**: subdirectorio en donde se dejan los datos cacheados y adaptados para las TVApps por el helper, así como imágenes y otros recursos gráficos obtenidos de la fuente de datos. Si una TVApp no tiene helper, este directorio no existe.
* **mb\_hd**: subdirectorio en donde está el front-end (código ejecutable, datos fijos y recursos gráficos) de la TVApp. La TVApp se debe desarrollar usando HTML y Javascript. La estructura dentro de este subdirectorio es libre, si bien se recomienda separar los recursos gráficos, los ficheros JavaScript, los ficheros de estilo CSS y los textos multiidioma, en cuatro subdirectorios diferentes. Normalmente se suelen nombrar img, js, css y lang, respectivamente.

1. ENTORNO DE DESARROLLO DE LA PLATAFORMA DE TVApps

En este capítulo se describe las herramientas, documentación, API, etc. que estás a disposición de los desarrolladores de TVApps para crear nuevas aplicaciones interactivas para la TV.

* 1. Contenido del SDK

Para poder desarrollar TVApps se ofrece un SDK que está formado por los siguientes elementos:

* **Guía de desarrollo de TVApps:** documento actual.
* **Guía de estilo de TVApps:** documento que contiene recomendaciones sobre aspecto de diseño e interfaz de usuario a tener en cuenta en el desarrollo de TVApps.
* **Guía de referencia del API Javascript de TVApps:** contiene la descripción detallada, completa y actualizada de todas las funciones del API Javascript que los desarrolladores pueden utilizar para integrar las TVApps con el Contenedor y para hacer uso de algunas capacidades de la Plataforma de TVApps.
  1. Herramientas de desarrollo

Para poder desarrollar nuevas TVApps, se pone a disposición de los terceros/desarrolladores una Plataforma de TVApps accesible a través de Internet que forma parte del entorno de desarrollo y sobre la cual se programarán, probarán y gestionarán las TVApps.

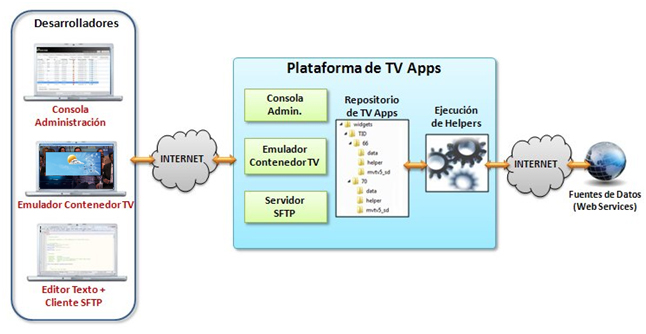
Esta Plataforma de TVApps ofrece los siguientes elementos principales:

* **Portal de desarrolladores de TVApps**: sitio Web disponible a través de Internet que ofrece toda la documentación del SDK de TVApps, un blog en el cual se publicarán periódicamente noticias relacionadas con el desarrollo de TVApps, un foro donde se pueden plantear dudas, un lista de preguntas y respuestas básicas y los enlaces de todas las herramientas del entorno de desarrollo de TVApps disponibles.
* **Consola de administración:** aplicación Web disponible a través de Internet que permite a un usuario con perfil de tercero gestionar diferentes aspectos de la Plataforma. Los elementos que los desarrolladores pueden gestionar/consultar a través de la consola son los siguientes:
  + TVApps propios
  + Helpers asociados a las TVApps propios
  + Usuarios finales de prueba
  + Datos asociados al propio tercero/desarrollador
  + Consulta de categorías de TVApps disponibles
* **Emulador del Contenedor TV para PC:** aplicación Web que permite a los desarrolladores ver el comportamiento de TVApps en el PC de una forma muy similar a cómo se comporta en el descodificador, pero sin necesidad de utilizar un descodificador. El Emulador es una máquina virtual que está disponible a través de un gestor de máquinas virtuales como el virtualBox (disponible a través de internet).
* **Servidor SFTP:** permiteacceder directamente al Repositorios de TVApps de la Plataforma de TVApps que forma el entorno de desarrollo. Cada tercero/desarrollador tendrá un usuario distinto para acceder al directorio del Repositorio de TVApps donde residen sus TVApps. Sobre este directorio, el tercero/desarrollador tiene permisos totales de lectura y escritura de archivos.

De esta forma, los usuarios pueden subir, editar o modificar nuevas versiones de sus TVApps, mientras desarrollan, sin necesidad de hacer un redespliegue cada vez que realizan un cambio en el mismo.

Se podrá usar cualquier cliente SFTP, pero hay que tener en cuenta que hay que configurarlo como “FTPS Explicito”. Por ejemplo se puede usar Filezilla, pudiéndose combinar con un editor de texto, por ejemplo Notepad++, de tal forma que al salvar un fichero se guarda automáticamente en el Repositorio de TVApps a través de SFTP.

Cuando un administrador da de alta un nuevo tercero en la Plataforma de TVApps, al tercero se le enviará un correo electrónico con todos los datos para acceder a estos tres elementos.



1. Entorno de desarrollo de la Plataforma de TVApps
   1. API del Contenedor TV

El API Javascript del contenedor para TVApps HTML define un conjunto de operaciones que comunican la TVApp con el Contenedor TV, y permite a los desarrolladores utilizar de forma muy sencilla ciertas capacidades de la Plataforma de TVApps y del descodificador híbrido.

A continuación se describe dicho API.

* + 1. Códigos de teclas

A continuación en enumeran y describen los identificadores de todas las teclas del mando a distancia que pueden ser utilizadas dentro de las TVApps.

* + - 1. Teclas de control

Estas son las teclas fundamentales para el manejo de cualquier TVApp. La mayoría de las operaciones dentro de una TVApp deben ser realizadas utilizando estas seis teclas.

**Pincho.KEY\_UP / Pincho.PC\_KEY\_UP**

Código del botón de la flecha de arriba del mando a distancia y el del teclado cuando se usa el emulador.

**Pincho.KEY\_DOWN / Pincho.PC\_KEY\_DOWN**

Código del botón de la flecha de abajo del mando a distancia y del teclado cuando se usa el emulador.

**Pincho.KEY\_LEFT / Pincho.PC\_KEY\_LEFT**

Código del botón de la flecha de izquierda del mando a distancia y del teclado cuando se usa el emulador.

**Pincho.KEY\_RIGHT / Pincho.PC\_KEY\_RIGHT**

Código del botón de la flecha de derecha del mando a distancia y del teclado cuando se usa el emulador.

**Pincho.KEY\_SELECT / Pincho.PC\_KEY\_SELECT**

Código del botón OK del mando a distancia y del teclado cuando se usa el emulador.

**Pincho.KEY\_BACK / Pincho.PC\_KEY\_BACK**

Código del botón BACK del mando a distancia y del teclado cuando se usa el emulador.

**Pincho.KEY\_NUM\_0 / Pincho.KEY\_NUM\_1 / Pincho.KEY\_NUM\_2 /**

**Pincho.KEY\_NUM\_3 / Pincho.KEY\_NUM\_4 / Pincho.KEY\_NUM\_5 /**

**Pincho.KEY\_NUM\_6 / Pincho.KEY\_NUM\_7 / Pincho.KEY\_NUM\_8 /**

**Pincho.KEY\_NUM\_9 / Pincho.PC\_KEY\_NUM\_0 / Pincho.PC\_KEY\_NUM\_1 /**

**Pincho.PC\_KEY\_NUM\_2 / Pincho.PC\_KEY\_NUM\_3 / Pincho.PC\_KEY\_NUM\_4 /**

**Pincho.PC\_KEY\_NUM\_5 / Pincho-PC\_KEY\_NUM\_6 / Pincho.PC\_KEY\_NUM\_7 /**

**Pincho.PC\_KEY\_NUM\_8 / Pincho.PC\_KEY\_NUM\_9**

Códigos de las teclas numéricas del mando a distancia y las correspondientes al teclado cuando se usa el emulador.

* + - 1. Teclas de control de reproducción

**Pincho.KEY\_PLAYPAUSE**

Código del botón PLAY-PAUSE del mando a distancia.

**Pincho.KEY\_STOP**

Código del botón STOP del mando a distancia.

**Pincho.KEY\_FWD**

Código del botón FORWARD del mando a distancia.

**Pincho.KEY\_RWD**

Código del botón REWIND del mando a distancia.

* + 1. Funciones generales de manejo de una TVApp

**Pincho.hook(object, eventName, func)**

Registra una función que será llamada cuándo el evento dado tiene lugar. Permite registrar los controladores de eventos para típicos clicks, movimientos del ratón... y también para cambios de nivel, características de actualización... (Pasando la TVApp como primer argumento).

**Pincho.unhook(object, eventName, func)**

Borra un controlador de evento.

**Pincho.closeMe()**

Cierra la TVApp y muestra el menu principal.

**Pincho.trigger(evt)**

Desencadena el evento **evt**.

**Pincho.log(msg)**

Escribe una línea en el fichero de log del emulador.

**Pincho.publish(topic)**

Permite publicar un evento basándose en el patrón publish/subscribe

**Pincho.subscribe(topic,context,func)**

Permite subscribirse a un evento para ser notificado cuando suceda.

**Pincho.unsubscribe(func)**

Permite borrar la subscripción a un evento.

**Pincho.isGancho()**

Indica si la TVApp ha sido lanzada desde el menú o desde la parte dinámica de la home.

**Pincho.init()**

Función a la que se llama cuando la aplicación se ha cargado y inicializa los datos asociados a la TVApp en el SDK.

**Pincho.show()**

Indica al SDK que la TVApp está lista para mostrar y se muestra a pantalla completa.

* + 1. Funciones para el manejo de propiedades del TVApp

**Pincho.properties = {}**

Objeto que actúa como un registro de las características y propiedades de la TVApp, es decir, permite asociar propiedades y valores a instancias de TVApps, según el paradigma clave=valor. Dichas propiedades y sus valores se guardarán en la Plataforma de TVApps y se mantendrán entre distintas sesiones de la misma instancia de TVApp.

Se entiende por instancia de una TVApp la asociación de una TVApp concreta a un usuario determinado. Cada ejecución de una instancia de una TVApp será una sesión distinta.

**Pincho.properties.all()**

Obtiene todas las propiedades de la instancia de la TVApp.

**Pincho.properties.get(prop, def)**

Obtiene la propiedad **prop** de la instancia de la TVApp. Si no existe, devuelve el valor por defecto **def**.

**Pincho.properties.set(prop, value)**

Guarda el valor **value** para la propiedad **prop** de la instancia de la TVApp.

Además existen las funciones **Pincho.loadProperties(callback)** que permite obtener en caliente las propiedades asociadas a la aplicación y al usuario que hay en la base de datos en un momento determinado y se mandan a la función de callback indicada y **Pincho.saveProperties(callback)** que permite guardar las propiedades en la base de datos en el momento. Una vez almacenados las properties se manda un 0 a la función de callback si el proceso se ha realizado correctamente.

* + 1. Otras funciones

**Pincho.LANGUAGE**

Obtiene el idioma que tiene seleccionado el usuario.

**Pincho.USERNAME**

Obtiene el nombre del usuario.

**Pincho.json.encode(data)**

Codifica una variable Javascript a formato JSON.

**Pincho.json.decode(json)**

Decodifica una cadena JSON en la variable Javascript que representa.

**Pincho.parseUrl(url)**

Comprueba que la url está bien formada y devuelve un json con el protocolo usado, dominio, puerto, argumentos y hash.

**Pincho.getApplicationKey()**

Obtiene la clave de aplicación.

**Pincho.include(file)**

Incluye un fichero Javascript (solo usado en entorno de desarrollo)

**Pincho.require(file)**

Sincroniza con el código externo Javascript para que sea cargado.

**Pincho.stats(action, widgetName, props)**

Manda estadísticas de la TVApp al Contenedor.

**Pincho.clone(ob)**

Hace una copia del objeto **ob**.

**Pincho.virtualKeyboard**(elemento, espacio, callback, passObj, doc)

Crea un teclado virtual sobre un elemento HTML. Se pasa el string a editar, indicando si es de tipo password o no. También se puede especificar la función de callback a la que llamar al salir.

**Pincho.reloadPreview**()

Notifica al Contenedor que la previsualización debe ser cargada.

**Pincho.goToExternalURL(url)**

Carga una url externa y finaliza el Contenedor.

**Pincho.encodeKey (stringToEncode)**

**Pincho.decodeKey(stringToDecode)**

Codifica/decodifica un string con AES en modo ECB de 128 bits con el algoritmo de Rijndael.

* 1. Pasos para el desarrollo de TVApps

Para poder desarrollar TVApps, el paso preliminar necesario consiste en dar de alta un nuevo tercero/desarrollador en la Plataforma de TVApps del entorno de desarrollo de TVApps. Esta operación la debe realizar un administrador de dicha plataforma y tras el alta se enviará al nuevo tercero/desarrollador un correo electrónico con todos los datos para acceder al entorno de desarrollo. Una vez se tenga acceso al portal de desarrolladores de TVApps se podrá acceder a toda la documentación disponible.

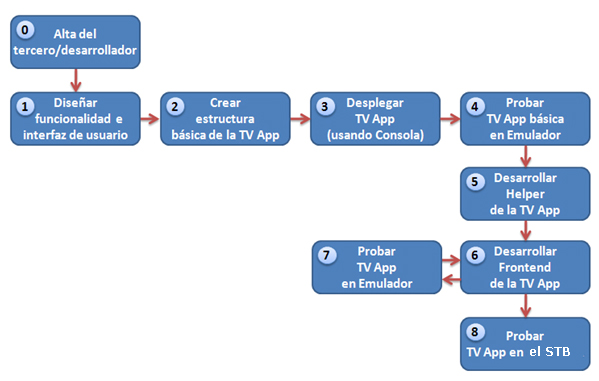
Una vez que el tercero/desarrollador está dado de alta en la Plataforma de TVApps, puede empezar a desarrollar una nueva TVApp siguiendo los siguientes pasos:

1. El primer paso consiste en **determinar la funcionalidad** que la nueva TVApp va a proporcionar. Una vez decidido esto, se debe **diseñar la interfaz gráfica de la TVApp**, es decir, cómo estará distribuida la funcionalidad en las diferentes pantallas de la TVApp, los elementos gráficos que se utilizarán, el aspecto gráfico general de la TVApp, etc.
2. El siguiente paso consiste en la **creación de la estructura básica** del frontend de la TVApp. Se usarán archivos JS, CSS y HTML. Se recomienda partir de la TVApp Base para crear esta primera versión de la TVApp. Dicha versión elemental se podrá editar en local.
3. Una vez creada la versión elemental de la TVApp, se deberá empaquetar en un fichero ZIP, de acuerdo a las normas establecidas, y **desplegar la TVApp** en la Plataforma de TVApps del entorno de desarrollo utilizando la **Consola de Administración**.
4. A continuación de podrá **probar la TVApp** utilizando el **Emulador del Contenedor**.
5. Seguidamente, es recomendable empezar a **desarrollar el helper de la TVApp** (backend), es decir, el proceso que obtiene datos externos de la Web y los adapta para la TVApp. Este paso sólo aplicará si la TVApp utiliza datos dinámicos obtenidos de Internet. Para el desarrollo de helpers la Plataforma de TVApps sólo soporta PHP.

Para programar el helper se deberá usar un editor de texto o un IDE integrado con el servidor SFTP de la Plataforma de TVApps.

El proceso de desarrollo del helper de una TVApp es un proceso iterativo que consistirá en programar partes del helper y probar el resultado utilizando un navegador web, hasta alcanzar el resultado final deseado.

1. Una vez se tenga desarrollado el helper, se deberá empezar a **desarrollar del frontend de la TVApps** usando los datos extraídos de las fuentes de datos externas. Para programar el frontend de la TVApps también se deberá usar un editor de texto o un IDE integrado con el servidor SFTP de la Plataforma de TVApps.
2. Según se avanza en el desarrollo de una TVApp, los resultados parciales se deberán **probar en el Emulador del Contenedor**. Los pasos 6 y 7 serán iterativos y se repetirán hasta conseguir el resultado final deseado.
3. El último paso consistirá en probar la TVApp en un descodificador híbrido, con el fin de verificar que la TVApp generada no sólo funciona bien en el Emulador, sino también en una caja híbrida real.



1. Pasos para el desarrollo de una TVApp
   1. Teclado virtual

**Pincho.showKeyboard(conf)**

Como parámetro esta llamada recibe un objeto conf que contiene las opciones de configuración del teclado.

* + 1. Configuración del teclado virtual

A continuación se enumeran los distintos parámetros (en cursiva la key del objeto de configuración)

* *document:* (Obligatorio) Referencia al document de la tvapp.
* *context:* (Obligatorio) Referencia del contexto de la tvapp.
* *formType:* (Obligatorio) Tipo de formulario asociado al teclado. Los valores posibles son:
  + *0:* *estándar*: Tiene un campo de texto multiline.
  + *1: search*: Un campo de texto de una sola línea.
  + *2:*  *login:* Dos campos de texto, un checkbox y un botón.
  + *3:* *comment:* Un textarea y puede contener elementos enfocables fuera del formulario y del teclado.
* *callbackData: (Obligatorio)* Función de callback a la que se mandan los datos introducidos en el formulario cuando se pulsa sobre el OK del teclado en los tipos 0, 1 y 3 o sobre el botón del formulario en el tipo 2.
* *formWidth:* ancho del formulario.
* *formHeight:* alto del formulario.

*(Si se indica el alto y el ancho del formulario, el teclado virtual se centra en el espacio disponible de la pantalla)*

* *posX:* posición X del teclado en la pantalla.
* *posY:* posición Y del teclado en la pantalla.

*(Si se indican los valores para posX y posY, el teclado virtual se muestra en esa posición)*

*Es obligatorio indicar o el alto/ancho del formulario o la posición x e y para que se muestre el teclado.*

* *headerText:* Es el texto que ha de aparecer en la cabecera del formulario. Es un valor opcional.
* *resources\_url:* En esta carpeta han de existir dos subcarpetas css e img donde se encuentran las imágenes del formulario (en img) y el fichero style.css (en css) donde se encuentran los estilos correspondientes al formulario. En el apartado estilos del teclado se adjuntan los nombres de las clases que corresponden a los elementos de los distintos tipos de formulario y teclado. Este parámetro es obligatorio si se quiere usar un teclado customizado.
* *mask : true/false.* Por defecto es true si no se indica nada. Indica si se ha de mostrar o no la máscara que oscurece el fondo al mostrar el teclado virtual.

*Configuración específica para el formulario de login:*

* *strings:* Es un objeto que contiene las cadenas de strings que se muestran en el formulario de login. Las keys son:
  + *remember\_password : ‘Recordar mi contraseña’*
  + *enter : Entrar*
  + *enter\_password : Pulse ‘OK’ para introducir la contraseña*
  + *enter\_user : Pulse ‘OK’ para introducir el usuario*
  + *error\_mail\_vacio : No has introducido el usuario*
  + *error\_mail\_ko : El e-mail introducido no es correcto*
  + *error\_pwd : No has introducido la contraseña*

*Si no se pasa este parámetro, las cadenas se rellenan con el valor por defecto. (Los valores por defecto son los que se muestran)*

* *focusOn :* 0 en el formulario, 1 en el teclado. Por defecto, el foco está en el formulario.
* *data :* Es un objeto que contiene los datos con los que se ha de mostrar el formulario. Las keys son:
  + *user*
  + *password*
  + *check*
* *En el resto de teclados se pueden introducir textos por defecto en el campo indicando en el campo data de la configuración un objeto con la key text. Por ejemplo, para introducir un texto por defecto en un teclado de búsqueda deberíamos introducir en el campo data lo siguiente: {‘text’:’Introduce tu búsqueda’}*
* *validateMail:* true/false. Indica si el valor de campo user es un e-mail o no. Se usa para que se valide si el email es correcto antes de mandar los datos a la función de callback. Entre los estilos del teclado ha de existir los estilos referentes al alert de error de datos mal introducidos. Por defecto el campo es true.

*Configuración específica para el formulario de comentarios:*

* *circMovHoriz:* true/false si se permite o no el movimiento circular en horizontal en el teclado. Por defecto es true.
* *circMovVert:* true/false si se permite o no el movimiento circular en vertical en el teclado. Por defecto es true.
* *callbackOutLimits:*función de callback que se llama cuando se ha llegado al borde del teclado por la izquierda cuando no hay movimiento circular (pendiente permitir la configuración del lado por el que hay otro elemento enfocable). Por defecto no hay porque el movimiento circular está habilitado.
  + 1. Ejemplos de configuración y de personalización
       1. Teclado de login con datos rellenados

var \_data = {'user':'myuser@gmail.es','password':'mypass','check':true};

var conf = {

'callbackData' : this.formularioLogin,

'document' : document,

'context' : this,

'formType' : 2,

'formWidth' : 550,

'formHeight' : 470,

'resources\_url' : 'keyboard\_resources',

'data':\_data,

'strings': {

'remember\_password' : strings.REMEMBER\_PASSWORD,

'enter' : strings.ENTER,

'enter\_password' : strings.ENTER\_PASSWORD,

'enter\_user':strings.ENTER\_USER,

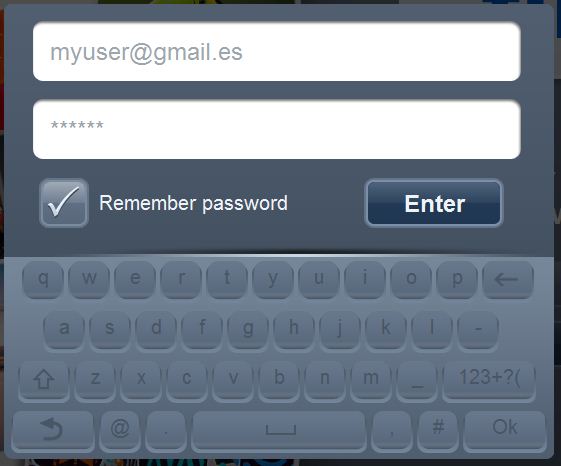
'error\_mail\_vacio':strings.ERROR\_MAIL\_VACIO,

'error\_mail\_ko':strings.ERROR\_MAIL\_KO,

'error\_pwd':strings.ERROR\_PWD

}

}



* + - 1. Teclado de búsqueda

var conf = {

'callbackData' : this.searchReq,

'document' : document,

'context' : this,

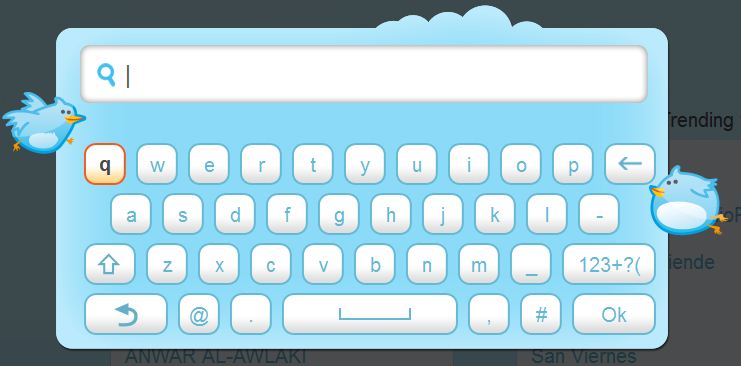
'formType' : 1,

'formWidth' : 726,

'formHeight' : 347,

'resources\_url' : 'keyboard\_resources'

};



* + - 1. Teclado de comentarios con elemento enfocable a la izquierda

var conf = {

'callbackData' : this.setCommentRequest,

'callbackOutLimits': this.focusComents,

'document' : document,

'context' : this,

'formType' : 3,

'formWidth' : 650,

'formHeight' : 470,

'resources\_url' : 'keyboard\_resources',

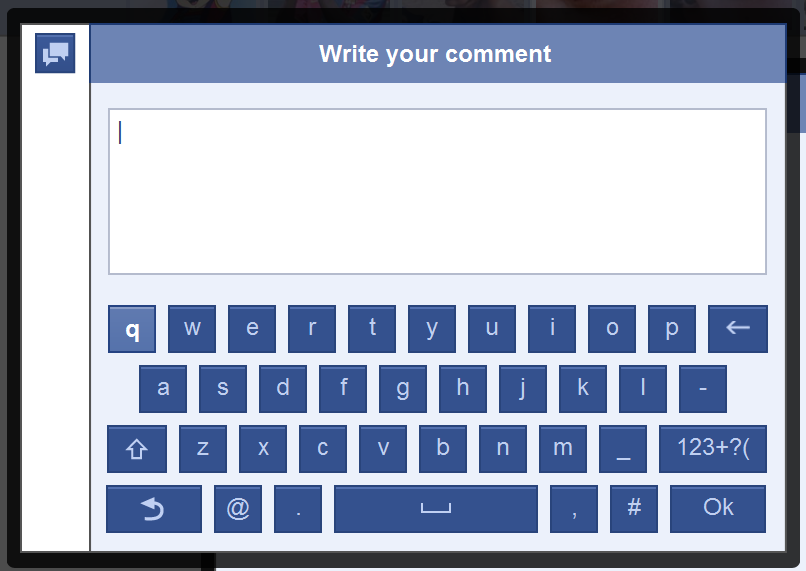
'circMovHoriz' : false,

'circMovVert' : true,

'headerText' : strings.HEADER\_COMMENTS,

'mask': false

};



* + - 1. Teclado estándar

var conf = {

'callbackData' : this.callbackKeyb,

'document' : document,

'context' : this,

'formType' : 0,

};

* + - 1. Teclado estándar customizado

var conf = {

'callbackData' : this.callbackKeyb,

'document' : document,

'context' : this,

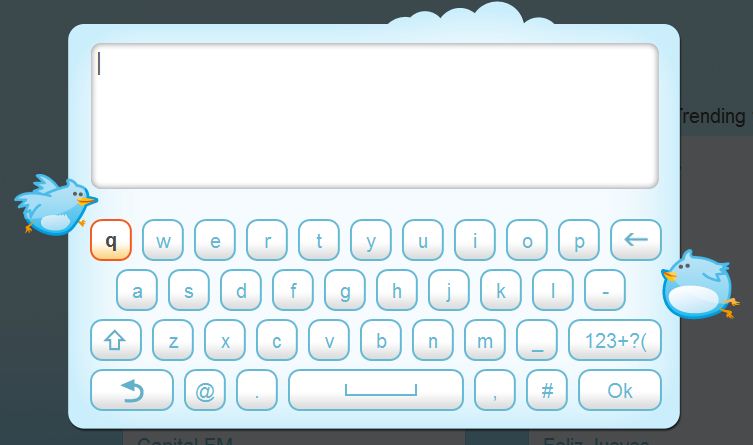
'formType' : 0,

'formWidth' : 726,

'formHeight' : 431,

'resources\_url' : 'keyboard\_resources'

};



* + 1. Personalización del teclado
       1. Estilos de las teclas

Las clases del teclado que hay que definir en la carpeta definida en *keyboard\_resources/css/style.css* son:

**.key1\_active, .key2\_active, .key3\_active, .key4\_active y .key5\_active**

Para definir el estilo de las teclas de tamaño 1, 2, 3, 4 y 5 cuando está el teclado activo, pero no están enfocadas.

**.key1\_inactive, .key2\_inactive, .key3\_inactive, .key4\_inactive y .key5\_inactive**

Para definir el estilo de las teclas de tamaño 1, 2, 3, 4 y 5 cuando está el teclado no está activo.

**.key1\_focused, .key2\_focused, .key3\_focused, .key4\_focused y .key5\_focused**

Para definir el estilo de las teclas de tamaño 1, 2, 3, 4 y 5 cuando está el teclado activo y la tecla tiene el foco.

**.spaceKey, .deleteKey,.shiftKey y.cancelKey**

Para definir la imagen y la posición de la imagen de espacio, back, shift y salir del teclado cuando está el teclado activo, pero no está la tecla enfocada.

**.spaceKey\_focused, .deleteKey\_focused,.shiftKey\_focused y.cancelKey\_focused**

Para definir la imagen y la posición de la imagen de espacio, back, shift y salir del teclado cuando está el teclado activo y está la tecla enfocada.

**.spaceKey\_inactive, .deleteKey\_inactive,.shiftKey\_inactive y.cancelKey\_inactive**

Para definir la imagen y la posición de la imagen de espacio, back, shift y salir del teclado cuando está el teclado no está activo.

**.keyText y .keyText\_focused**

Para definir el estilo del texto de las teclas cuando está y cuando no está la tecla enfocada.

**.row0, .row1, .row2y .row3**

Para definir la posición de cada una de las filas de teclas.

**#teclado\_wrapper**

Para definir el espacio destinado al teclado.

**.globo\_3\_left, .globo\_3\_center, globo\_3\_right, . .globo\_2\_left, .globo\_2\_center, globo\_2\_right, .globo\_4\_left, .globo\_4\_center, globo\_4\_right, .globo\_5\_left, .globo\_5\_center, globo\_5\_right, .globusKey\_wrapper, .globusKey\_wrapper\_2**

Para definir los estilos que corresponden a los globos de 2, 3, 4 y 5 letras con la fleche a la izquierda, en el centro o a la derecha y el estilo referente al contenedor de las teclas de dentro del globo. El estilo de las teclas de dentro del globo es el definido en .key1\_active y key1\_focused.

* + - 1. Estilos del formulario de login

**#loginForm**

Estilo general (tamaño, posición…)

**.loginOn, .loginOff**

Imagen de fondo del teclado con el foco en el formulario y con el foco en el teclado.

**#loginForm #form\_wrapper**

Estilo del espacio destinado al formulario

**#loginForm #header\_form, #loginForm #header\_form\_text**

Espacio de la cabecera y estilo del texto.

**#loginForm .input\_active, #loginForm .input\_focused, #loginForm .input\_inactive\_focus, #loginForm .input\_inactive**

Estilo de los inputs de texto en los diferentes estados (con foco o sin foco y active/inactive)

**#loginForm .input\_txt, #loginForm .input\_txt\_focused, #loginForm .input\_txt\_inactive**

Estilo del texto de los inputs de texto en los diferentes estados.

**#loginForm .checkBox\_active, #loginForm .checkBox\_inactive, #loginForm .checkBox\_focused**

Estilo del cuadro de checkbox en los diferentes estados.

**#loginForm .noCheck , #loginForm .check, #loginForm .check\_inactive**

Estilo del check interior del checkbox cuando no está marcado, está marcado y activo y está marcado e inactivo.

**#loginForm .checkBox\_text, #loginForm .checkBox\_text\_inactive**

Estilo del label del checkbox

**#loginForm .btnEnter\_active, #loginForm .btn\_enter\_txt\_active, #loginForm .btnEnter\_focused, #loginForm .btn\_enter\_txt\_focused, #loginForm .btnEnter\_inactive , #loginForm .btn\_enter\_txt\_inactive**

Estilos del botón y el texto del botón del formulario en todos sus estados.

**#loginForm #formulario\_footer**

Estilos del pie del formulario

**#loginForm #keyboard\_wrapper**

Estilos del espacio destinado al teclado

* + - 1. Estilos del formulario de búsqueda

**#searchForm**

Estilos generales del teclado de búsqueda (fondo, posición, ancho, alto…)

**#searchForm #form\_wrapper**

Estilo del contenedor del formulario

**#searchForm #header\_form, #searchForm #header\_form\_text**

Estilo del espacio destinado a la cabecera y al texto de la cabecera

**#searchForm #search\_input , #searchForm #search\_txt**

Estilo del input de texto y del texto.

**#searchForm #keyboard\_wrapper**

Espacio destinado al teclado

* + - 1. Estilos del teclado de comentarios

**#commentForm, .commentOn, .commentOff**

Estilos del fondo general del teclado e imágenes cuando el foco está en un lado o en el otro.

**#commentForm #form\_wrapper**

Estilo general del contenedor del formulario

**#commentForm #header\_form , #commentForm #header\_form\_text**

Estilo de la cabecera del formulario y del texto de la cabecera

**#commentForm .input, #commentForm .input\_txt**

Estilo del input de texto y del texto

**#commentForm #comments\_wrapper, #commentForm #comment\_icon**

Estilo del element externo enfocable

**#commentForm #keyboard\_wrapper**

Estilo general del contenedor del teclado

* 1. Creación de popups con mensaje

El SDK dispone de una función que facilita la creación de popups para mostrar mensajes.

**Pincho.showMessage(msg,mode,callback,timeout,type, conf)**

Los parámetros que recibe esta función son:

* *Msg:* Mensaje que se ha de mostrar
* *Mode:* Botones que tendrá el popup. Los posibles valores son:
  + -1: sin botón.
  + 0: con botón de Aceptar.
  + 1: con botón de Aceptar y botón de Cancelar
  + 2: Con botón Si y botón No.
  + 3: Con botón Continuar
* *Callback:* Función a la que se llamará al cerrar el popup. Esta función recibe un número que indica el modo del que se ha salido del popup. Los valores son:
  + *0:* se ha pulsado sobre el primer botón
  + *1:* se ha pulsado sobre el segundo botón
  + *-*1 : se ha salido pulsando BACK
  + *-*2: se ha salido al acabar el timeout establecido
  + *-4:* se ha eliminado pulsando cualquier tecla
* *Timeout:* Tiempo en ms tras el cual el mensaje desaparece. Si se indica -1, cualquier tecla hace desaparecer el popup.
* *Type:* Icono que se muestra. Los valores posibles son:
  + *0:* sin imagen
  + *1:* con la imagen indicada en infoIcon
  + *2:* con la imagen indicada en warningIcon
  + *3:* con la imagen indicada en errorIcon
* *Conf:* Objeto con configuración extra
  + ‘document’ : es la referencia al document de la tvapp (document)
  + ‘context’ : es la referencia al contexto de la tvapp (this)
  + ‘styleId’ : es el prefijo con el que se creará el popup y con el que se puede indicar distintos estilos de popup para una misma aplicación en el css. Si no se indica ningún valor, se coge el estilo de popup por defecto definido en el container.
    1. Personalización de popups

Las clases que hay que definir en la aplicación y que aplican a los popup mostrados son:

**#alrt\_dialog**

Estilo general del contenedor del popup

**#alrt\_top, #alrt\_bottom , #alrt\_dialogBg**

Clases que hacen referencia al fondo del popup. Se definen tres clases para dar la posibilidad de crear un fondo usando tres imágenes que correspondan a la parte superior, la parte central y la de abajo.

**.alrt\_dialogTxt**

Estilo del texto del mensaje

**.alrt\_dialogBtn, .alrt\_dialogBtn\_selected**

Estilos de los botones y del texto que contienen

**#alrt\_dialogButtons**

Estilo del contenedor de botones

**.alrt\_infoIcon , .alrt\_warningIcon ,.alrt\_errorIcon**

Estilos de las imágenes y posición del icono de cada uno de los distintos tipos.

Todas estas clases pueden repetirse y especificar al popup al que se refieren anteponiendo a las clases que se desee el prefijo #alert\_ + el string indicado en styleId. Por ejemplo:

#alert\_myCustomPopup .alrt\_infoIcon sería la imagen de un popup de tipo 1 creado con styleId ‘myCustomPopup’.

* 1. Creación de popups de cargando

El SDK ofrece una función que facilita la creación de mensajes de carga.

**Pincho.showLoading() o Pincho.loading.show(doc)**

*doc:* Referencia al DOMDocument de la aplicación (document)

Muestra un gif animado en el centro de la pantalla.

Para ocultar un popup de cargando cuando ya no es necesario hay que realizar una llamada a la función del SDK.

**Pincho.hideLoading() o Pincho.loading.hide()**

* + 1. Creación de popups de cargando animados

Adicionalmente se ofrece la posibilidad de mostrar un popup de cargando con un icono animado creado por la sucesión de varias imágenes estáticas.

**Pincho.loading.createAnimated(conf)**

Esta función recibe un objeto de configuración que contiene los siguientes parámetros:

* *Document:* referencia al DOMDocument de la aplicación (document)
* *styleId:* estilo que se aplica al popup de cargando
* *text:* Texto a mostrar en el popup
* *mask:* true/false indica si se muestra una máscara o no
* *opacityMask :* un valor entre 0 y 1 que indica el grado de opacidad de la máscara. Por defecto es 0.7.
* *timeoutAnimation:* tiempo en ms entre imagen e imagen de la animación
* *pauseAnimation:* tiempo en ms entre una animación y la siguiente
* *images:* array que contiene el path de las imágenes que forman la animación. El path es desde la raíz de la aplicación. En caso de usar una única imagen de sprite, el array contendrá las posiciones de las imágenes.
* *spriteImage:* path de la imagen de sprite
* *isSprite:* booleano que indica si se usa sprite o no
* *height:* alto de la imagen de loading
* *width:* ancho de la imagen de loading

Para ocultar un popup de cargando cuando ya no es necesario hay que realizar una llamada a la función del SDK.

**Pincho.loading.hideAnimated()**

* + 1. Personalización

**#loading\_**

Estilo general del loading (tamaño, posición)

**#loading\_ #alrt\_icon**

Estilo del icono de loading (imagen solo para loading no animado)

**#loading\_ #alrt\_text\_loading**

Estilo para el texto que se mostrará en el loading

* 1. Reproducción de vídeo

Si se desea reproducir video en streaming en la aplicación los pasos a seguir son:

* Inicializar el objeto player con **Pincho.player.init();**
* Determinar el tamaño de la ventana de reproducción y crear un contenedor padre para el reproductor.
* Mediante la llamada **Pincho.player.getVideoElement(*conf*)** obtendremos el elemento que hay que añadir al DOM. Recibe como parámetro un objeto de configuración con los siguientes datos:
  + - doc: document de la aplicación
    - width: número que indica el ancho de la ventana de reproducción (en píxeles)
    - height: número que indica el alto de la ventana de reproducción (en píxeles)
* Iniciar la reproducción del video con la llamada **Pincho.player.start(*uri, callbackFunc)*** donde uri es la url del streaming de video y callbackFunc es la función de callback a la que se llamará cuando se reciban los eventos de *onPlayOk y onPlayFailed.*
* Control de la reproducción de video gestionando los eventos de teclas.
  + 1. Workaround forward y rewind en streaming

Para dar soporte al forward y rewind en streaming, el workaround propuesto es habilitar en la barra de progreso un marcador que se mueva cada vez más rápido a lo largo de la barra de progreso del contenido mientras la tecla está pulsada (hacia adelante o hacia atrás) y en el momento en que se suelte la tecla, hacer un goToPosition del video a la posición correspondiente en la barra de progreso.

* + 1. Gestión de teclas

La aplicación en la que se vaya a reproducir video debe registrar (mediante Pincho.hook) las funciones gestoras de teclas. El fwd y rwd no están soportados en streaming de video, es por eso que se habilita la recepción del evento de keyUp para estas dos teclas para poder detectar pulsaciones largas de tecla y poder realizar el workaround propuesto como sustitución del fwd y rwd en streaming.

La aplicación debe registrar:

Pincho.hook(Pincho, ‘key’, onKeyDown);

Pincho.hook(Pincho,'keyUp',onKeyUp); (solo para fwd y rwd)

Las teclas de reproducción de video son:

* Pincho.KEY\_PLAYPAUSE - Pincho.PC\_KEY\_PLAYPAUSE
* Pincho.KEY\_STOP - Pincho.PC\_KEY\_STOP
* Pincho.KEY\_FWD - Pincho.PC\_KEY\_FWD
* Pincho.KEY\_RWD - Pincho.PC\_KEY\_RWD
* Pincho.KEY\_NEXT - Pincho.PC\_KEY\_NEXT
* Pincho.KEY\_PREV - Pincho.PC\_KEY\_PREV
* Pincho.KEY\_REC - Pincho.PC\_KEY\_REC
  + 1. Funciones de control de reproducción
* Pincho.player.playPause(callbackFunc)

Pone en pausa la reproducción si se está reproduciendo o la reanuda si está pausada. Recibe como parámetro una función de callback que se llamará cuando se reciben los eventos *onSpeedChanged, onSpeedChangeFailed, onSetSpeedOk y onSetSpeedError.*

* Pincho.player.stop(callbackFunc)

Para la reproducción de un contenido. Recibe como parámetro una función de callback que se llamará cuando se reciben los eventos de *onStopOk y onStopFailed.*

* Pincho.player.skip(ms\_to\_skip, callbackFunc)

Avanza o retrocede ms\_to\_skip la reproducción actual (ms\_to\_skip puede ser un número positivo para ir hacia adelante o un número negativo si se quiere retroceder). También recibe una función de callback como parámetro que se llamará cuando se reciben los eventos *onPositionChanged y onPositionChangeFailed.*

* Pincho.player.getPosition

Devuelve la posición en ms actual de la reproducción de video.

* Pincho.player.getDuration

Devuelve la duración en ms total del contenido en reproducción.

* Pincho.player.getPercentPosition

Devuelve la posición en porcentaje.

* Pincho.player.goToPosition(pos, callbackFunc)

Pone la reproducción en la posicion en ms indicada. Recibe dos parámetros, pos es la posición en ms en la que se desea situar la reproducción y callbackFunc es la función de callback a la que se llamará cuando se reciban los eventos *onPositionChanged y onPositionChangeFailed.*

* + 1. Eventos

Las funciones de callback reciben un objeto que contiene los siguientes datos:

* name es el nombre del evento que ha saltado
* event es el objeto que contiene información del evento
  + - 1. onSpeedChanged

Reporta un cambio en la velocidad de la reproducción.

* event.target : Objeto en el que el manejador de eventos fue registrado.
* event.SpeedChangedInfo.playSessionHandle
* event.SpeedChangedInfo.oldSpeed
* event.SpeedChangedInfo.newSpeed
  + - 1. onSpeedChangeFailed
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.SpeedChangeFailedInfo.playSessionHandle
* event.SpeedChangeFailedInfo.requestedSpeed
* event.SpeedChangeFailedInfo.reason:
  + -1: No soportado

(SPEED\_CHANGE\_FAILED\_REASON\_NOT\_SUPPORTED)

* + -2: Inactivo

(SPEED\_CHANGE\_FAILED\_REASON\_INACTIVE)

* + -3: PlaySessionHandle inválido

(SPEED\_CHANGE\_FAILED\_REASON\_INVALID\_PLAY\_SESSION\_HANDLE)

* + -4: Genérico

(SPEED\_CHANGE\_FAILED\_REASON\_GENERIC)

* + - 1. onSetSpeedOk
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.handle
  + - 1. onSetSpeedFailed
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.handle
* event.error.domain
* event.error.message
  + - 1. onStopOk
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.handle
  + - 1. onStopFailed
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.handle
* event.error.domain
* event.error.message
  + - 1. onPositionChanged
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.PositionChangedInfo.playSessionHandle
* event.PositionChangedInfo.newPosition
  + - 1. onPositionChangeFailed
* event.target: Objeto en el que el manejador de eventos fue registrado.
* event.PositionChangeFailedInfo.playSessionHandle
* event.PositionChangeFailedInfo.requestedPosition.whence
  + SEEK\_INVALID
  + SEEK\_SET: Desde el principio
  + SEEK\_END: Al final
  + SEEK\_CUR: En la posición actual
* event.PositionChangeFailedInfo.requestedPosition.type
  + POSITION\_TYPE\_INVALID
  + POSITION\_TYPE\_TIME\_BASED
* event.PositionChangeFailedInfo.requestedPosition.timePosition
* event.PositionChangeFailedInfo.reason
  + -1: No soportado

(POSITION\_CHANGE\_FAILED\_REASON\_NOT\_SUPPORTED)

* + -2: Inactivo

(POSITION\_CHANGE\_FAILED\_REASON\_INACTIVE)

* + -3: Fuera de límites

(POSITION\_CHANGE\_FAILED\_REASON\_OUT\_OF\_BOUNDS)

* + -4: Número inválido de parámetros

(POSITION\_CHANGE\_FAILED\_REASON\_INVALID\_NO\_OF\_IP\_PARAMS)

* + -5: PlaySessionHandle inválido

(POSITION\_CHANGE\_FAILED\_REASON\_INVALID\_PLAY\_SESSION\_HANDLE)

* + -6: Genérico

(POSITION\_CHANGE\_FAILED\_REASON\_GENERIC)

1. DESARROLLO DE HELPERS
   1. Conceptos Generales

Los helpers son procesos que se ejecutan periódicamente y que extraen datos de fuentes de datos externas (por ejemplo RSS).

Cuando se ejecuta un helper se generan ficheros (pueden ser uno solo o varios), los cuales contienen la información relevante que se ha extraído de la fuente de datos externa, codificada en formato JSON y con los datos estructurados de forma óptima para la TVApp.

La recomendación de utilizar ficheros de datos en formato JSON, se debe a que en los descodificadores es mucho más eficiente parsear este formato de datos que parsear XML. El parseo de un fichero XML para un descodificador implica un gran consumo de recursos, lo cual limita el rendimiento de la TVApp.

La ejecución de los helpers es controlada en todo momento por la Plataforma de TVApps, mediante una librería desarrollada en PHP, la cual se encarga de facilitar funciones para el desarrollo (parseo y edición de datos, manejo de directorios, almacenado de imágenes, etc.), y principalmente de llevar a cabo un control de los errores que puedan ocurrir a lo largo de la ejecución, notificando al servidor el resultado de ejecución, y dejando constancia de los errores producidos en un fichero local de logs.

Se recomienda el uso de helpers para guardar datos en local dentro de la Plataforma de TVApps, es decir, para cachear datos dentro de esta plataforma, y así conseguir un acceso más rápido a dichos datos.

Sólo se permite un helper por cada TVApp. Si bien, dicho helper puede incluir todas las funcionalidades que se consideren oportunas y podrá tener la estructura de clases y la división de dichas clases en los ficheros que se estimen necesarios.

* 1. Gestión de Helpers

La gestión de Helpers se lleva a cabo a través de la solapa Fuentes Datos de la Consola de Administración. Los usuarios de la consola con el perfil de tercero/desarrollador tienen permiso de acceso y consulta sobre dicha solapa.

En la solapa de Fuentes de datos, los terceros/desarrolladores tienen acceso a las siguientes funcionalidades:

* Consulta de la lista de todos los helpers propios
* Consulta de todos los datos asociados a un helper propio
* Consulta del log de actividad de un helper propio.

A continuación se describe las diferentes pantallas para llevar a cabo las acciones sobre la gestión de Fuentes de Datos.

* + 1. Lista de Helpers

En esta pantalla se muestran todos los helpers asociados a las TVApps del tercero/desarrollador, así como los datos más importantes de cada helper.

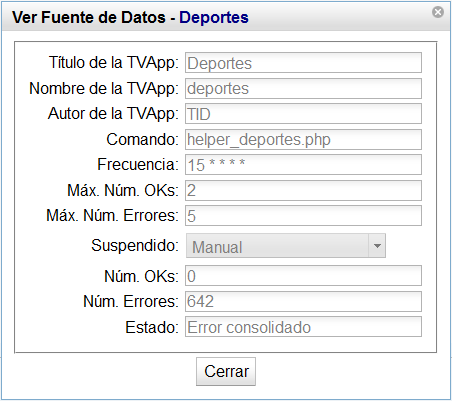
Esta pantalla incluye también un filtro sobre la lista de helpers y los botones para acceder a las demás pantallas (consulta de datos de un helper y consulta de log de actividad de un helper).



1. Lista de Helpers
   * 1. Consulta de datos de un Helper

Mediante esta opción se muestra un cuadro de diálogo con todos los datos de un Helper. Ninguno de los datos se podrá modificar.

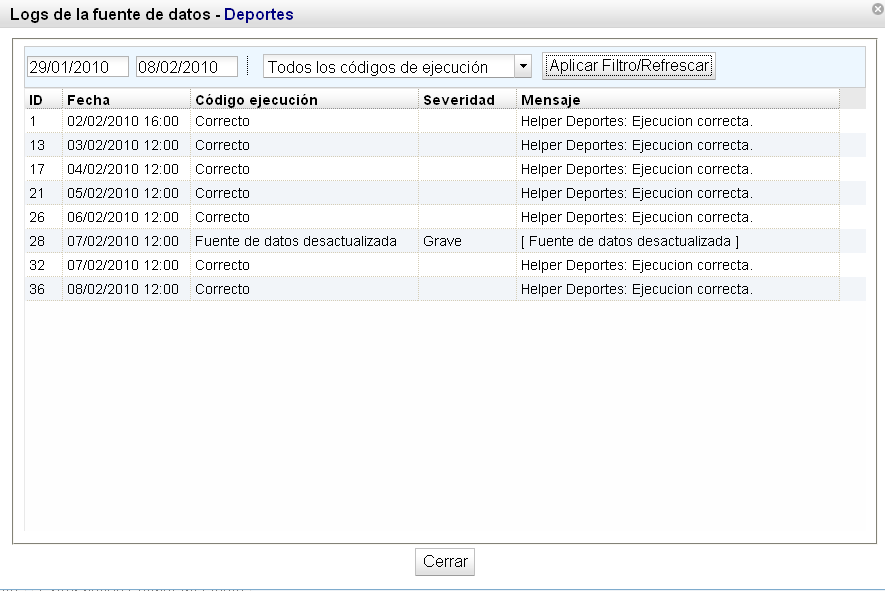
Se accede a esta funcionalidad a través del botón que aparece en la lista de helpers.



1. Consulta de datos de un Helper
   * 1. Consulta de log de un Helper

A través de esta opción se pueden consultar todos los logs de actividad de un Helper que cumplen un filtro previo. A través de este filtro el usuario puede elegir los logs de los que desea información.

Se accede a esta funcionalidad a través del botón que aparece en la lista de helpers.



1. Pantalla de Consulta de Log de un Helper
   1. Librería de Helpers

La Librería de Helpers define un conjunto de clases y funciones que permiten simplificar las diferentes etapas del ciclo de ejecución de un helper, y que permite a los desarrolladores definir, de manera muy sencilla, las fuentes de datos externas a acceder, el método de parseo de dichas fuentes, la codificación de los datos obtenidos a JSON, la escritura de la estructura JSON en un fichero JavaScript y la gestión de errores que puedan ocurrir durante todo este proceso.

La librería está desarrollada sobre PHP 5.3.2, y se encuentra ubicada en el siguiente directorio dentro de la plataforma de TVApps:

**/var/www/html/pinchos-pserver/libs/helpers/helpers.php**

La librería modela el ciclo de ejecución del helper a través de una clase denominada **Helper**, la cual tiene definida los siguientes atributos:

|  |  |
| --- | --- |
| **Atributo** | **Descripción** |
| **$\_url** | URL de la fuente de datos. |
| **$\_widgetName** | Nombre de la TVApp al que corresponde el helper. |
| **$\_outputFile** | Nombre del fichero JavaScript donde se guardará la información. |
| **$\_sourceType** | Tipo de fuente de datos externa: XML o HTML. |
| **$\_rootTag** | Nombre de la etiqueta XML que usaremos como raíz para el parseo de datos. |
| **$\_arrayParsed** | Contiene la información parseada de la fuente almacenada como un variable de tipo *array*. |
| **$\_jsonParsed** | Contiene la información codificada a JSON, almacenada en esta variable como un dato de tipo *string*. |
| **$\_sourceDelay** | Define el máximo intervalo de desactualización que puede tolerar la fuente de datos (en horas). |
| **$\_utils** | Contiene una instancia de la clase helperUtils. |
| **$\_httpRequestURL** | URL a la cual se enviarán las notificaciones al servidor a través de un POST. |

Para crear una nueva instancia del objeto Helper se requieren inicialmente tres parámetros obligatorios:

**$helper = new helper($\_widgetName, $\_url, $\_outputFile);**

Para definir en cualquier momento algún atributo, modificar su valor o leer su valor actual, la librería aplica el encapsulamiento de datos, de modo que todos sus atributos son accesibles a través de los métodos “set” y “get” correspondientes, a excepción de:

* Los atributos privados de la clase Helper.
* El atributo público **\_utils,** el cual es accesible directamente.

|  |  |  |
| --- | --- | --- |
| **Atributo** | **Set** | **Get** |
| **$\_url** | $helper->setUrl($src) | $helper->getUrl () |
| **$\_widgetName** | $helper->setWidgetName($wdg) | $helper->getWidgetName() |
| **$\_outputFile** | $helper->setOutputFile($name) | $helper->getOutputFile() |
| **$\_sourceType** | $helper->setSourceType($type) | $helper->getSourceType () |
| **$\_rootTag** | $helper->setRootTag($tag) | $helper-> getRootTag () |
| **$\_arrayParsed** | $helper->setArrayParsed($array) | $helper->getArrayParsed() |
| **$\_jsonParsed** | $helper->setJsonParsed($json) | $helper->getJsonParsed() |
| **$\_sourceDelay** | $helper->setSourceDelay($d) | $helper->getSourceDelay() |

Con respecto al tipo de fuente de datos externa, inicialmente el atributo ***$\_sourceType*** tiene como valor “XML”, pero en caso de utilizar una fuente de datos HTML, habrá que modificar su valor por “HTML”.

Para el caso de una fuente de datos externos XML, es necesario definir el nombre de la etiqueta XML a partir de la cual se comienza el parseo de la fuente de datos:

**$helper->setRootTag(“<**nombre\_etiqueta**>”)**

Una vez definidos los atributos principales, procedemos a ejecutar el parseo de la fuente a través de la llamada al método **parse** de la clase Helper**.**

* + 1. Método parse

En el caso de que nos encontremos con una fuente de datos XML, el método **parse** es el que se encarga de iniciar el parseo de la fuente.

Para ello, primero se comprueba la disponibilidad de la fuente accediendo a la URL. Luego se verifica la integridad de la fuente de datos (XML correctamente formado).

Una vez cargada la fuente, se obtiene el nodo raíz a partir del cual se iniciará el parseo de dicha fuente. Tener en cuenta que al definir el atributo \_rootTag, si **la etiqueta se encuentra una sola vez** dentro de la fuente XML, se devolverá **un solo nodo XML** el cual contenga todo el contenido del XML a partir de ese nodo; en cambio **si la etiqueta se encuentra repetida varias veces**, devolverá un **array de nodos**, cada uno de los cuales tendrá como raíz a $\_rootTag.

De este modo, cada elemento del array obtenido se parseará de acuerdo a una función de callback cuyo nombre será pasado como segundo parámetro del método parse.

La función parse siempre genera un array asociativo conteniendo los elementos que se han extraído del nodo analizado y retorna finalmente el código de ejecución de esta acción.

Adicionalmente, una vez se ha parseado un nodo, se dispone de una segunda función de callback opcional, la cual se ejecuta cuando el nodo parseado por el primer callback ya ha sido almacenado en memoria en el atributo \_arrayParsed.

Por tanto, una llamada a la función parse quedaría definida de la siguiente manera:

**$helper->parse($parser\_callback, $post\_callback=null)**

Donde ***parser\_callback*** contiene la lógica para parsear cada uno de los nodos obtenidos, y ***post\_callback*** contiene código a ejecutarse una vez se tiene el nodo parseado. Como el segundo callback es opcional, si no se requiere su uso, simplemente debe omitirse.

* + 1. Códigos de Ejecución

Debido a que la librería incorpora un control de los errores que puedan ocurrir durante todo el proceso, cada vez que se ejecuta algún método de la librería, se devuelve un código de ejecución que refiere el estado resultante de la ejecución de una acción en particular.

Para ello la librería define un array de resultados en la variable global **$RESPONSES**. Los códigos de ejecución se pueden obtener son los siguientes:

|  |  |
| --- | --- |
| **Código** | **Descripción** |
| **100** | OK (Ejecución Correcta). |
| **101** | La URL no se encuentra disponible. |
| **102** | Etiqueta XML inicial incorrecta. |
| **103** | Error codificando JSON: json no válido. |
| **104** | Error codificando JSON: array no válido. |
| **105** | Error generando fichero: datos incorrectos o nulos. |
| **106** | Error generando fichero: Sintaxis JSON incorrecta. |
| **107** | Etiquetas de apertura y cierre no coinciden. |
| **108** | Fuente de datos desactualizada. |

* + 1. Control de Errores

En cuanto se produce un error, si la librería determina que este error impide continuar con la ejecución del helper (URL no válida, fuente RSS errónea, etc.) automáticamente se suspende la ejecución, manteniendo el fichero JavaScript resultante de la última ejecución correcta.

Luego se envía una notificación al servidor de la Plataforma TVApps indicando el código resultante de la ejecución, la severidad y un mensaje opcional con datos adicionales del error.

Una ejecución correcta del helper devolverá un código **100**. De lo contrario, se devolverá el código del error producido de acuerdo a la tabla anterior que contiene los códigos de ejecución.

Hay que tener en cuenta que si bien la librería envía automáticamente notificaciones al servidor cuando se produce algún error, **ésta no envía una notificación al servidor cuando la ejecución del helper ha sido correcta**.

Para poder indicar una ejecución satisfactoria del helper, debemos llamar al método

**$helper->notify\_correct\_execution()**

El cual se encargará de enviar la notificación al servidor de la ejecución correcta, y generar una entrada en el fichero de logs indicando el resultado de esta ejecución.

* + 1. Funciones adicionales de la librería

La librería de Helpers pone a disposición del desarrollador, además de las funciones generales para el parseo de fuentes, un conjunto de funciones adicionales, definidas en la clase helperUtils. De modo que dentro del atributo **$helpers->\_utils** se tiene creada una instancia a dicha clase helperUtils, y para utilizar estas funciones se deben referenciar de la siguiente manera: **$helpers->\_utils->nombre\_funcion()**

Las funciones definidas en la nueva clase helperUtils, en *utils.php* son:

* **public function printLog($txt)**

Permite imprimir logs en pantalla (optimizado para mostrar resultados en una consola linux).

* **public function getResponseMsg($code, $mode=false)**

Obtiene el mensaje de error a partir del código de ejecución devuelto.

* **public function setTextColor($txt, $color)**

Modifica el color del texto de un log. ***$color*** puede ser ‘r’ (rojo), ‘g’ (verde), ‘b’ (azul), etc.

* **public function file\_list($path, $type=false)**

Devuelve la lista de ficheros contenidos en un directorio (***$path***). ***$type*** permite filtrar un solo tipo de ficheros (jpg, png, etc).

* **public function replace\_chars($find, $rep, $data)**

Reemplaza todas las ocurrencias de ***$rep***  en la cadena de texto ***$find*** por el texto ***$data****.*

* **public function sort\_string\_asc($data, $hive)**

Ordena el array ***$data*** de forma ascendente.

* **public function array\_msort($array, $cols)**

Adaptación de la función array\_multisort (ordenamiento de datos) para PHP 5.3

* **public function sortResults(array $data, $name, $order, $mode)**

Método de ordenación del array ***$data***.

* **public function removeHTMLtags($str)**

Remueve todos los tags HTML de una cadena de texto (a excepción de <p> y <br>, que sí son soportados por todos los descodificadores) y todo el código javascript que pueda haberse incluido.

* **public function strip\_cdata($string)**

Extrae todo el texto que se encuentra dentro de la etiqueta CDATA.

* **public function strip\_only($str, $tags)**

Remueve de la cadena ***$str*** todos los tags HTML indicados en ***$tags*** (separados por comas).

* **public function saveImage($url, $path)**

Guarda en un directorio local ***$path*** una imagen ubicada en la dirección externa ***$url***.

* **public function resizeImg($name, $file, $width, $height)**

Redimensiona la imagen ubicada en ***$name*** y la guarda con el nombre ***$file***, de acuerdo al ancho y alto indicado, sin deformarla.

* **public function dateplus($date, $dd=0, $mm=0, $yy=0, $hh=0, $mn=0, $ss=0)**

Permite añadir o restar a una fecha ***$date*** años, meses, días, horas, minutos y segundos.

* **public function deleteDirectory($dir)**

Elimina el directorio indicado en ***$dir***.

Las siguientes funciones son métodos públicos de la clase Helper:

* **public function loadXML($path)**

Permite cargar una fuente de datos externa XML de la URL indicada en ***$path***.

* **public function loadHTML($path)**

Permite cargar una fuente de datos externa HTML de la URL indicada en ***$path***.

* **public function notify\_correct\_execution()**

Notifica al servidor que el Helper se ha ejecutado correctamente.

Los siguientes métodos también son métodos púbicos de la clase helper y son los que realizan el control de errores (devuelven un código de ejecución):

* **public function post\_request($data)**

Permite enviar una notificación al servidor indicando el resultado de ejecución de una tarea.

* **public function validate\_delay($fecha\_fuente)**

Verifica que la fuente de datos esté actualizada. Recibe como parámetro la fecha de la fuente (en formato Date) y el criterio a tomar en cuenta para determinar si está actualizada o no depende de la definición del atributo del helper $\_sourceDelay.

* **public function parse($parser\_callback, $post\_callback=null)**

Inicia el parseo de la fuente. Recibe como parámetros una función callback que se encarga del parseo de cada nodo de la fuente (***$parser\_callback***) y otra función adicional de callback que permite realizar alguna acción una vez se ha terminado de parsear un nodo (***$post\_callback***)**.**

* **public function JSON\_encode()**

Codifica el contenido del atributo **$\_arrayParsed** del objeto helper a JSON, y guarda el resultado codificado en el atributo **$\_jsonParsed.**

* **public function open\_file()**

Crea un fichero y retorna su file descriptor. El nombre del fichero creado será el definido en el atributo **$this->\_outputFile** del objeto helper.

* **public function close\_file($fd)**

Cierra el fichero definido en **$this->\_outputFile.**

* **public function write\_file($varName)**

Escribe en un fichero JavaScript el JSON contenido en **$\_jsonParsed**. ***$varName*** indicará el nombre de la variable a la que se asignará el JSON en el contexto de JavaScript.

* 1. Recomendaciones de Desarrollo

En este apartado veremos algunas recomendaciones útiles al momento de desarrollar un helper.

En primer lugar, se recomienda definir los parámetros principales de configuración del helper como constantes, como el caso de la URL, el nombre de la TVApp al que corresponde el helper, el nombre del fichero de salida JavaScript, etc.

Los pasos a seguir en el desarrollo de un Helper deben tener la siguiente estructura:

* Crea una instancia del objeto helper.

***$helper = new helper(\_NAME, \_URL, \_FILE)***

* Si la fuente de datos externa es HTML, redefinimos este parámetro que por defecto tiene como valor “XML”.

***$helper->setSourceType(“HTML”)***

* Si utilizamos una fuente de datos XML, indicamos la etiqueta inicial a partir del cual empezaremos a parsear.

***$helper->setRootTag($tag)***

|  |  |
| --- | --- |
| <?xml version="1.0" encoding="UTF-8"?>  <noticias>  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia>  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia>  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia>  </noticias> | **Si elegimos como $root\_tag la etiqueta <noticias>, obtendremos como resultado un nodo con un solo elemento que contiene la estructura:**  <noticias>  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia>  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia>  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia>  </noticias> |
| **Si elegimos como $root\_tag la etiqueta <noticia>, obtendremos como resultado un array con tres nodos, cada uno de ellos con la siguiente estructura:**  <noticia>  <titulo></titulo>  <texto ></texto>  </noticia> |

* Cuando ya tenemos todos los parámetros anteriores definidos, podemos hacer uso de la función parse. Esta función se encarga del parseo del contenido de cada elemento de la etiqueta del XML definida como **$root\_tag.**

***$result = $helper->parse( “helper\_parser”)***

La función de callback debe estar definida de la siguiente manera:

***function helper\_parser($node, $array){***

***/\**** *Código destinado al parseo de un nodo* ***\*/***

***}***

***$node***  contiene la estructura que se define dentro la etiqueta XML señalada como root\_tag.

***$array***  contiene todos los nodos anteriores que ya han sido parseado de la fuente XML.

Si en algún momento necesitamos obtener en qué iteración nos encontramos (esto es, qué número de nodo estamos por parsear en el callback), bastará con obtener la longitud del parámetro *$array.*

En $result se almacena el código de ejecución del método parse. Si es igual a 100, la ejecución ha sido correcta. En caso contrario, la librería ya habrá efectuado por su parte el control de errores, y reportará en $result el código del error obtenido. De manera que es recomendable realizar una validación de este código de ejecución, para determinar si el helper debe seguir ejecutándose, o si debemos terminarlo y mostrar un mensaje de error en pantalla.

* Codifica a JSON el resultado del parseo de la fuente y la almacena en **$\_jsonParsed**.

***$helper->JSON\_encode()***

* Escribe el contenido del JSON en el fichero JavaScript cuyo nombre fue indicado al crear el objeto helper.

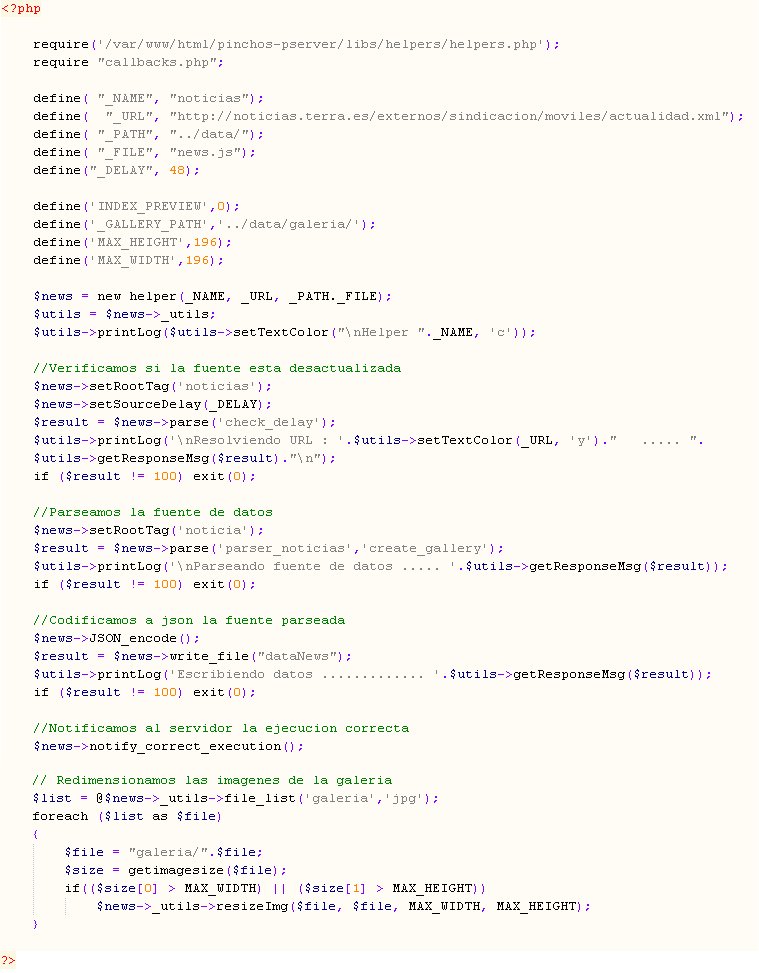
***$helper->write\_file($varName)***

* Una vez se ha generado el fichero de salida, si con ello ha concluido la ejecución del helper, es recomendable notificar al servidor que el Helper se ha ejecutado correctamente. Para ello, se debe utilizar la función de la librería:

***$helper->notify\_correct\_execution();***

* 1. Ejemplo de desarrollo de Helper

A continuación se muestra un ejemplo sencillo de implementación de un helper:



1. DESARROLLO DE TVAPPS PARA EL DESCODIFICADOR HÍBRIDO

A lo largo de este documento se ha presentado varias veces el concepto de TVApps y se ha detallado las funcionalidades generales que ofrece. En este capítulo se detalla cómo gestionar TVApps a través de la consola de administración, los formatos y estilos de las TVApps, las limitaciones técnicas y cómo desarrollar el front-end de una TVApp.

* 1. Gestión de TVApps

La gestión de TVApps se lleva a cabo a través de la solapa TVApps de la Consola de Administración. Los usuarios de la consola con el perfil de terceros/desarrollador tienen permiso de consulta y modificación sobre las TVApps propias.

Las funcionalidades disponibles son las siguientes:

* Consulta de la lista de todas las TVApps propias.
* Consulta y modificación de los datos de las TVApps propias.
* Consulta y modificación de los grupos de las TVApps propias.
* Gestión de modelos de descodificador soportados por las TVApps propias.
* Gestión de las plataformas activas para las TVApps propias.
* Despliegue y redespliegue de una TVApp propia.
* Eliminación de una TVApp propia.

A continuación se describe las diferentes pantallas para llevar a cabo las acciones sobre las TVApps.

* + 1. Lista de TVApps

En esta pantalla se muestran todas las TVApps del tercero/desarrollador, así como los datos más importantes de cada TVApp.

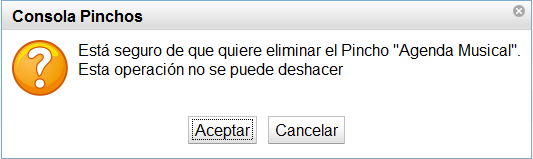
Esta pantalla incluye también un filtro sobre la lista de TVApps y los botones para acceder a las demás pantallas (borrado de una TVApp, redespliegue de una TVApp, modificación de datos de una TVApp, etc.).



1. Pantalla de Lista de TVApps
   * 1. Eliminación de una TVApp

Mediante esta opción el usuario puede eliminar una TVApp, previa confirmación de la operación.

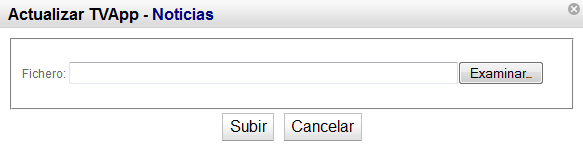
Se accede a esta funcionalidad a través del botón que aparece en la lista de TVApps.



1. Eliminación de una TVApp
   * 1. Redespliegue de una TVApp

Mediante esta opción se muestra un cuadro de diálogo que permite seleccionar un fichero zip que contiene el código de una TVApp para actualizarla.

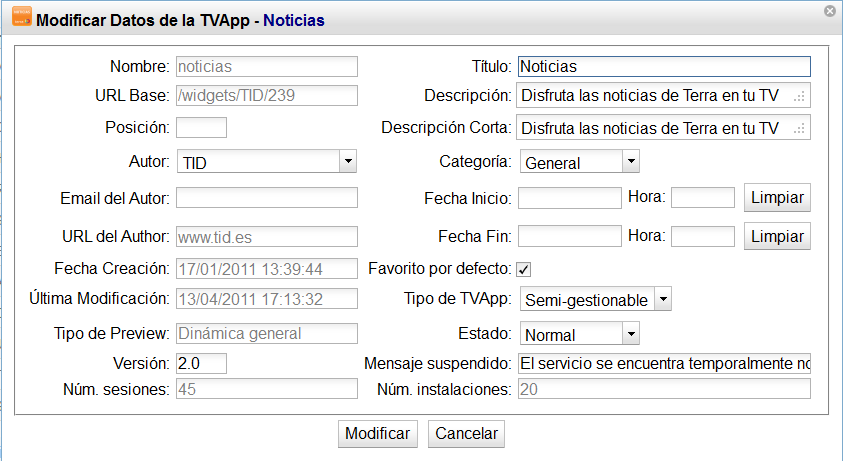
Se accede a esta funcionalidad a través del botón que aparece en la lista de TVApps.



1. Redespliegue de una TVApp
   * 1. Consulta/modificación de datos de una TVApp

Mediante esta opción se muestra un cuadro de diálogo con todos los datos de una TVApp. Algunos de estos datos se pueden modificar.

Se accede a esta funcionalidad a través del botón que aparece en la lista de TVApps.



1. Consulta/modificación de datos de un TVApp
   * 1. Consulta/modificación de modelos de STB soportados por un TVApp

Mediante esta opción se muestra un cuadro de diálogo que contiene dos listas, una de ella muestra los STB soportados por una TVApp, y en la otra los que no están soportados. El usuario podrá mover elementos de una lista a otra.

Se accede a esta funcionalidad a través del botón que aparece en la lista de TVApps.

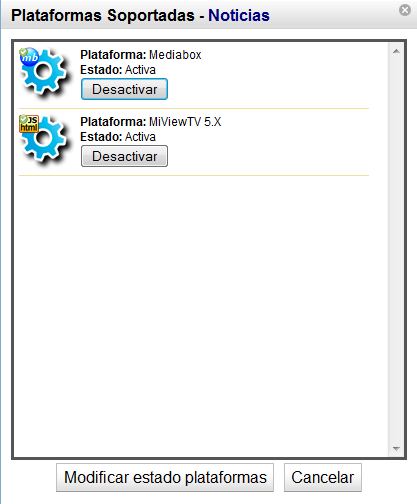


1. Consulta/modificación de modelos STB soportados por una TVApp
   * 1. Consulta/modificación de plataformas activas para una TVApp

La plataforma de TVApps permite soportar varias plataformas de TV al mismo tiempo (IPTV, Satélite, Mediabox, etc.).

Mediante esta opción se muestra un cuadro de diálogo que contiene una lista con todas las plataformas en las cuales la TVApp está soportada y un indicador que muestra si la TVApp está activa en cada una de estas plataformas o no. El usuario podrá activar o desactivar plataformas.

Se accede a esta funcionalidad a través del botón que aparece en la lista de TVApps.



1. Consulta/modificación de plataformas activas para una TVApp

En general son será necesario hacer ningún cambio de configuración en este cuadro de diálogo mientras se está desarrollando una TVApp para el descodificador híbrido (plataforma OpenTV)

* + 1. Consulta/modificación de grupos para una TVApp

Mediante esta opción se muestra un cuadro de diálogo que contiene dos listas, una de ella muestra los grupos asociados a una TVApp, y en la otra los que no están asociados a esa TVApp. El usuario podrá mover elementos de una lista a otra.

Se accede a esta funcionalidad a través del botón  que aparece en la lista de TVApps.



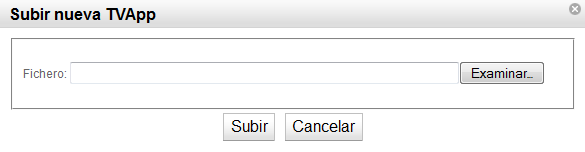
1. Consulta/modificación de grupos de una TVApp

En general son será necesario hacer ningún cambio de configuración en este cuadro de diálogo mientras se está desarrollando una TVApp para el descodificador híbrido.

* + 1. Despliegue de una TVApp nueva

Mediante esta opción se muestra un cuadro de diálogo que permite seleccionar un fichero zip que contiene el código de una nueva TVApp.

Se accede a esta funcionalidad a través del botón “Subir nueva TVApp” que aparece en el pie de la página.



1. Despliegue de una TVApp nueva
   1. Limitaciones técnicas

Actualmente, el navegador Web que existe en los descodificadores híbridos es un Ekioh Web Browser.

El diseño de las TVApps debe restringir el uso de características básicas comunes del navegador Web.

Dichas características son:

* HTTP 1.1 [RFC2068, RF 2616], con soporte para ciertos tipos MIME.

|  |  |
| --- | --- |
| Nombre | Tipos MIME |
| Text | text/plain |
| HTML | text/html |
| GIF Image | image/gif |
| JPEG Image | image/jpeg |
| PNG Image | image/png |

* HTTPS [RFC2818 http sobre TLS]
* HTML 4.01 [RFC2616] (GIF animados, JPEG, PNG, GIF)
* CSS1 [RFC2381], CSS2 y CSS3 (parcialmente [a1], soportadas por Mozilla 1.5)
* DOM 1, DOM 2 y DOM 3 (parcialmente, soportadas por Mozilla 1.5)
* JavaScript 1.5
* XML 1.0 with CSS
* Cookies (no permanentes, ya que el sistema no tiene medio de almacenamiento local).

Para contenidos multimedia, los formatos soportados son:

* Transporte: HTTP.
* Flujo de transporte de video: MPEG-2 y MPEG-4.
* Formatos de codificación de vídeo: AVI, DivX, ASF, WMV, MKV y 3GP.
* Formatos de codificación de audio: MP3, WMA, AC3 y AAC.

Otra característica importante es que el usuario debe moverse por los diferentes elementos que son capaces de recibir el ‘foco’ (selección resaltada) con las cuatro teclas de flechas (izquierda, derecha, arriba y abajo) y no con el ratón. Esto significa que hay menos libertad de movimiento, ya que para llegar a un elemento el usuario debe navegar por los elementos intermedios en la dirección que los une, el usuario no puede saltar directamente desde un elemento a otro.

* 1. Formato de una TVApp

En el descodificador híbrido se muestra inicialmente un menú con las aplicaciones asociadas al usuario y un icono que las representa.

Usando las teclas de flecha izquierda y derecha se puede cambiar la TVApp seleccionada y pulsando la tecla de selección (OK) se accede a la TVApp seleccionada.



1. Pantalla de menú principal de TVApps

Las TVApps también son accesibles desde la Home del híbrido, estando la opción de TVApps enfocada, pulsando flecha arriba, se pueden ver las aplicaciones recomendadas y se puede acceder pulsando OK sobre ellas. Cuando se accede a una TVApp de este modo y se consulta la propiedad isGancho del SDK (Pincho.isGancho()), este devolverá true.



1. Pantalla de TVApps en parte dinámica de la home
   1. Recomendaciones de estilo

El objetivo de este apartado es proporcionar al desarrollador algunas recomendaciones de estilo con el fin de que se puedan crear TVApps que tengan el estilo general de las TVApps y la información que muestren se pueda leer correctamente en los televisores.

La fuente soportada por el descodificador es “Arial” y ha de usarse un tamaño suficiente para ser leída en el televisor.

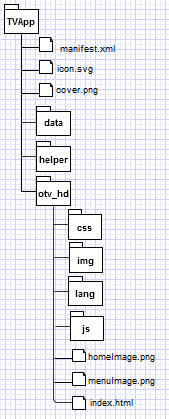
Para el uso de imágenes se recomienda los formatos JPEG y PNG con una transparencia de 8 bits. Se recomienda que las imágenes no superen un tamaño máximo de 100 Kbytes.

Se recomienda usar un fondo en la TVApp que permita la visualización correcta del contenido.

En el desarrollo de aplicaciones para TV han de respetarse los márgenes de seguridad para que el contenido se muestre completamente en los televisores sean del tipo que sean. Los márgenes recomendados son 36px de margen horizontal y 64px como margen vertical.

* 1. Estructura de una TVApp

Una TVApp es una aplicación web formada por un conjunto de fichero HTML, JavaScript, CSS y recursos gráficos. Para desplegar una TVApp, todos estos ficheros se deben empaquetar en un único fichero ZIP. La estructura de una TVApp es la siguiente:

****

1. Estructura de una TVApp

* **Fichero manifest.xml** (obligatorio)**:** contiene metadatos asociados a la TVApp y que describe la estructura de la TVApp de cara al usuario.
* **Fichero icon.svg** (fichero de imagen vectorial) (obligatorio) que será utilizado para generar las distintas versiones (tamaños) del icono de la TVApp. El icono debe ser cuadrado y se recomienda que tenga una resolución minima de 200x200 píxeles.
* **Fichero cover.png** (opcional)**:** contiene la imagen que se utilizará en el catálogo de TVApps, en la pantalla de detalle de la TVApp. Debe estar en formato PNG y con una resolución de 200x200 píxeles. Si este fichero no se incluye se utilizará el icono en tamaño 200x200 para representar la TVApp en el catálogo.
* **Directorio /helper** (opcional) contendrá el helper de la TVApp, es decir, uno o varios ficheros PHP que se ejecutan periódicamente y que acceden a las fuentes de datos asociadas a la TVApp.
* **Directorio /data** (opcional) contendrá los datos obtenidos de las fuentes de datos y cacheados en el servidor de TVApps.
* **Directorio** **/otv\_hd** con el código ejecutable de la TVApp para el descodificador híbrido, basado en tecnología web estándar. Este directorio debe contener:
  + Un fichero **index.html**: página web que contiene la estructura de la TVApp.
  + Carpetas **css, img, js, lang (opcionales)**: se recomienda que los archivos externos estén en directorios separados por tipo.
  + Un fichero **homeImage.png** (obligatorio)**:** contiene la imagen que se utilizará en la pantalla principal del descodificador híbrido para representar las TVApps destacadas. Debe estar en formato PNG.
  + Un fichero **menuImage.png** (obligatorio)**:** contiene la imagen que se utilizará en la pantalla del menú horizontal de TVApps. Debe estar en formato PNG.
    1. manifest.xml

Este archivo define el comportamiento y la estructura de cara al usuario de la TVApp.

El identificador de la TVApp es usado por el servidor para desplegarlo en la Plataforma, debe ser único en la plataforma y es aconsejable que proporcione información sobre la TVApp.



1. manifest.xml

A continuación se describe cada una de las etiquetas que conforman el archivo:

* + - **id** (obligatorio): nombre que identifica a la TVApp (no puede contener espacios)
    - **category** (opcional): nombre de la categoría a la que pertenece la TVApp. Debe ser una de las existentes en la Plataforma de TVApps. Si la categoría no se declara, se usa la categoría por defecto (“General”).
    - **title** (obligatorio): nombre de la TVApp de cara a los usuarios.
    - **description** (obligatorio): descripción larga de la TVApp.
    - **short\_description** (obligatorio): descripción corta de la TVApp.
    - **version** (obligatorio): nombre o número de la versión de la TVApp (texto libre)
    - **preview** (opcional): número que identifica el tipo de previsualización de la TVApp. Los posibles valores de este atributo son:
      * + 1: Estática
        + 2: Dinámica general
        + 3: Dinámica personalizable

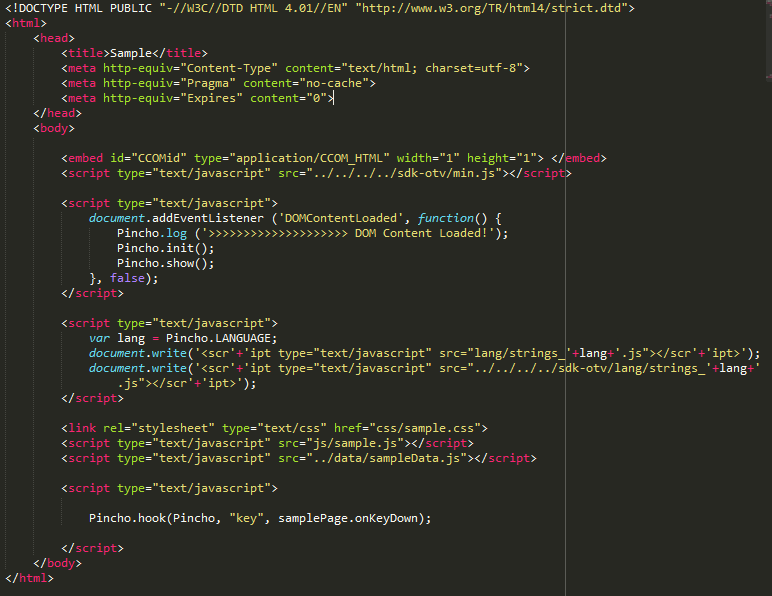
En el caso de la plataforma OpenTV (la del descodificador híbrido) no aplica el tipo de preview. Pondremos siempre 1.

* + - **author** (obligatorio): nombre corto que identifica al autor/tercero de la TVApp. Debe ser uno de los existentes en la plataforma. Para que un tercero pueda tener TVApps desplegados en la Plataforma de TVApps es necesario que previamente haya sido dado de alta.
    - **datein** (opcional): fecha de inicio del periodo durante el cual la TVApp estará activa. Si no se declara, la TVApp no tendrá restricción de fecha de inicio. Debe ir en formato: yyyy-mm-dd hh:mm:ss
    - **dateout** (opcional): fecha de fin del periodo durante el cual la TVApp estará activa. Si no se declara, la TVApp no tendrá restricción de fecha de fin. Debe ir en formato: yyyy-mm-dd hh:mm:ss
    - **levels** (obligatorio): descripción general de la estructura de una TVApp. Dentro de este elemento se debe declarar al menos un elemento del siguiente tipo:
      * **level** (obligatorio, uno al menos): indica el nivel o layout utilizado en la TVApp (de 1 a 4, con cualquier combinación).

En el caso de la plataforma OpenTV (descodificador híbrido) no hay más layouts que el de pantalla completa, pondremos 1.

* + - **platforms** (obligatorio): lista de plataformas soportadas por la TVApp. Dentro de este elemento se debe declarar al menos un elemento del siguiente tipo:
      * **platform**: contendrá un atributo obligatorio, **name**. El valor de este atributo para las aplicaciones del descodificador híbrido debe ser: OpenTV.
    - **helper** (opcional): determina los distintos parámetros del helper asociado a la TVApp para su despliegue automático al mismo tiempo que el front-end de la TVApp. Contendrá los siguientes elementos:
      * **execute** (opcional): nombre del fichero PHP que contiene el helper de la TVApp y que se ejecutará periódicamente. Por defecto, el fichero de ejecución será: *helper\_<widget\_id>.php*
      * **cron\_options** (obligatorio): frecuencia de ejecución de un helper. Deberá estar en el formato admitido por el crontab. Por ejemplo: 05 04 \* \* \* (es decir, que el helper se ejecutará todos los días a las 04:05). El significado de los cinco campos de esta datos es el siguiente:
        + Primer campo **minutos** (0-59)
        + Segundo campo **horas** (0-23)
        + Tercer campo **días del mes** (1-31)
        + Cuarto campo **meses** (1-12)
        + Quinto campo **días de la semana** (0-6, donde 0=Domingo))
      * **max\_num\_errors** (obligatorio): Número de ejecuciones erróneas consecutivas de un helper para considerar el error como consolidado.
      * **max\_num\_oks** (obligatorio): Número de ejecuciones correctas consecutivas de un helper para considerar que el helper se ha recuperado de un error consolidado previo.
    - **tv\_channel** (opcional): nombre de un canal de TV con el cual está relacionada la TVApp. Si una TVApp está relacionada con un canal de TV, entonces el usuario sólo tendrá acceso a dicha TVApp en el caso en que también tenga acceso al canal de TV indicado. El control de acceso es gestionado por la Plataforma de TVApps.
    - **domains** (opcional): lista de dominios externos que se acceden directamente desde la TVApp. Si un dominio no se incluye en esta lista y la TVApp lo intenta usar, la plataforma **no** lo permitirá. Este elemento no se suele usar puesto que en general las TVApps acceden a las fuentes de datos a través de los helpers.
    1. index.html

Este archivo define la estructura fundamental de la TVApp HTML. Los archivos externos necesarios para la ejecución de la TVApp han de ser definidos en la cabecera.



1. index.html

Para acceder al SDK se debe importar el fichero **min.js**.

Las aplicaciones desarrolladas para el descodificador híbrido han de incluir la siguiente línea en el body:

<embed id="CCOMid" type="application/CCOM\_HTML" width="1" height="1"> </embed>

Para acceder a la fuente de datos de la TVApp, debemos importarla de la misma manera. En el ejemplo se usa **sampleData.js**. El fichero *js/sampleData.js* describe todas las funciones necesarias para la ejecución de la TVApp.

Como podemos ver, es recomendable trabajar dentro de index.html únicamente con los elementos HTML **DIV, IMG** y **TABLE.** Otros elementos sí que son soportados por el descodificador, pero pueden llevar a comportamientos extraños en los diferentes modelos de descodificadores (especialmente, se recomienda evitar el uso de los elementos **<A>**, **<B>** e **<INPUT>**).

Otra consideración a tener en cuenta es tratar de mantener en el index.html una estructura HTML simple, sencilla y limpia. A medida que definamos más elementos dentro de esta estructura, la carga inicial de la TVApp se hará mucho más lenta.

En este sentido, debemos recordar que también podemos añadir elementos HTML dinámicamente a través de las funciones JavaScript *document.createElement* y *node.appendChild* (esto debe ser hecho a través del fichero widget.js que veremos a continuación).

Por tanto, es recomendable definir los elementos estáticos explícitamente en el fichero index.html, mientras que aquellos elementos variables los manejaremos con las funciones DOM de JavaScript.

Sin embargo, hay que tener en cuenta que si bien tener definidos menos elementos en index.html y definir los demás dinámicamente permite una carga inicial mucho más rápida, esto aumenta la complejidad del código javascript de widget.js, e introduce una demora en la navegación dentro de la TVApp mientras se crean dichos elementos. Por lo que establecer un equilibrio en la definición de estos elementos dependerá de las funcionalidades requeridas por cada TVApp, y del criterio del desarrollador.

El inicio de la ejecución del código javascript se marca con el siguiente código, que llama a la carga del primer nivel, una vez que todo el contenido del DOM se ha cargado:

*document.addEventListener('DOMContentLoaded', function(){*

*Pincho.log('loaded!');*

*Pincho.init();*

*Pincho.show();*

*}, false);*

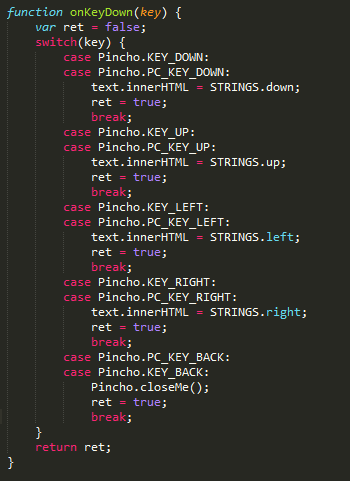
Cuando se selecciona una TVApp ya sea desde el menú o desde la parte dinámica de la home, el contenedor de TVApps mostrará una imagen de cargando que se ocultará y se mostrará la TVApp cuando se realice la llamada a Pincho.show().

* + 1. widget.js

En este archivo se definen las funciones que son invocadas cuándo el usuario abre la TVApp HTML o cuándo está navegando por la TVApp y pulsa cualquier tecla. Inicialmente, debemos registrar las funciones que controlarán estos eventos de pulsado de teclas. Para ello utilizamos la función *Pincho.hook* que provee el SDK:

* **Pincho.hook(Pincho, "key", onKeyDown);**

De esta manera registramos la función *teclas* como manejador de los eventos de pulsación de teclas.



1. widget.js

Cuando se pulsa una tecla, el contenedor llama a la función *onKeyDown,* pasando cómo parámetro la tecla pulsada. Dependiendo de la tecla que ha sido pulsada y de la situación actual de la TVApp en ese momento, la función puede hacer cosas diferentes.

La gestión del BACK ha de hacerse explícitamente en la TVApp. Cuando se desee cerrar la aplicación y regresar a la pantalla desde la que se lanzó la TVApp, debe usarse la función closeMe.

Determinadas teclas en el descodificador híbrido realizan acciones por defecto y pueden provocar la salida de la aplicación. La TVApp no ha de hacer nada para controlar estas teclas especiales:

* + - EPG en el mando a distancia / G en el teclado: muestra un popup para confirmar la salida de la aplicación y en caso de aceptar, muestra la EPG.
    - Botón HOME en el mando a distancia / ESC en el teclado: muestra un popup para confirmar la salida de la aplicación y en caso de aceptar, muestra la home del descodificador híbrido.
    - Botón EXIT del mando a distancia / T en el teclado: muestra un popup para confirmar la salida de la aplicación y en caso de aceptar, va al LiveTV.
  1. Acceso a fuentes de datos

Desde las TVApps HTML, hay dos formas de acceder a fuentes de datos:

* Acceso estático: el archivo local de datos JSON se incluye directamente en el archivo.
* Acceso dinámico: el archivo local de datos JSON se incluye dinámicamente a través de una función.
  + 1. Acceso estático a datos

El acceso a datos de forma estática consiste en incluir el archivo de datos local JSON dentro del fichero index.html. El archivo debe incluir una variable JavaScript que contenga el dato:



1. Acceso estático

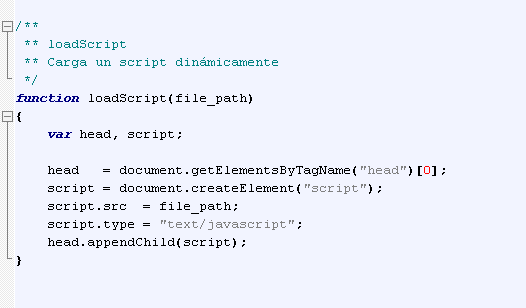
En este ejemplo, el archivo **../data/news.js** está incluido estáticamente.

* + 1. Acceso dinámico a datos

El acceso a datos dinámicamente consiste en cargar un fichero JSON con la información de una fuente de datos a raíz de una acción determinada dentro de la navegación de una TVApp (puede ser al pulsar la tecla “OK” para acceder a una opción dentro de la TVApp).

Para ello debemos definir una función que se encargue de obtener el fichero de datos y agregarlo mediante las funciones DOM de JavaScript a la etiqueta HEAD (o la etiqueta BODY) del fichero index.html.

Una propuesta de esta función se presenta en el siguiente ejemplo:



1. Acceso dinámico

Una consideración a tener en cuenta es que la carga de los ficheros en el descodificador es asíncrona. Es decir, una vez que se solicita la carga de un fichero con la función *loadScript,* no se esperará a que se cargue completamente y se compruebe dicho fichero para seguir con la ejecución de la TVApp, sino que inmediatamente pasará a ejecutar las siguientes líneas de código definidas en el fichero js.

Esto puede causar que en muchas ocasiones, si el fichero requerido es suficientemente grande, no se haya cargado completamente el JSON contenido en dicho fichero una vez que en el código de widget.js se utilice su contenido, lo cual desencadenaría en un error fatal.

Para ello, se deberá utilizar un retardo para la ejecución del código posterior a la llamada *loadScript* (por ejemplo, con la función JavaScript *setTimeout*), o se deberá utilizar una estructura *try/catch* para control la correcta carga de los datos contenidos en el fichero requerido dinámicamente.

glosario de términos y acrónimos

API

Application Programming Interface

BBDD / BD

Bases de Datos

CSC

Centro de Servicios Centrales

CSL

Centro de Servicios locales

CSS

Cascading Style Sheets

DTS

Definición Tecnológica de la Solución

EAP

Extensible Authentication Protocol

EPG

Electronic Program Guide

FTP

File Transfer Protocol

HTML

HyperText Markup Language

HTTP

HyperText Transfer Protocol

HTTPS

HyperText Transfer Protocol Secure

IPTV

Internet Protocol Television

JS

JavaScript

JSON

JavaScript Object Notation

MVC

Modelo Vista Controlador

PC

Ordenador Personal

TVApp

Es un servicio personal, de acceso sencillo que permite a los usuarios disfrutar de información y contenidos disponibles en Internet, de una forma fácil y guiada, con una experiencia de uso adaptada a la TV.

PHP

Acrónimo recursivo de PHP Hypertext Pre-processor

REST

Representational State Transfer

RSS

Really Simple Syndication

SDK

Software Development Kit

SGBD

Sistema Gestor de Base de Datos

SGBDR

Sistema Gestor de Base de Datos Relacionales

SSH

Secure Shell

STB

Set Top Box

TCP

Transmission Control Protocol

URL

Uniform Resource Locator

VOD

Vídeo on Demand

VPN

Virtual Private Network

W3C

World Wide Web Consortium (organismo de estandarización de tecnologías web)

XML

eXtensible Markup Language

ZIP

Formato de compresión ZIP