GENERACIÓN AUTOMATIZADA de CARTELERÍA DIGITAL en CINES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

okode

PROYECTO FINAL DE CARRERA

Ingeniería Técnica Informática de Gestión

DSIC-184
2011

Alumno ALEJANDRO DÍAZ RUIZ
Tutor de la UPV ROBERTO AGUSTÍN VIVÓ HERNANDO
Director del proyecto PEDRO JORQUERA HERVÁS

Tabla de Contenidos

1. Antecedentes

Introducción

El desarrollo de este PFC titulado: “Generación automatizada de cartelería digital en cine”, forma parte de un proyecto de gran alcance para la gestión de una red de cines a nivel nacional que está desarrollando la empresa Okode, y en el que también está involucrado desde su inicio el alumno de Ingeniería Técnica en Informática de Gestión, Alejandro Díaz Ruiz, que forma parte de la empresa (en prácticas) desde septiembre de 2009.

El PFC que se describe en este documento se identifica con el código DSIC-184 (del curso 2010-2011). Está dirigido por el profesor del DSIC, Roberto Agustín Hernando, y desarrollado en la empresa Okode, bajo la supervisión de su director, Pedro Jorquera Hervás.

La empresa

Okode es una consultora tecnológica experta en desarrollo de soluciones para Internet y dispositivos móviles. Entre las actividades de Okode destacan el desarrollo de aplicaciones móviles para iPhone, iPad y Android, portales Web 2.0, integración de sistemas de información geográfica basados en Google Maps y aplicaciones de marketing interactivo para redes sociales (Facebook).

El proyecto

Okode actualmente se encuentra inmersa en un proyecto de gran alcance encargado de la gestión de una red de cines a gran escala a nivel nacional (actualmente unos 300 cines y con vistas de que aumente a lo largo del próximo año). El cliente de Okode es el que gestiona dicha red y lo hace con el objetivo de ofrecer a los cines, entre muchas otras utilidades, contratación de contenidos alternativos (ópera, deportes, conciertos, etc.) y un sistema de cartelería digital con canales de información y publicidad.

El proyecto se basa en un portal de gestión de cines y contenidos principalmente, desde el cual, los encargados de cada cine gestionan la programación de cada una de las salas de sus cines, a través de unos calendarios y unos listados de contenidos disponibles para su programación. Los contenidos son gestionados por los administradores del portal, y disponen de una ficha completa de información.

Esta información, que se gestiona desde el portal desde hace unas versiones, se utiliza para un front-end público que expone la programación de cada cine, para la gestión interna de la plataforma, y para exponer utilidades hacia los exhibidores (cines). Los anunciantes y la publicidad también se gestionan de forma centralizada y se dan de alta en el mismo portal.

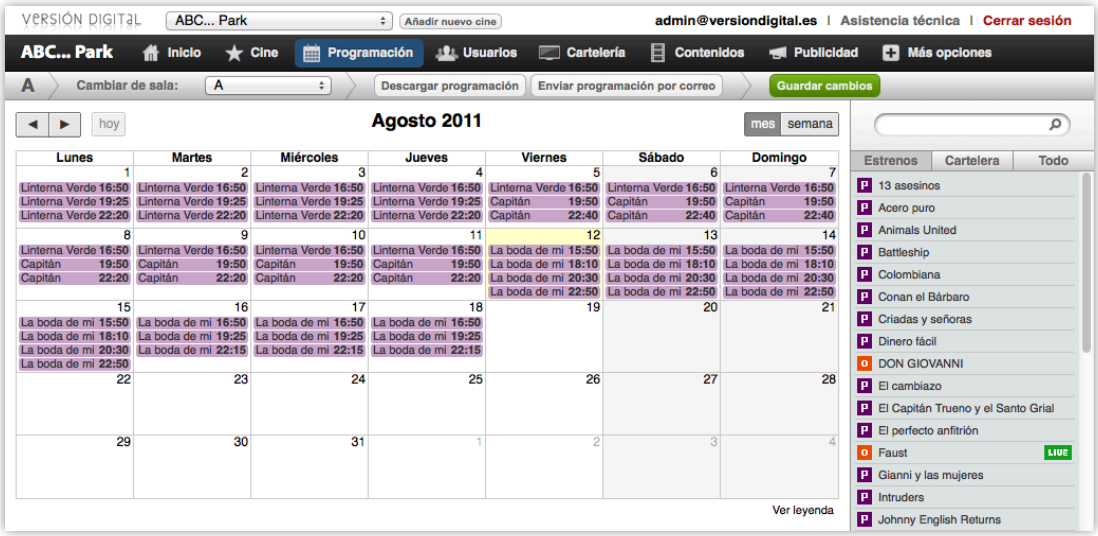


Ilustración 1 | ***Página de programación de los contenidos en una sala***

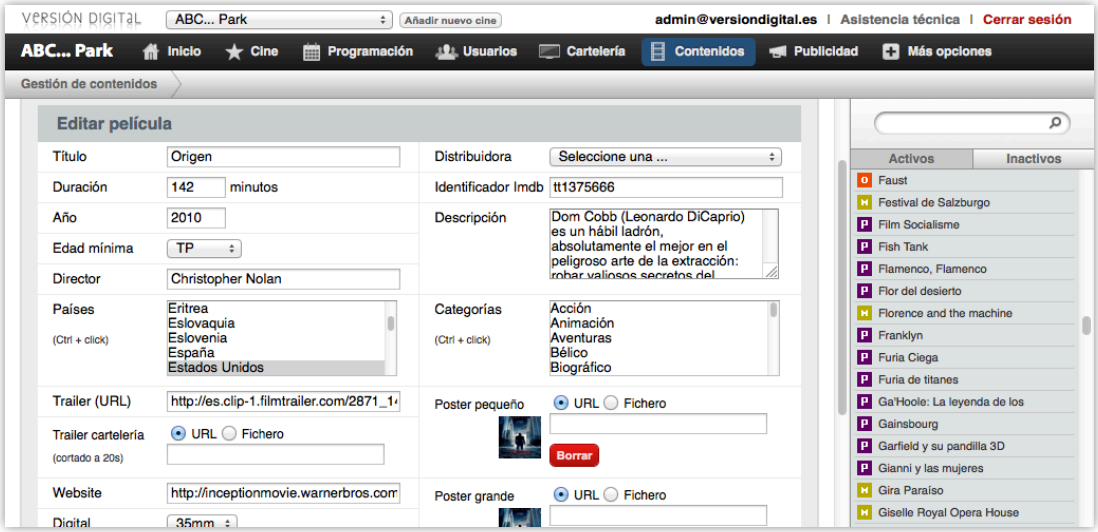


Ilustración 2 | ***Página de alta/edición de contenidos programables***

Sin embargo, la generación del contenido que se utiliza en la cartelería digital actualmente se está realizando manualmente a través del montaje de vídeos que mezclan tanto la información de cartelera (que ya se encuentra disponible en el portal) y los vídeos de publicidad. Esta solución inicial para la cartelería digital comienza a resultar un grave problema por el coste temporal y en recursos humanos que conlleva la generación y por las incidencias inherentes a un sistema manual.

Descripción de los componentes y tecnologías usadas en el desarrollo del portal

El portal de gestión de cines y contenidos digitales (proyecto del que depende este PFC) está desarrollado con Java EE 6, usa PostgeSQL 8.4 como SGBD, corre sobre un servidor de aplicaciones Glassfish 3 usando un Apache 2 como proxy.

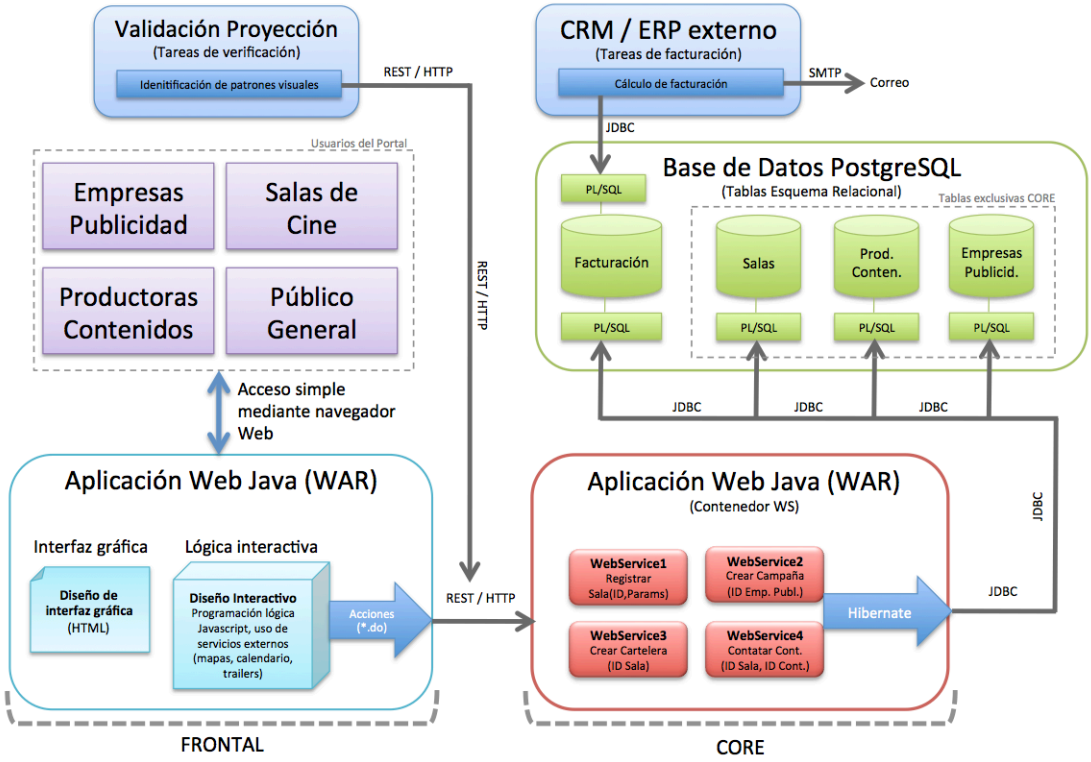


*Ilustración 3 | **Diagrama de componentes del proyecto de gestión de cines***

A parte cabe destacar, entre otras tecnologías, el uso de JSF 2.0, JPA 2.0, EJB 3.1, Maven 3.x, Web Services / REST, XHTML – CSS – JavaScript (jQuery).

Se trata de un proyecto de gran envergadura con algunos módulos que proporcionan funcionalidades especificas (Gestión de la programación, Gestión de contenidos digitales , Sistema de venta de entradas, **Generación de cartelería digital**, etc).

2. Objetivo

El objetivo que persigue esta propuesta de Proyecto Final de Carrera es llevar a cabo la implementación de un sistema de generación automatizada de cartería digital obteniendo la información necesaria desde el portal actualmente en producción. No solo se deberá mostrar la programación en las pantallas de los cines sin intervención manual, sino que se deberá tener en cuenta la publicidad según el algoritmo de distribución de publicidad que marque el portal.

A continuación describiremos el escenario inicial y los requisitos que marcarán la solución a implementar.

Requisitos previos y limitaciones

La solución final a implementar se ve condicionada por varios factores:

- Como requisito inicial por parte del cliente contamos con buscar una solución basada en unos ***players*** de bajo coste sin capacidad de mostrar contenido dinámico. Se trata de hardware cuya única capacidad funcional es reproducir vídeo. La actualización de cada *player* se realiza diariamente descargando el contenido de un FTP central.

- La plataforma consta de N implantaciones (tantas como cines participantes) con M *players* cada una. El número de *players* en local dependerá del tamaño y necesidades del cine en concreto y se deberá contemplar ofrecer una amplia casuística de representación (fragmentación de la cartelera, vistas en horizontal y vertical, diferentes layouts, etc.).

- Cada *player* de una misma implantación podrá requerir representar información diferenciada dando opciones.

- La plataforma debe ser la misma para todos los *players* y su gestión debe estar centralizada en la empresa contratante.

- Aunque los *players* no están conectados permanentemente y son principalmente offline, se precisa buscar una solución para que los cambios en la programación lleguen al *player* lo más rápidamente posible.

- Existe un *algoritmo de distribución de publicidad* que debe ser integrado en la generación de la escaleta de vídeos, lo cual dificulta generar un único vídeo en loop tal y como se hace actualmente.

- El vídeo final debe tener una resolución de 1280x720 (720p) y una calidad óptima, pues se reproducirán en pantallas de gran tamaño. Se deberán tener en cuenta los problemas de escalabilidad analizando los tiempos de generación y transferencia en función del tamaño y compresión de los vídeos y la carga y volumen de trabajo del sistema.

3. Alcance

Dados los requisitos definidos, la solución a implementar en el presente proyecto se puede resumir en los siguientes puntos que definen el alcance del mismo:

- **Se expondrá a través de servicios web la información** que no esté actualmente expuesta para que puedan ser consumidos por el ***media server*** (máquina encargada e la generación de los vídeos de cartelería).

- **Se diseñará un sistema de plantillas basado en un motor de scripting** que facilite la generación de vídeos y la extensibilidad del sistema. Se evaluará integrar el Frame Server de AVISynth entre otras alternativas.

- **Se diseñará un sistema de perfiles por grupos de pantallas**, para que de esta manera, los cines puedan disponer de diferentes grupos de pantallas con sus perfiles que emitan diferente contenido. (Por ejemplo: un grupo emite la cartelería por película y sala, y otras emiten todo el rato el listado de todas las salas y sesiones del día o semana).

- **La cartelería se generará en un *media server*** en base a los contenidos del portal y de una lógica de lanzamiento dependiente de las necesidades de actualización y las posibilidades de escalabilidad.

- **Se generarán múltiples vídeos**, tantos como sean necesarios por cines y contenidos, mostrando horarios, póster, además de otra información como la duración, la edad recomendada, etc. Al no mezclar toda la parrilla en un único vídeo en loop será posible reutilizar los clips que no varíen entre actualizaciones con el ahorro en CPU y ancho de banda que esto implicará.

- **El *media server* transferirá vía FTP** los vídeos generados para cada cine, en base a la lógica y frecuencia de generación que se considere, al repositorio central para exponerlos a los *players*. Junto a los vídeos de cartelera se transferirán los de publicidad y los metadatos que conformen la ***parrilla de emisión*** que según las lógicas definidas haya configurado el portal.

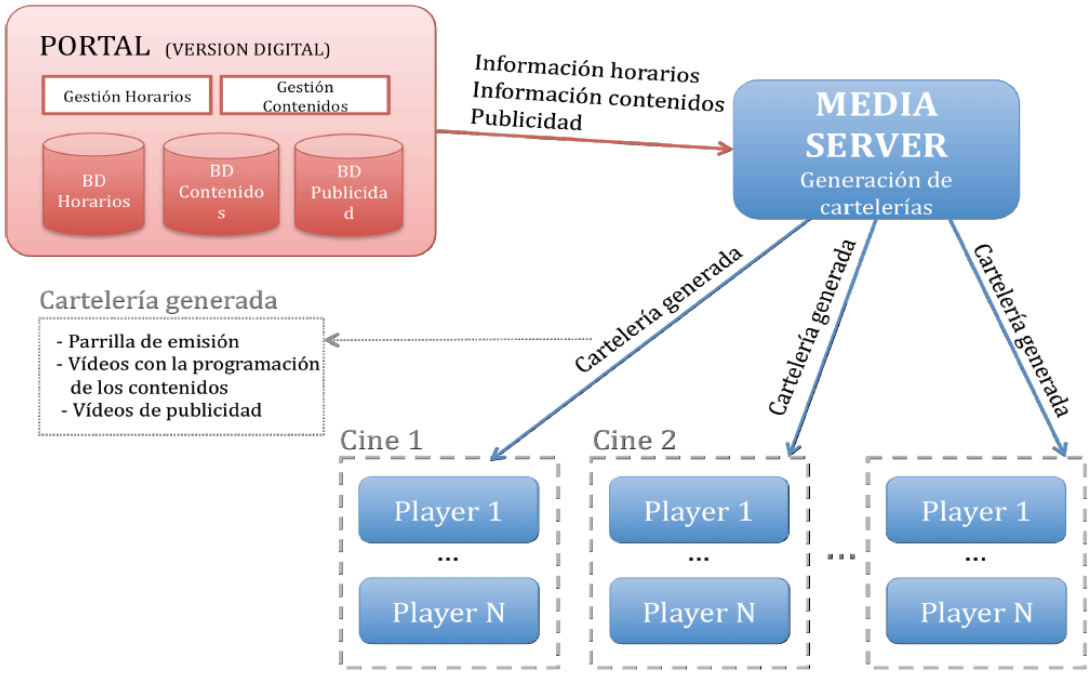


Ilustración 4 | ***Diagrama de componentes del proyecto de gestión de cines***

Resulta importante resaltar la necesidad de realizar pruebas de escalabilidad considerando tanto el ancho de banda como el tiempo y consumo de CPU. Se deberá encontrar la mejor combinación en cuanto a la solución software de generación, los *códecs* de compresión utilizados y la disposición de los recursos (servidores remotos, cachés intermedias, servidor local) para llegar a la solución más óptima. Es un punto crítico del sistema, teniendo en cuenta dos factores:

1. La programación cambia generalmente los viernes y en muchas ocasiones no se dispone de ella hasta el día anterior, con lo que el tiempo desde que se decide generar los vídeos de cartelería hasta que puedan verse en los *players* del cine es crítico.

2. Inicialmente, sin contar modificaciones no planificadas, se generará cada semana un vídeo por cada contenido y sala que lo emita (unos 12 clips diferentes por cine de media). Teniendo en cuenta los 300 cines de partida, son 3600 vídeos que se deben gestionar semanalmente desde el ***media server*** para que lleguen a los cines en el momento oportuno.

4. Entorno de desarrollo

A continuación se detallan las herramientas más importantes que se utilizan en el desarrollo de este proyecto.

Para este proyecto se ha trabajo tanto con Mac OS X como con Windows, dependiendo de las necesidades y limitaciones de cada herramienta.

Herramientas de organización y seguimiento

Subversión

Multiplataforma
Sistema de control de versiones del repositorio de código. Automatiza las tareas de guardar, recuperar, registrar, identificar y mezclar versiones de archivos. En todo momento se puede consultar el historial de modificaciones que ha sufrido un proyecto, para comprobar los avances realizados o volver a versiones previas para arreglar posibles fallos.

Atlassian **Jira**

Aplicación web
Aplicación para el seguimiento de errores, de incidentes y para la gestión operativa de proyectos. Permite gestionar tareas o errores desde el momento de su creación hasta su finalización, asignando el asunto a su responsable correspondiente y con la posibilidad de añadir comentarios, capturas o incluso apuntarte como espectador de forma que seas convenientemente informado a través de correo electrónico de todos los cambios que se produzcan.

Atlassian **Confluence**

Aplicación web
Confluence es un wiki corporativo que facilita la colaboración dentro de equipos de trabajo y compartir el conocimiento. Facilita la creación, el intercambio y la búsqueda de contenido.

Herramientas auxiliares de construcción

Maven 3

Multiplataforma
Maven es una herramienta de software para la gestión y construcción de proyectos Java. Utiliza un POM (Project Object Model) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Atlassian **Bamboo**

Multiplataforma
Servidor de integración continua y control de versiones que permite aumentar la productividad y mejorar la calidad del código. Al compilar y probar automáticamente el código mientras va cambiando, Bamboo proporcionará informaciones instantáneas (informes) para los desarrolladores y también permitirá una colaboración rápida.

Herramientas de desarrollo

Eclipse IDE for Java EE

Multiplataforma
Entorno de desarrollo integrado (IDE) de código abierto. Es multiplataforma y es el IDE que se utiliza para el desarrollo de todo el proyecto (portal web de gestión de cines y contenidos).
Con Eclipse se implementarán los módulos necesarios en el portal que añadirán la funcionalidad necesaria para la generación de cartelería (los servicios de exposición de información para el media server, gestión de plantillas/perfiles/cine, sistema de logs y feedback del proceso de generación, ...).

Microsoft **Visual Studio** 2010

Windows
Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.
Visual Studio será utilizado para la implementación del *media server*, que correrá sobre una máquina Windows, debido a que será el encargado de generar los vídeos de cartelería, y las herramientas seleccionadas requieren un entorno Windows (AVISynth y VirtualDub, que se comentan a continuación).

Herramientas para la generación de vídeo

AVISynth

Windows
AVISynth es una poderosa herramienta de post-producción de vídeo. Proporciona formas ilimitadas de edición y procesamiento de vídeos. Trabaja como un Frame Server, proporcionando edición instantánea sin la necesidad de ficheros temporales. AVISynth por sí mismo no proporciona una interfaz gráfica (GUI) pero en vez de eso, usa un sistema de scripting no lineal que permite edición avanzada. Puede parecer tedioso a primera vista, pero es extraordinariamente poderoso y es una forma muy buena para manejar proyectos precisa, consistente, y reproduciblemente. Como los scripts basados en texto son comprensibles, los proyectos están inherentemente autodocumentados.

VirtualDub

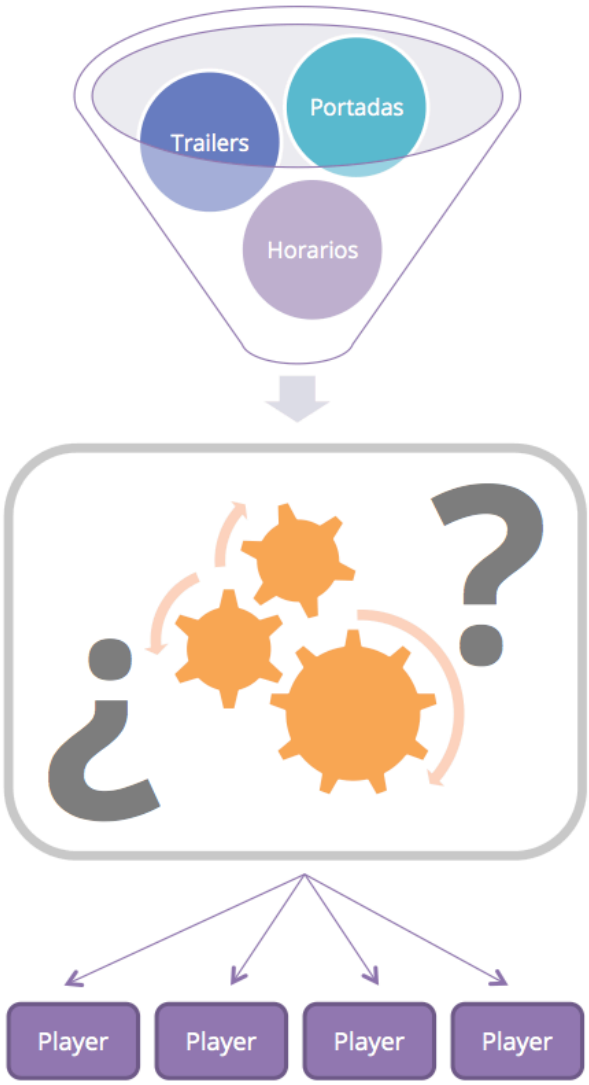
Windows

VirtualDub es una herramienta de código abierto para capturar vídeo y procesarlo, se ejecuta en Microsoft Windows. Dispone de funciones muy avanzadas, es capaz de usar plugins para añadir diferentes técnicas de procesado de vídeo, y puede trabajar con cualquier fichero AVI, independientemente del códec que use, mientras esté instalado. También puede ejecutarse mediante la línea de comandos lo que permite que pueda ser usado para procesamiento por lotes y depuración; característica que determinó su elección.

5. Estudio y análisis técnico

Como punto de partida y antes de empezar en el desarrollo tanto del Media Server como de los servicios necesarios en el portal, se debe realizar un estudio inicial, conociendo las limitaciones técnicas y las opciones de desarrollo que tenemos, principalmente a la hora de seleccionar la herramienta para scriptar los vídeos de la cartelería y las opciones y limitaciones que nos brinda.

Antes de tomar la decisión de que herramientas y/o tecnologías a emplear para la generación de vídeo, debemos conocer las limitaciones que tenemos. En primer lugar tocará conocer el *player* con el que trabajaremos, impuesto por el cliente, y que será el mismo en todos los cines. Conocerlo, significa saber y comprobar que formatos soporta, que ventajas nos puede ofrecer y que limitaciones pueden aparecer.



El siguiente paso será decidir la herramienta de scriptado de vídeo, entre varias alternativas. Habrá que valorar junto al cliente ciertas características.

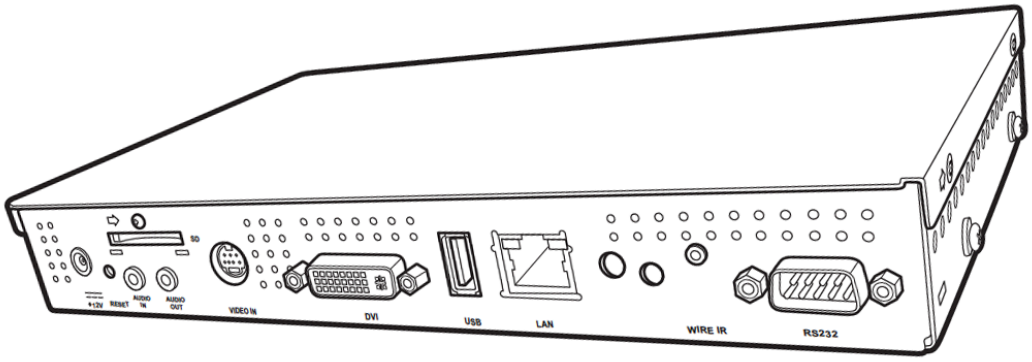
Seguido a esto, toca realizar pruebas de generación del vídeo scriptado en el paso anterior, con códecs soportados por el *player*, y probarlo sobre el mismo.

Y por último, y probablemente lo más importante, será realizar pruebas de carga y transferencia de los vídeos generados, valorando factores de compresión con la mínima pérdida de calidad posible. Se realizarán pruebas de carga y transferencia superior al volumen inicial esperado, para estudiar la escalabilidad y fiabilidad del sistema que se va a desarrollar.

Player de Optoma

La única limitación técnica por parte del cliente ha sido la elección del *player* (hardware) que emitirá en los cines la cartelería digital a través de las pantallas. Se trata de unos *players* fabricados por uno de sus proveedores, **Optoma**, una empresa líder mundial en la fabricación de proyectores y reproductores de vídeo/audio.

Optoma Digital Signage Processor SignShow D5000



Este *player* está concebido para la cartelería digital en sitios públicos debido a sus características, las cuales encajan perfectamente para el objetivo de este proyecto. Las características claves son las siguientes:

- Reproduce **varios formatos de vídeo**, aceptando estándares de compresión de gran calidad, lo que proporcionan una mayor flexibilidad para la decisión de desarrollo final.

- Incorpora conexión en red, pudiendo **sincronizarse de forma remota** continuamente (vía FTP).

- Permite la **programación en bucle** (mediante un archivo XML) de los vídeos cargados en el *player*. Permitiendo incluso, parametrizar el día y hora de inicio y de caducidad de dicha programación.

- Soporta resoluciones de 720p y 1080p.

Cada uno de estos *players*, emitirá el mismo contenido para un conjunto de pantallas en un cine. El número de *players* y grupo de pantallas dependerá del tamaño y características de cada cine.

La posibilidad de la programación por XML de la emisión en bucle de los vídeos de cartelería, soluciona un problema inicial, que era saber como se podría reproducir una lista de reproducción programable, pudiendo añadir el loop de publicidad con la lógica correspondiente, se llegó incluso a plantear,

en una aproximación inicial, la construcción del vídeo con todos los vídeos con la secuencia de todo el día, caso extremo que hubiera complicado los tiempos de generación y transferencia del contenido por FTP.

Group	Type	Códec	Container	Extension
WMV	Video	Windows Media Video 9	ASF	.WMV, .ASF
	Video	Windows Media Video 9 Advanced Profile		
	Audio	Windows Media Video 9		
MPEG-1	Video	MPEG-1 Video	MPG	.MPG
	Audio	MPEG-1 Audio layer 2		
MPEG-2	Video	MPEG-2 Video	MPG, M2P	.MPG, .M2P
	Audio	MPEG-1/2 Audio layer 2	M2V	.M2V
	Audio	MPEG-1/2 Audio layer 3		
MPEG-2	Video	MPEG-2 Video (Transport Stream)	TS	.TS
	Audio	MPEG-1/2 Audio layer 2		
	Audio	MPEG-1/2 Audio layer 3		
MPEG-2	Video	MPEG-2 Video (Program Stream)	VOB	.VOB
	Audio	MPEG-1/2 Audio layer 2		
	Audio	MPEG-1/2 Audio layer 3		
MPEG-4	Video	DivX 5.0	Divx, AVI	.DivX, .AVI
	Audio	MPEG-1/2 Audio layer 2		
	Audio	MPEG-1/2 Audio layer 3		
MPEG-4	Video	H.264	MOV, AVI, TS	.MOV, .AVI, .TS
	Audio	AAC		
MPEG-4	Video	MP4	MP4	.MP4
	Audio	AAC		
VC-1	Video	VC-1	TS	.TS

Tabla 1 | ***Formatos de vídeo y audio soportados***

AVISynth vs Final Cut Studio

Inicialmente se descartaron algunas opciones para la generación scriptada de vídeo por diferentes limitaciones que las hicieron incompatibles con el proyecto. Finalmente el debate se centró en dos posibles opciones, AVISynth o una solución montado con Final Cut Studio, sus características, ventajas y desventajas se comentan a continuación:

AVISynth

Características

- Un servidor (PC barato + software) con Windows para la generación de vídeo.
- Este servidor descarga la información del portal y transfiere los vídeos generados a los *players* de cada cine.
- El software incluye un motor de generación de vídeo que produce los contenidos finales renderizados con la parametrización del cine (horarios + publicidad).
- Partiendo de diseños en Adobe Photoshop (imágenes) y transiciones en Adobe Premiere se compilará todo en scripts de AVISynth.
- Cuando el cine cambia sus horarios el server recibe notificación para generar los vídeos y los players los reproducen.

Ventajas

- PC más barato y flexibilidad.
- Software libre.
- Posibilidad de usar la potencia de VirtualDub para la optimización y compresión de vídeo.
- Se minimizan las tareas manuales a confeccionar los tráilers y los anuncios (compartidos por todos los cines).
- Cada plantilla de cartelería sólo se scriptará la primera vez, y será el portal el que parametrice la plantilla cada vez.
- El refinamiento manual, o la eliminación de un vídeo de cartelería resultaría más sencillo, en caso de una excepción por datos mal introducidos.

Desventajas

- Cada plantilla es programadas de forma genérica y limitan ciertos parámetros en cada una (por ejemplo: número de sesiones máximas por día). Se deben crear “subplantillas” si se quieren extender las opciones, aunque sólo la primera vez.

Final Cut Studio + Final Cut Studio Server

Características

- Necesidad de un sistema Mac OS X.
- Integración de un servidor de contenidos en la red local con Final Cut Server.
- Los tráilers y los anuncios se suben al Final Cut Server.
- Para generar un vídeo de cartelería: se abre el proyecto genérico de Final Cut, se descarga el script desde el portal, se ejecuta el script y el video generado se sube a Final Cut Server.
- El servidor de Final Cut Server sube automáticamente a los players los vídeos de cada cine.

Ventajas

- El portal puede integrarse completamente con Final Cut Server (es un sistema abierto) y generar los scripts sin conflictos de identificadores, obteniendo además, mejoras para el portal, como la generación de las vistas previas, miniaturas, etc.

Desventajas

- Equipo y licencias de software más caros.
- La integración del sistema se complicaría.
- Aunque las plantillas serían más flexibles, esto dificulta su diseño y parametrización de cada tipo de plantilla y su futura modificación.

AVISynth, la opción elegida

Finalmente, tras diversas reuniones con el cliente final, se llegó a la elección de AVISynth como la mejor opción por dos factores determinantes:

- **Menor riesgo a asumir de inicio**, al ser un software libre y poder realizar las primeras pruebas en un PC convencional, resulta una decisión más cómoda para el cliente.

- **La creación de varias plantillas y sus variantes es más rápida**, factor importante para la posible personalización de cada cine para ajustarse por parte de nuestro cliente a sus clientes (los cines).

AVISynth, primeras consideraciones y pruebas de concepto

Una vez tomada la decisión de que sería AVISynth la herramienta a usar, es momento de realizar las primeras aproximaciones de lo que sería una plantilla final, para conocer las limitaciones y posibilidades del mismo, así como los tiempos y costes de generación que pudiera acarrear emplear o no ciertas características de la herramienta, pues hay que tener en cuenta que el tiempo de generación es un factor muy importante a tener en cuenta a la hora de tomar la decisión de cómo hacer la plantilla y de que extras prescindir, ya que la carga de trabajo del Media Server en ciertos momentos puede ser muy grande (estamos hablando de generar muchos vídeos de cartelería para cada cine de forma diaria o semanal, de una red de unos 300 cines).

AVISynth es un sistema de scripting de vídeo con el que montaremos las diferentes plantillas de cartelería, pero la generación del vídeo resultante será trabajo de VirtualDub, el cual, tardará más o menos tiempo depende de la complejidad del script y del estándar de compresión que usemos desde VirtualDub para la generación del vídeo resultante.

Factores que deberemos considerar en las pruebas para medir tiempos:

- El fondo del vídeo será una imagen o un conjunto de imágenes, habrá que valorar los costes de:
    - Usar sólo una imagen que cubra todo el lienzo del vídeo.
    - Usar un conjunto de imágenes (casilla de horarios, casilla de edad, casilla de título de película, ...).
    - Probar con diferentes formatos de imágenes (.jpg, .bmp, .png)
- Costes de reescalado de trailers y portadas de las películas para adaptarlos a la plantilla. Esto será algo necesario que tendremos que asumir, pero quizá existan diferentes técnicas, habrá que buscar la mejor.
- Costes para añadir textos.
- Efectos de transición de entrada y salida para cada plantilla.

Conclusiones con las primeras pruebas de concepto

- El número de imágenes a añadir es indiferente si no se reescalan.
- El formato preferible para imágenes es .BMP (no necesita conversión).
- Los textos se pueden añadir como subtítulos (con formato propio: fuente, tamaño, color, borde y posición) y no tiene coste significante.
- Los efectos de *FadeIn* y *FadeOut* de no tienen coste significante.

- La portada de los contenidos digitales (que se descargan del portal) están en formato JPG, es un coste que hay que asumir en las plantillas que incorporen la portada.
- Los costes de reescalado de tráiler y portadas son perfectamente asumibles conforme se expone en el ejemplo que aparece a continuación.

Ejemplo de script de AVISynth / **plantilla.avs**

```
varFondo = "C:\work\templateWEEK4ROWS\plantilla\fondo.bmp"
varTrailer = "C:\work\templateWEEK4ROWS\origen.mp4"
varPoster = "C:\work\templateWEEK4ROWS\origen.jpeg"
varSala = "Sala 1"
varDay1Sessions = "17:00    19:00    21:00"
varDay2Sessions = "17:00    19:00    21:00    23:00"
varDay3Sessions = "17:00    19:00    21:00"
varDay4Sessions = "19:00    21:00"
varAgeLimit = "TP"
varDuration = "93 min"

LoadPlugin ("C:\work\CarteleriaDigital\ffms2.dll")
trailer = FFVideoSource (varTrailer, colorspace="YUY2", width=716,
height=403, resizer="BICUBIC")
poster = ImageSource(varPoster,
pixel_type="RGB32").ConvertToYUY2(matrix="Rec709",
interlaced=false).Lanczos4Resize(444, 640)
background = ImageSource(varFondo, pixel_type="RGB32", fps=24, start
= 0, end = 480).ConvertToYUY2(matrix="Rec709", interlaced=false)
background.Layer(trailer, op="add", level=256, x=524, y=278,
threshold=0, use_chroma=true).Layer(poster, op="add", level=256,
x=40, y=40, threshold=0, use_chroma=true)

Subtitle(varSala, x=540, y=45, font="Tahoma", size=40, lsp=0,
text_color=$ff821d)
Subtitle(varDay1Sessions, x=1220, y=105, align=9, font="Tahoma",
size=23, lsp=0, text_color=$ffffff)
Subtitle(varDay2Sessions, x=1220, y=137, align=9, font="Tahoma",
size=23, lsp=0, text_color=$ffffff)
Subtitle(varDay3Sessions, x=1220, y=169, align=9, font="Tahoma",
size=23, lsp=0, text_color=$ffffff)
Subtitle(varDay4Sessions, x=1220, y=201, align=9, font="Tahoma",
size=23, lsp=0, text_color=$ffffff)
Subtitle(varAgeLimit, x=1204, y=44, align=8, font="Tahoma", size=20,
text_color=$ffffff)
Subtitle(varDuration, x=1205, y=71, align=8, font="Tahoma", size=16,
text_color=$ffffff)

FadeIn(12).FadeOut(12)
```

El ejemplo anterior es un script de AVISynth para generar un vídeo de cartelería digital que muestra la programación de un contenido en una sala durante una semana.

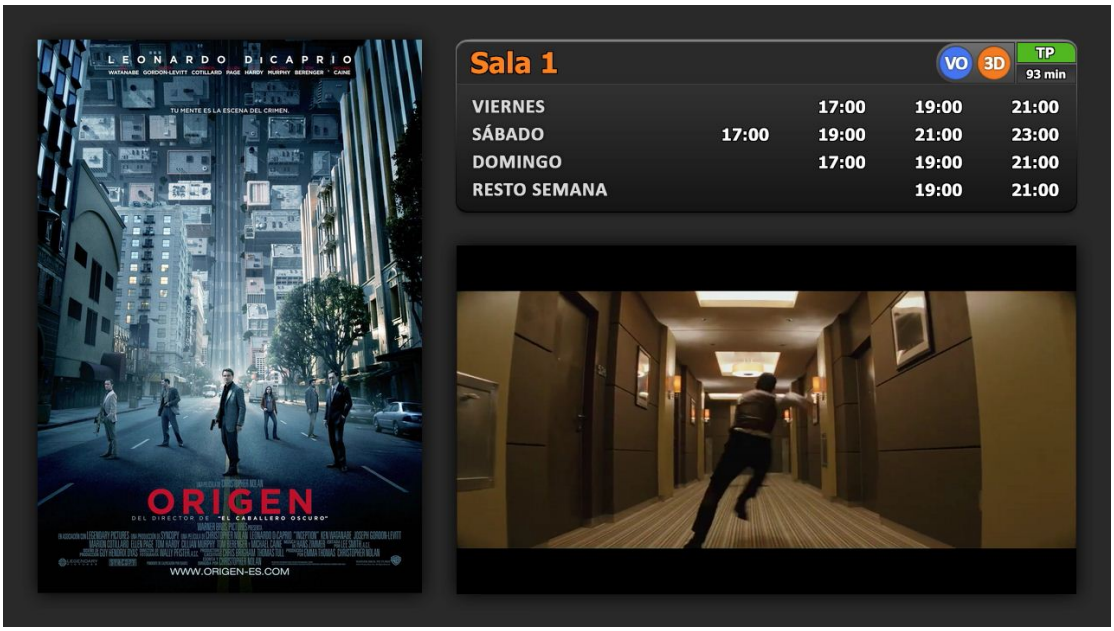


Ilustración 5 | ***Captura resultante del script de AVISynth anterior***

En el script, el primer bloque define las variables que usará el propio script: las rutas locales de los recursos (que deberán ser descargados desde el portal), el título y demás información del contenido (límite de edad, duración, si es 3D, etc), así como las sesiones por día. Toda esta información será cambiante en cada script, y será el portal el que deberá rellenar estas primeras líneas del script para cada vídeo que el Media Server deba generar, según la información programada en cada sala de cada cine en el portal.

El resto de código de cada plantilla será estático, no cambiará en función del contenido. Estará preparado para las diferentes opciones que pueda plantear cada plantilla, de manera que el portal, a la hora de preparar el contenido que deberá generar el Media Server, tendrá que rellenar los primeros parámetros (variables) de cada script.

VirtualDub, pruebas de compresión y ejecución por línea de comandos

Una vez tenemos el script de AVISynth (archivos con extensión .AVS), las pruebas a realizar son las de generación del archivo de vídeo resultante, archivo que debe ser compatible con la lista de estándares de vídeo compatibles con los *players* de Optoma (la lista se encuentra unas páginas antes, donde se comentan las características de dichos *players*).

Para la selección del estándar que se usará en la generación de vídeo, se optó directamente por uno de los más modernos y más extendidos actualmente (MPEG-4), concretamente las pruebas se realizaron con MP4, H.264 y DivX 5.0.

Los vídeos se generarán a una resolución de 720p y serán reproducidos en pantallas de 42" como mínimo, por lo que debemos buscar el equilibrio perfecto entre la mínima pérdida de calidad posible, y un óptimo tiempo de generación y un peso del archivo resultante lo más ajustado posible. Estos dos últimos puntos son muy importantes, por que la carga de trabajo del Media Server será de bastantes vídeos cada vez para todos los cines, y se prevé que el número de cines vaya aumentando, por lo que hay que optimizar al máximo el tiempo de generación. De igual manera, el tamaño de los archivos resultantes es importante, aunque se contará con una gran conexión de red, no es lo mismo transferir por FTP archivos de 10MB que de 100MB.

H.264, el códec escogido

Tras varias pruebas y mediciones realizadas con los tres códecs preseleccionados (MP4, H.264 y DivX 5.0) no se han apreciado grandes diferencias en cuanto tiempos de generación (estamos hablando de entre 10-20 segundos depende del nivel de compresión en una máquina de 4 cores, que podría correr cuatro VirtualDub a la vez, y 6GB de RAM). No se han apreciado grandes diferencias en cuanto a tiempos, pero con el códec H.264 se percibe que con una compresión alta, la perdida de calidad es menor que con los otros dos.

Las pruebas con H.264 y una compresión óptima, generaban vídeos de 1280x720 píxeles, de 20 segundos de duración, con un peso aproximado de **5MB** en unos **13 segundos** (en la máquina descrita anteriormente, obviamente estos tiempos son mejorables con mejor hardware).

Ejecución por consola de VirtualDub

VirtualDub ofrece la posibilidad de ejecutar generaciones de vídeo por línea de comandos, pasándole un archivo de configuración .VCF, dónde viene especificado el códec a usar, así como la compresión y otros parámetros (en la página siguiente se muestra el archivo de configuración usado), e indicando la ruta del script de entrada y la ruta del vídeo que se generará.

La característica con la que no contábamos y la cual nos facilitará el trabajo a la hora de la implementación de la lógica del Media Server, es que ofrece la posibilidad de crear una cola de trabajo y ejecutarla en el momento que queramos, pudiendo encolar grupos de trabajo según la carga de trabajo total, permitiendo definir diferentes formas de sincronización del sistema.

Ejecución simple

```
> vdub.exe /s vdub_conf.vcf /p "C:/script.avs" "C:/video.avi" /r
```

Cola de trabajo

```
> vdub.exe /s vdub_conf.vcf /p "C:/script1.avs" "c:/video1.avi"
> vdub.exe /s vdub_conf.vcf /p "C:/script2.avs" "c:/video2.avi"
> vdub.exe /s vdub_conf.vcf /p "C:/scriptN.avs" "c:/videoN.avi"
> vdub.exe /r
```

Los parámetros que se emplean en estas instrucciones:

- /s – para indicar la ruta del archivo de configuración
- /p – para indicar archivo de entrada y archivo de salida
- /r – para ejecutar la instrucción o la cola de trabajos actual
-

Archivo de configuración de VirtualDub / **vdub_conf.vcf**

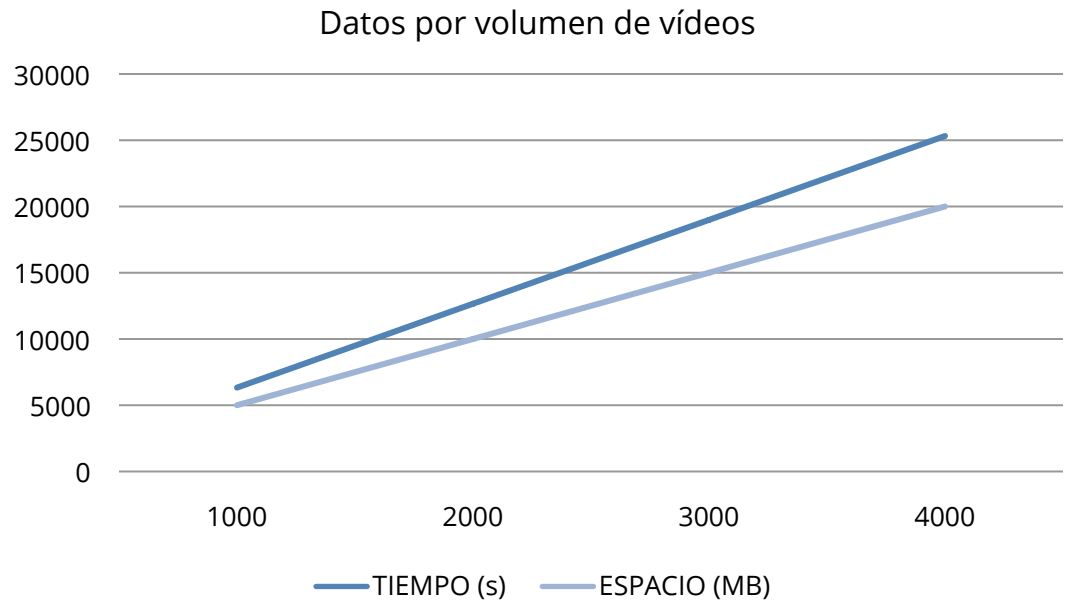
```
VirtualDub.audio.SetSource(1);
VirtualDub.audio.SetMode(0);
VirtualDub.audio.SetInterleave(1,500,1,0,0);
VirtualDub.audio.SetClipMode(1,1);
VirtualDub.audio.SetConversion(0,0,0,0,0);
VirtualDub.audio.SetVolume();
VirtualDub.audio.SetCompression();
VirtualDub.audio.EnableFilterGraph(0);
VirtualDub.video.SetInputFormat(0);
VirtualDub.video.SetOutputFormat(7);
VirtualDub.video.SetMode(3);
VirtualDub.video.SetSmartRendering(0);
VirtualDub.video.SetPreserveEmptyFrames(0);
VirtualDub.video.SetFrameRate2(0,0,1);
VirtualDub.video.SetIVTC(0, 0, 0, 0);
VirtualDub.video.SetCompression(0x34363278,0,10000,0);
VirtualDub.video.SetCompData(464,"AAAAACADAAAAAAAALlx4MjY0LnN0YXRzAAB
zAHQAYQB0AHMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAQAAAAEAAAAAAAAAAQAAAAEAAAAAAAAAgAAAAQAAAABAAAAEAAAAAEAAAABAAAAAQA
AAAAAAAAAAAAAAAAAAEAAAAAAAAAAQAAACgAAAAeAAAAPAAAAAEAAAABAAAAAQAAAAEA
AABIMjY0AAAAAAEAAAABAAAAAAAAAAAAAAA=");
VirtualDub.video.filters.Clear();
VirtualDub.audio.filters.Clear();
```

Pruebas de carga y conclusiones

En el momento de realización de las pruebas, se calculó que el número de vídeos que se generarían por semana, teniendo en cuenta los cines preparados para contar con el sistema y sus características, sería de unos 1000 vídeos diferentes. Por lo que se optó por hacer la primera prueba de carga con **2000 vídeos** a generar (el doble), para ver con que tiempos estábamos trabajando.

Las pruebas se realizaron con el PC anteriormente descrito (4 cores y 6GB de RAM), con 2 procesos simultáneos (en lugar de usar los 4 posibles).

Dicha generación tardó **3 horas y 31 minutos**, utilizando un espacio en disco de unos **10GB**. Estos datos de tiempo y espacio utilizado son lineales, por lo que las estimaciones variando el número de videos generados se pueden calcular de forma directa.



Pruebas de transferencia

Además de la generación se habilitó un FTP en Okode contra el que se simuló una subida al servidor FTP final del que obtienen las programaciones los Optoma, para evaluar el rendimiento de la red del cliente para este propósito.

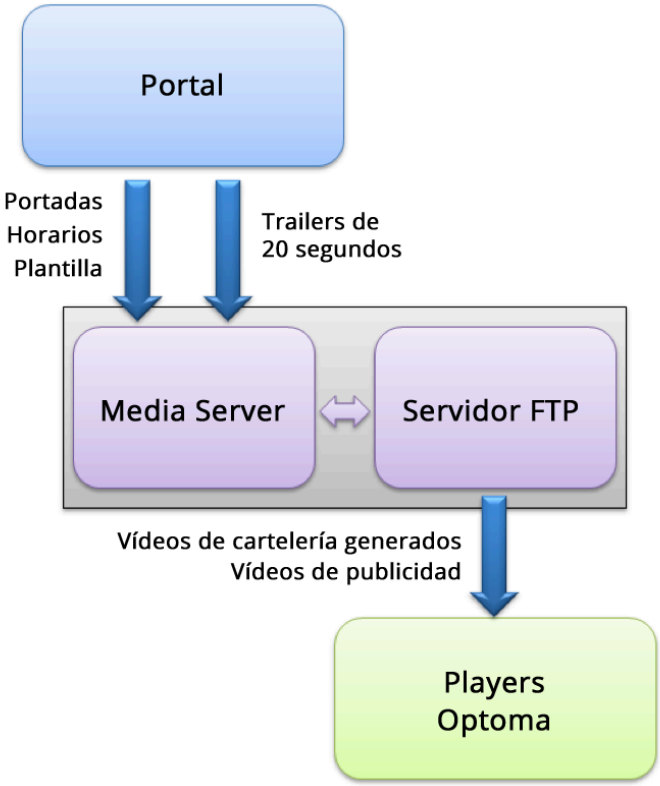
La subida de un único video (5MB) se realizó a las 14:10 horas, considerado un horario de baja utilización de la red de trabajo, y supuso un coste de **45 segundos** aproximadamente (tasa de transferencia: **116 KB/s**), lo que, extrapolando, supone un coste de **24 horas** y media aproximadamente para la subida de los **10GB** de videos generados en la prueba de carga.

Conclusiones

De estas primeras pruebas se obtiene un tiempo total aproximado de 28 horas, aunque aún es un tiempo obtenido por las primeras pruebas, sí que muestra unas considerables necesidades de tiempo, especialmente para la subida de los videos generados.

Pensando en la escalabilidad y fiabilidad del sistema, resulta un punto especialmente importante el coste temporal de la subida de los vídeos generados a un servidor que ofrezca por FTP las programaciones, ya que con esta primera prueba se aprecia que el tiempo es considerable y mayor tiempo supone más posibilidades de fallos de transferencia y más tiempo con la red del cliente colapsada, la cual, ya se mostró incapaz, durante las pruebas, de ofrecer navegación web y subida FTP simultáneas.

Lo que parece una conclusión clara es que el sistema, terminadas las pruebas pertinentes, deberá tener en una misma red local el componente del Media Server y el Servidor de FTP, ya que el intercambio de comunicación entre ellos es uno de los puntos más críticos y eliminarlo mejora en gran medida la solución.



6. Diseño e implementación

6.1. Servicios web en el portal, diseño e implementación

Introducción al diseño del portal

La lógica que se va a desarrollar en la parte del portal, se desarrolla como un módulo o componente añadido a la lógica ya existente.

El portal cuenta con una estructura de 3 capas:

- **Capa de presentación**
  - XHTML + CSS para la parte estática
  - JavaScript + Ajax para la lógica interactiva
- **Capa de negocio**.
  - Java EE 6
    - WebServices
    - EJBs
- **Capa de persistencia**.
  - PostgreSQL

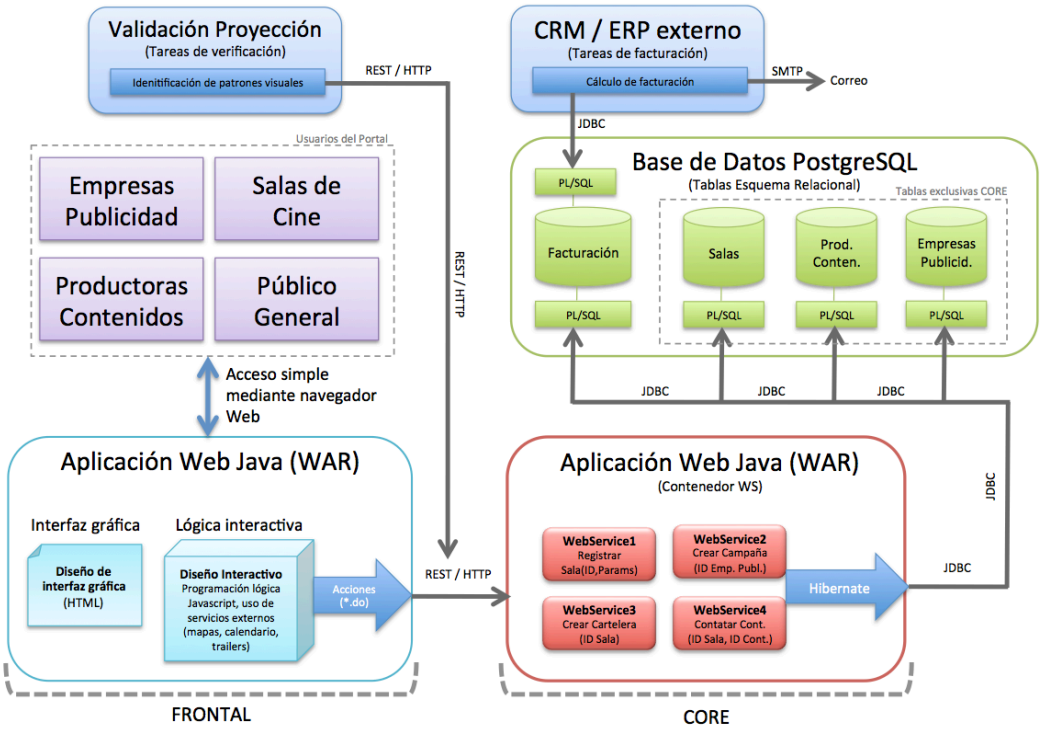


Ilustración 6 | ***Diagrama de componentes del proyecto de gestión de cines***

El portal cuenta con varios módulos con diferentes fines y con lógica separada, al cual se le añadirá uno nuevo con las funciones de cartelería.

Este módulo, estará integrado dentro del EJB Container de la aplicación Java EE 6**,** contará con varios **EJBs** con la lógica necesaria y varios **WebServices** para exponer y recibir la información necesaria por HTTP (para la comunicación remota con el Media Server), que pasará a formar parte de la capa de negocio del portal. Sin embargo, también habrá que realizar cambios en las otras dos capas para adaptar el sistema a estas nuevas funcionalidades. En la capa de presentación, se deberán maquetar nuevas páginas y formularios para servir de interfaz a la configuración del sistema de cartelería. Y, lógicamente, habrá que ampliar el esquema de la Base de Datos con las nuevas tablas y relaciones que harán falta.

¿Qué es un **EJB**?

Los EJB (Enterprise JavaBean) proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJB es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

¿Qué es un **WebService**?

Un servicio web (en inglés, Web service) es una pieza de software que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Componente de cartelería digital en el portal

Primero, se van a definir la lógica que se tiene que desarrollar en este componente, que se va a separar en dos grandes bloques:

- **Sistema de plantillas**
- **Sistema de publicidad**

Podrían considerarse dos módulos que funcionan por separado, puesto que el sistema de plantillas, es el encargado de montar el script con la plantilla seleccionada por los cines y con la información programada en sus salas; y el sistema de publicidad es independiente, no es gestionado por los cines, sólo por los anunciantes y funciona por separado. Será la lógica global del sistema de cartelería el que, a la hora de construir el "paquete de trabajo" que el Media Server procesará, unirá las partes para generar la parrilla de cartelería que corresponda a la programación de los cines, y la publicidad que deberá aparecer.

El portal, deberá preparar toda la información que el Media Server necesite para la generación de la cartelería. Únicamente creará un paquete comprimido con los scripts de las plantillas y una serie de información en XML de los recursos que necesita. Este "paquete" lo llamaremos **tarea**, y se explicará con detalle más adelante.

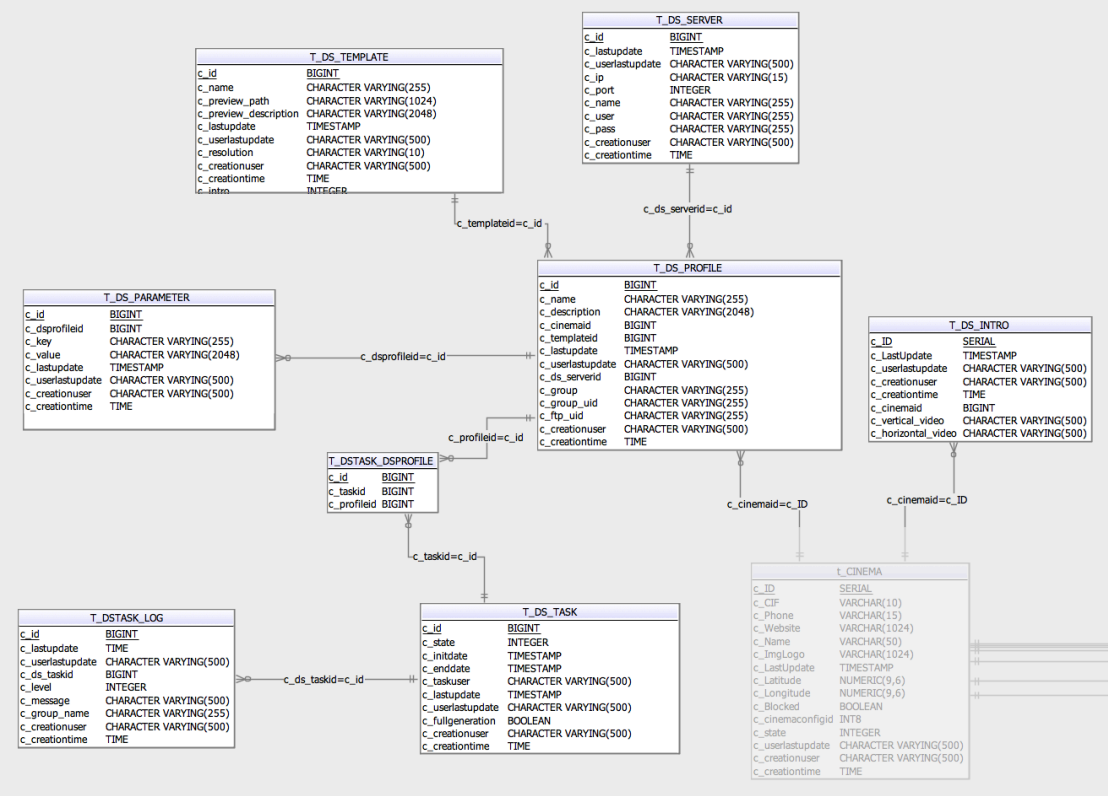
Sistema de plantillas

Cines, Perfiles y Plantillas

El sistema de plantillas, debe permitir a los cines, determinar que tipo de plantilla aparece en sus pantallas. En el análisis inicial se contaba con esto, pero según el proyecto fue madurando, se necesitó introducir el concepto de perfiles de cartelería en los cines. De manera que, un único cine puede disponer de diferentes perfiles con una plantilla asociada, cada uno de ellos para uno o varios *players* Optoma con su grupo de pantallas conectado, para dotar de diferente cartelería digital según la ubicación de las pantallas.

Por ejemplo, separar las pantallas del hall y las de las taquillas; o para cines muy grandes, separar uno por sectores la programación de la sala 1 hasta la N, y de la N+1 hasta la M.

Modelo en base de datos



Se han añadido 8 nuevas tablas en la BD para la gestión de las plantillas:

- **T_DS_PROFILE**: representa los perfiles, directamente asociados con el cine.
- **T_DS_TEMPLATE**: representa las plantillas, que tienen los scripts parametrizables, y que se asocian a los perfiles.
- **T_DS_PARAMETER**: representa los parámetros que pueden tener asociados los perfiles, dependiendo de su tipo.
- **T_DS_SERVER**: representa los diferentes servidores con los que cuenta el sistema para la transferencia FTP, cada perfil de cada cine se asociará manualmente a un servidor decidido por el administrador.
- **T_DS_INTRO**: corresponde al vídeo de cartelería que se iniciará como introducción a la reproducción de la cartelería y que será totalmente parametrizable con la imagen corporativa de cada cine. Por motivos de prioridades, el desarrollo de este punto se ha pospuesto para la versión siguiente.
- **T_DS_TASK_DSPROFILE**, **T_DS_TASK** y **T_DS_TASK_LOG** son las tablas necesarias para la gestión de las tareas, que explicaremos más adelante.

Asociación de perfiles a los cines

Los cines que han adoptado el sistema de cartelería digital en sus cines, deben tener asociado al menos un perfil para mostrar en todas sus pantallas. U optar por separar por grupos sus pantallas y agrupar por perfiles que se asocian directamente a sus players (cada grupo de pantallas que emiten la misma programación necesita un *player*).

La gestión de los perfiles se realiza por parte de los administradores del sistema de forma fácil a través del portal.

Ilustración 7 | ***Listado de perfiles y formulario de alta de nuevo perfil en el portal***

El *nombre* y *descripción*, no son más que descriptivos. Los parámetros de *Servidor*, *Grupo* y *Subruta FTP* son referidos al servidor FTP donde se cargará la cartelería del cine, y es una decisión técnica (normalmente geográfica). Y el parámetro ***Plantilla*** es un selector con las diferentes plantillas scriptadas, ya diseñadas, con las que cuenta el sistema.

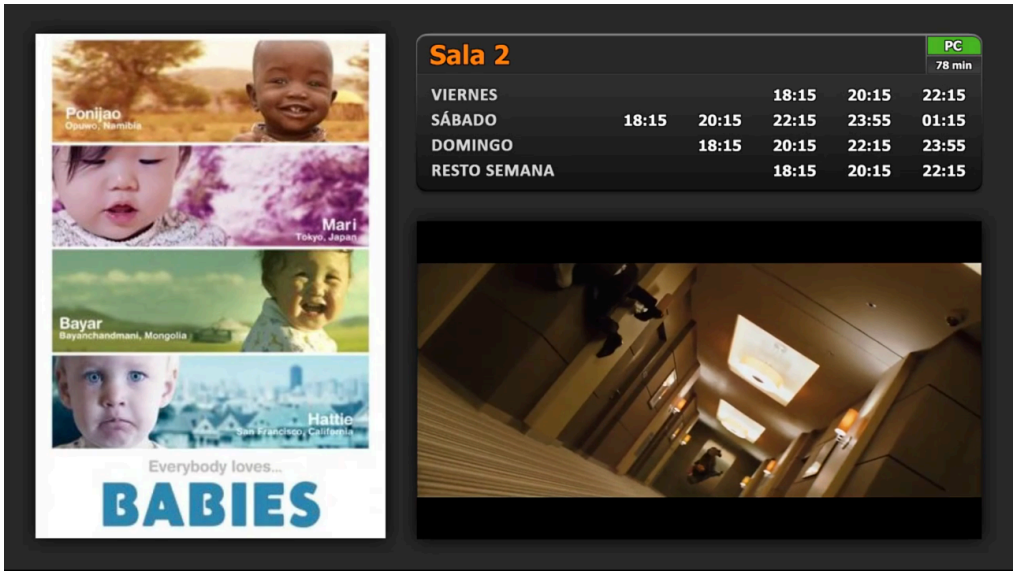
Plantillas disponibles

Las plantillas tienen todas la misma resolución (1280x720px) y pueden ser de diferentes tipos:

- **Horizontal** o **Vertical**. Las plantillas verticales no son realmente tal, se gira 90 grados el texto, imágenes y vídeos, para que en las pantallas puestas en vertical se vea bien el contenido.
- De un **contenido** (película) **y sala** (que suelen incorporar el tráiler y la portada) o de la **programación de varias salas** (que suelen llevar el nombre de la película, la sala y las horas en las que se emite)
- Semanales, diarias, de 2 días, … Aunque actualmente, se trabaja sólo con plantillas semanales.
- Plantillas con vídeos de publicidad integrados o no. Las plantillas que no tienen insertado la publicidad en su interior, se intercala entre los vídeos de cartelería.

Hasta la fecha (verano 2011), el sistema cuenta con 3 plantillas en marcha y funcionando perfectamente, a continuación se muestra una captura y breve descripción de cada una de ellas:

1Pase-ContenidoSala-Horizontal

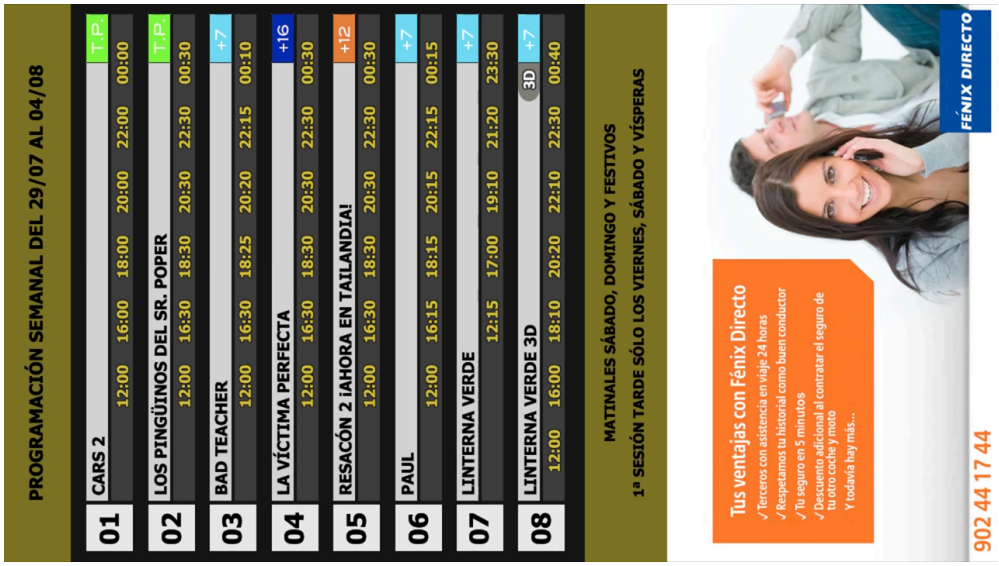


- **HORIZONTAL**. Muestra una película en una sala
- 6 sesiones máximas por día
- Programación semanal en 4 filas (viernes, sábado, domingo, resto)
- Tráiler y portada
- Edad recomendada por colores, también muestra si es en 3D o VO

8Pases-ParrillaCine-Horizontal

- **HORIZONTAL**. Muestra 8 películas/sala
- 5 sesiones máximas por día
- Programación semanal en 4 filas (viernes, sábado, domingo, resto)
- Edad recomendada por colores, también muestra si es en 3D o VO
- **Anuncio insertado** en la plantilla

8Pases-ParrillaCine-Vertical



- **VERTICAL**. Muestra 8 películas/sala
- 7 sesiones máximas por día
- Programación semanal en 1 fila
- Permite mensaje para aclarar diferencias de programación entre días
- Edad recomendada por colores, también muestra si es en 3D o VO
- **Anuncio insertado** en la plantilla

¿Cómo se crean las plantillas con la programación de los cines desde el portal?

El portal cuenta con unas plantillas de AVISynth (en el capítulo de **Estudio y análisis técnico** se documentó un ejemplo de script) donde se han parametrizado las variables, de forma, que cuando el portal reciba instrucciones de generar la cartelería (de forma manual o programada), consulte de que cines tiene que generar las plantillas, compruebe sus perfiles, lea y procese las plantillas asociadas, y compruebe que información necesita para rellenar esas plantillas. Al procesarlas, irá realizando consultas a base de datos, para saber los contenidos y sus programaciones en las salas del cine hasta completar la información de la plantilla.

Cuando se han creado las plantillas pertinentes, el portal se encargará de *empaquetar* los scripts, junto con otra meta-información en ***tareas*** par que el Media Server realice la generación de los vídeos de cartelería. Más adelante, se explica con más detalle el objetivo de las *tareas*.

1Pase-ContenidoSala-Horizontal.avs / Antes de ser procesada

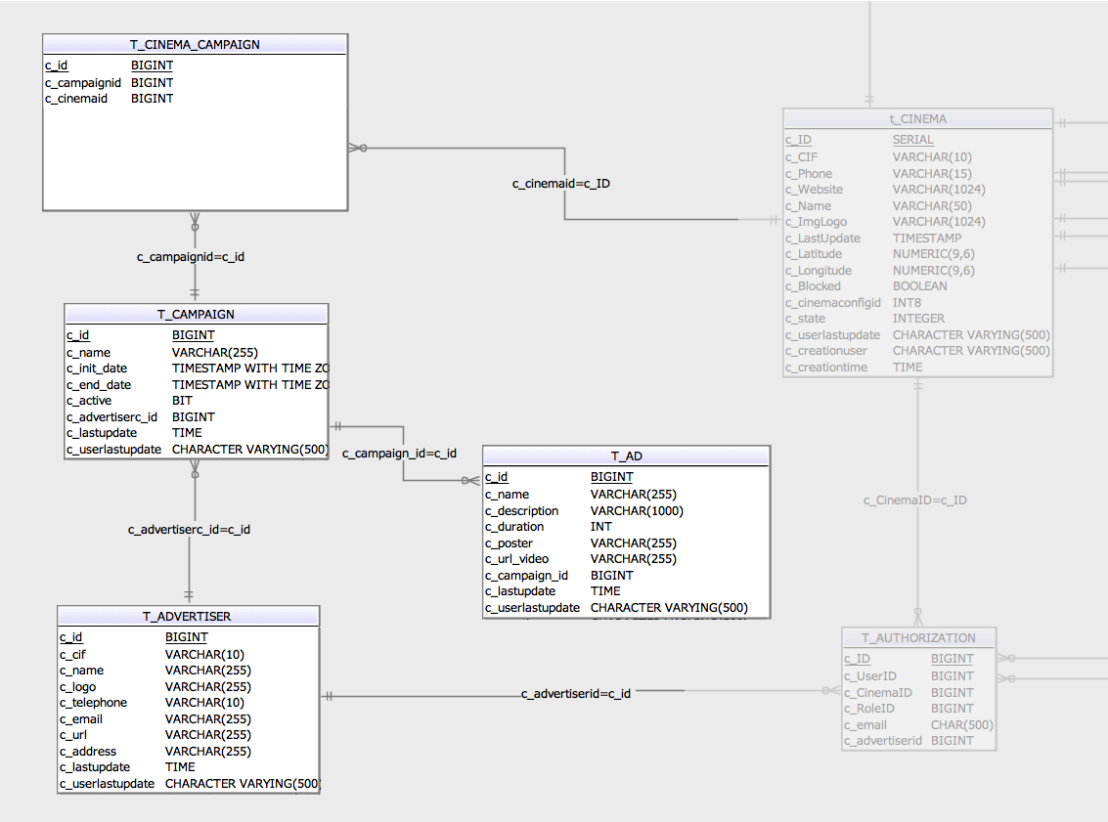
```
varFondo = "@varFondo"
varTrailer = "@varTrailer"
varPoster = "@varPoster"
varSala = "@varSala"
varDay1Sessions = "@varDay1Sessions"
varDay2Sessions = "@varDay2Sessions"
varDay3Sessions = "@varDay3Sessions"
varDay4Sessions = "@varDay4Sessions"
varAgeLimit = "@varAgeLimit"
varDuration = "@varDuration"

LoadPlugin ("C:\work\CarteleriaDigital\ffms2.dll")
trailer = FFVideoSource (varTrailer, colorspace="YUY2", width=716,
height=403, resizer="BICUBIC")
poster = ImageSource(varPoster,
pixel_type="RGB32").ConvertToYUY2(matrix="Rec709",
interlaced=false).Lanczos4Resize(444, 640)
background = ImageSource(varFondo, pixel_type="RGB32", fps=24, start
= 0, end = 480).ConvertToYUY2(matrix="Rec709", interlaced=false)
background.Layer(trailer, op="add", level=256, x=524, y=278,
threshold=0, use_chroma=true).Layer(poster, op="add", level=256,
x=40, y=40, threshold=0, use_chroma=true)

...

FadeIn(12).FadeOut(12)
```

Sistema de publicidad

El sistema permite la creación, por parte de los anunciantes, de campañas publicitarias. Los anunciantes dan de alta sus campañas contratando una serie de impactos (emisiones que tendrán en las pantallas de los cines). Estas campañas pueden definir su alcance: a nivel nacional, local, en ciertos cines concretos, o por circuitos de cines definidos en el portal.

En sí, la gestión de publicidad, no ha generado muchos problemas. La parte más compleja ha sido definir el algoritmo para el loop de publicidad y que el sistema lo supiera gestionar correctamente para cada cine, según los impactos contratados por cada anunciante.

Modelo en base de datos



Se han añadido 4 nuevas tablas en la BD para la gestión de la publicidad:

- **T_ADVERTISER**: representa los usuarios del portal con el rol adecuado para la gestión de la publicidad. Dan de altas campañas.
- **T_CAMPAIGN**: representa las campañas dadas de alta.
- **T_AD**: representa el anuncio que toda campaña incorpora.
- **T_CINEMA_CAMPAIGN**: tabla que guarda la relación establecida por parte de las campañas con los cines, según el ámbito y los impactos contratados por el anunciante.

Creación/edición de campañas

A continuación se explica la interfaz diseñada para que los anunciantes puedan dar de alta las campañas y contraten ciertos impactos. El formulario web de alta, cuenta con 2 pasos.

Primer paso

En el primer paso se introducen los **datos de la campaña**: nombre de la campaña y anunciante (la gestión de anunciantes es otro formulario independiente) los **datos del anuncio**: imagen/logo estática, y los vídeos del anuncio en 2 formatos, vertical y horizontal para los diferentes perfiles de cartelería, y una descripción del anuncio. Y por último, la **duración de la campaña**, cuyo requisito es que se contrata por semanas completas (empezando por los viernes, que es cuando se renueva la programación de los cines con los estrenos de la semana), se puede contratar de 1 a N semanas.

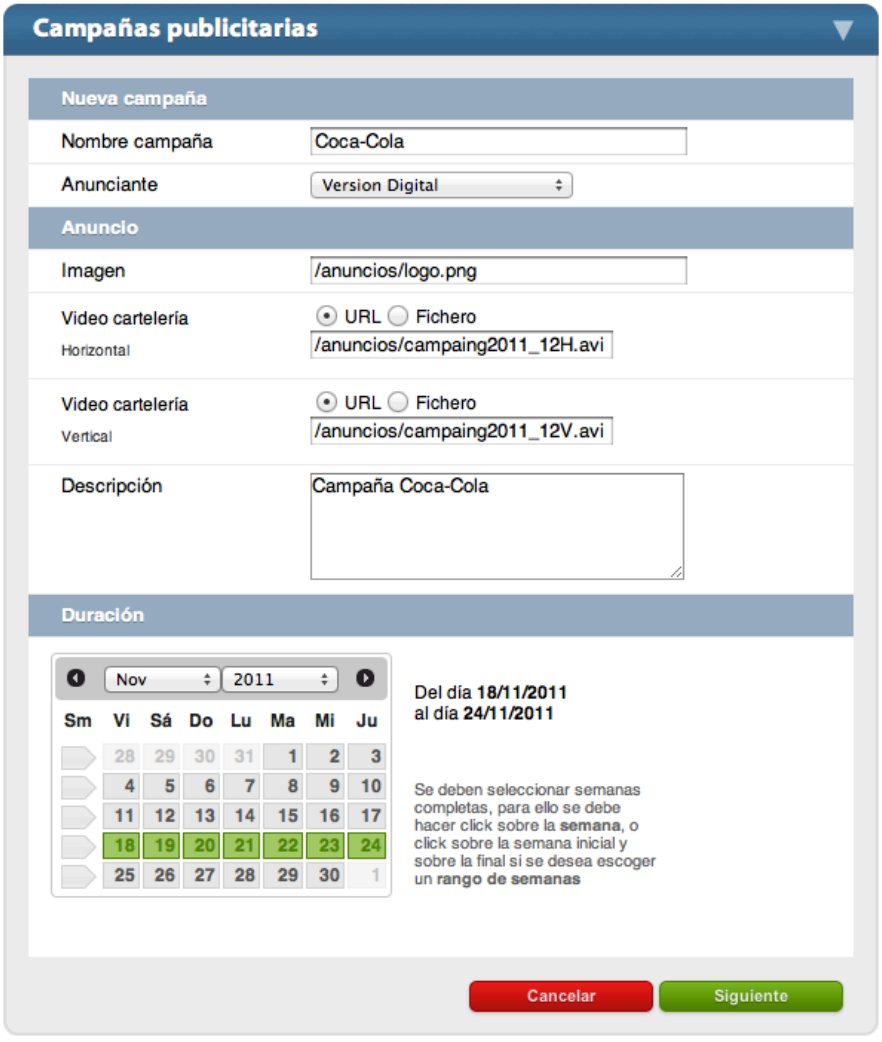


Ilustración 8 | ***Captura del primer paso del formulario de alta de campaña***

Segundo paso

En este segundo y último paso del formulario de alta de campaña, se debe indicar el alcance de la misma.

El alcance se mide en número de impactos por hora durante la duración de la campaña. El ámbito puede ser:

- **Nacional**
  El número de impactos contratado se repartirán de forma uniforme entre todos los cines de la plataforma.
- **Personalizado**
  Se podrán seleccionar cines concretos, filtrando por ciudad y seleccionándolos.
- **Por circuito**
  Se repartirán entre todos los cines de un circuito. Un circuito es una red de cines agrupados por algún motivo (geográfico, colaborativo o de algún tipo de organización).

Dependiendo del filtrado realizado, y de la disponibilidad de impactos libres por parte de esos cines, se dispondrá de un número máximo de impactos recomendado para que la distribución sea uniforme.

Ilustración 9 | ***Captura del primer paso del formulario de alta de campaña***

Formularios de gestión de anunciantes y circuitos

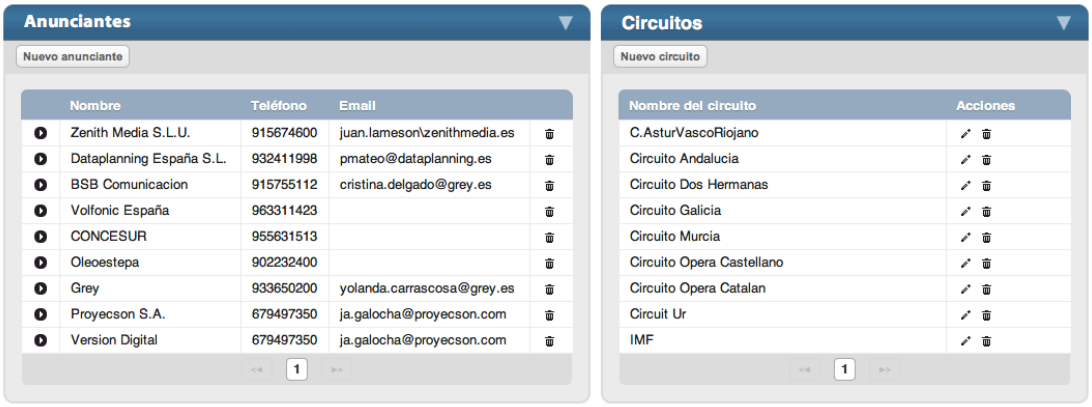


*Ilustración 10 | **Captura de los listados de anunciantes y circuitos en el portal***

Ambos módulos se muestran en el portal web como dos listados, con opciones de borrado y edición, así como un botón para el alta de nuevos, que muestra un sencillo formulario para rellenar.

Visualización del loop de publicidad

Una característica de administración importante que requería el portal, era la de tener una herramienta que permitiera ver el loop de publicidad programado para un cine en un día concreto, para que de forma visual, se facilitarán ciertas comprobaciones por parte de los administradores del sistema.

*Ilustración 11 | **Captura de la representación visual del loop de publicidad***

En la captura anterior, se muestra el loop de publicidad de un cine en un fecha concreta, y como se reparten las 90 impresiones por hora disponibles entre 5 campañas publicitarias.

A continuación se explica cómo se obtiene la secuencia del loop.

Cálculo del loop de publicidad

1) Obtener todas las campañas programadas en el cine para 1 día

Entradas: 1 cine, 1 día

Salidas: N campañas

2) Calcular el máximo común divisor de los impactos de todas las campañas

Entradas: todas las campañas

Salidas: m.c.d. (Integer)

3) Sumar el resultado de las divisiones entre los impactos de cada campaña con el m.c.d.

Entradas: todas las campañas

Salidas: loopSize (tamaño del loop de publicidad)

4) Aplicar algoritmo para crear un Array de tamaño loopSize en el que aparezcan las campañas en el orden en el que deben ser programadas.

Entradas: todas las campañas, loopSize

Salidas: ArrayList de tamaño loopSize con las campañas ordenadas (con los enlaces a los mismos objetos replicados cuando sea necesario)

Para la creación del array resultante se seguirán los siguientes pasos:

- Se crearán nº de campañas – 1 arrays.
- Se recorrerá la campaña con más impactos se insertará un impacto en cada uno de los arrays.
- Se recorrerán el resto de campañas de mayor a menor número de impactos y se insertará un impacto en la posición (2*númeroDePasada) – 1 (siempre que existan elementos previos, si no se insertan en la última posición)
- Una vez se hayan recorrido todas las campañas se recorrerán todos los arrays en orden para obtener el array resultante.

A continuación se explica con unos ejemplos:

Ejemplo 1

Campaña	**Impactos en loop**
A	5
B	1
C	2
D	4

3 arrays

- Insertamos impactos de la campaña con más impactos

A	A
A	A
A	

- Pasada 1, insertamos en las posiciones (2*1) – 1 = 1 los impactos de la campaña con más impactos (la campaña D)

A	D	A
A	D	A
A	D	

- Pasada 2: (2*2) – 1 = 3, continuamos con la campaña D y siguientes

A	D	A	D
A	D	A	C
A	D	C	

- Pasada 3: (2*3) – 1 = 5

A	D	A	D	B
A	D	A	C	
A	D	C		

- Recorremos los arrays para obtener la secuencia resultante

Secuencia resultante: A D A D B A D A C A D C

Ejemplo 2

Campaña	Impactos en loop
A	5
B	2
C	1

2 arrays

- Insertamos impactos de la campaña con más impactos

A	A	A	
A	A		

- Pasada 1, insertamos en las posiciones (2*1) – 1 = 1 los impactos de la campaña con más impactos (la campaña B)

A	B	A	A
A	B	A	

- Pasada 2: (2*2) – 1 = 3, continuamos con la campaña C

A	B	A	C	A
A	B	A		

- Recorremos los arrays para obtener la secuencia resultante

Secuencia resultante: A B A C A A B A

Ejemplo 3

Campaña	Impactos en loop
A	6
B	1

1 array

- Insertamos impactos de la campaña con más impactos

A	A	A	A	A	A

- Pasada 1, insertamos en las posiciones (2*1) – 1 = 1 los impactos de la campaña con más impactos (la campaña B)

A	B	A	A	A	A	A

Secuencia resultante: A B A A A A A

Ejemplo 4

Campaña	**Impactos en loop**
A	7
B	1
C	1
D	1

3 arrays

- Insertamos impactos de la campaña con más impactos

A	A	A
A	A	
A	A	

- Pasada 1, insertamos en las posiciones (2*1) – 1 = 1 los impactos de la campañas con más impactos (todas)

A	B	A	A
A	C	A	A
A	D	A	

- Recorremos los arrays para obtener la secuencia resultante

Secuencia resultante: A B A A A C A A A D A

Generación de la cartelería. Preparación de las tareas para el Media Server

El portal generará los scripts y listará los recursos necesarios en XMLs para que el Media Server genere los vídeos pertinentes y los empaquetará en lo que llamamos una ***tarea***. Este proceso, en principio, se realizará de forma automática semanalmente (aunque también se puede ejecutar de forma manual). Actualmente, el proceso se inicia la madrugada del miércoles al jueves para que rápidamente el Media Server lleve a cabo toda la generación y transferencia de los vídeos de cartelería, pudiendo disponer de esta forma de 24h (hasta el viernes, día en el que la programación de los cines suele cambiar con estrenos) para terminar su labor. De forma, que los cines, antes de que acabe cada miércoles, deben tener la cartelera de cada una de sus salas programada en su calendario web del portal.

A continuación se muestra una captura de un módulo del portal preparado como herramienta para los administradores del sistema que permite la generación manual de uno o varios cines, por si hubiera que corregir algún error o algún problema con la cartelería de un cine. Si se selecciona algún cine y se pulsa al botón *Generar cartelería*, el portal deja prepara una ***tarea***, para la próxima vez que el Media Server compruebe si hay alguna ***tarea*** para que la procese.

Ilustración 12 | ***Captura del módulo para la generación manual (administrador)***

¿A qué llamamos **TAREA**?

Llamamos tarea al paquete que el portal prepara con la información que necesita el Media Server para la generación de los vídeos. Se trata de archivos ZIP (nombrado por el identificador de dicha tarea) con los scripts y archivos de configuración (nada de recursos gráficos, vídeos, etc.), que el Media Server descargará por HTTP desde el portal.

Pasos para la descarga de una tarea

1) El **portal** ha dejado preparado una tarea (tarea 44).
2) El **Media Server** pregunta por HTTP (WebService) al portal si hay tareas pendientes (con estado DOWNLOAD_***PENDING***). Está consulta se hace cada X minutos (configurable).
3) El **portal** responde con la lista de los ids de las tareas pendientes de descarga. En este caso la respuesta es **"44"**.
4) El **Media Server** descarga la tarea desde la URL dónde el **portal** sirve las tareas comprimidas. Ej: *http://www.urldelportal/.../tasks/**44.zip***
5) En caso de que la tarea se haya descargado con éxito, el **Media Server**, mediante http notifica al portal que ha sido descargado con éxito, y que a partir de ahora, la tarea es responsabilidad suya. De manera que el portal, marcará la tarea como ***DOWNLOADED***.

Tanto PENDING como DOWNLOADED son dos estados con los que se marcan las tareas, para obtener un *feedback* entre Media Server y portal durante el proceso de generación de una tarea.

```
IN_PROGRESS = 0,
SCRIPT_GENERATION_ERROR = 1,
DOWNLOAD_PENDING = 2,
DOWNLOADED = 3,
VIDEO_BUILD_PENDING = 4,
VIDEO_BUILD_ERROR = 5,
VIDEO_BUILD_READY = 6,
COMPLETE = 7,
OPTOMA_UPDATED = 8,
OPTOMA_UPDATED_FAILS = 9,
OPTOMA_UPDATED_WITH_ERRORS = 10
```

Los 3 primeros estados, corresponden a los primeros pasos del ciclo de vida de la tarea y son estados con los que el portal puede actualizar una tarea. El resto, corresponde al resto del proceso de generación por parte del Media Server, que se explicará más adelante.

Estructura de una tarea

A continuación se va a explicar, con un ejemplo, la estructura de una tarea generada por el portal una vez descomprimida, para entender qué es lo que el Media Server espera encontrar.

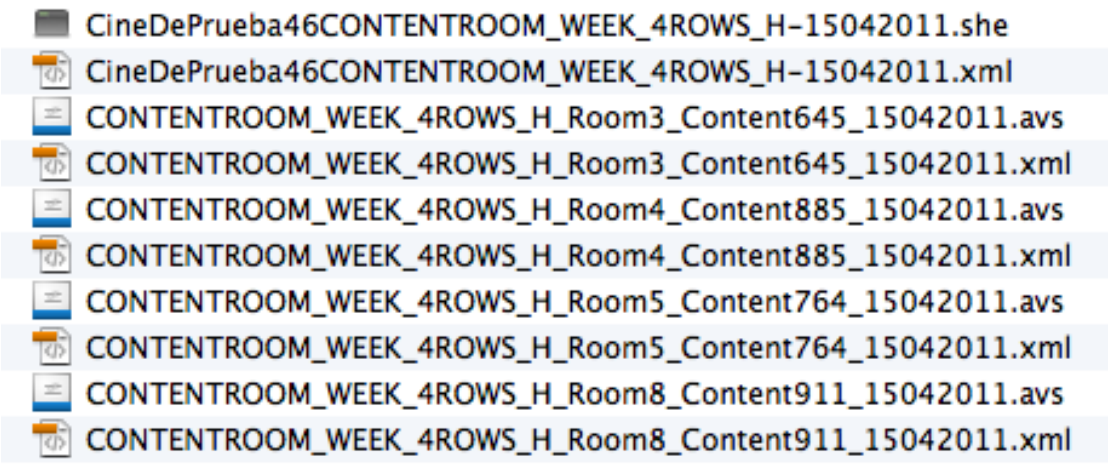


*Ilustración 13 | **Captura del contenido de una tarea sencilla***

Lo que se muestra arriba es el contenido de una tarea muy sencilla. En este caso se ha generado sólo la cartelera de un cine (aunque podrías ser muchos en el mismo paquete) con un perfil asociado y con sólo 4 vídeos de cartelería.

En este ejemplo, se ha preparado la cartelería para el cine llamado **CineDePrueba**, con el id **46** y el perfil **CONTENTROOM_WEEK_4ROWS_H** para la semana del 15 de abril de 2011. Los dos primeros archivos (she y xml) son de configuración para el *player* de Optoma.

El archivo .SHE es un archivo con formato de XML que parametriza ciertos aspectos para la reproducción, que será siempre igual, excepto el parámetro que determina el archivo XML que contiene la secuencia de la escaleta de vídeos que se reproducirán (en este caso los 4 que se generarán, más la publicidad intercalada).

El resto de archivos son los pertenecientes a los vídeos a generar por el **FrameServer** del Media Server. Como se puede observar, hay 4 archivos AVS con su XML asociado con el mismo nombre (en el nombre se indica el nombre de la plantilla del perfil, el ID de la sala, el ID del contenido y la fecha).

Los archivos AVS son los scripts ya preparados para la generación de vídeo, y los XML listan los recursos necesarios para tal vídeo (portada, tráilers, anuncios, recursos gráficos de la plantilla, etc.). En el siguiente apartado se explica el procesado que el Media Server realiza sobre estos archivos.

6.2. Media Server, diseño e implementación

Introducción

Se trata de un servidor únicamente dedicado a la generación de la cartelería digital. Concretamente es un PC con Windows 7 como Sistema Operativo, en el cual correrá la aplicación que se ha desarrollado con el objetivo mencionado, además de otras herramientas que se requieren en el proceso y se comentan más adelante.

Esta aplicación se ha desarrollado con C# .NET en Visual Studio 2010, y, aunque se puede ejecutar de forma manual (mediante aplicación de escritorio *Windows Forms*), ha sido desarrollada para ser ejecutada como una aplicación de servicio Windows, con funcionalidad para ser activada por red de forma sencilla (desde el portal).

¿Qué es un **Servicio Windows**?

Los servicios de Microsoft Windows, antes conocidos como servicios NT, permiten crear aplicaciones ejecutables de larga duración, que se ejecutan en sus propias sesiones de Windows. Estos servicios pueden iniciarse automáticamente cuando el equipo arranca, se pueden pausar y reiniciar, y no muestran ninguna interfaz de usuario. Estas características hacen que los servicios resulten perfectos para ejecutarse en un servidor o donde se necesite una funcionalidad de ejecución larga que no interfiera con los demás usuarios que trabajen en el mismo equipo. También puede ejecutar servicios en el contexto de seguridad de una cuenta de usuario específica, diferente de la del usuario que inició la sesión o de la cuenta predeterminada del equipo.

Instalación necesaria

Para la ejecución del Media Server y todo su proceso de generación de vídeo, la máquina que funcione que tal debe tener instalado y/o configurado:

- Windows (en este caso Windows 7)
- Framework .NET (ya instalado con Windows 7)
- Gran capacidad de espacio libre en el disco duro para la generación de un gran volumen de vídeos por iteración
- Conexión a internet (con gran ancho de banda)
- AVISynth (última versión estable 2.5.8)
- VirtualDub (última versión estable 1.9.11)
- Códec de vídeo H.264

Como ya se ha explicado con anterioridad, AVISynth es la herramienta con la que se scripta el vídeo, y VirtualDub la herramienta con la que se generará.

Estructura de la aplicación

La aplicación en sí, repite de forma constante el proceso de generación y subida de los vídeos al Servidor FTP. Estos vídeos que se tienen que generar vienen especificados en las tareas generadas en el portal, como ya se ha explicado con anterioridad en el diseño e implementación del mismo.

El proceso se puede simplificar a la hora de explicarlo en 5 fases:

- **Fase 1.** Comprobación y descarga de tareas pendientes
- **Fase 2.** Procesado y preparación de tareas
- **Fase 3.** Añadir vídeos a la cola de VirtualDub
- **Fase 4.** Ejecutar cola de trabajo de VirtualDub
- **Fase 5.** Sincronizar con el FTP

Aunque las fases son claramente secuenciales, debido a que la ejecución de la cola de trabajo de VirtualDub (fase 4) y la sincronización con el FTP (fase 5), consumen un gran tiempo del trabajo, y pudiendo ésta última bloquear el proceso durante horas innecesariamente ya que se podría estar generando una nueva tarea (si es que la hubiera) mientras tanto; se ha decidido trabajar con dos hilos de ejecución, uno para las 4 primeras fases (todo el proceso hasta la generación) y otro para la fase 5, sincronización con el FTP.

Estos dos hilos funcionarán de forma paralela y asíncrona, de forma que el primero irá comprobando si existen tareas y generando el contenido, y el segundo subirá el contenido cuando compruebe que el primero ha dejado algo generado, sin tener que esperar el uno al otro, simplemente comprobar si tiene trabajo. En caso de no tener trabajo que realizar, el proceso dormirá un tiempo (previamente parametrizado) hasta la siguiente iteración.

Los hilos en C# .NET se implementan por medio de la clase ***BackgroundWorker***, a continuación se adjunta el código de la inicialización de dichos hilos, una explicación de la propia clase y un diagrama del funcionamiento de los dos hilos implementados.

Creación de los dos hilos de ejecución en C# .NET

```
_mainProcess = new BackgroundWorker();
_mainProcess.WorkerSupportsCancellation = true;
_mainProcess.DoWork += new DoWorkEventHandler(LoopProcess);
_mainProcess.RunWorkerAsync();

_ftpUpload = new BackgroundWorker();
_ftpUpload.WorkerSupportsCancellation = true;
_ftpUpload.DoWork += new DoWorkEventHandler(UploadFtpProcess);
_ftpUpload.RunWorkerAsync();
```

Funcionamiento de la clase **BackgroundWorker**

La clase BackgroundWorker permite ejecutar una operación en un subproceso dedicado e independiente. La ejecución de operaciones que exigen mucho tiempo, como las descargas y las transacciones de las bases de datos, puede hacer que la interfaz de usuario deje de responder. Si desea una interfaz de usuario con gran capacidad de respuesta y debe realizar operaciones que exigen mucho tiempo, la clase BackgroundWorker proporciona una solución práctica.

Para ejecutar una operación en segundo plano, debe crear BackgroundWorker. Puede realizar escuchas de los eventos que notifican el progreso de la operación y que indican cuándo se completa la operación.

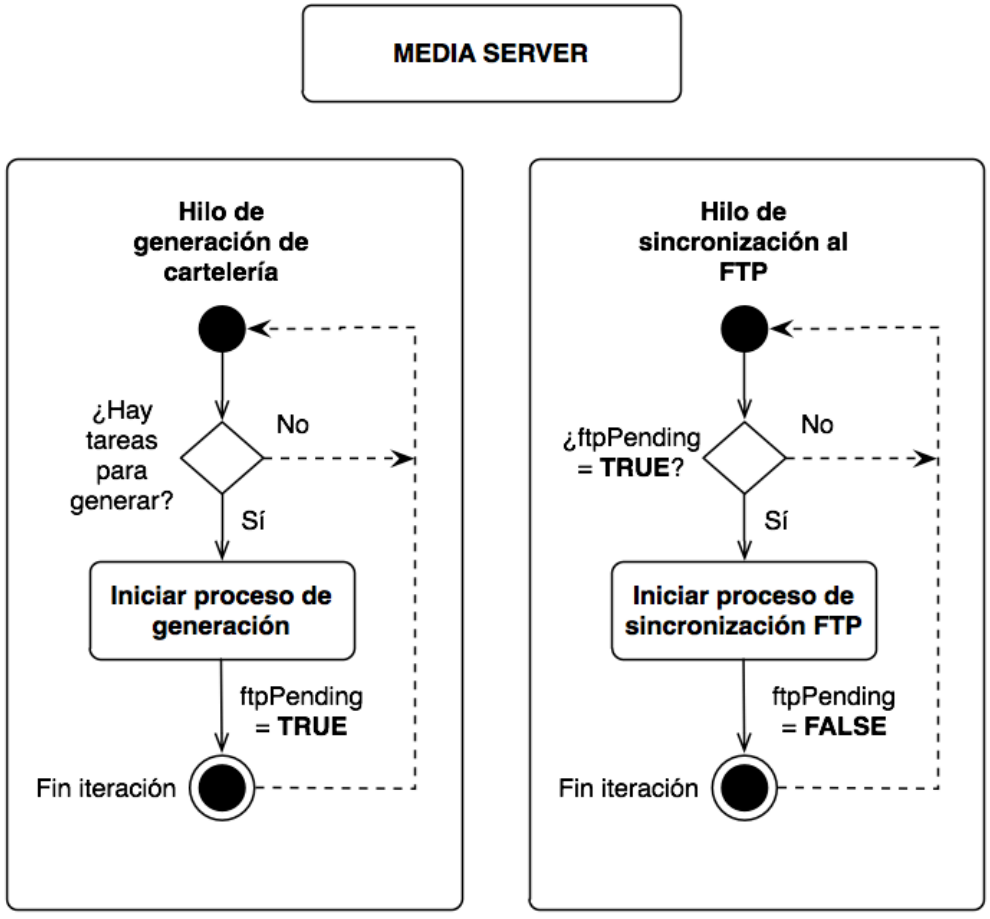


*Diagrama 1 | **Estructura de la aplicación - Hilos de ejecución***

En las páginas siguientes se explicará de forma conceptual y con algunos diagramas apoyados por pseudocódigo, la lógica de funcionamiento de cada una de las cinco fases en las que se ha dividido el proceso para su explicación.

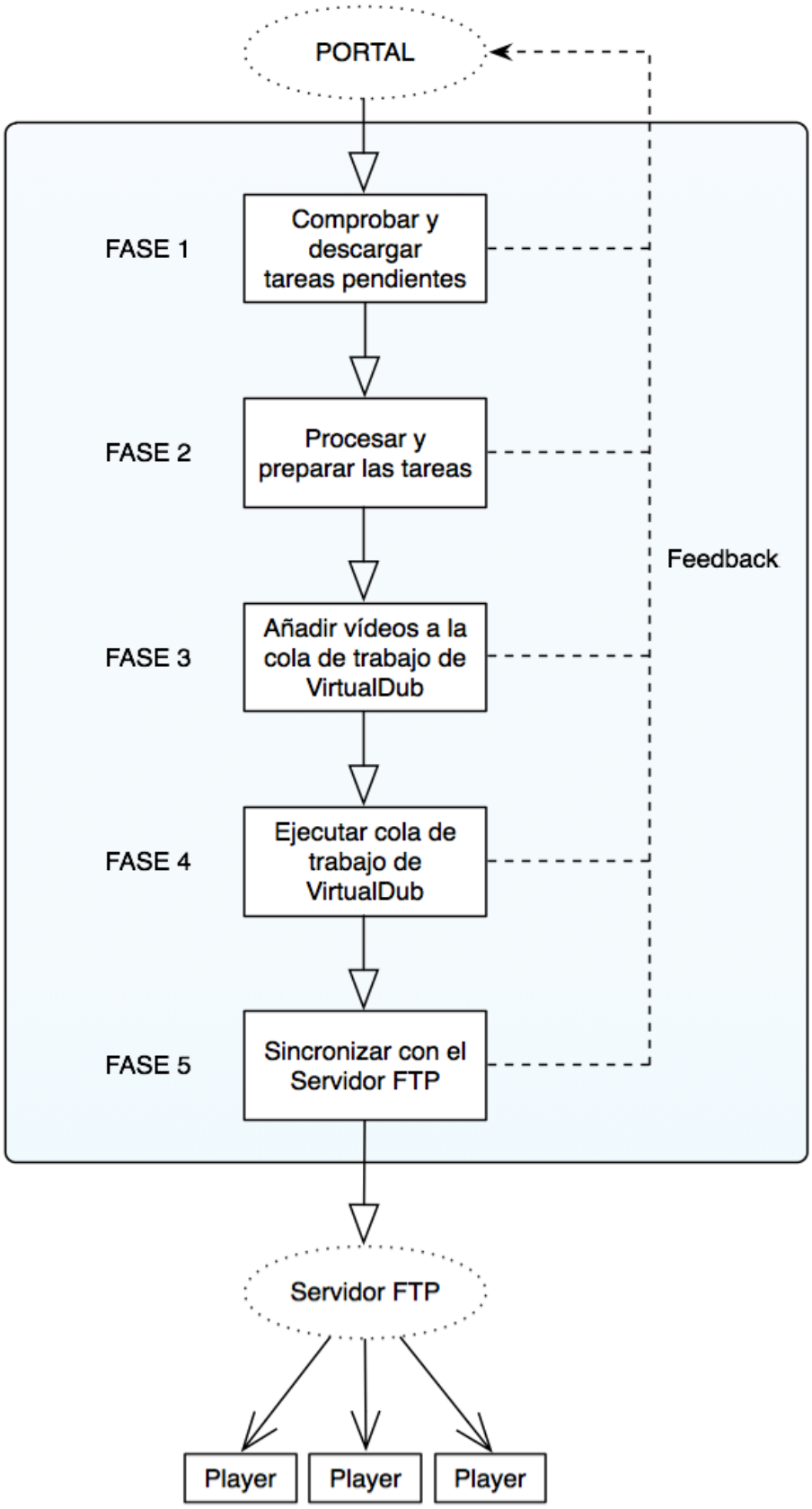


Diagrama 2 | ***Diagrama simplificado de la estructura del Media Server***

FASE 1. Comprobación y descarga de tareas pendientes

La primera fase de la iteración del proceso de generación de la cartelería digital es el de comprobación y descarga de tareas pendientes. En el momento de la comprobación puede haber **una**, **muchas** o **ninguna** tareas pendientes para generar en el portal.

Para la comprobación se hace un ***WebRequest*** al servicio web del portal que informa de las tareas que tiene preparadas y listas para descargar (en forma de lista de identificadores).

En caso de haber tareas, se descargan y se marcan (con otra *WebRequest*) indicando el id de la tarea en concreto y su nuevo estado ***DOWNLOADED*** a la espera de la siguiente fase. En caso de no haber ninguna tarea pendiente, el proceso acaba aquí, esperando la próxima iteración programada o hasta que se solicite manualmente.

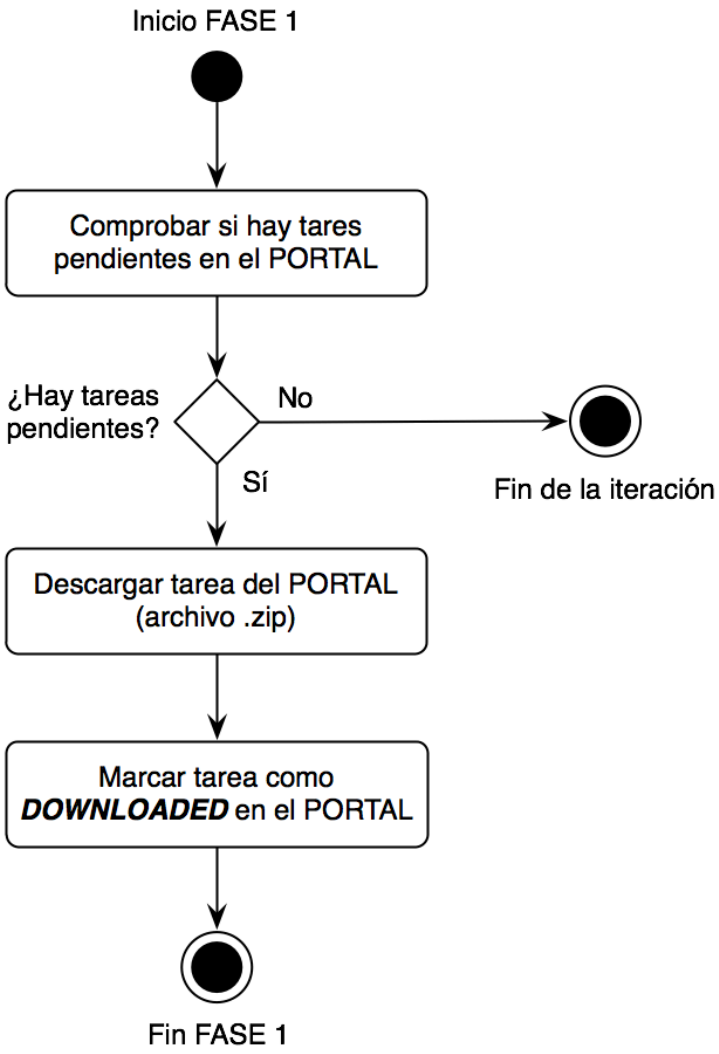


Diagrama 3 | ***Diagrama de Estado - FASE 1***

FASE1 / **APROXIMACIÓN SIMPLIFICADA del código para su comprensión**

```
List<int> pendingTasksList = CheckTasks();
if (pendingTasksList.Count == 0) {
  ...
  continue;
}
foreach(DSTask task in pendingTasksList) {
  currentTasks.Add(task);
  task.UpdateTaskState(TaskState.DOWNLOADED);
}
```

```
/// Returns List with pending tasks id, via WebRequest
private List<int> CheckTasks();
/// Update current task state, via WebRequest
public void UpdateTaskState(TaskState newState);
```

FASE 2. Procesado y preparación de tareas

Esta segunda fase consiste en la descompresión de las tareas que nos ha dejado preparadas el portal y su preparación, descargando los recursos que sean necesarios para cada una de ellas. El archivo comprimido de cada tarea, sólo lleva una serie de archivos de configuración y el script para generar el vídeo. **Cada script tiene asociado un XML** donde se especifican los recursos necesarios que necesita para su generación (posters, tráilers, vídeos de publicidad, imágenes de fondo de la plantilla, ...).

Estos archivos XML indican los recursos que se deben descargar para cada tarea, tanto su URL de descarga, como su path dentro del sistema de archivos, por lo que antes de descargar un recurso, el media server deberá comprobar si ya lo tiene (para evitar descargas inútiles y su pérdida de tiempo correspondiente). Por ejemplo, un tráiler o un póster de una película de estreno, será solicitado por muchas plantillas de muchos cines.

Después de comprobar que se han descargado o ya se tenían en local, se marcará la tarea en el portal como ***VIDEO_BUILD_PENDING***, si hubiera algún error en la tarea, como ***VIDEO_BUILD_ERROR***. Si fallasen todas las tareas de la iteración, se acabaría la misma.

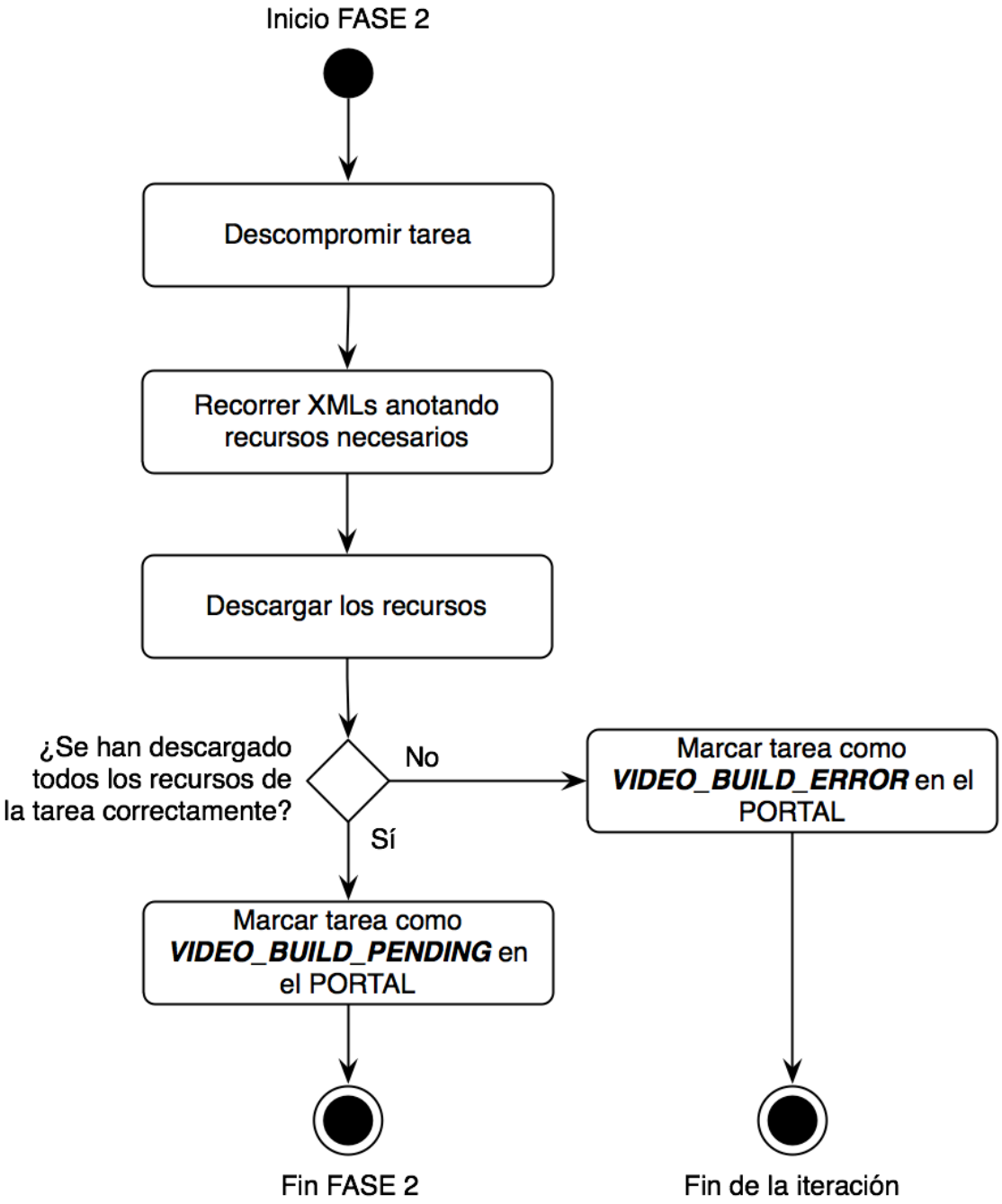


Diagrama 4 | ***Diagrama de Estado - FASE 2***

FASE2 / **APROXIMACIÓN SIMPLIFICADA** del código para su comprensión

```
foreach(DSTask task in pendingTasksList) {
   ...
   try {
      ProcessZipFile(task);
      foreach(string xml in GetXmlFilesFromPath(task.TaskFolder))
      {
         ...
         ProcessXmlFile(task, xml);
      }
      task.UpdateTaskState(TaskState.VIDEO_BUILD_PENDING);
   } catch(Exception e) {
      task.UpdateTaskState(TaskState.VIDEO_BUILD_ERROR);
      continue;
   }
```

```
/// Download and unzip task
private void ProcessZipFile(DSTask task);
/// Return List with xml paths of a task
private List<string> GetXmlFilesFromPath(string taskPath)
/// Process xml file and download resources specified in xml
private void ProcessXmlFile(DSTask task, string xmlFile)
```

FASE 3. Añadir vídeos a la cola de trabajo de VirtualDub

En esta tercera fase, antes de añadir los scripts a la cola de trabajo de VirtualDub (donde se encolarán todos los scripts .AVS para generar los archivos de vídeo pertinentes con toda la cartelería de las tareas), se comprobará y validarán los scripts, para tener la seguridad de que la preparación todo se ha realizado correctamente, y en caso de que se produjera un error a continuación, sería por algún problema en la ejecución de VirtualDub, pudiendo volver a recuperar el trabajo anterior.

Si se corrobora que todos los scripts están bien formados y que los recursos que necesitan los .AVS se han descargado (validando la información que suministraban los XML) se marcará en el portal la tarea como ***VIDEO_BUILD_READY***, y en caso de encontrase algún error como ***VIDEO_BUILD_ERROR***, finalizando así la iteración.

Como se ha comentado antes, en el caso de haberse marcado como ***VIDEO_BUILD_PENDING*** y fallase en la generación, se podrá repetir esta tarea desde la siguiente fase, omitiendo las anteriores.

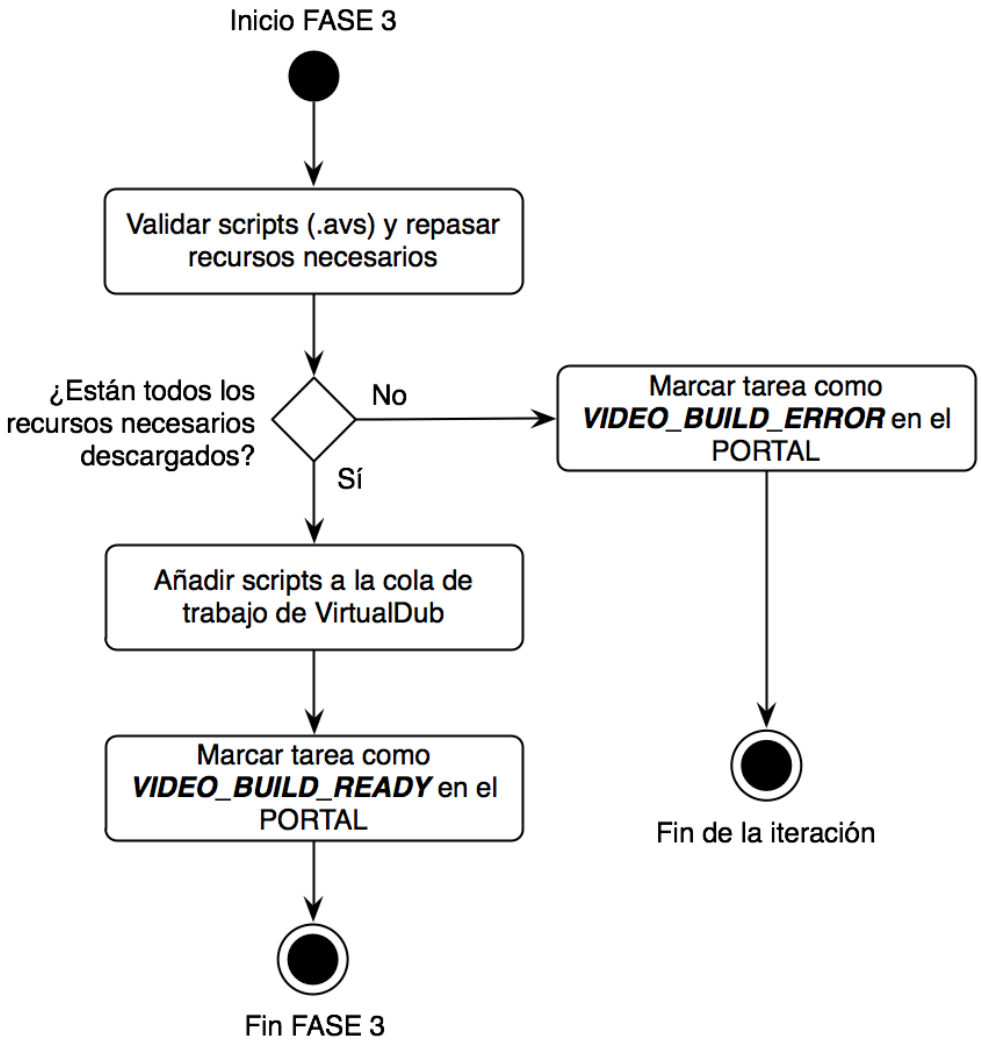


*Diagrama 5 | **Diagrama de Estado - FASE 3***

FASE3 / **APROXIMACIÓN SIMPLIFICADA** del código para su comprensión

```
foreach (DSTask task in currentTasks) {
   foreach (DSVideo video in task.Videos) {
      if(isAVSFileAndResourcesReady(video.AvsFile)) {
         AddVideoToVirtualDubJobList(video);
         task.UpdateTaskState(TaskState.VIDEO_BUILD_READY);
      } else {
         task.UpdateTaskState(TaskState.VIDEO_BUILD_ERROR);
         continue;
```

```
/// Checks if AVS and all resources are ready
private bool isAVSFileAndResourcesReady(string avsPath);
/// Adds video to VirtualDub jobList, via CmdLine
private void AddVideoToVirtualDubJobList(DSVideo video);
```

FASE 4. Ejecutar cola de trabajo de VirtualDub

Esta es la fase de la iteración de más riesgo, ya que se lleva a cabo la ejecución de la cola de trabajo de VirtualDub, y dependiendo de la carga de trabajo, puede ser de muchas horas.

En este punto, el hilo de ejecución que lleva a cabo la iteración (pueden ser múltiples hilos de ejecución, el sistema está optimizado para 4 hilos con 4 procesos de VirtualDub en paralelo), ejecuta la cola de trabajos de VirtualDub, esperando el hilo a la finalización del proceso y comprobando su *ExitCode* (VirtualDub devuelve 0 si ha podido acabar su ejecución) para saber si se ha ejecutado bien.

En el caso de haberse ejecutado correctamente, se comprobará el log de VirtualDub resultante de la ejecución, pues aunque el proceso haya acabado con éxito, puede que VirtualDub haya tenido algún problema con algún proyecto y no haya generado el vídeo. En ese caso, VirtualDub marca el vídeo con un *status* de ERROR, indicando el motivo, el cuál el sistema deberá recuperar para notificárselo al portal. En función del tipo de error que se produzca en este paso se podrá solucionar de forma manual, si a través de la notificación en el log del portal se descubre algún error de configuración.

En caso de haberse generado todo correctamente se marcará la tarea como ***COMPLETE***. En este momento, el sistema ya tendrá preparado toda la cartelería necesaria, así como sus anuncios publicitarios y resto de recursos para sincronizar con el FTP.

FASE4 / **APROXIMACIÓN SIMPLIFICADA** del código para su comprensión

```
foreach (DSTask task in currentTasks) {
   foreach (DSVideo video in task.Videos) {
      // execute cmdline process
      Process p = Process.start("c:/vdub.exe /p ...");
      if (p.WaitForExit() && p.ExitCode == 0) {
         if(ValidateVirtualDubJobsLog()) {
            task.UpdateTaskState(TaskState.COMPLETE);
         } else {
            task.UpdateTaskState(TaskState.VIDEO_BUILD_ERROR);
            continue;
         }
      } else {
         task.UpdateTaskState(TaskState.VIDEO_BUILD_ERROR);
         continue;
      }
   }
```

```
/// Checks VirtualDub log to check all is right
private bool ValidateVirtualDubJobsLog ();
/// Update current task state, via WebRequest
public void UpdateTaskState(TaskState newState);
```

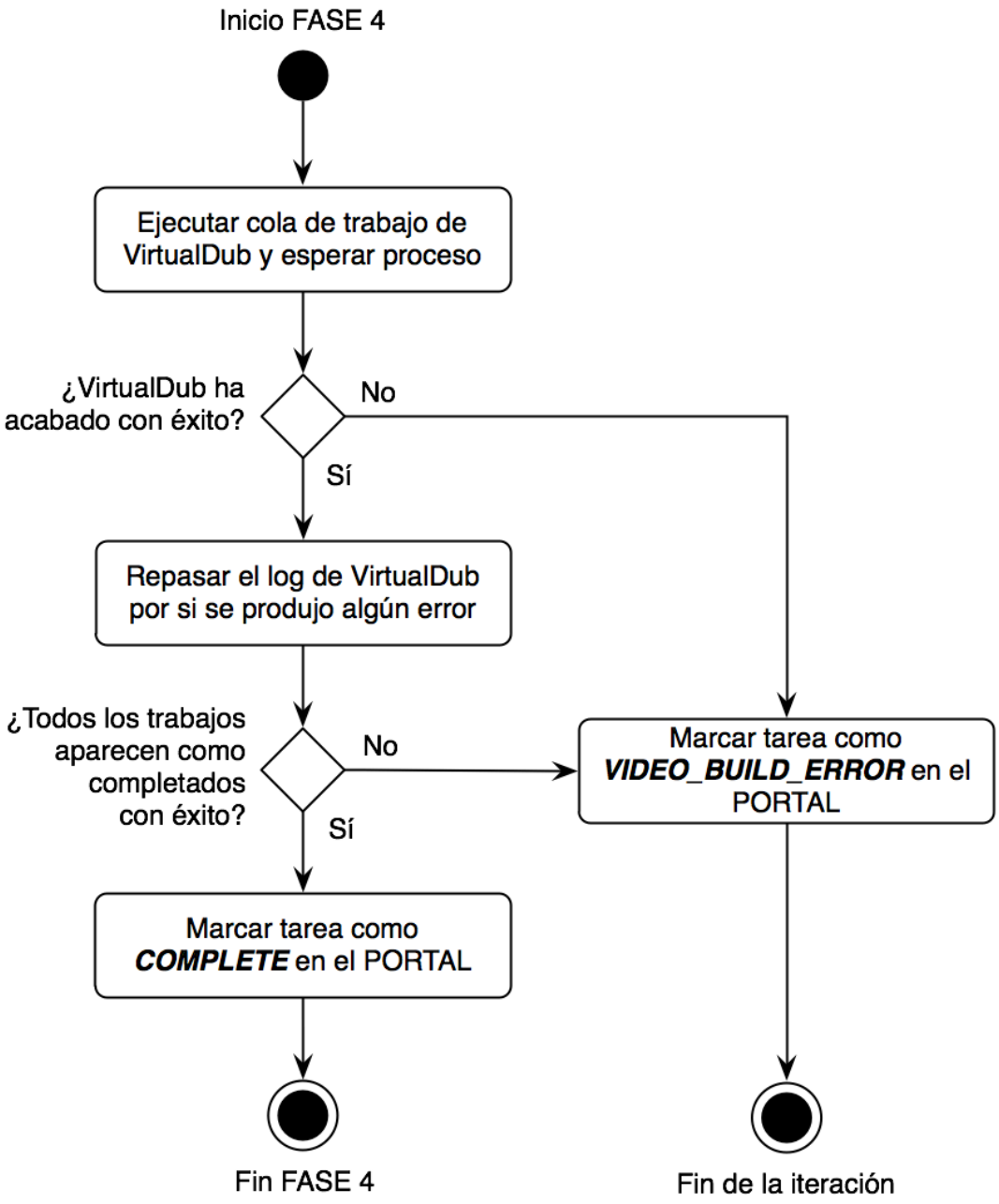


*Diagrama 6 | **Diagrama de Estado - FASE 4***

FASE 5. Sincronizar con el Servidor FTP

Esta quinta y última fase de la iteración del proceso es la encargada de sincronizar el Servidor de FTP (que como ya se ha comentado con anterioridad, está en la misma red local que el Media Server), al cual se conectarán los *players* de Optoma para descargar su cartelería asignada (conectándose cada uno a su *path* único dentro del servidor).

El primer paso consiste en la conexión al FTP, que no debería fallar puesto que se supone que el sistema va a estar siempre en marcha, en el caso de que por algún motivo no se pudiera conectar, la tarea a subir se marcaría como ***OPTOMA_UPDATED_FAILS***, en este caso el portal mostraría un log indicando que el problema está al conectar.

Una vez conectado se comenzará la subida de la tarea. El proceso puede durar desde pocos minutos a muchas horas, dependiendo de la carga de trabajo (por eso de la subida al FTP se encarga un hilo de ejecución diferente, como se comenta en la explicación inicial del Media Server).

Si la subida se completa con éxito, el estado de la tarea se marca en el portal como ***OPTOMA_UPDATED***, con lo que la tarea se dará por completada con éxito y los players de los cines ya estarán preparados para la emisión de la cartelería generada.

FASE5 / **APROXIMACIÓN SIMPLIFICADA** del código para su comprensión

```
foreach (DSTask task in currentTasks) {
   try {
      ConnectToFTP();
      CleanCinemasFolders(task.CinemasPath);
      if(UploadCinemasFiles(task.Videos)) {
         task.UpdateTaskState(TaskState.OPTOMA_UPDATED);
      } else {
         task.UpdateTaskState(TaskState.OPTOMA_UPDATED_ERRORS);
      }
   } catch(FTPException ex) {
      task.UpdateTaskState(TaskState.OPTOMA_UPDATED_FAILS);
   }
}
```

```
/// Connect to FTP (throws FTPException)
private void ConnectToFTP();
/// Clean folders with old signages (throws FTPException)
private void CleanCinemasFolders(List<String> paths);
/// Uploads new signages. Returns true if all right
private bool UploadCinemasFiles (List<DSVideo> videos);
```

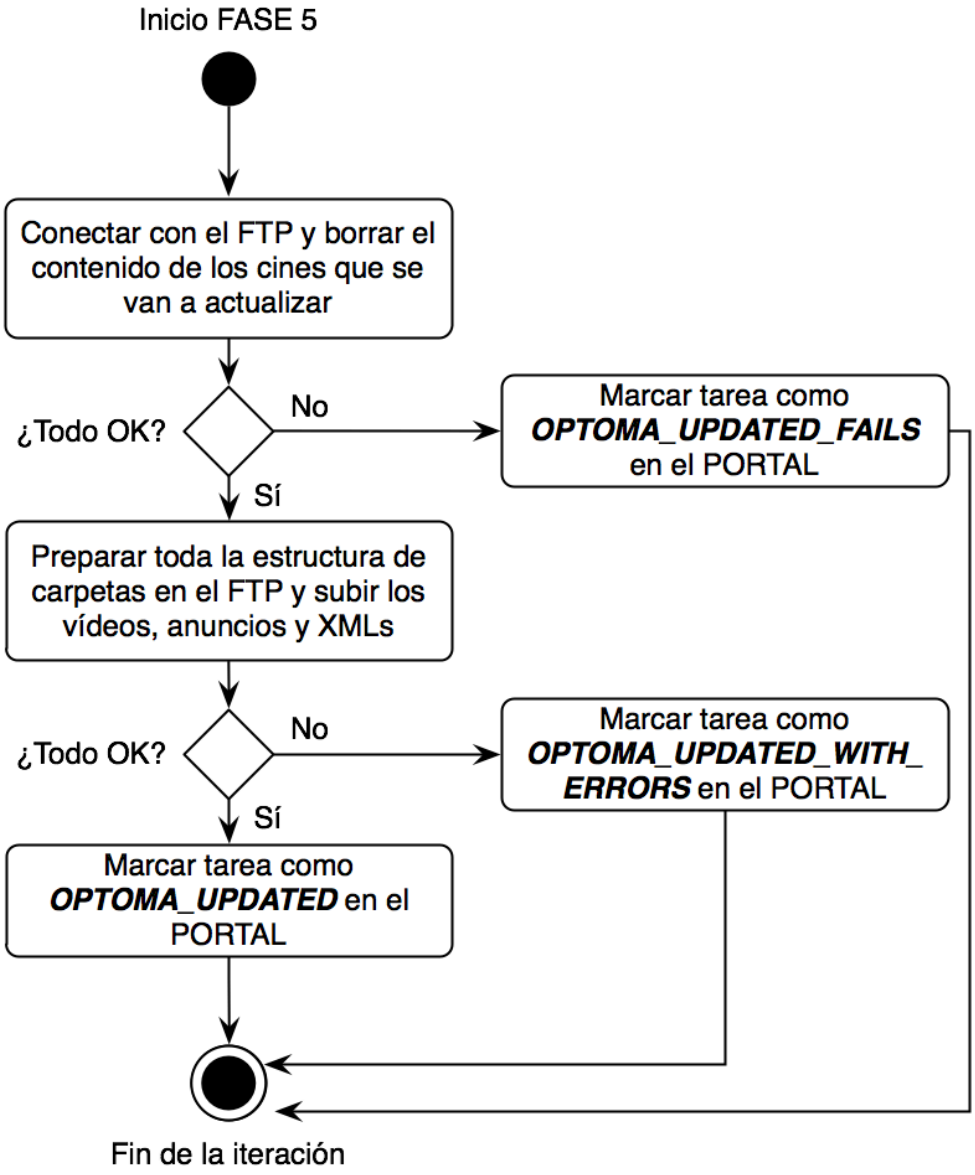


Diagrama 7 | ***Diagrama de Estado - FASE 5***

6.3. Comunicación entre el portal y el Media Server.

La comunicación entre el portal y el Media Server se realiza por HTTP. El portal expone vía ***servlet*** varías URLs para funciones determinadas. En concreto son 3 las funciones que el portal expone:

- **Comprobación de tareas pendientes para descargar**
- **Actualización del estado de una tarea**
- **Actualización de logs de una tarea**

Comprobación de tareas pendientes para descargar

Se trata de un método del servlet expuesto por URL:

```
@GET
@Path("pending")
@Produces("text/plain")
public String getAllDownloadPendingTasks() { ... }
```

La URL para comprobar las tareas pendientes será *http://urlportal/.../pending*. El Media Server, cada X tiempo, irá haciendo una petición web (por GET) a dicha URL, la respuesta (String) podrá ser:

- Cadena vacía. No hay tareas
- Una única tarea, ej: “44”.
- Varias tareas, separadas por “-”. Ej: “44-45-47”

El Media Server procesará la respuesta y descargará la o las tareas, desde un URL dónde el portal deja todas las tareas comprimidas (idTarea.zip).

Actualización del estado de una tarea

```
@GET
@Path("updateTask/{idtask}/{state}/{log}")
@Consumes("text/plain")
@Produces("text/plain")
public String updateTaskState(
          @PathParam("idtask") final Long idTask,
          @PathParam("state") final Integer newState,
          @PathParam("log") final String log
          )
          {  ...  }
```

En este caso, se trata igualmente de una URL expuesta por el *servlet* (por GET) que permite al Media Server actualizar el estado de las tareas que está procesando. La URL tendrá la forma *http:/.../**updateTask**/idTask/state/log*, donde *idTask*, *state* y *log* son parámetros de entrada.

Como se ha comentado anteriormente, las tareas tienen una colección de estados codificados por un número. De manera que, por ejemplo: si el Media Server ha terminado la generación de los vídeos, debe marcarlo como ***COMPLETE*** (código 7), entonces hará una petición http de esta forma:

http://.../updateTask/44/7/ Successfully+completed

La respuesta por parte del portal será "OK" o "ERROR" en el caso de que ocurriera algún error.

Actualización de logs de una tarea

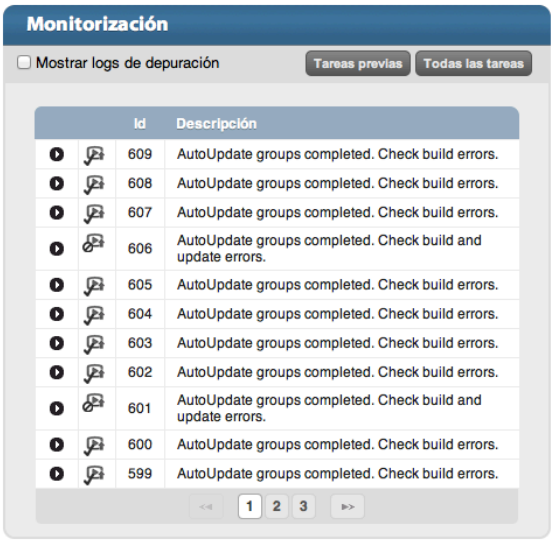
```
@GET
@Path("newTaskLog/{idtask}/{level}/{message}/{group}")
@Consumes("text/plain")
@Produces("text/plain")
public String newTaskLog(
          @PathParam("idtask") final Long idTask,
          @PathParam("level") final Integer level,
          @PathParam("message") final String message,
          @PathParam("group") final String group
          )
          {  ...  }
```

Este método sirve para actualizar líneas de log de una tarea concreta, para cada línea de log, deberá ejecutarse una petición HTTP (GET) a la URL expuesta por este método (*http:/.../newTaskLog/idTask/level/message/group*).

- **idTask**. El id de la tarea.
- **level**. Indica el nivel del log (DEBUG,INFO,WARNING,ERROR,FATAL)
- **message**. Mensaje informativo que el Media Server considere.
- **group**. Se refiere al grupo de players Optoma al que pertenece. Sirve de información técnica para posibles fallos, y viene dado en los archivos de configuración de las tareas.

La respuesta por parte del portal será "OK" o "ERROR" en el caso de que ocurriera algún error.

Los logs de las tareas pueden verse desde las herramientas de administración del portal.



Listado de los de las tareas

Icono	Estado
	Generación de cartelería completada
	Generación de cartelería completada con fallos
	Generación de scripts de cartelería en proceso
	Fallo en la generación de scripts de cartelería
	Generación de videos de cartelería en proceso
	Fallo en la generación de videos de cartelería
	Descarga de recursos del Media Server en curso
	Actualización de Optomas finalizada con éxito
	Actualización de Optomas finalizada con avisos
	Fallo detectado en la actualización de Optomas

Leyenda

590 AutoUpdate groups completed.

AutoUpdate activation groups finalized.	17:09 - 01/09/2011
AutoUpdate activation groups starts (2 groups).	17:09 - 01/09/2011
Upload files to FTP end	17:09 - 01/09/2011
Upload files to FTP starts	17:03 - 01/09/2011
Video generation job list ready	16:54 - 01/09/2011
All xml files processed	16:53 - 01/09/2011
Zip file downloaded and unzipped	16:53 - 01/09/2011
Downloading zip file	16:53 - 01/09/2011
Starting process task	16:53 - 01/09/2011
1 pages will be generated for last/current week for cinema Al-Andalus Coín and template 2.	16:52 - 01/09/2011
1 pages will be generated for next week for cinema Al-Andalus Coín and template 2.	16:52 - 01/09/2011

Ejemplo del detalle de una tarea completada con éxito

562 A content without trailer for signage was found

No available next week advertisement schedule detected for cinema Lara and profile 1.	13:25 - 30/08/2011
Content Ahora los padres son ellos do not have trailer for digital signage.	13:25 - 30/08/2011

Ejemplo del detalle de una tarea que ha fallado

Estos son los logs de las tareas que el Media Server envía al portal, y este muestra de forma visual como herramienta de administración.

Pero no son los únicos logs con los que cuenta el sistema. El Media Server dispone de un sistema de log a nivel local (.txt) detallando toda su ejecución por si ocurriera algún error inesperado.

De igual manera, el portal incluye un sistema de logs que por supuesto usan de igual manera los servicios desarrollados para la generación de la cartelería.

7. Resultados y conclusiones

Después de la primera versión final lanzada y puesta en marcha, se puede decir que el proyecto ha sido o está siendo (e proyecto seguirá evolucionando) todo un éxito, viendo que el sistema responde a las grandes expectativas y requisitos iniciales a los que desde Okode noes enfrentamos.

El sistema muestra una gran estabilidad y se diseñó teniendo en cuenta que la escalabilidad del sistema era un pilar básico en el desarrollo, y todo indica que se ha conseguido.

Los datos que tenemos por parte del cliente (junio 2011) son los siguientes:

- **Más de 120 cines** usando el sistema de cartelería digital.
- **Más de 300 players** instalados en los cines.
- **Más de 700 pantallas** conectadas a los players.
- Se prevé que **más 8 millones de espectadores el próximo año** visualizarán en algún momento el contenido mostrado por las pantallas en alguno de los cines en todo el territorio español.

Valoración personal

No es el primer proyecto que realizo en la empresa. Llevo en Okode desde hace 2 años (septiembre - 2009) como becario, y desde el principio he participado en diversos proyectos con cierta responsabilidad, y todos ellos han sido muy gratificantes a nivel personal y profesional, principalmente en cuanto a experiencia se refiere.

Okode es una empresa de innovación especializada en tecnologías de la información, por lo que la mayoría de los proyectos que se realizan necesitan un análisis y una investigación previa, para adquirir conocimientos y estudiar la viabilidad de tecnologías ha emplear.

Este proyecto ha tenido una parte de análisis e investigación importante, pues hubo que estudiar principalmente el tema de la generación de vídeo por scripting y la viabilidad de montar un sistema basado en ello. Muchas pruebas de concepto y pruebas con diferentes tecnologías y herramientas para dicho objetivo, valorando pros y contras. Está parte fue bastante larga y algo pesada, muchas herramientas eran algo bruscas, con documentación escasa, o problemas de compatibilidad. Pero viendo el sistema acabado, la puesta en marcha y los frutos, sólo se puede valorar como una experiencia gratificante.

En cuanto a las dificultades aparecidas a lo largo del desarrollo (que ha sido muchas) tanto en el Portal (Java EE 6) y en el Media Server (C# .NET), he contado con la ayuda de varios de mis compañeros de trabajo, sin los cuales hubiera sido imposible llevar a cabo el proyecto ni aprender todo lo que he llegado a aprender a lo largo de estos meses.

Quiero concluir diciendo que la experiencia ha sido gratificante viendo los resultados obtenidos, y muy positiva gracias a todo lo que profesionalmente me ha aportado.

8. Objetivos futuros

Después de las valoraciones realizadas con las primeras versiones estables del sistema que ya están en funcionamiento con un gran éxito de aceptación, el objetivo principal de la plataforma es conseguir la mayor expansión posible y así ir haciendo cada vez más atractiva la contratación de anunciantes.

El proyecto seguirá avanzando y madurando al menos un año más, adaptándose a necesidades más particulares y mejorando su rendimiento, pensando principalmente en la **escalabilidad** y **fiabilidad** del sistema, para su progresión futura, y en nuevas plantillas de cartelería o en la extensión de las mismas para que proporcionen más flexibilidad en los datos mostrados.

Las tareas acordadas a corto y medio plazo en las próximas versiones del proyectos se pueden resumir de la siguiente manera:

- Paralelizar la generación de videos

Actualmente existe un único proceso encargado de generar un archivo de video a partir de un archivo .avs, aunque el sistema esta desarrollado para llegar a trabajar con **cuatro procesos paralelos**. Se deben realizar pruebas de carga sobre el sistema, para determinar los requisitos hardware del servidor para estudiar la viabilidad de usar 2 o 4 procesos.

- No subir al FTP recursos duplicados (anuncios)

El proceso de generación actual crea copias idénticas de los videos de los anuncios programados para cada cine en sus correspondientes carpetas de recursos y posteriormente se suben al FTP. Los anuncios son iguales para todos los cines que los tengan programados y no es eficiente realizar copias del video y subirlas por separado al FTP cuando se podrían subir una única vez y realizar las copias internamente en el FTP o enlazar los archivos de video.

- Crear un proceso de generación para cambios

Actualmente el proceso se genera para todos los perfiles configurados en todos los cines. Se debe pensar en un mecanismo que responda a cambios fuera de plazo y regenere únicamente aquello que haya cambiado en lugar de todo el conjunto.

- No subir al FTP recursos no modificados

El mecanismo actual de subida al FTP hace una subida completa de los recursos detectados en la carpeta correspondiente a una semana concreta. Si se realiza un mecanismo para regenerar pequeños cambios fuera de plazo, estos cambios se generarán y sobrescribirán los correspondientes recursos dentro de las carpetas correspondientes a la semana del cambio. Habría que ampliar el funcionamiento de la subida al FTP para que fuera capaz de subir al FTP únicamente los recursos modificados y aquellos que no cambiaron no volver a subirlos.

- Mecanismo de decisión previa del tiempo de actualización de los players de Optoma

Se debe crear un mecanismo que permita indicar a la tarea que se quiere generar el tiempo que debe transcurrir entre la finalización del proceso de generación de la cartelería y la actualización de los players.

- Edición y visualización de perfiles

Permitir actualizar los detalles de los perfiles de cartelería, en lugar de tener que borrar el perfil y volver a crearlo. Además es necesario poder ver algunos detalles de los perfiles en la pantalla de detalles como la ruta del FTP. Esto se refiere a la configuración a través del portal por parte de los cines.

- Comprobación completa de recursos pendientes al generar una tarea

Se está informando de los recursos que faltan uno a uno según la generación de las tareas. Habría que modificar esta información de recursos para que informe de todos los problemas a la vez para que no sea necesario realizar varios intentos cuando faltan varios recursos.

- Mostrar en los logs de las tareas los cines a los que afecta

En logs de información para poder saber qué tareas se generaron para qué cines. Actualmente, el sistema de logs identifica incidencias por tarea. La complejidad del sistema ha ido incrementando y se debe mejorar el sistema de logs por cines y perfiles.

- Plantillas de programación de todas las salas de diferentes tamaños

Para ciertos cines estas plantillas (las que muestran todas las películas y todas las sesiones en una misma pantalla) no sirven. Las primeras

versiones se adaptaron según ciertos cines que empleaban esta visualización, el problema es que al tener que comprimir toda la información en una misma pantalla, el número de sesiones máximas por día y película se limita a 5-6 y para algunos cines, son pocas. También existen ciertos cines con una programación limitada, y estás plantillas quedan muy vacías. Se deberá estudiar la flexibilidad del scripts de estas plantillas o crear diferentes tipos en función de las necesidades.

- Mecanismo de control de cartelería pendiente de generar

Sería interesante un mecanismo para poder llevar un control sencillo de para qué cines se ha generado la cartelería, que cines han modificado las sesiones después de generarla, qué cines han dado por finalizada su programación, qué cines generaron su cartelería y falló algún punto del proceso.

- Clip de entrada

Para la actual plantilla horizontal de sala-contenido crear un clip inicial con información sobre el cine y las franjas de fecha a las que se refiere la cartelería. Este clip inicial se consideraría como parte de los contenidos para no alterar la frecuencia de visualización de la publicidad.

- Customización de las plantillas. Imagen corporativa.

Ciertas plantillas deberán aceptar la parametrización por colores y/o inserción del logo del cine para apropiar el diseño de la cartelería a su imagen corporativa.

9. Referencias y bibliografía

Okode, Innovación en Tecnologías de la Información

Página oficial de la empresa
http://www.okode.com

AVISynth

Página del producto
http://www.avisynth.org
Documentación en castellano
http://avisynth.org/mediawiki/Main_Page/es

VirtualDub

http://www.virtualdub.org/

Final Cut Studio

Página del producto
http://www.apple.com/finalcutstudio
Generación de plantillas XML desde Final Cut Studio
http://developer.apple.com/library/mac/#documentation/AppleApplications/Reference/FinalCutPro_XML/

Optoma

Página principal
http://www.optoma.es
Página del player - *Digital Signage Processor SignShow D5000*
http://www.optoma.es/dsMPdetails.aspx?PC=SignShow%20D5000

MDSN Library – Documentación .NET

http://msdn.microsoft.com/es-es/library/

Java Enterprise Edition 6 – API Specification

http://java.sun.com/javaee/6/docs/api/

Glassfish 3

http://glassfish.java.net/

Subversion

http://subversion.apache.org/

Atlassian Jira

http://www.atlassian.com/software/jira/

Atlassian Confluence

http://www.atlassian.com/software/confluence/

Atlassian Bamboo

http://www.atlassian.com/software/bamboo/

Maven 3

http://maven.apache.org/

Eclipse IDE for Java EE Developers

http://eclipse.org/downloads/moreinfo/jee.php

Microsoft Visual Studio 2010

http://msdn.microsoft.com/es-es/vstudio/

Anexo I

Documentación – Java EE 6

Core de cartelería digital en el Portal

Class Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- com.okode.versiondigital.core.advertisement.AdvertisementService
- com.okode.versiondigital.core.advertisement.AdvertisementServiceException
- com.okode.versiondigital.core.digitalsignage.exceptions.AdvertisementVideoException
- com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplateVariables
- com.okode.versiondigital.core.digitalsignage.DigitalSignageService
- com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageServiceException
- com.okode.versiondigital.core.digitalsignage.model.DigitalSignageTemplate
  - com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplate
  - com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplate
  - com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplate

- com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageTemplateException
- com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService
- com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlInfo
- com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlMetadata
- com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlResource
- com.okode.versiondigital.core.digitalsignage.model.DigitalSignageZipEntry
- com.okode.versiondigital.core.digitalsignage.DsIntroVideoVariables
- com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplateVariables
- com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplateVariables
- com.okode.versiondigital.core.digitalsignage.exceptions.NoShortTrailerAvailableException
- com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupInfo
- com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupsCache
- com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationException
- com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

com.okode.versiondigital.core.advertisement.AdvertisementService
com.okode.versiondigital.core.advertisement.AdvertisementServiceException
com.okode.versiondigital.core.digitalsignage.exceptions.AdvertisementVideoException
com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplate
com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplateVariables
com.okode.versiondigital.core.digitalsignage.DigitalSignageService
com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageServiceException
com.okode.versiondigital.core.digitalsignage.model.DigitalSignageTemplate
com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageTemplateException
com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService
com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlInfo
com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlMetadata
com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlResource
com.okode.versiondigital.core.digitalsignage.model.DigitalSignageZipEntry
com.okode.versiondigital.core.digitalsignage.DsIntroVideoVariables
com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplate
com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplateVariables
com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplate
com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplateVariables
com.okode.versiondigital.core.digitalsignage.exceptions.NoShortTrailerAvailableException
com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupInfo
com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupsCache
com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationException
com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService

Class Documentation

com.okode.versiondigital.core.advertisement.AdvertisementService Class Reference

Inherits BaseService.

Public Member Functions

- void **setUserAsAdvertisementAdmin** (User currentUser, User user) throws AdvertisementServiceException
- void **setUserAsAdvertisementAdmin** (Long currentUserId, Long userId) throws AdvertisementServiceException
- void **removeAdminAdvertisementRole** (User currentUser, User user) throws AdvertisementServiceException
- void **removeAdminAdvertisementRole** (Long currentUserId, Long userId) throws AdvertisementServiceException
- Advertiser **findAdvertiserById** (Long advertiserId) throws AdvertisementServiceException
- void **addAdvertiser** (Advertiser adv, String currentUserEmail) throws AdvertisementServiceException
- void **deleteAdvertiser** (Long advertiserId) throws AdvertisementServiceException
- void **bindAdvertiser** (Advertiser adv, User user) throws AdvertisementServiceException
- void **unbindAdvertiser** (Advertiser adv, User user) throws AdvertisementServiceException
- List< Advertiser > **findAllAdvertiserBasic** () throws AdvertisementServiceException
- List< Advertiser > **findAllAdvertisers** (int page, int pageSize, SingularAttribute<?super Advertiser,?> orderedBy, boolean ascendent) throws AdvertisementServiceException
- int **countAllAdvertisers** ()
- Advertiser **updateAdvertiser** (Advertiser adv, String currentUserEmail) throws AdvertisementServiceException
- Long **createCampaign** (Long advertiserId, Campaign campaign, Advertisement advertisement, String currentUserEmail) throws AdvertisementServiceException
- void **setCampaignActivation** (Campaign campaign, Boolean activation, String currentUserEmail) throws AdvertisementServiceException
- Campaign **findCampaignById** (Long campaignId) throws AdvertisementServiceException
- Campaign **findCampaignByIdFullAttach** (Long campaignId) throws AdvertisementServiceException
- int **countAllCampaigns** (boolean hidePastCampaigns, boolean hideInactiveCampaigns)
- List< Campaign > **findAllCampaign** (int page, int pageSize, SingularAttribute<?super Campaign,?> orderedBy, boolean ascendent, boolean hidePastCampaigns, boolean hideInactiveCampaigns) throws AdvertisementServiceException
- Campaign **updateCampaign** (Campaign campaign, String currentUserEmail) throws AdvertisementServiceException
- void **deleteCampaignAndAdvertisement** (Long campaignId) throws AdvertisementServiceException

- int **getFreePlacesForAdvertisement** (Date date, Long cinemaId) throws AdvertisementServiceException
- int **getFreePlacesForAdvertisement** (Date date, List< Long > cinemaIds) throws AdvertisementServiceException
- int **getMinimumFreePlacesForAdvertisement** (Date initDate, Date endDate, List< Long > cinemaIds) throws AdvertisementServiceException
- int **getFreePlacesForAdvertisement** (Date initDate, Date endDate, List< Long > cinemaIds) throws AdvertisementServiceException
- List< MinimumFreeImpactsByCity > **getFreePlacesForAdvertisement** (Date initDate, Date endDate) throws AdvertisementServiceException
- List< MinimumFreeImpactsByCircuit > **getFreePlacesForCircuitAdvertisement** (Date initDate, Date endDate) throws AdvertisementServiceException
- List< MinimumFreeImpactsByCinema > **getFreePlacesForAdvertisement** (Date initDate, Date endDate, Long cityId) throws AdvertisementServiceException
- List< MinimumFreeImpactsByCinema > **getFreePlacesForCircuitAdvertisement** (Date initDate, Date endDate, Long circuitId) throws AdvertisementServiceException
- List< Campaign > **getScheduledAdvertisement** (Cinema cinema, Date date) throws AdvertisementServiceException
- List< Campaign > **getScheduledAdvertisement** (Long cinemaId, Date date) throws AdvertisementServiceException
- List< Campaign > **getScheduledAdvertisementWithAdvertisement** (Long cinemaId, Date date) throws AdvertisementServiceException
- List< Advertisement > **getScheduledAdvertisementDistinct** (Long cinemaId, Date date) throws AdvertisementServiceException
- int **scheduleCampaign** (Long campaignId, List< Long > cinemaIds) throws AdvertisementServiceException
- Integer **getEstimatedImpacts** (Campaign campaign, Date initDate, Date endDate) throws AdvertisementServiceException
- List< CampaignImpacts > **getEstimatedImpactsGroupByCinema** (Campaign campaign, Date initDate, Date endDate) throws AdvertisementServiceException
- Advertisement **findAdvertisementById** (Long advertisementId) throws AdvertisementServiceException
- Advertisement **updateAdvertisement** (Advertisement advertisement, String currentUserEmail) throws AdvertisementServiceException
- void **deleteAdvertisementAndCampaign** (Advertisement advertisement) throws AdvertisementServiceException
- int **countCampaignCinemas** (Long campaignId)
- void **disableCampaign** (Long campaignId)

Detailed Description

Advertisement utilities

Member Function Documentation

void com.okode.versiondigital.core.advertisement.AdvertisementService.addAdvertiser (Advertiser *adv*, String *currentUserEmail*) throws AdvertisementServiceException

Add a new advertiser.

Parameters:

adv	

Exceptions:

AdvertisementServiceException	

void com.okode.versiondigital.core.advertisement.AdvertisementService.bindAdvertiser (Advertiser *adv*, User *user*) throws AdvertisementServiceException

Adds a advertiser to an user advertiser list.

Parameters:

adv	
user	

int com.okode.versiondigital.core.advertisement.AdvertisementService.countAllAdvertisers ()

Get count all advertisers

Returns:

int com.okode.versiondigital.core.advertisement.AdvertisementService.countAllCampaigns (boolean *hidePastCampaigns*, boolean *hideInactiveCampaigns*)

Count all campaigns

Parameters:

hidePastCampaigns	
hideInactiveCampaigns	

Returns:

int com.okode.versiondigital.core.advertisement.AdvertisementService.countCampaignCinemas (Long *campaignId*)

Get number of cinemas of a campaign.

Parameters:

campaigId	

Returns:

Long com.okode.versiondigital.core.advertisement.AdvertisementService.createCampaign (Long *advertiserId*, Campaign *campaign*, Advertisement *advertisement*, String *currentUserEmail*) throws AdvertisementServiceException

Create a new campaign.

Parameters:

advertiserId	
campaign	
advertisement	

Exceptions:

AdvertisementServiceException	

Returns:

void com.okode.versiondigital.core.advertisement.AdvertisementService.deleteAdvertisementAndCampaign (Advertisement *advertisement*) throws AdvertisementServiceException

Delete advertisement and its campaign.

Parameters:

advertisement	

Exceptions:

AdvertisementServiceException	

void com.okode.versiondigital.core.advertisement.AdvertisementService.deleteAdvertiser (Long *advertiserId*) throws AdvertisementServiceException

Delete an advertiser.

Parameters:

advertiserId	

Exceptions:

AdvertisementServiceException	

void com.okode.versiondigital.core.advertisement.AdvertisementService.deleteCampaignAndAdvertisement (Long *campaignId*) throws AdvertisementServiceException

Delete campaign and its advertisement.

Parameters:

campaigId	

Exceptions:

AdvertisementServiceException	

void com.okode.versiondigital.core.advertisement.AdvertisementService.disableCampaign (Long *campaignId*)

Disable a campaign.

Parameters:

campaignId	

Advertisement com.okode.versiondigital.core.advertisement.AdvertisementService.findAdvertisementById (Long *advertisementId*) throws AdvertisementServiceException

Find advertisement by id.

Parameters:

advertisementId	

Returns:

Exceptions:

AdvertisementServiceException	

Advertiser com.okode.versiondigital.core.advertisement.AdvertisementService.findAdvertiserById (Long *advertiserId*) throws AdvertisementServiceException

Find advertiser by id.

Parameters:

advertiserId	

Returns:

Exceptions:

AdvertisementServiceException	

List<Advertiser> com.okode.versiondigital.core.advertisement.AdvertisementService.findAllAdvertiserBasic () throws AdvertisementServiceException

Find all advertisers.

Parameters:

pageNumber	
pageSize	

Returns:

Exceptions:

AdvertisementServiceException	

List<Advertiser> com.okode.versiondigital.core.advertisement.AdvertisementService.findAllAdvertisers (int *page*, int *pageSize*, SingularAttribute<?super Advertiser,?> *orderedBy*, boolean *ascendent*) throws AdvertisementServiceException

Find all advertisers.

Parameters:

page	
pageSize	
orderedBy	
ascendent	

Returns:

Exceptions:

AdvertisementServiceException	

List<Campaign> com.okode.versiondigital.core.advertisement.AdvertisementService.findAllCampaign (int *page*, int *pageSize*, SingularAttribute<?super Campaign,?> *orderedBy*, boolean *ascendent*, boolean *hidePastCampaigns*, boolean *hideInactiveCampaigns*) throws AdvertisementServiceException

Find all campaigns

Parameters:

page	
pageSize	
orderedBy	
ascendent	

Returns:

Campaign com.okode.versiondigital.core.advertisement.AdvertisementService.findCampaignById (Long *campaignId*) throws AdvertisementServiceException

Find campaign by id.

Parameters:

campaignId	

Returns:

Exceptions:

AdvertisementServiceException	

Campaign
com.okode.versiondigital.core.advertisement.AdvertisementService.findCampaignByIdFullAttach (Long *campaignId*) throws AdvertisementServiceException

Find campaign by id with Advertiser and Advertisement info

Parameters:

campaignId	

Returns:

Integer
com.okode.versiondigital.core.advertisement.AdvertisementService.getEstimatedImpacts (Campaign *campaign*, Date *initDate*, Date *endDate*) throws AdvertisementServiceException

Get estimated impacts for a campaign in all cinemas.

Parameters:

campaign	
initDate	
endDate	

Returns:

List<CampaignImpacts>
com.okode.versiondigital.core.advertisement.AdvertisementService.getEstimatedImpactsGroupByCinema (Campaign *campaign*, Date *initDate*, Date *endDate*) throws AdvertisementServiceException

Get estimated impacts for a campaign grouping by cinema.

Parameters:

campaign	
initDate	
endDate	

Returns:

int
com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesForAdvertisement (Date *date*, Long *cinemaId*) throws AdvertisementServiceException

Get number of free places for advertisement.

Parameters:

date	
cinemaId	

Returns:

Exceptions:

AdvertisementServiceException	

int com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesForAdvertisement (Date *date*, List< Long > *cinemaIds*) throws AdvertisementServiceException

Get number of free places for advertisement.

Parameters:

date	
cinemaIds	

Returns:

Exceptions:

AdvertisementServiceException	

int com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesForAdvertisement (Date *initDate*, Date *endDate*, List< Long > *cinemaIds*) throws AdvertisementServiceException

Get number of free places for advertisement.

Parameters:

initDate	
endDate	
cinemaIds	

Returns:

Exceptions:

AdvertisementServiceException	

List<MinimumFreeImpactsByCity> com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesForAdvertisement (Date *initDate*, Date *endDate*) throws AdvertisementServiceException

Get number of free places for advertisement.

Parameters:

initDate	
endDate	

Returns:

Exceptions:

AdvertisementServiceException	

List<MinimumFreeImpactsByCinema> com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesFor

Advertisement (Date *initDate*, Date *endDate*, Long *cityId*) throws AdvertisementServiceException

Get number of free places for advertisement.

Parameters:

initDate	
endDate	

Returns:

Exceptions:

AdvertisementServiceException	

List<MinimumFreeImpactsByCircuit> com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesForCircuitAdvertisement (Date *initDate*, Date *endDate*) throws AdvertisementServiceException

Get number of free places for advertisement.

Parameters:

initDate	
endDate	

Returns:

Exceptions:

AdvertisementServiceException	

List<MinimumFreeImpactsByCinema> com.okode.versiondigital.core.advertisement.AdvertisementService.getFreePlacesForCircuitAdvertisement (Date *initDate*, Date *endDate*, Long *circuitId*) throws AdvertisementServiceException

Get number of free places for advertisement by circuitId.

Parameters:

initDate	
endDate	

Returns:

Exceptions:

AdvertisementServiceException	

int com.okode.versiondigital.core.advertisement.AdvertisementService.getMinimumFreePlacesForAdvertisement (Date *initDate*, Date *endDate*, List< Long > *cinemaIds*) throws AdvertisementServiceException

Get minimum number of free places for advertisement.

Parameters:

initDate	
endDate	
cinemaIds	

Returns:

Exceptions:

AdvertisementServiceException	

List<Campaign> com.okode.versiondigital.core.advertisement.AdvertisementService.getScheduledAdvertisement (Cinema *cinema*, Date *date*) throws AdvertisementServiceException

Get advertisement scheduled into a cinema.

Parameters:

cinema	

Returns:

Exceptions:

AdvertisementServiceException	

List<Campaign> com.okode.versiondigital.core.advertisement.AdvertisementService.getScheduledAdvertisement (Long *cinemaId*, Date *date*) throws AdvertisementServiceException

Get advertisement scheduled into a cinema.

Parameters:

cinema	

Returns:

Exceptions:

AdvertisementServiceException	

List<Advertisement> com.okode.versiondigital.core.advertisement.AdvertisementService.getScheduledAdvertisementDistinct (Long *cinemaId*, Date *date*) throws AdvertisementServiceException

Get advertisement scheduled into a cinema with advertisement info attached.

Parameters:

cinema	

Returns:

Exceptions:

AdvertisementS	

erviceException	

List<Campaign>
com.okode.versiondigital.core.advertisement.AdvertisementService.getScheduledAdvertisementWithAdvertisement (Long *cinemaId*, Date *date*) throws AdvertisementServiceException

Get advertisement scheduled into a cinema with advertisement info attached.

Parameters:

cinema	

Returns:

Exceptions:

AdvertisementServiceException	

void
com.okode.versiondigital.core.advertisement.AdvertisementService.removeAdminAdvertisementRole (User *currentUser*, User *user*) throws AdvertisementServiceException

remove ADMIN_ADVERTISER role to a user.

Parameters:

currentUser	Current logged user
user	User to modify

void
com.okode.versiondigital.core.advertisement.AdvertisementService.removeAdminAdvertisementRole (Long *currentUserId*, Long *userId*) throws AdvertisementServiceException

remove ADMIN_ADVERTISER role to a user.

Parameters:

currentUser	Current logged user
user	User to modify

int
com.okode.versiondigital.core.advertisement.AdvertisementService.scheduleCampaign (Long *campaignId*, List< Long > *cinemaIds*) throws AdvertisementServiceException

Schedule an existing campaign into many cinemas.

Parameters:

adv	
campaign	
cinemaIds	

Exceptions:

AdvertisementServiceException	

void
com.okode.versiondigital.core.advertisement.AdvertisementService.setCampaignActiv

ation (Campaign *campaign*, Boolean *activation*, String *currentUserEmail*) throws AdvertisementServiceException

Set campaign activation.

Parameters:

campaign	

Exceptions:

AdvertisementServiceException	

void com.okode.versiondigital.core.advertisement.AdvertisementService.setUserAsAdvertisementAdmin (User *currentUser*, User *user*) throws AdvertisementServiceException

Add ADMIN_ADVERTISER role to a user.

Parameters:

currentUser	Current logged user
user	User to modify

void com.okode.versiondigital.core.advertisement.AdvertisementService.setUserAsAdvertisementAdmin (Long *currentUserId*, Long *userId*) throws AdvertisementServiceException

Add ADMIN_ADVERTISER role to a user.

Parameters:

currentUser	Current logged user
user	User to modify

void com.okode.versiondigital.core.advertisement.AdvertisementService.unbindAdvertiser (Advertiser *adv*, User *user*) throws AdvertisementServiceException

Adds a advertiser to an user advertiser list.

Parameters:

adv	
user	

Advertisement com.okode.versiondigital.core.advertisement.AdvertisementService.updateAdvertisement (Advertisement *advertisement*, String *currentUserEmail*) throws AdvertisementServiceException

Update advertisement info.

Parameters:

advertisement	

Exceptions:

AdvertisementServiceException	

Advertiser
com.okode.versiondigital.core.advertisement.AdvertisementService.updateAdvertiser (Advertiser *adv*, String *currentUserEmail*) throws AdvertisementServiceException

Update advertiser info.

Parameters:

adv	

Exceptions:

AdvertisementServiceException	

Campaign
com.okode.versiondigital.core.advertisement.AdvertisementService.updateCampaign (Campaign *campaign*, String *currentUserEmail*) throws AdvertisementServiceException

Update campaign info.

Parameters:

campaign	

Exceptions:

AdvertisementServiceException	

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/advertisement/AdvertisementService.java

com.okode.versiondigital.core.advertisement.AdvertisementServiceException Class Reference

Inherits ServiceException.

Public Member Functions

- **AdvertisementServiceException** (Exception he)

Detailed Description

Exception for Cinema Service problems.

Constructor & Destructor Documentation

com.okode.versiondigital.core.advertisement.AdvertisementServiceException.AdvertisementServiceException (Exception *he*)

Exception for Advertisement Service problems.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/advertisement/AdvertisementServiceException.java

com.okode.versiondigital.core.digitalsignage.exceptions.AdvertisementVideoException Class Reference

Inherits ServiceException.

Public Member Functions

- **AdvertisementVideoException** ()
- **AdvertisementVideoException** (String message)
- **AdvertisementVideoException** (Exception ex)

Detailed Description

Exception for Digital Signage Advertisement video problems.

Constructor & Destructor Documentation

com.okode.versiondigital.core.digitalsignage.exceptions.AdvertisementVideoException.AdvertisementVideoException ()

Exception for Digital Signage Advertisement video problems.

com.okode.versiondigital.core.digitalsignage.exceptions.AdvertisementVideoException.AdvertisementVideoException (String *message*)

Exception for Digital Signage Advertisement video problems.

com.okode.versiondigital.core.digitalsignage.exceptions.AdvertisementVideoException.AdvertisementVideoException (Exception *ex*)

Exception for Digital Signage Advertisement video problems.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/exceptions/AdvertisementVideoException.java

com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplate Class Reference

Inheritance diagram for com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplate:

Public Member Functions

- List< **DigitalSignageZipEntry** > **generateDigitalSignageScript** (Date initDate, DsProfile profile, DsTask dsTask, ParamBundle messages, String clientTimeZone) throws NoShortTrailerAvailableException, OptomaIntegrationException, AdvertisementVideoException, Exception
- **DigitalSignageZipEntry generateDigitalSignageScript** (DsTask dsTask, Date initDate, Date endDate, Long contentId, Long cinemaRoomId, Long cinemaId, ParamBundle messages, String clientTimeZone, boolean oldFileGeneration, DsProfile profile, List< Advertisement > advertisement) throws NoShortTrailerAvailableException, OptomaIntegrationException, Exception

Static Public Attributes

- static final String **BACKGROUND** = "\\ContentRoom_week_4rows_H\\"
- static final String **SCRIPT_SOURCE** = "/ContentRoom_week_4rows_H.avs"
- static final String **VAR_FONDO** = "@varFondo"
- static final String **VAR_TRAILER** = "@varTrailer"
- static final String **VAR_POSTER** = "@varPoster"
- static final String **VAR_SALA** = "@varSala"
- static final String **VAR_DAY1** = "@varDay1Sessions"
- static final String **VAR_DAY2** = "@varDay2Sessions"
- static final String **VAR_DAY3** = "@varDay3Sessions"
- static final String **VAR_DAY4** = "@varDay4Sessions"
- static final String **VAR_AGE_LIMIT** = "@varAgeLimit"
- static final String **VAR_DURATION** = "@varDuration"
- static final String **METADATA_CONTENT_ID** = "contentId"
- static final String **METADATA_CONTENT_AGE_LIMIT** = "contentAgeLimit"
- static final String **METADATA_CONTENT_DURATION** = "contentDuration"
- static final String **METADATA_ROOM_ID** = "roomId"
- static final String **METADATA_ROOM_NAME** = "roomName"
- static final String **TRAILERS_FOLDER** = "DigitalSignage.LocalTrailersFolder"
- static final String **ADVERTISING_FOLDER** = "Advertisement.LocalDefaultFolder"
- static final String **BACKGROUND_BASE_URL** = "http://www.versiondigital.es/carteleria/ContentRoom_week_4rows_H/"

Detailed Description

Digital Signage script generator for Content-CinemaRoom-Horizontal template.

Member Function Documentation

DigitalSignageZipEntry com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplate.generateDigitalSignageScript (DsTask *dsTask*, Date *initDate*, Date *endDate*, Long *contentId*, Long *cinemaRoomId*, Long *cinemaId*, ParamBundle *messages*, String *clientTimeZone*, boolean *oldFileGeneration*, DsProfile *profile*, List< Advertisement > *advertisement*) throws NoShortTrailerAvailableException, OptomaIntegrationException, Exception

Create AVS file and related XML file with content and advertisement info.

Parameters:

dsTask	
initDate	
endDate	
contentId	
cinemaRoomId	
cinemaId	
messages	
clientTimeZone	
oldFileGeneration	
profile	
advertisement	

Returns:

Exceptions:

NoShortTrailerAvailableException	
OptomaIntegrationException	
Exception	

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/contentandroomhorizontal/ContentAndRoomHorizontalTemplate.java

com.okode.versiondigital.core.digitalsignage.contentandroomhorizontal.ContentAndRoomHorizontalTemplateVariables Class Reference

Public Member Functions

- String **getVarFondo** ()
- void **setVarFondo** (String varFondo)
- String **getVarTrailer** ()
- void **setVarTrailer** (String varTrailer)
- String **getVarPoster** ()
- void **setVarPoster** (String varPoster)
- String **getVarSala** ()
- void **setVarSala** (String varSala)
- String **getVarDay1Sessions** ()
- void **setVarDay1Sessions** (String varDay1Sessions)
- String **getVarDay2Sessions** ()
- void **setVarDay2Sessions** (String varDay2Sessions)
- String **getVarDay3Sessions** ()
- void **setVarDay3Sessions** (String varDay3Sessions)
- String **getVarDay4Sessions** ()
- void **setVarDay4Sessions** (String varDay4Sessions)
- String **getVarAgeLimit** ()
- void **setVarAgeLimit** (String varAgeLimit)
- String **getVarDuration** ()
- void **setVarDuration** (String varDuration)
- Date **getInitDate** ()
- void **setInitDate** (Date initDate)
- Date **getEndDate** ()
- void **setEndDate** (Date endDate)
- String **getBackgroundDownload** ()
- void **setBackgroundDownload** (String backgroundDownload)
- String **getPosterDownload** ()
- void **setPosterDownload** (String posterDownload)
- String **getFtpFolder** ()
- void **setFtpFolder** (String ftpFolder)
- String **getGroupName** ()
- void **setGroupName** (String groupName)

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/contentandroomhorizontal/ContentAndRoomHorizontalTemplateVariables.java

com.okode.versiondigital.core.digitalsignage.DigitalSignageService Class Reference

Inherits BaseService.

Public Member Functions

- void **createDsTask** (DsTask task) throws DigitalSignageServiceException
- DsTask **updateDsTask** (DsTask task) throws DigitalSignageServiceException
- List< DsTask > **findAllDsTaskByState** (int state) throws DigitalSignageServiceException
- DsTask **findDsTaskById** (Long idTask) throws DigitalSignageServiceException
- DsTemplate **findTemplateById** (Long templateid) throws DigitalSignageServiceException
- DsTemplate **findTemplateByIdWithParameters** (Long templateid) throws DigitalSignageServiceException
- List< DsProfile > **findProfilesOfCinemas** (List< Long > cinemaIds) throws DigitalSignageServiceException
- List< DsServer > **findAllServers** () throws DigitalSignageServiceException
- List< DsTemplate > **findAllTemplates** () throws DigitalSignageServiceException
- void **deleteProfile** (Long profileId) throws DigitalSignageServiceException
- List< DsProfile > **findCinemaProfilesWithTemplateAndServer** (Long cinemaId) throws DigitalSignageServiceException
- void **addDsTaskLog** (Long idTask, Integer level, String message) throws DigitalSignageServiceException
- void **addDsTaskLog** (Long idTask, Integer level, String message, String group) throws DigitalSignageServiceException
- void **addProfile** (DsProfile profile, Long cinemaId) throws DigitalSignageServiceException
- List< DsTask > **getAllDsTasks** () throws DigitalSignageServiceException
- List< DsTask > **getNewDsTasks** () throws DigitalSignageServiceException
- List< DsTask > **getPreviousDsTasks** () throws DigitalSignageServiceException
- List< DsTaskLog > **getDsTasksLogs** (Long taskId) throws DigitalSignageServiceException
- List< DsTaskLog > **getDsTasksLogs** (Long taskId, Boolean showDebugLogs) throws DigitalSignageServiceException
- DsServer **getDsServer** (Long dsServerId) throws DigitalSignageServiceException
- void **autoUpdatePlayers** (Long taskId, String hour, String minute) throws DigitalSignageServiceException
- void **autoUpdatePlayers** (Long taskId) throws DigitalSignageServiceException
- void **disableAutoUpdate** () throws DigitalSignageServiceException
- void **saveDsIntro** (DsIntro intro, Cinema cinema, User currentUser)
- DsTask **findCinemaTask** (Long cinemaId) throws DigitalSignageServiceException

---

Detailed Description

Digital signage data access utilities

Member Function Documentation

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.addDsTaskLog (Long *idTask*, Integer *level*, String *message*) throws DigitalSignageServiceException

Create new log for a task.

Parameters:

idTask	
level	
message	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.addDsTaskLog (Long *idTask*, Integer *level*, String *message*, String *group*) throws DigitalSignageServiceException

Create new log for a task.

Parameters:

idTask	
level	
message	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.addProfile (DsProfile *profile*, Long *cinemaId*) throws DigitalSignageServiceException

Create new profile and add it to a cinema.

Parameters:

profile	
cinemaId	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.autoUpdatePlayers (Long *taskId*, String *hour*, String *minute*) throws DigitalSignageServiceException

Automatic upload task info to final players.

Parameters:

taskId	

Exceptions:

DigitalSignageServiceException	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.autoUpdatePlayers (Long *taskId*) throws DigitalSignageServiceException

Manual upload task info to final players. Time for update will be established for current time + 3 minutes

Parameters:

taskId	

Exceptions:

DigitalSignageSer	

viceException	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.createDsTask (DsTask *task*) throws DigitalSignageServiceException

Save DsTask into DB.

Parameters:

task	

Exceptions:

DigitalSignageSer viceException	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.deleteProfile (Long *profileId*) throws DigitalSignageServiceException

Delete a profile.

Parameters:

profileId	

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.disableAutoUpdate () throws DigitalSignageServiceException

Disable AutoUpdate for all groups.

Exceptions:

DigitalSignageSer viceException	

List<DsTask> com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findAllDsTaskByState (int *state*) throws DigitalSignageServiceException

Find all dsTasks by state.

Parameters:

state	

Returns:

List<DsServer> com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findAllServers () throws DigitalSignageServiceException

Get all DsServers.

Returns:

Exceptions:

DigitalSignageSer viceException	

List<DsTemplate>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findAllTemplates ()
throws DigitalSignageServiceException

Get all DsTemplates.

Returns:

List<DsProfile>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findCinemaProfiles
WithTemplateAndServer (Long *cinemaId*) throws DigitalSignageServiceException

Get cinema profiles with server and template info attached.

Parameters:

cinemaId	Cinema's identifier.

Returns:

DsTask
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findCinemaTask
(Long *cinemaId*) throws DigitalSignageServiceException

find current cinema's Task

Parameters:

cinemaId	

Returns:

Exceptions:

NoResultException	: the selected cinema doesn't have asociated tasks
Exception	: Unexpected exception
DigitalSignageServiceException	

DsTask
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findDsTaskById
(Long *idTask*) throws DigitalSignageServiceException

Find DsTask by id.

Parameters:

idTask	

Returns:

List<DsProfile>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findProfilesOfCine
mas (List< Long > *cinemaIds*) throws DigitalSignageServiceException

Find all profiles of the selected cinemas.

Parameters:

cinemaIds	

Returns:

DsTemplate com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findTemplateById (Long *templateid*) throws DigitalSignageServiceException

Find DsTemplate by id.

Parameters:

templateid	

Returns:

DsTemplate com.okode.versiondigital.core.digitalsignage.DigitalSignageService.findTemplateById WithParameters (Long *templateid*) throws DigitalSignageServiceException

Find DsTemplate by id with parameters attached.

Parameters:

templateid	

Returns:

Exceptions:

DigitalSignageServiceException	

List<DsTask> com.okode.versiondigital.core.digitalsignage.DigitalSignageService.getAllDsTasks () throws DigitalSignageServiceException

Get all tasks.

Returns:

DsServer com.okode.versiondigital.core.digitalsignage.DigitalSignageService.getDsServer (Long *dsServerId*) throws DigitalSignageServiceException

Find DsServer by id.

Parameters:

dsServerId	

Returns:

List<DsTaskLog>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.getDsTasksLogs
(Long *taskId*) throws DigitalSignageServiceException

Get task logs.

Parameters:

taskId	

Returns:

List<DsTaskLog>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.getDsTasksLogs
(Long *taskId*, Boolean *showDebugLogs*) throws DigitalSignageServiceException

Get task logs.

Parameters:

taskId	

Returns:

List<DsTask>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.getNewDsTasks ()
throws DigitalSignageServiceException

Get the most recent tasks.

Returns:

List<DsTask>
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.getPreviousDsTask
s () throws DigitalSignageServiceException

Get older tasks.

Returns:

void com.okode.versiondigital.core.digitalsignage.DigitalSignageService.saveDsIntro
(DsIntro *intro*, Cinema *cinema*, User *currentUser*)

Save ds intro info.

Parameters:

intro	
cinema	
currentUser	

DsTask
com.okode.versiondigital.core.digitalsignage.DigitalSignageService.updateDsTask
(DsTask *task*) throws DigitalSignageServiceException

Update DsTask info.

Parameters:

task	

Returns:

Exceptions:

DigitalSignageServiceException	

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/DigitalSignageService.java

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageServiceException Class Reference

Inherits ServiceException.

Public Member Functions

- **DigitalSignageServiceException** ()
- **DigitalSignageServiceException** (String message)
- **DigitalSignageServiceException** (Exception ex)

Detailed Description

Exception for Digital Signage Service problems.

Constructor & Destructor Documentation

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageServiceException.DigitalSignageServiceException ()

Exception for Content Service problems.

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageServiceException.DigitalSignageServiceException (String *message*)

Exception for Content Service problems.

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageServiceException.DigitalSignageServiceException (Exception *ex*)

Exception for Content Service problems.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/exceptions/DigitalSignageServiceException.java

com.okode.versiondigital.core.digitalsignage.model.DigitalSignageTemplate Interface Reference

Inheritance diagram for com.okode.versiondigital.core.digitalsignage.model.DigitalSignageTemplate:

Package Functions

- List< **DigitalSignageZipEntry** > **generateDigitalSignageScript** (Date initDate, DsProfile profile, DsTask dsTask, ParamBundle messages, String clientTimeZone) throws Exception

---

Detailed Description

All digital signage script generators must implement this interface.

The documentation for this interface was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/model/DigitalSignageTemplate.java

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageTemplateException Class Reference

Inherits ServiceException.

Public Member Functions

- **DigitalSignageTemplateException** ()
- **DigitalSignageTemplateException** (Exception he)

Detailed Description

Exception for DigitalSignageTemplate Service problems.

Constructor & Destructor Documentation

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageTemplateException.DigitalSignageTemplateException ()

Exception for DigitalSignageTemplate Service problems.

com.okode.versiondigital.core.digitalsignage.exceptions.DigitalSignageTemplateException.DigitalSignageTemplateException (Exception *he*)

Exception for DigitalSignageTemplate Service problems.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/exceptions/DigitalSignageTemplateException.java

com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService Class Reference

Inherits BaseService.

Public Types

- enum **AvailableTemplates** { **CONTENTROOM_WEEK_4ROWS_H**, **EIGHT_PROGRAMGRID_VERTICAL**, **EIGHT_PROGRAMGRID_HORIZONTAL** }

Public Member Functions

- String **generateDigitalSignageScriptsForCinemas** (Date initDate, String clientTimeZone, String requestUser, List< Long > cinemaIds)
- String **generateScripts** (List< DsProfile > profiles, Date initDate, String clientTimeZone, String requestUser, Boolean fullGeneration)
- String **generateAllDigitalSignageScripts** (Date initDate, String clientTimeZone, String requestUser)
- List< **DigitalSignageZipEntry** > **generateDigitalSignageEntry** (Date initDate, DsProfile profile, String clientTimeZone, DsTask dsTask) throws NoShortTrailerAvailableException, DigitalSignageTemplateException
- List< DsProfile > **findAllProfiles** ()
- void **disableAutoUpdateTimer** ()

Static Public Member Functions

- static String **GetShortTrailer** (Content content, String directoryPath)
- static String **getAdvertisementVideo** (Advertisement adv, String directoryPath, boolean portraitMode)
- static **DigitalSignageZipEntry generateIntroScript** (**DsIntroVideoVariables** variables, ParamBundle messages) throws JAXBException, Exception
- static String **getClipVideo** (String videoPath, ParamBundle messages)

Static Public Attributes

- static final String **jndiBase** = "java:module/"
- static final String **CONTENT_TITLE_SUBSTRING** = "..."
- static final String **METADATA_CINEMA_ID** = "cinemaId"
- static final String **TRAILERS_EXTENSION** = ".mp4"
- static final String **BACKGROUND_IMAGES_FOLDER** = "DigitalSignage.LocalBackgroundImagesFolder"
- static final String **DS_RESOURCES_FOLDER** = "/digitalSignage"
- static final String **POSTERS_FOLDER** = "DigitalSignage.LocalPostersFolder"
- static final String **CLIPS_FOLDER** = "DigitalSignage.LocalClipsFolder"
- static final String **ADVERTISING_FOLDER** = "Advertisement.LocalDefaultFolder"
- static final String **DSGENERATION_FOLDER** = "dsFolderGenerationBase"
- static final String **POSTERS_EXTENSION** = ".jpeg"
- static final String **BACKGROUND_EXTENSION** = ".png"
- static final String **GENERATED_VIDEOS_EXTENSION** = ".avi"
- static final String **SCRIPTS_EXTENSION** = ".avs"
- static final String **TEMPLATES_FOLDER** = ""

- static final String **SHE_TEMPLATE** = "/Base.she"
- static final String **SHE_XML_TEMPLATE** = "/Base.xml"
- static final String **H_INTRO_AVS** = "/Intro-H.avs"
- static final String **V_INTRO_AVS** = "/Intro-V.avs"
- static final String **INTRO_TITLE** = "PROGRAMACIÓN DEL %s AL %s"
- static final String **VAR_INTRO_TITLE** = "@varTitle"
- static final String **VAR_INTRO_BACK** = "@varFondo"
- static final String **VAR_INTRO_FONT_COLOR** = "@varColor"
- static final String **VAR_INTRO_FONT_SIZE** = "@varSize"
- static final String **VAR_INTRO_TEXT_IN** = "@varTextIn"
- static final String **VAR_INTRO_TEXT_OUT** = "@varTextOut"
- static final String **VAR_INTRO_TITLE_X** = "@varXTitle"
- static final String **VAR_INTRO_TITLE_Y** = "@varYTitle"
- static final String **VAR_INTRO_FONT** = "@varFont"
- static final String **MSG_INTRO_FONT_COLOR** = "DigitalSignage.IntroFontColor"
- static final String **MSG_INTRO_FONT_SIZE** = "DigitalSignage.IntroFontSize"
- static final String **MSG_INTRO_TEXT_IN** = "DigitalSignage.IntroTextIn"
- static final String **MSG_INTRO_TEXT_OUT** = "DigitalSignage.IntroTextOut"
- static final String **MSG_INTRO_TITLE_X** = "DigitalSignage.IntroTitleX"
- static final String **MSG_INTRO_TITLE_Y** = "DigitalSignage.IntroTitleY"
- static final String **MSG_INTRO_TEXT_FONT** = "DigitalSignage.IntroTextFont"
- static final String **SHE_CURRENT_NAME** = "@CurrentName"
- static final String **SHE_CURRENT_NAME_OLD** = "@CurrentNameOld"
- static final String **SHE_PROGRAM_NAME_OLD** = "@ProgramNameOld"
- static final String **SHE_CURRENT_NAME_NEW** = "@CurrentNameNew"
- static final String **SHE_PROGRAM_NAME_NEW** = "@ProgramNameNew"
- static final String **SHE_NEW_PROGRAM_ACTIVATE_DATE** = "@NewProgramActivateDate"
- static final String **SHE_XML_RESOLUTION** = "@Resolution"
- static final String **SHE_XML_SOURCESNUM** = "@SourcesNum"
- static final String **SHE_XML_SOURCES** = "@SourceVideos"
- static final String **SHE_XML_SOURCE_TEMPLATE** = "<Source id=\"%s\" duration=\"20\" fileName=\"%s\" />"
- static final String **RED_AGE** = "red"
- static final String **GREEN_AGE** = "green"
- static final String **CONTENT_3D** = "_3D"
- static final String **CONTENT_VO** = "_VO"
- static final SimpleDateFormat **sessionDateFormat** = new SimpleDateFormat("HH:mm")
- static final SimpleDateFormat **programInitDateFormat** = new SimpleDateFormat("yyyy-MM-dd")
- static final SimpleDateFormat **fileNameDateFormat** = new SimpleDateFormat("ddMMyyyy")
- static final SimpleDateFormat **logDateFormat** = new SimpleDateFormat("dd-MM-yyyy")
- static final SimpleDateFormat **titleDateFormat** = new SimpleDateFormat("dd/MM")
- static final String **POSTER_BASE_URL** = "http://www.versiondigital.es/vdserv/lgimage?id=%s"
- static final String **RESOURCE_BACKGROUND** = "background"
- static final String **RESOURCE_POSTER** = "poster"
- static final String **RESOURCE_TRAILER** = "trailer"
- static final String **ADVERTISEMENT** = "advertisement"
- static final String **RESOURCE_IMAGE** = "image"

- static final String **RESOURCE_INTRO_VIDEO** = "intro"
- static final String **SPANISH_LOCALE** = "es-ES"

Detailed Description

Digital signage template utilities.

Member Enumeration Documentation

enum com::okode::versiondigital::core::digitalsignage::DigitalSignageTemplatesService::AvailableTemplates

Allowed states for DsTasks

Member Function Documentation

List<DigitalSignageZipEntry> com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService.generateDigitalSignageEntry (Date *initDate*, DsProfile *profile*, String *clientTimeZone*, DsTask *dsTask*) throws NoShortTrailerAvailableException, DigitalSignageTemplateException

Generate script for Digital Signage.

Parameters:

contentId	
cinemaRoomId	
cinemaId	
templateType	

Returns:

Exceptions:

DigitalSignageTemplateException, NoShortTrailerAvailableException	

static DigitalSignageZipEntry com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService.generateIntroScript (DsIntroVideoVariables *variables*, ParamBundle *messages*) throws JAXBException, Exception [static]

Create Zip Entry for cinema intro video.

Parameters:

variables	

Returns:

Exceptions:

JAXBException	
Exception	

static String com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService.getAdvertisementVideo (Advertisement *adv*, String *directoryPath*, boolean *portraitMode*) [static]

Parameters:

adv	

Returns:

static String com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService.getClipVideo (String *videoPath*, ParamBundle *messages*) [static]

Parameters:

adv	

Returns:

static String com.okode.versiondigital.core.digitalsignage.DigitalSignageTemplatesService.GetShortTrailer (Content *content*, String *directoryPath*) [static]

Parameters:

content	

Returns:

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/DigitalSignageTemplatesService.java

com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlInfo Class Reference

Inherits Serializable.

Public Member Functions

- String **getUsedTemplate** ()
- void **setUsedTemplate** (String usedTemplate)
- Date **getInitDate** ()
- void **setInitDate** (Date initDate)
- Date **getEndDate** ()
- void **setEndDate** (Date endDate)
- Date **getLastEventModified** ()
- void **setLastEventModified** (Date lastEventModified)
- List< **DigitalSignageXmlResource** > **getResources** ()
- void **setResources** (List< **DigitalSignageXmlResource** > resources)
- List< **DigitalSignageXmlMetadata** > **getMetadata** ()
- void **setMetadata** (List< **DigitalSignageXmlMetadata** > metadata)
- String **getCinemaFolderName** ()
- void **setCinemaFolderName** (String cinemaFolderName)
- String **getVideoFolder** ()
- void **setVideoFolder** (String videoFolder)
- String **getFinalVideoName** ()
- void **setFinalVideoName** (String finalVideoName)
- String **getSheFileName** ()
- void **setSheFileName** (String sheFileName)
- String **getSheXmlFileNameNew** ()
- void **setSheXmlFileNameNew** (String sheXmlFileNameNew)
- String **getSheXmlFileNameOld** ()
- void **setSheXmlFileNameOld** (String sheXmlFileNameOld)
- Boolean **getOldVideo** ()
- void **setOldVideo** (Boolean oldVideo)
- String **getAvsFile** ()
- void **setAvsFile** (String avsFile)
- String **getFtpDestinationPath** ()
- void **setFtpDestinationPath** (String ftpDestinationPath)
- String **getFtpCrendentialsUser** ()
- void **setFtpCrendentialsUser** (String ftpCrendentialsUser)
- String **getFtpCrendentialsPass** ()
- void **setFtpCrendentialsPass** (String ftpCrendentialsPass)
- String **getGroupName** ()
- void **setGroupName** (String groupName)
- Boolean **getCopyAdvertisementVideos** ()
- void **setCopyAdvertisementVideos** (Boolean copyAdvertisementVideos)

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/model/DigitalSignageXmlInfo.java

com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlMetadata Class Reference

Inherits Serializable.

Public Member Functions

- String **getName** ()
- void **setName** (String name)
- String **getValue** ()
- void **setValue** (String value)
- String **getDescription** ()
- void **setDescription** (String description)
- **DigitalSignageXmlMetadata** (String name, String value, String description)

Detailed Description

Class for xml metadata.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/model/DigitalSignageXmlMetadata.java

com.okode.versiondigital.core.digitalsignage.model.DigitalSignageXmlResource Class Reference

Inherits Serializable.

Public Member Functions

- String **getName** ()
- void **setName** (String name)
- String **getLocalPath** ()
- void **setLocalPath** (String localPath)
- String **getDownloadPath** ()
- void **setDownloadPath** (String downloadPath)

Detailed Description

Class for xml resources.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/model/DigitalSignageXmlResource.java

com.okode.versiondigital.core.digitalsignage.model.DigitalSignageZipEntry Class Reference

Public Member Functions

- String **getScriptFileName** ()
- void **setScriptFileName** (String scriptFileName)
- String **getScriptFileContent** ()
- void **setScriptFileContent** (String scriptFileContent)
- String **getXmlFileName** ()
- void **setXmlFileName** (String xmlFileName)
- String **getXmlFileContent** ()
- void **setXmlFileContent** (String xmlFileContent)
- String **getGeneratedVideoName** ()
- void **setGeneratedVideoName** (String generatedVideoName)

Detailed Description

One entry in the dsTask zip file, with a script file and a xml file.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/model/DigitalSignageZipEntry.java

com.okode.versiondigital.core.digitalsignage.DsIntroVideoVariables Class Reference

Public Member Functions

- IntroType **getIntroType** ()
- void **setIntroType** (IntroType introType)
- Date **getInitDate** ()
- void **setInitDate** (Date initDate)
- Date **getEndDate** ()
- void **setEndDate** (Date endDate)
- String **getUsedTemplate** ()
- void **setUsedTemplate** (String usedTemplate)
- String **getFtpFolder** ()
- void **setFtpFolder** (String ftpFolder)
- String **getGroupName** ()
- void **setGroupName** (String groupName)
- String **getFtpPass** ()
- void **setFtpPass** (String ftpPass)
- String **getFtpUser** ()
- void **setFtpUser** (String ftpUser)
- String **getVideoLocalPath** ()
- void **setVideoLocalPath** (String videoLocalPath)
- String **getDownloadVideoPath** ()
- void **setDownloadVideoPath** (String downloadVideoPath)
- Long **getCinemaId** ()
- void **setCinemaId** (Long cinemaId)
- String **getFinalVideoName** ()
- void **setFinalVideoName** (String finalVideoName)
- String **getAvsFileName** ()
- void **setAvsFileName** (String avsFileName)
- String **getSheFileName** ()
- void **setSheFileName** (String sheFileName)
- String **getXmlSheFileName** ()
- void **setXmlSheFileName** (String xmlSheFileName)
- Boolean **getOldGeneration** ()
- void **setOldGeneration** (Boolean oldGeneration)
- String **getVideoFolder** ()
- void **setVideoFolder** (String videoFolder)
- **DigitalSignageTemplatesService.AvailableTemplates getTemplate** ()
- void **setTemplate** (**DigitalSignageTemplatesService.AvailableTemplates** template)

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/DsIntroVideoVariables.java

com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplate Class Reference

Inheritance diagram for com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplate:

Public Member Functions

- List< **DigitalSignageZipEntry** > **generateDigitalSignageScript** (Date initDate, DsProfile profile, DsTask dsTask, ParamBundle messages, String clientTimeZone) throws Exception

Static Public Attributes

- static final String **RESOURCES_FOLDER** = "\\8ProgramGrid-H\\"
- static final String **SCRIPT_SOURCE** = "/8ProgramGrid-H.avs"
- static final String **jndiBase** = "java:module/"
- static final String **DOWNLOAD_FILES_BASE_URL** = "http://www.versiondigital.es/carteleria/8ProgramGrid-H/"
- static final String **BASE_LOCAL_PATH** = "C:\\work\\CarteleriaDigital\\Plantillas\\8ProgramGrid-H\\"
- static final String **TEMPLATE_TITLE** = "PROGRAMACIÓN SEMANAL DEL %s AL %s"
- static final String **BACKGROUND** = "bg.bmp"
- static final String **PARTIAL_RATE_ND** = "ND"
- static final String **PARTIAL_RATE_PC** = "PC"
- static final String **PARTIAL_RATE_TP** = "TP"
- static final String **PARTIAL_RATE_7** = "7"
- static final String **PARTIAL_RATE_12** = "12"
- static final String **PARTIAL_RATE_16** = "16"
- static final String **PARTIAL_RATE_18** = "18"
- static final String **PARTIAL_RATE_X** = "X"
- static final String **PARTIAL_RATE_3D** = "_3D"
- static final String **PARTIAL_RATE_VO** = "_VO"
- static final String **PARTIAL_RATE_EXTENSION** = ".bmp"
- static final String **RATE_EMPTY** = "empty.bmp"
- static final String **RATE_PC** = "PC.bmp"
- static final String **RATE_PC_3D** = "PC_3D.bmp"
- static final String **RATE_PC_VO** = "PC_VO.bmp"
- static final String **RATE_PC_3D_VO** = "PC_3D_VO.bmp"
- static final String **RATE_TP** = "TP.bmp"
- static final String **RATE_TP_3D** = "TP_3D.bmp"
- static final String **RATE_TP_VO** = "TP_VO.bmp"
- static final String **RATE_TP_3D_VO** = "TP_3D_VO.bmp"
- static final String **RATE_7** = "7.bmp"
- static final String **RATE_7_3D** = "7_3D.bmp"
- static final String **RATE_7_VO** = "7_VO.bmp"
- static final String **RATE_7_3D_VO** = "7_3D_VO.bmp"

- static final String **RATE_12** = "12.bmp"
- static final String **RATE_12_3D** = "12_3D.bmp"
- static final String **RATE_12_VO** = "12_VO.bmp"
- static final String **RATE_12_3D_VO** = "12_3D_VO.bmp"
- static final String **RATE_16** = "16.bmp"
- static final String **RATE_16_3D** = "16_3D.bmp"
- static final String **RATE_16_VO** = "16_VO.bmp"
- static final String **RATE_16_3D_VO** = "16_3D_VO.bmp"
- static final String **RATE_18** = "18.bmp"
- static final String **RATE_18_3D** = "18_3D.bmp"
- static final String **RATE_18_VO** = "18_VO.bmp"
- static final String **RATE_18_3D_VO** = "18_3D_VO.bmp"
- static final String **RATE_X** = "X.bmp"
- static final String **RATE_X_3D** = "X_3D.bmp"
- static final String **RATE_X_VO** = "X_VO.bmp"
- static final String **RATE_X_3D_VO** = "X_3D_VO.bmp"
- static final String **VAR_FONDO** = "@varFondo"
- static final String **VAR_RATE1** = "@varRate1"
- static final String **VAR_RATE2** = "@varRate2"
- static final String **VAR_RATE3** = "@varRate3"
- static final String **VAR_RATE4** = "@varRate4"
- static final String **VAR_RATE5** = "@varRate5"
- static final String **VAR_RATE6** = "@varRate6"
- static final String **VAR_RATE7** = "@varRate7"
- static final String **VAR_RATE8** = "@varRate8"
- static final String **VAR_ADVERTISEMENT** = "@varAdvertisement"
- static final String **VAR_HEADER_TEXT** = "@varHeaderText"
- static final String **VAR_TITLE1** = "@varTitle1"
- static final String **VAR_ROOM1** = "@varRoom1"
- static final String **VAR_SESSION1_1** = "@varSessions1_1"
- static final String **VAR_SESSION1_2** = "@varSessions1_2"
- static final String **VAR_SESSION1_3** = "@varSessions1_3"
- static final String **VAR_SESSION1_4** = "@varSessions1_4"
- static final String **VAR_TITLE2** = "@varTitle2"
- static final String **VAR_ROOM2** = "@varRoom2"
- static final String **VAR_SESSION2_1** = "@varSessions2_1"
- static final String **VAR_SESSION2_2** = "@varSessions2_2"
- static final String **VAR_SESSION2_3** = "@varSessions2_3"
- static final String **VAR_SESSION2_4** = "@varSessions2_4"
- static final String **VAR_TITLE3** = "@varTitle3"
- static final String **VAR_ROOM3** = "@varRoom3"
- static final String **VAR_SESSION3_1** = "@varSessions3_1"
- static final String **VAR_SESSION3_2** = "@varSessions3_2"
- static final String **VAR_SESSION3_3** = "@varSessions3_3"
- static final String **VAR_SESSION3_4** = "@varSessions3_4"
- static final String **VAR_TITLE4** = "@varTitle4"
- static final String **VAR_ROOM4** = "@varRoom4"
- static final String **VAR_SESSION4_1** = "@varSessions4_1"
- static final String **VAR_SESSION4_2** = "@varSessions4_2"
- static final String **VAR_SESSION4_3** = "@varSessions4_3"

- static final String **VAR_SESSION4_4** = "@varSessions4_4"
- static final String **VAR_TITLE5** = "@varTitle5"
- static final String **VAR_ROOM5** = "@varRoom5"
- static final String **VAR_SESSION5_1** = "@varSessions5_1"
- static final String **VAR_SESSION5_2** = "@varSessions5_2"
- static final String **VAR_SESSION5_3** = "@varSessions5_3"
- static final String **VAR_SESSION5_4** = "@varSessions5_4"
- static final String **VAR_TITLE6** = "@varTitle6"
- static final String **VAR_ROOM6** = "@varRoom6"
- static final String **VAR_SESSION6_1** = "@varSessions6_1"
- static final String **VAR_SESSION6_2** = "@varSessions6_2"
- static final String **VAR_SESSION6_3** = "@varSessions6_3"
- static final String **VAR_SESSION6_4** = "@varSessions6_4"
- static final String **VAR_TITLE7** = "@varTitle7"
- static final String **VAR_ROOM7** = "@varRoom7"
- static final String **VAR_SESSION7_1** = "@varSessions7_1"
- static final String **VAR_SESSION7_2** = "@varSessions7_2"
- static final String **VAR_SESSION7_3** = "@varSessions7_3"
- static final String **VAR_SESSION7_4** = "@varSessions7_4"
- static final String **VAR_TITLE8** = "@varTitle8"
- static final String **VAR_ROOM8** = "@varRoom8"
- static final String **VAR_SESSION8_1** = "@varSessions8_1"
- static final String **VAR_SESSION8_2** = "@varSessions8_2"
- static final String **VAR_SESSION8_3** = "@varSessions8_3"
- static final String **VAR_SESSION8_4** = "@varSessions8_4"

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/eightprogramgrid horizontal/EightProgramGridHorizontalTemplate.java

com.okode.versiondigital.core.digitalsignage.eightprogramgridhorizontal.EightProgramGridHorizontalTemplateVariables Class Reference

Public Member Functions

- String **getVarFondo** ()
- void **setVarFondo** (String varFondo)
- String **getAdvertisement** ()
- void **setAdvertisement** (String advertisement)
- String **getHeaderText** ()
- void **setHeaderText** (String headerText)
- List< String > **getVarRates** ()
- void **setVarRates** (List< String > varRates)
- List< String > **getVarContentTitles** ()
- void **setVarContentTitles** (List< String > varContentTitles)
- List< String > **getVarRooms** ()
- void **setVarRooms** (List< String > varRooms)
- List< List< String > > **getVarSessions** ()
- void **setVarSessions** (List< List< String >> varSessions)
- Date **getInitDate** ()
- void **setInitDate** (Date initDate)
- Date **getEndDate** ()
- void **setEndDate** (Date endDate)
- String **getFtpFolder** ()
- void **setFtpFolder** (String ftpFolder)
- String **getGroupName** ()
- void **setGroupName** (String groupName)
- List< String > **getDownloadVarRates** ()
- void **setDownloadVarRates** (List< String > downloadVarRates)

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/eightprogramgridhorizontal/EightProgramGridHorizontalTemplateVariables.java

com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplate Class Reference

Inheritance diagram for com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplate:

Public Types

- enum **AvailableMarks** { **markNone**, **markVO3D**, **markVO**, **mark3D** }

Public Member Functions

- List< **DigitalSignageZipEntry** > **generateDigitalSignageScript** (Date initDate, DsProfile profile, DsTask dsTask, ParamBundle messages, String clientTimeZone) throws AdvertisementVideoException, OptomaIntegrationException, Exception

Static Public Attributes

- static final String **NIGHT_SESSION_KEY** = "night_session"
- static final String **MORNING_SESSION_KEY** = "morning_session"
- static final String **RESOURCES_FOLDER** = "\\8ProgramGrid-V\\"
- static final String **SCRIPT_SOURCE** = "/8ProgramGrid-V.avs"
- static final String **jndiBase** = "java:module/"
- static final String **DOWNLOAD_FILES_BASE_URL** = "http://www.versiondigital.es/carteleria/8ProgramGrid-V/"
- static final String **BACKGROUND_LOCAL_PATH** = "C:\\work\\CarteleriaDigital\\Plantillas\\8ProgramGrid-V\\background.bmp"
- static final String **MARK_NONE_LOCAL_PATH** = "C:\\work\\CarteleriaDigital\\Plantillas\\8ProgramGrid-V\\mark_none.bmp"
- static final String **MARK_3D_LOCAL_PATH** = "C:\\work\\CarteleriaDigital\\Plantillas\\8ProgramGrid-V\\mark_3D.bmp"
- static final String **MARK_3D_VO_LOCAL_PATH** = "C:\\work\\CarteleriaDigital\\Plantillas\\8ProgramGrid-V\\mark_3D_VO.bmp"
- static final String **MARK_VO_LOCAL_PATH** = "C:\\work\\CarteleriaDigital\\Plantillas\\8ProgramGrid-V\\mark_VO.bmp"
- static final String **TEMPLATE_TITLE** = "PROGRAMACIÓN SEMANAL DEL %s AL %s"
- static final String **BACKGROUND** = "background.bmp"
- static final String **Mark_3D** = "mark_3D.bmp"
- static final String **Mark_3D_VO** = "mark_3D_VO.bmp"
- static final String **Mark_VO** = "mark_VO.bmp"
- static final String **Mark_NONE** = "mark_none.bmp"
- static final String **VAR_Mark_3D** = "mark3D"
- static final String **VAR_Mark_3D_VO** = "markVO3D"
- static final String **VAR_Mark_VO** = "markVO"
- static final String **VAR_Mark_NONE** = "markNone"
- static final String **RateTP** = "rate_TP.bmp"
- static final String **Rate7** = "rate_7.bmp"
- static final String **Rate12** = "rate_12.bmp"
- static final String **Rate16** = "rate_16.bmp"

- static final String **Rate18** = "rate_18.bmp"
- static final String **RatePC** = "rate_PC.bmp"
- static final String **RateND** = "rate_ND.bmp"
- static final String **RateX** = "rate_X.bmp"
- static final String **RateNONE** = "rate_none.bmp"
- static final String **VAR_ANUNCIO** = "@varAnuncio"
- static final String **VAR_TITLE** = "@varTitle"
- static final String **VAR_MATINAL** = "@varMatinal"
- static final String **VAR_GOLFA** = "@varGolfa"
- static final String **VAR_RATE1** = "@varRate1"
- static final String **VAR_RATE2** = "@varRate2"
- static final String **VAR_RATE3** = "@varRate3"
- static final String **VAR_RATE4** = "@varRate4"
- static final String **VAR_RATE5** = "@varRate5"
- static final String **VAR_RATE6** = "@varRate6"
- static final String **VAR_RATE7** = "@varRate7"
- static final String **VAR_RATE8** = "@varRate8"
- static final String **VAR_ROOM1** = "@varRoom1"
- static final String **VAR_CONTENT_TITLE1** = "@varContentTitle1"
- static final String **VAR_SESSIONS1** = "@varSessions1"
- static final String **VAR_MARK1** = "@mark1"
- static final String **VAR_ROOM2** = "@varRoom2"
- static final String **VAR_CONTENT_TITLE2** = "@varContentTitle2"
- static final String **VAR_SESSIONS2** = "@varSessions2"
- static final String **VAR_MARK2** = "@mark2"
- static final String **VAR_ROOM3** = "@varRoom3"
- static final String **VAR_CONTENT_TITLE3** = "@varContentTitle3"
- static final String **VAR_SESSIONS3** = "@varSessions3"
- static final String **VAR_MARK3** = "@mark3"
- static final String **VAR_ROOM4** = "@varRoom4"
- static final String **VAR_CONTENT_TITLE4** = "@varContentTitle4"
- static final String **VAR_SESSIONS4** = "@varSessions4"
- static final String **VAR_MARK4** = "@mark4"
- static final String **VAR_ROOM5** = "@varRoom5"
- static final String **VAR_CONTENT_TITLE5** = "@varContentTitle5"
- static final String **VAR_SESSIONS5** = "@varSessions5"
- static final String **VAR_MARK5** = "@mark5"
- static final String **VAR_ROOM6** = "@varRoom6"
- static final String **VAR_CONTENT_TITLE6** = "@varContentTitle6"
- static final String **VAR_SESSIONS6** = "@varSessions6"
- static final String **VAR_MARK6** = "@mark6"
- static final String **VAR_ROOM7** = "@varRoom7"
- static final String **VAR_CONTENT_TITLE7** = "@varContentTitle7"
- static final String **VAR_SESSIONS7** = "@varSessions7"
- static final String **VAR_MARK7** = "@mark7"
- static final String **VAR_ROOM8** = "@varRoom8"
- static final String **VAR_CONTENT_TITLE8** = "@varContentTitle8"
- static final String **VAR_SESSIONS8** = "@varSessions8"
- static final String **VAR_MARK8** = "@mark8"

Detailed Description

Digital Signage script generator for 8ProgramGrid-Vertical template.

Member Enumeration Documentation

enum
com::okode::versiondigital::core::digitalsignage::eightprogramgridvertical::EightProgramGridVerticalTemplate::AvailableMarks

Available marks

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/eightprogramgridvertical/EightProgramGridVerticalTemplate.java

com.okode.versiondigital.core.digitalsignage.eightprogramgridvertical.EightProgramGridVerticalTemplateVariables Class Reference

Public Member Functions

- String **getVarAnuncio** ()
- void **setVarAnuncio** (String varAnuncio)
- String **getVarTitle** ()
- void **setVarTitle** (String varTitle)
- String **getVarMatinal** ()
- void **setVarMatinal** (String varMatinal)
- String **getVarGolfa** ()
- void **setVarGolfa** (String varGolfa)
- List< String > **getVarRates** ()
- void **setVarRates** (List< String > varRates)
- List< String > **getDownloadVarRates** ()
- void **setDownloadVarRates** (List< String > downloadVarRates)
- List< String > **getVarRooms** ()
- void **setVarRooms** (List< String > varRooms)
- List< String > **getVarContentTitles** ()
- void **setVarContentTitles** (List< String > varContentTitles)
- List< String > **getVarSessions** ()
- void **setVarSessions** (List< String > varSessions)
- List< String > **getMarks** ()
- void **setMarks** (List< String > marks)
- String **getDownloadMarkNone** ()
- void **setDownloadMarkNone** (String downloadMarkNone)
- String **getDownloadMarkVO3D** ()
- void **setDownloadMarkVO3D** (String downloadMarkVO3D)
- String **getDownloadMarkVO** ()
- void **setDownloadMarkVO** (String downloadMarkVO)
- String **getDownloadMark3D** ()
- void **setDownloadMark3D** (String downloadMark3D)
- String **getDownloadBackground** ()
- void **setDownloadBackground** (String downloadBackground)
- Date **getInitDate** ()
- void **setInitDate** (Date initDate)
- Date **getEndDate** ()
- void **setEndDate** (Date endDate)
- String **getFtpFolder** ()
- void **setFtpFolder** (String ftpFolder)
- String **getGroupName** ()
- void **setGroupName** (String groupName)

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/eightprogramgridvertical/EightProgramGridVerticalTemplateVariables.java

com.okode.versiondigital.core.digitalsignage.exceptions.NoShortTrailerAvailableException Class Reference

Inherits ServiceException.

Public Member Functions

- **NoShortTrailerAvailableException** ()
- **NoShortTrailerAvailableException** (Exception he)

Detailed Description

Exception for Digital Signage trailer not available.

Constructor & Destructor Documentation

com.okode.versiondigital.core.digitalsignage.exceptions.NoShortTrailerAvailableException.NoShortTrailerAvailableException ()

Exception for Digital Signage trailer not available.

com.okode.versiondigital.core.digitalsignage.exceptions.NoShortTrailerAvailableException.NoShortTrailerAvailableException (Exception *he*)

Exception for Digital Signage trailer not available.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/exceptions/NoShortTrailerAvailableException.java

com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupInfo Class Reference

Public Member Functions

- String **getGroupName** ()
- void **setGroupName** (String groupName)
- String **getFtpPath** ()
- void **setFtpPath** (String ftpPath)
- String **getGroupUid** ()
- void **setGroupUid** (String groupUid)
- String **getFtpUid** ()
- void **setFtpUid** (String ftpUid)
- String **getFtpUser** ()
- void **setFtpUser** (String ftpUser)
- String **getFtpPass** ()
- void **setFtpPass** (String ftpPass)

Detailed Description

Class for Optoma groups info.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/optoma/OptomaGroupInfo.java

com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupsCache Class Reference

Inherits BaseService.

Public Member Functions

- **OptomaGroupInfo findGroupInfo** (String groupName) throws DigitalSignageServiceException, OptomaIntegrationException
- void **fillGroups** () throws DigitalSignageServiceException, OptomaIntegrationException

---

Member Function Documentation

void com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupsCache.fillGroups () throws DigitalSignageServiceException, OptomaIntegrationException

Get info of all groups.

OptomaGroupInfo com.okode.versiondigital.core.digitalsignage.optoma.OptomaGroupsCache.findGroupInfo (String *groupName*) throws DigitalSignageServiceException, OptomaIntegrationException

Get cached group info.

Parameters:

groupName	

Returns:

Exceptions:

OptomaIntegrationException	

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/optoma/OptomaGroupsCache.java

com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationException Class Reference

Inherits ServiceException.

Public Member Functions

- **OptomaIntegrationException** ()
- **OptomaIntegrationException** (String message)
- **OptomaIntegrationException** (Exception ex)

Detailed Description

Exception for Optoma integration problems.

Constructor & Destructor Documentation

com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationException.OptomaIntegrationException ()

Exception for Optoma integration problems.

com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationException.OptomaIntegrationException (String *message*)

Exception for Optoma integration problems.

com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationException.OptomaIntegrationException (Exception *ex*)

Exception for Optoma integration problems.

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/optoma/OptomaIntegrationException.java

com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService Class Reference

Inherits BaseService.

Public Member Functions

- List< **OptomaGroupInfo** > **getGroupsInfo** (String ip, String port) throws OptomaIntegrationException
- Boolean **setAutoUpdateGroupOn** (String ip, String port, String groupUid, String ftpUid, String hour, String minute) throws OptomaIntegrationException
- Boolean **setAutoUpdateGroupOn** (String ip, String port, String groupUid, String ftpUid) throws OptomaIntegrationException
- GregorianCalendar **getServerTime** (String ip, String port) throws OptomaIntegrationException
- void **setAutoUpdateGroupOff** (String ip, String port, String groupUid, String ftpUid) throws OptomaIntegrationException

---

Member Function Documentation

List<OptomaGroupInfo> com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService.getGroupsInfo (String *ip*, String *port*) throws OptomaIntegrationException

Get optoma groups info.

Returns:

Exceptions:

OptomaIntegrationException	

GregorianCalendar com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService.getServerTime (String *ip*, String *port*) throws OptomaIntegrationException

Get Optoma server time in UTC.

Parameters:

ip	
port	

Returns:

Exceptions:

OptomaIntegrationException	

void com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService.setAu

toUpdateGroupOff (String *ip*, String *port*, String *groupUid*, String *ftpUid*) throws OptomaIntegrationException

Disable auto update for an Optoma group.

Parameters:

ip	
port	
groupUid	
ftpUid	

Exceptions:

OptomaIntegrationException	

Boolean com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService.setAutoUpdateGroupOn (String *ip*, String *port*, String *groupUid*, String *ftpUid*, String *hour*, String *minute*) throws OptomaIntegrationException

Enable auto update for an Optoma group.

Parameters:

ip	WebServices server IP.
port	WebServices server port.
groupUid	Group UID.
ftpUid	FTP UID.

Exceptions:

OptomaIntegrationException	

Boolean com.okode.versiondigital.core.digitalsignage.optoma.OptomaIntegrationService.setAutoUpdateGroupOn (String *ip*, String *port*, String *groupUid*, String *ftpUid*) throws OptomaIntegrationException

Enable auto update for an Optoma group.

Parameters:

ip	WebServices server IP.
port	WebServices server port.
groupUid	Group UID.
ftpUid	FTP UID.

Exceptions:

OptomaIntegrationException	

The documentation for this class was generated from the following file:

- /core/src/main/java/com/okode/versiondigital/core/digitalsignage/optoma/OptomaIntegrationService.java

Anexo II

Documentación – C#

Core del Media Server

Namespace Index

Packages

Here are the packages with brief descriptions (if available):

MediaServerEngine
MediaServerEngine.Log

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MediaServerEngine.DSResource (Class for DSResources)
MediaServerEngine.DSTask
MediaServerEngine.DsTaskFtpFolderInfo (FTP crendentials and folder info)
MediaServerEngine.DSVideo (Class with processed info of XML file)
MediaServerEngine.Log.Logger (Singleton Logger class. Provides an event for log**)**
MediaServerEngine.MediaEngine
MediaServerEngine.MediaEngineConfig (Class for media engine configuration)
MediaServerEngine.Log.VersionDigitalLogger (Class for communication functions)

Namespace Documentation

Package MediaServerEngine

Packages

- package **Log**

Classes

- class **DSResource**
- *Class for DSResources.* class **DSTask**
- class **DsTaskFtpFolderInfo**
- *FTP crendentials and folder info.* class **DSVideo**
- *Class with processed info of XML file.* class **MediaEngine**
- class **MediaEngineConfig**
- *Class for media engine configuration.* class **MediaEngineUtil**

Package MediaServerEngine.Log

Classes

- class **Logger**
- *Singleton **Logger** class. Provides an event for log subscription.* class **VersionDigitalLogger**

Class for Version Digital communication functions.

Class Documentation

MediaServerEngine.DSResource Class Reference

Class for DSResources.

Public Types

- enum **ResourceTypes** { **trailer**, **poster**, **advertisement**, **background**, **image** }

Properties

- string **LocalPath** [get, set]
  Local file path.
- string **DownloadPath** [get, set]
  Downloadeable file location.
- bool **Downloaded** [get, set]
  Indicates if resource was downloaded.
- bool **DefaultUsed** [get, set]
  Indicates if resource was copied from the default resource.
- ResourceTypes **ResourceType** [get, set]
  Resource type.

Detailed Description

Class for DSResources.

Property Documentation

bool MediaServerEngine.DSResource.DefaultUsed [get, set]

Indicates if resource was copied from the default resource.

bool MediaServerEngine.DSResource.Downloaded [get, set]

Indicates if resource was downloaded.

string MediaServerEngine.DSResource.DownloadPath [get, set]

Downloadeable file location.

string MediaServerEngine.DSResource.LocalPath [get, set]

Local file path.

ResourceTypes MediaServerEngine.DSResource.ResourceType [get, set]

Resource type.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/DSResource.cs

MediaServerEngine.DSTask Class Reference

Public Types

- enum **TaskState** { **IN_PROGRESS** = 0, **SCRIPT_GENERATION_ERROR** = 1, **DOWNLOAD_PENDING** = 2, **VIDEO_BUILD_PENDING** = 3, **VIDEO_BUILD_ERROR** = 4, **COMPLETE** = 5, **COMPLETE_WITHERRORS** = 6, **OPTOMA_AUTOUPDATED** = 7, **OPTOMA_AUTOUPDATED_FAILS** = 8, **OPTOMA_AUTOUPDATED_WITHERRORS** = 9 }

Possible tasks states (copy of Versión Digital enum). Public Member Functions

- **DSTask** (int idTask, **MediaEngineConfig** configuration)
- void **UpdateTaskState** (**TaskState** newState, string message)
  Update global task state in Version Digital.
- void **ReportLog** (**MediaServerEngine.Log.VersionDigitalLogger.TaskLogLevel** level, string message)
  Send a log for this task to Version Digital.
- void **ReportLog** (**MediaServerEngine.Log.VersionDigitalLogger.TaskLogLevel** level, string message, string group)
  Send a log for this task to Version Digital.

Properties

- int **IdTask** [get, set]
  Identifier of the task.
- **TaskState State** [get]
  Task state.
- double **Progress** [get]
  Task progress.
- string **TaskFolder** [get]
  Task Folder.
- string **TaskUrl** [get, set]
  Task URL.
- string **TaskZip** [get, set]
  Task Zip file path.
- SortedList< string,
- **DsTaskFtpFolderInfo** > **Cinemas** [get, set]
  Cinemas and its ftp crendentials.
- SortedList< string,
- **DsTaskFtpFolderInfo** > **FtpFolders** [get, set]
  FTP Folders info.
- DateTime **NewProgDate** [get, set]
  New program init date.
- DateTime **OldProgDate** [get, set]
  Old program init date.
- Collection< string > **FtpUploadedResourcesPath** [get, set]

FTP Uploaded paths.

Member Enumeration Documentation

enum MediaServerEngine::DSTask::TaskState

Possible tasks states (copy of Versión Digital enum).

Member Function Documentation

void MediaServerEngine.DSTask.ReportLog (MediaServerEngine.Log.VersionDigitalLogger.TaskLogLevel *level*, string *message*)

Send a log for this task to Version Digital.

Parameters:

level	
message	

void MediaServerEngine.DSTask.ReportLog (MediaServerEngine.Log.VersionDigitalLogger.TaskLogLevel *level*, string *message*, string *group*)

Send a log for this task to Version Digital.

Parameters:

level	
message	

void MediaServerEngine.DSTask.UpdateTaskState (TaskState *newState*, string *message*)

Update global task state in Version Digital.

Parameters:

newState	
message	

Property Documentation

SortedList<string, DsTaskFtpFolderInfo> MediaServerEngine.DSTask.Cinemas [get, set]

Cinemas and its ftp crendentials.

SortedList<string, DsTaskFtpFolderInfo> MediaServerEngine.DSTask.FtpFolders [get, set]

FTP Folders info.

Collection<string> MediaServerEngine.DSTask.FtpUploadedResourcesPath [get, set]

FTP Uploaded paths.

int MediaServerEngine.DSTask.IdTask [get, set]

Identifier of the task.

DateTime MediaServerEngine.DSTask.NewProgDate [get, set]

New program init date.

DateTime MediaServerEngine.DSTask.OldProgDate [get, set]

Old program init date.

double MediaServerEngine.DSTask.Progress [get]

Task progress.

TaskState MediaServerEngine.DSTask.State [get]

Task state.

string MediaServerEngine.DSTask.TaskFolder [get]

Task Folder.

string MediaServerEngine.DSTask.TaskUrl [get, set]

Task URL.

string MediaServerEngine.DSTask.TaskZip [get, set]

Task Zip file path.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/DSTask.cs

MediaServerEngine.DsTaskFtpFolderInfo Class Reference

FTP crendentials and folder info.

Public Member Functions

- **DsTaskFtpFolderInfo** (string ftpFolder, string ftpUser, string ftpPass)

Properties

- string **FtpFolder** [get, set]
  FTP destination Folder.
- string **FtpUser** [get, set]
  FTP Crendentials User.
- string **FtpPass** [get, set]
  FTP Crendentials Pass.

Detailed Description

FTP crendentials and folder info.

Property Documentation

string MediaServerEngine.DsTaskFtpFolderInfo.FtpFolder [get, set]

FTP destination Folder.

string MediaServerEngine.DsTaskFtpFolderInfo.FtpPass [get, set]

FTP Crendentials Pass.

string MediaServerEngine.DsTaskFtpFolderInfo.FtpUser [get, set]

FTP Crendentials User.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/DsTaskFtpFolderInfo.cs

MediaServerEngine.DSVideo Class Reference

Class with processed info of XML file.

Public Types

- enum **Options** { **FALSE**, **TRUE**, **FAIL** }
- enum **States** { **OK**, **WARNING**, **ERROR** }

Public Member Functions

- **DSVideo** (XmlDocument xmlDoc, **DSTask** task)

Properties

- string **CinemaFolderName** [get, set]
  Cinema folder name.
- string **FinalVideoName** [get, set]
  Final name for generated video.
- DateTime **InitDate** [get, set]
  Video generated for schedule which starts on this date.
- DateTime **EndDate** [get, set]
  Video generated for schedule which ends on this date.
- DateTime **LastEventModified** [get, set]
  Date of the last event modified between init and end dates.
- string **SheFileName** [get, set]
  She file name.
- string **SheXmlFileName** [get, set]
  XML program file name.
- string **UsedTemplate** [get, set]
  Name of the used template.
- string **VideoFolder** [get, set]
  Name of the end video resources folder.
- string **FinalVideoPath** [get, set]
  Full path of the generated video.
- string **AvsFile** [get, set]
  Avs file name.
- bool **IsNew** [get, set]
  Indicates if video is for new schedule or current/old schedule.
- Options **AddedToVirtualDub** [get, set]
  Indicates if video was added to virtual dub job list.
- string **FtpDestinationPath** [get, set]
  End ftp path.
- string **Cinema** [get]
  Name of the cinema.
- **DSTask Task** [get, set]

DsTask.

- SortedList< string, **DSResource** > **Resources** [get]
  Parsed resources.
- string **FtpCrendentialsUser** [get, set]
  User credential for FTP.
- string **FtpCrendentialsPass** [get, set]
  Password credential for FTP.
- string **OptomaGroupName** [get, set]
  Optoma group name.
- bool **CopyAdvertisement** [get, set]
  Indicates if advertisement must be copied.

Detailed Description

Class with processed info of XML file.

Property Documentation

Options MediaServerEngine.DSVideo.AddedToVirtualDub [get, set]

Indicates if video was added to virtual dub job list.

string MediaServerEngine.DSVideo.AvsFile [get, set]

Avs file name.

string MediaServerEngine.DSVideo.Cinema [get]

Name of the cinema.

string MediaServerEngine.DSVideo.CinemaFolderName [get, set]

Cinema folder name.

bool MediaServerEngine.DSVideo.CopyAdvertisement [get, set]

Indicates if advertisement must be copied.

DateTime MediaServerEngine.DSVideo.EndDate [get, set]

Video generated for schedule which ends on this date.

string MediaServerEngine.DSVideo.FinalVideoName [get, set]

Final name for generated video.

string MediaServerEngine.DSVideo.FinalVideoPath [get, set]

Full path of the generated video.

string MediaServerEngine.DSVideo.FtpCrendentialsPass [get, set]

Password credential for FTP.

string MediaServerEngine.DSVideo.FtpCrendentialsUser [get, set]

User credential for FTP.

string MediaServerEngine.DSVideo.FtpDestinationPath [get, set]

End ftp path.

DateTime MediaServerEngine.DSVideo.InitDate [get, set]

Video generated for schedule which starts on this date.

bool MediaServerEngine.DSVideo.IsNew [get, set]

Indicates if video is for new schedule or current/old schedule.

DateTime MediaServerEngine.DSVideo.LastEventModified [get, set]

Date of the last event modified between init and end dates.

string MediaServerEngine.DSVideo.OptomaGroupName [get, set]

Optoma group name.

SortedList<string, DSResource> MediaServerEngine.DSVideo.Resources [get]

Parsed resources.

string MediaServerEngine.DSVideo.SheFileName [get, set]

She file name.

string MediaServerEngine.DSVideo.SheXmlFileName [get, set]

XML program file name.

DSTask MediaServerEngine.DSVideo.Task [get, set]

DsTask.

string MediaServerEngine.DSVideo.UsedTemplate [get, set]

Name of the used template.

string MediaServerEngine.DSVideo.VideoFolder [get, set]

Name of the end video resources folder.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/DSVideo.cs

MediaServerEngine.Log.Logger Class Reference

Singleton **Logger** class. Provides an event for log subscription.

Public Types

- enum **Level** { **TRACE** = 0, **DEBUG**, **INFO**, **WARNING**, **ERROR**, **FATAL**, **OFF** }

Public Member Functions

- void **Debug** (String message)
- void **Warning** (String message)
- void **Error** (String message)
- void **Info** (String message)
- bool **isEnabledFor** (Level level)
- void **Log** (String message, Level level)

Static Public Member Functions

- static void **StartLogger** (string logBaseString)
- static **Logger GetLogger** ()

Properties

- long **CountInfo** [get]
- long **CountWarn** [get]
- long **CountErr** [get]

Detailed Description

Singleton **Logger** class. Provides an event for log subscription.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/Log/Logger.cs

MediaServerEngine.MediaEngine Class Reference

Public Types

- enum **FileExtension** { **zip**, **xml**, **avs**, **avi**, **she** }
- *Possible file extensions.* enum **XMLTags** { **digitalSignageXmlInfo**, **resource**, **downloadPath**, **localPath**, **cinemaFolderName**, **initDate**, **endDate**, **lastEventModified**, **sheFileName**, **sheXmlFileNameNew**, **sheXmlFileNameOld**, **usedTemplate**, **finalVideoName**, **avsFile**, **videoFolder**, **cinema**, **ftpCrendentialsUser**, **ftpCrendentialsPass**, **ftpDestinationPath**, **groupName**, **copyAdvertisementVideos** }

XML tags (xml attributes) Public Member Functions

- void **InitProcess** ()
  Launch background workers and start process.
- void **DeleteFTP** ()
  Testing function.

Public Attributes

- const string **LOG_CONFIG_FILE** = "Config\\nlogconfig.xml"
  ***Log** configuration file.*
- const int **WEBSERVICE_TIMEOUT** = 15000
- const string **URL_SEPARATOR** = "/"

Properties

- bool **StopProcess** [get, set]
  Flag to stop process.
- int **ProcessCount** [get, set]
  Process counter.
- **MediaEngineConfig Configuration** [get]
  Read only configuration of Media Engine.
- static **Logger Log** [get]
  Logger.
- static string **InstallPath** [get, set]
  Service install path.
- static **MediaEngine Instance** [get, set]
  Instance of the engine.

Detailed Description

Member Enumeration Documentation

enum MediaServerEngine::MediaEngine::FileExtension

Possible file extensions.

enum MediaServerEngine::MediaEngine::XMLTags

XML tags (xml attributes)

Member Function Documentation

void MediaServerEngine.MediaEngine.DeleteFTP ()

Testing function.

void MediaServerEngine.MediaEngine.InitProcess ()

Launch background workers and start process.

Member Data Documentation

const string MediaServerEngine.MediaEngine.LOG_CONFIG_FILE = "Config\\nlogconfig.xml"

Log configuration file.

Property Documentation

MediaEngineConfig MediaServerEngine.MediaEngine.Configuration [get]

Read only configuration of Media Engine.

string MediaServerEngine.MediaEngine.InstallPath [static, get, set]

Service install path.

MediaEngine MediaServerEngine.MediaEngine.Instance [static, get, set]

Instance of the engine.

Logger MediaServerEngine.MediaEngine.Log [static, get]

Logger.

int MediaServerEngine.MediaEngine.ProcessCount [get, set]

Process counter.

bool MediaServerEngine.MediaEngine.StopProcess [get, set]

Flag to stop process.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/MediaEngine.cs

MediaServerEngine.MediaEngineConfig Class Reference

Class for media engine configuration.

Public Types

- enum **ConfigurationParamaters** { **CheckTaskWebservice**, **UpdateTaskWebservice**, **LogTaskWebservice**, **DownloadTaskUrl**, **DownloadsPath**, **DsPath**, **ProcessFrequency**, **FtpCrendentialsUser**, **FtpCrendentialsPass**, **FtpBasePath**, **VirtualDubPath**, **VirtualDubPath2**, **VirtualDubPath3**, **VirtualDubPath4**, **VirtualDubConfigPath**, **FtpUploadFrequency**, **DefaultTrailer**, **DefaultAdvertisement** }

Configuration parameters (Attributes of MediaEngineConfig). Properties

- string **CheckTaskWebservice** [get, set]
  Webservice URL to check pending tasks.
- string **UpdateTaskWebservice** [get, set]
  Webservice URL to update task info.
- string **LogTaskWebservice** [get, set]
  Webservice URL to log task.
- string **DownloadTaskUrl** [get, set]
  URL to download pending task zip file.
- string **DownloadsPath** [get, set]
  Downloads local folder path.
- string **DsPath** [get, set]
  Digital Signage local folder path.
- int **ProcessFrequency** [get, set]
  Process execution frequency (seconds).
- string **VirtualDubPath** [get, set]
  VirtualDub full path.
- string **VirtualDubPath2** [get, set]
  VirtualDub full path 2.
- string **VirtualDubPath3** [get, set]
  VirtualDub full path 3.
- string **VirtualDubPath4** [get, set]
  VirtualDub full path 4.
- string **VirtualDubConfigPath** [get, set]
  VirtualDub configuration file path.
- int **FtpUploadFrequency** [get, set]
  FTP Upload frequency (miliseconds)
- string **DefaultTrailer** [get, set]
  Path for default trailer video.
- string **DefaultAdvertisement** [get, set]
  Path for default advertisement video.

Detailed Description

Class for media engine configuration.

Member Enumeration Documentation

enum MediaServerEngine::MediaEngineConfig::ConfigurationParamaters

Configuration parameters (Attributes of **MediaEngineConfig**).

Property Documentation

string MediaServerEngine.MediaEngineConfig.CheckTaskWebservice [get, set]

Webservice URL to check pending tasks.

string MediaServerEngine.MediaEngineConfig.DefaultAdvertisement [get, set]

Path for default advertisement video.

string MediaServerEngine.MediaEngineConfig.DefaultTrailer [get, set]

Path for default trailer video.

string MediaServerEngine.MediaEngineConfig.DownloadsPath [get, set]

Downloads local folder path.

string MediaServerEngine.MediaEngineConfig.DownloadTaskUrl [get, set]

URL to download pending task zip file.

string MediaServerEngine.MediaEngineConfig.DsPath [get, set]

Digital Signage local folder path.

int MediaServerEngine.MediaEngineConfig.FtpUploadFrequency [get, set]

FTP Upload frequency (miliseconds)

string MediaServerEngine.MediaEngineConfig.LogTaskWebservice [get, set]

Webservice URL to log task.

int MediaServerEngine.MediaEngineConfig.ProcessFrequency [get, set]

Process execution frequency (seconds).

string MediaServerEngine.MediaEngineConfig.UpdateTaskWebservice [get, set]

Webservice URL to update task info.

string MediaServerEngine.MediaEngineConfig.VirtualDubConfigPath [get, set]

VirtualDub configuration file path.

string MediaServerEngine.MediaEngineConfig.VirtualDubPath [get, set]

VirtualDub full path.

string MediaServerEngine.MediaEngineConfig.VirtualDubPath2 [get, set]

VirtualDub full path 2.

string MediaServerEngine.MediaEngineConfig.VirtualDubPath3 [get, set]

VirtualDub full path 3.

string MediaServerEngine.MediaEngineConfig.VirtualDubPath4 [get, set]

VirtualDub full path 4.

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/MediaEngineConfig.cs

MediaServerEngine.Log.VersionDigitalLogger Class Reference

Class for Version Digital communication functions.

Public Types

- enum **TaskLogLevel** { **DEBUG** = 0, **INFO** = 1, **WARNING** = 2, **ERROR** = 3, **FATAL** = 4, **PROGRESS_INFO** = 5 }

Possible log levels (copy of Versión Digital enum). Static Public Member Functions

- static void **UpdateTaskState** (int taskId, **MediaServerEngine.DSTask.TaskState** status, string log)
  Update task state in Version Digital.
- static void **ReportLog** (int taskId, **TaskLogLevel** level, string message)
  Send task log to Version Digital.
- static void **ReportLog** (int taskId, **TaskLogLevel** level, string message, string optomaGroup)
  Send task log to Version Digital.

Detailed Description

Class for Version Digital communication functions.

Member Enumeration Documentation

enum MediaServerEngine::Log::VersionDigitalLogger::TaskLogLevel

Possible log levels (copy of Versión Digital enum).

Member Function Documentation

static void MediaServerEngine.Log.VersionDigitalLogger.ReportLog (int *taskId*, TaskLogLevel *level*, string *message*) [static]

Send task log to Version Digital.

Parameters:

taskId	
level	

message	

static void MediaServerEngine.Log.VersionDigitalLogger.ReportLog (int *taskId*, TaskLogLevel *level*, string *message*, string *optomaGroup*) [static]

Send task log to Version Digital.

Parameters:

taskId	
level	
message	

static void MediaServerEngine.Log.VersionDigitalLogger.UpdateTaskState (int *taskId*, MediaServerEngine.DSTask.TaskState *status*, string *log*) [static]

Update task state in Version Digital.

Parameters:

taskId	
status	
log	

The documentation for this class was generated from the following file:

- /MediaServer/MediaServerEngine/Log/VersionDigitalLog