Universidad de Valladolid

E.T.S.I.T

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación

Diseño y desarrollo de una aplicación de escritorio para visualización y edición de archivos EPG (Electronic Program Guide)

Autor:

D. Diego Velayos San Juan

Tutora:

Dra. Dña. Míriam Antón García

Valladolid, Junio de 2016

Universidad de Valladolid

Universidad de Valladolid

Trabajo Fin de Grado

Título:	Diseño y desarrollo de una aplicación de escritorio para visualización y edición de archivos EPG (Electronic Program Guide)
Autor:	D. Diego Velayos San Juan
Tutora:	Dra. Dña. Míriam Antón Rodríguez
Departamento:	Teoría de la Señal y Comunicaciones e Ingeniería Telemática

Tribunal

Presidenta:	Míriam Antón Rodríguez
Vocal:	David González Ortega
Secretario:	Mario Martínez Zarzuela
Suplente 1:	María Ángeles Pérez Juárez
Suplente 2:	Francisco Javier Díaz Pernas

Fecha:	Junio 2016
Calificación:	

Resumen

Tras el apagón analógico televisivo, e inmediata instauración de la TDT (Televisión Digital Terrestre) en nuestros hogares, la cantidad y calidad de los servicios relacionados con los contenidos audiovisuales de las plataformas de televisión no han parado de crecer y mejorar. Uno de estos primeros servicios es la guía electrónica de programación, en inglés EPG (*Electronic Program Guide*), que ofrece la posibilidad de conocer gran cantidad de datos de interés acerca de la emisión que estamos visionando en cada momento.

Dicha información está contenida en un documento de texto utilizando un lenguaje de etiquetado XML (eX*tensible Markup Language*). Este archivo contiene además otros datos de mucha importancia para las plataformas televisivas digitales, pues permiten mejorar la experiencia del usuario añadiendo nuevas funcionalidades.

El objetivo de este Trabajo Fin de Grado es simplificar a los miembros de la iniciativa GVP (*Global Video Platform*) de Telefónica I+D, dónde realicé las prácticas profesionales, el manejo de los ficheros EPG mediante una aplicación de escritorio que permita visualizar y editar de forma cómoda la información contenida en ellos.

Palabras clave: EPG, TDT, guía, programación, TV, televisión, emisión, digital, Telefónica, GVP.

Abstract

After the analog television blackout, and immediate introduction of the DTV (Digital Terrestrial Television) in our homes, the quantity and quality of services related to live TV programming has grown. One of these services is the Electronic Program Guide, known as EPG, offering to the users tons of interesting information about the current TV program they are watching.

This information is contained in a text file using XML language (eXtensible Markup Language). This file also contains other relevant information determinant for the final user experience when using digital television platforms. Allow to create new services, improving the quality of service offered to the user.

The objective of this Final Project is to simplify the way the members of the GVP initiative (Global Video Platform) of Telefónica I+D explore and edit EPG files, by creating an intuitive software that provides an smart way to view and edit comfortably the information contained in the EPG documents.

Keywords: EPG, TDT, guide, programming, TV, television, broadcasting, digital, Telefónica, GVP.

Agradecimientos

''A mis padres y hermanos:
Por el apoyo y la confianza
que siempre me han dado;
Sin ellos, nada de esto sería posible.

A mi tutora, Míriam:
Por su tiempo y dedicación;
Por sus buenos consejos.

A Telefónica I+D,
Y en especial a Jose Antonio:
Por brindarme la oportunidad
de realizar este TFG.

A mis compañeros de clase:
Por hacer de mí paso por la universidad
una experiencia inolvidable.

A todas aquellas personas
que me empujaron en algún momento;
A los que siguen a mi lado,
Y a los que no.

A todos vosotros:
MUCHÍSIMAS GRACIAS''

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

Primera Parte

CAPÍTULO 1: INTRODUCCIÓN

1.1 – CONTEXTO DEL TFG

Este Trabajo Fin de Grado nace en el contexto de unas prácticas curriculares en la empresa Telefónica I+D, en su sede en Boecillo (Valladolid), desde Agosto de 2015 hasta Abril de 2016.

Telefónica I+D es una empresa del Grupo Telefónica dedicada a la Innovación y el Desarrollo. Nace en el año 1988 con la misión principal de contribuir a la modernización y mejorar la competitividad del Grupo Telefónica, mediante la aplicación de nuevas ideas en el desarrollo de productos y servicios avanzados.

Fui asignado a la iniciativa de GVP (*Global Video Platform*) que se encarga, a grandes rasgos, del soporte y desarrollo de funcionalidades para la plataforma de video de Movistar en los diferentes países en donde tiene otorgadas licencias, principalmente en Brasil bajo el nombre de Vivo.

Ya durante las primeras semanas de mi estancia en la empresa, se me propuso la opción de dedicar mis horas de prácticas a la realización de un proyecto de mutuo acuerdo, que pudiera servir a la iniciativa de GVP y a su vez como Trabajo Fin de Grado. Analizando las necesidades de la iniciativa, tomé la decisión consensuada de realizar una herramienta software utilizando Java como lenguaje de programación, con el objetivo de visualizar y editar un tipo de archivo frecuentemente usado en GVP. Más adelante hablaremos con detalle de este archivo.

Este documento recopila, de forma ordenada, toda la información referente a la realización de la herramienta software, bautizada con el nombre de EPG Viewer 1.0, desarrollada para Telefónica I+D en el contexto anteriormente descrito. Algunos datos sensibles pueden haber sido omitidos de forma deliberada.

1.2 – OBJETIVOS

La principal motivación de este proyecto gira en torno a la idea de “hacer algo útil para alguien”. Considero que la realización del TFG es una buena oportunidad de demostrar las destrezas adquiridas durante los años anteriores. La mejor forma de demostrarlo es desarrollando un proyecto viable que no sólo pueda aplicarse, sino que se aplique y se utilice.

En el caso particular de este TFG, y dentro del contexto de Telefónica I+D, se pretende cambiar la forma de ver y editar los archivos EPG (*Electronic Program Guide*). El equipo de QA (*Quality Assurance*) utiliza este tipo de archivos para testear las funcionalidades, mejoras y actualizaciones de la plataforma de vídeo. El objetivo de esta tarea es asegurar la calidad del producto final.

La edición de ficheros EPG es en ocasiones rutinaria y un tanto tediosa, puesto que se trata de archivos de texto plano que pueden contener hasta cientos de miles de líneas. Mediante el uso de la herramienta software que propone este documento, se puede conseguir editar este tipo de archivo de forma más cómoda y sencilla, con un considerable ahorro de tiempo. Esto permite aumentar el número de pruebas y test realizados, asegurando así una mayor calidad del producto final.

1.3 – Fases y Métodos

Para la alcanzar el objetivo descrito en el apartado anterior, partimos de una fase cero de toma de contacto con la iniciativa. En GVP se realizan reuniones diarias, dónde cada uno individualmente pone al corriente al resto de sus últimos avances. Esto permite discutir y consensuar los objetivos transversales y conseguir trabajar todos en la misma dirección.

Fase 1

En una primera fase de aproximación, de unas cuatro semanas de duración, aproveché estas reuniones para sondear las necesidades acerca de la herramienta software, informar de mis avances, y escuchar las peticiones y los consejos de los demás miembros del grupo. Estas aportaciones asentaron las bases del proyecto y marcaron las pautas de trabajo en el mismo.

Durante este periodo, y dado que mis conocimientos sobre programación en Java eran muy esenciales, dedique muchas horas a leer manuales, ver vídeo tutoriales y recopilar información acerca de la programación orientada a objetos. Realicé varias pruebas para acostumbrarme al uso de Java, su sintaxis y su metodología.

También dedique algunas horas al estudio en profundidad del tipo de archivo EPG para conocer su función, aplicación, uso y codificación. Toda la documentación acerca de estos archivos fue proporcionada por Telefónica I+D. Más adelante en el documento nos centraremos en explicar la estructura de estos ficheros.

Fase 2

Durante esta fase, centré mis esfuerzos en capturar el archivo EPG de texto en un objeto Java para poder manejarlo con comodidad. Esta tarea se puede llevar a cabo de diferentes formas, pero dependiendo del tipo de archivo, tamaño y uso, algunas formas resultan más pertinentes que otras. En el caso particular, y dada mi inexperiencia, realice varias pruebas utilizando diferentes métodos, para optar por el que mejor comportamiento tenía.

Fase 3

Una vez que el archivo EPG fue capturado con éxito en un objeto Java, la siguiente fase consistió en el desarrollo de las funcionalidades básicas para manejo y edición. Inicialmente, centrando la atención en la visualización e interfaz gráfica, y más tarde, en la edición y en algunas funcionalidades extra.

En esta fase tuvieron un peso importante las aportaciones y necesidades de los miembros de la iniciativa. Al inicio del proyecto fueron descritas unas necesidades básicas que el software debía incorporar, y durante el desarrollo del mismo fueron surgiendo otras funciones añadidas que mejoran la experiencia del usuario y dan calidad al producto final.

Fase 4

La última fase del proyecto se centró en documentar todo lo relativo al software EPG Viewer: manual de usuario, diagramas de clases, diagramas de flujo, funcionamiento interno, etc. Conviene recordar la importancia de generan una documentación de buena calidad, ya que de ello depende el futuro de la aplicación. Una buena infografía hace más sencillo comprender el funcionamiento interno simplificando el trabajo de futuros desarrolladores.

1.4 – Estructura del documento

El presente documento se organiza en tres partes bien diferenciadas. Cada una de ellas tiene un objetivo bien distinto y en conjunto pretenden guiar a lector desde lo más general, dando una idea global del proyecto, hasta lo más concreto y detallado acerca del tema del presente TFG.

La Primera Parte de este documento contiene los tres primeros capítulos. El presente capítulo (Capítulo 1) sirve de introducción al Trabajo Fin de Grado. Describe los objetivos, fases y métodos seguidos para el desarrollo del proyecto que se expone.

En el Capítulo 2 se realiza un recorrido por la historia de la televisión. Desde las primeras emisiones analógicas, hasta las últimas y más modernas emisiones digitales mediante enlaces satélites o de fibra óptica. Haremos hincapié en los servicios añadidos a la emisión, que mejoran la experiencia del telespectador. Conoceremos a fondo los inicios del Teletexto, su principio de funcionamiento, y la herencia que nos ha dejado en los servicios añadidos de la televisión digital de hoy en día.

El Capítulo 3 contiene los detalles acerca de las tecnologías usadas para el desarrollo del proyecto y cierra la Primera Parte de este documento.

A continuación, en la Segunda Parte, se muestran los detalles sobre el desarrollo del proyecto. Se incluye una descripción detallada sobre el estándar GLF para transmisión de la guía de programación, el entorno JAXB de Java utilizado para el tratamiento de ficheros XML, y un manual de usuario del software EPG Viewer que se propone como Trabajo Fin de Grado. Esta información está contenida en dos capítulos diferentes (Capitulo 4 y Capítulo 5).

A lo largo de la Tercera Parte de este documento se expondrán los datos relativos al presupuesto (Capítulo 6), unas conclusiones finales y posibles líneas futuras para continuar con el desarrollo del software (Capítulo 7). El Capítulo 8 contiene las referencias APA utilizadas durante el desarrollo de esta memoria de proyecto, y un glosario de términos utilizados a lo largo del texto. Con esto finaliza la Tercera Parte del presente Trabajo Fin de Grado.

Por último se incluyen dos anexos. El primero de ellos contiene una copia del documento de la patente de Microsoft del estándar GLF. El segundo documento contiene el contenido del documento XSD perteneciente al esquema GLF.

CAPÍTULO 2: LA GUÍA ELECTRÓNICA DE PROGRAMACIÓN

2.1 – INTRODUCCIÓN

Los avances tecnológicos en el campo de las comunicaciones han marcado y condicionado indudablemente las pautas del comportamiento humano. Esto no debe sorprendernos, pues cada avance en este campo responde a la necesidad humana de estar cada vez más y mejor conectado con el resto de humanos y, evidentemente, la repercusión de estos avances es latente en nuestra forma de comportarnos. Sólo tenemos que echar la vista atrás y pensar en cualquier revolución tecnológica asociada a la comunicación, para darnos cuenta del impacto que supone en nuestra sociedad y en nuestros hábitos diarios.

Pensemos, por ejemplo, en la radio. Este ingenio tecnológico de principios del siglo XX, es capaz de reproducir voces y sonidos en tiempo real que se producen a cientos de kilómetros de distancia. Irrumpe en la sociedad de la época modificando sus costumbres. La gente quedaba para reunirse alrededor del receptor de radio y escuchar juntos el boletín de noticias, música, o programas de entretenimiento radiofónicos. Todo un avance para la época.

Hoy en día la cantidad de medios de comunicación disponibles a nuestro alcance para entretenernos, informarnos o comunicarnos son casi innumerables. Televisión, prensa, radio, Internet y otras fuentes derivadas de esta última, por nombrar las principales. Todas ellas tienen su nexo común en Internet, que día a día va ganando terreno e importancia debido a que puede aglutinar a las otras en un mismo lugar. Como punto de partida de este documento, vamos a centrar la atención en uno de los principales medios de comunicación: LA TELEVISIÓN.

2.2 – LA TELEVISIÓN: DE ANALÓGICA A DIGITAL

Primeramente, vamos a diferenciar la televisión como medio de comunicación del aparato receptor propiamente dicho. En este documento, y salvo que se diga lo contrario, cuando hablamos de televisión, hacemos referencia al medio de comunicación que emplea mecanismos de difusión, para transmitir contenidos audiovisuales desde una estación emisora hasta los receptores de televisión finales. Esta transmisión se realiza mediante la propagación de ondas de origen electromagnético, utilizando diferentes medios físicos, como pueden ser cables de cobre coaxiales, fibras ópticas o el propio espacio libre, ya sea mediante un enlace satélite o un enlace terrestre.

La televisión (como aparato receptor), nace de la mano de una serie de inventos previos. Por un lado, Paul Gottlieb Nipkow patenta en 1884 un explorador de imagen que estaba formado por un disco metálico perforado. Estos agujeros cuadrangulares estaban dispuestos en espiral. El ingenio fue bautizado bajo el nombre de “disco de Nipkow”.

Años más tarde, en 1897, Carl Ferdinand Braun, un científico alemán, inventó el tubo de rayos catódicos (en inglés CRT, *Cathode Ray Tube*). Este dispositivo era una evolución del tubo de Crookes que incorporaba, entre otras modificaciones, una capa de fósforo sobre el frontal, que permite reproducir la imagen proveniente el haz de electrones. También dispone de un sistema de deplexión, que permite desviar el haz a los largo de la capa de fósforo frontal. Este sistema se basa en las propiedades magnéticas de dicho haz al atravesar el tubo.

Estos dos avances tecnológicos, sumados a otros tantos en el campo de la física, la química y el electromagnetismo, asentaron las bases para los primeros aparatos de televisión de la historia. Los primeros pasos apuntaban en dos direcciones bien diferentes: la televisión mecánica frente a la televisión electrónica. La primera basada en el invento de Nipkow, la segunda en el de Braun.

En la Ilustración 1 podemos ver el principio de funcionamiento de la primera televisión mecánica, fabricada por John Logie Baird a través de la *Baird Television Development Company*, utilizando el disco de Nipkow.

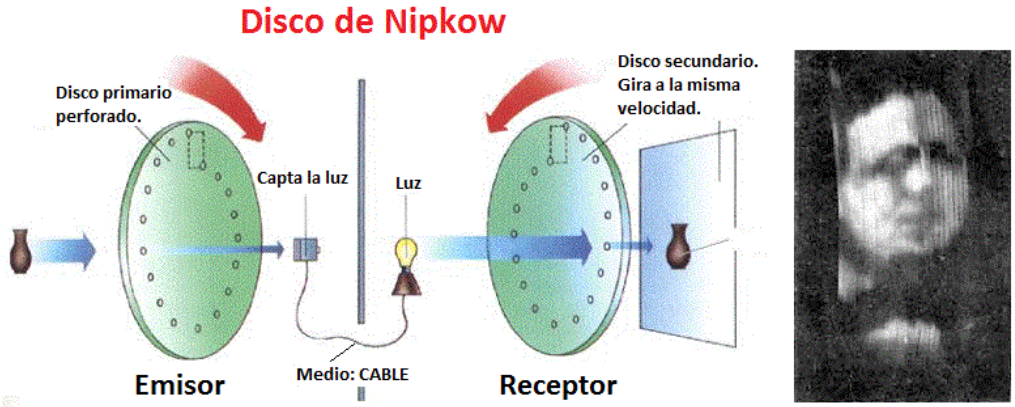


Ilustración 1: Esquema del disco de Nipkow. Imagen real.

Por otro lado, la Ilustración 2 muestra el diagrama de un televisor basado en CRT junto a una imagen real de una emisión de la BBC.

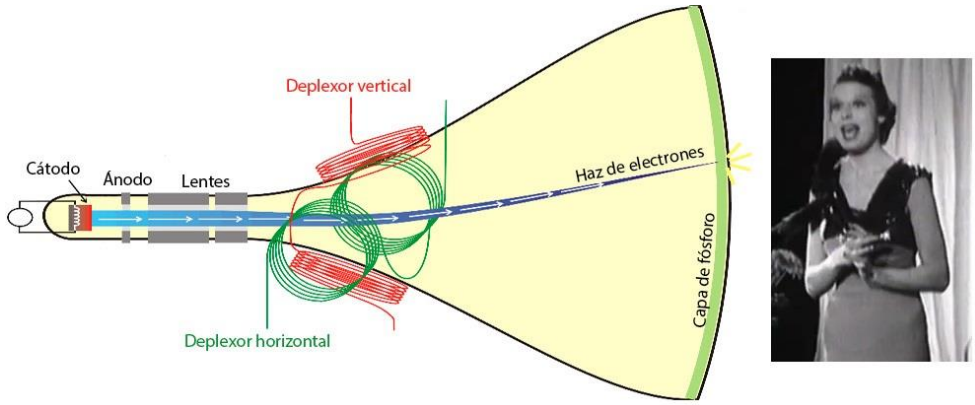


Ilustración 2: Esquema de un CRT. Imagen real

Finalmente, y dado que la calidad de imagen, entre otras cosas, era muy superior, la televisión basada en CRT marcó las pautas futuras de este medio de comunicación. Tal fue así, que el propio Baird, precursor de la televisión mecánica, acabo pasándose a la tendencia de televisión puramente electrónica. En 1944 comenzó a trabajar en el primer televisor a color 100% electrónico denominado *Telechrome*.

El resto de la historia es bien conocido por todos. Casi nadie ha oído hablar de televisiones mecánicas. La era televisiva había comenzado muy despacio y con muchas dificultades, pero apuntaba tener una gran papel en la sociedad del futuro.

En lo que respecta a la televisión como medio de transmisión, ha sufrido una gran evolución a lo largo del tiempo. Desde las primitivas emisiones analógicas, con escaso alcance y baja calidad, a las más actuales transmisiones digitales, de alta calidad, difusión mundial y con gran cantidad de servicios añadidos.

A lo largo de este apartado vamos a dar unas pequeñas pinceladas sobre los orígenes de la televisión, desde las primeras emisiones analógicas hasta su digitalización. Hablaremos de la televisión digital y de los servicios añadidos que ofrece, centrando la atención en uno de ellos: la guía electrónica de programación.

2.2.1 – Televisión analógica

Las primeras emisiones televisivas tuvieron lugar en los años 30. Utilizaban como medio de propagación el espacio libre mediante enlaces terrestres. Una estación base genera y transmite las ondas de radio mediante una antena. Estas se propagan por el espacio libre y son recibidas por antenas receptoras para luego ser demoduladas en el aparato receptor de televisión. Estas transmisiones eran en sus inicios puramente analógicas. Tanto la captación de la imagen, como el tratamiento previo a la transmisión, así como la propia transmisión, su recepción y reproducción, se realizaban de forma analógica.

La cámara de vídeo captura la luz reflejada por la escena a través del conjunto de lentes que forman la óptica, permitiendo enfocar la imagen en el sensor (transductor optoelectrónico). Se realiza un barrido de la imagen de forma secuencial, tomando medidas sobre la intensidad de la luz que llega a cada punto del cuadro.

Este barrido se conoce como barrido entrelazado, pues primero se escanean las líneas impares y después las líneas pares de la imagen. De esta manera se forma cada muestra, denominada cuadro (del término en inglés *frame)*. Cada cuadro se compone de dos campos (en inglés *field*). Un campo contiene las líneas pares y el otro las impares. Estos dos campos se entrelazan en el receptor formando de nuevo el cuadro. El número de líneas depende del estándar. Por poner un ejemplo, el conocido como PAL, establece en 625 el número de líneas.

Esta operación de barrido debe repetirse con una frecuencia tal que el espectador no aprecie ningún parpadeo. La frecuencia mínima para evitar este efecto se sitúa entorno a los 10 Hertzios (veces por segundo) y es fruto de un efecto llamado persistencia retiniana, por el cual las imágenes que vemos permanecen impresas en nuestra retina durante un periodo de tiempo de alrededor de 0.1 segundos. Normalmente, se utiliza la frecuencia de 24 o 30 Hertzios (dependiendo del país y del estándar usado) para una mayor fluidez de la imagen.

Esto genera una señal eléctrica que es proporcional a la intensidad de luz en cada punto de la imagen. Cabe destacar que esta señal se correspondería a una imagen monocromática (blanco y negro). Para conseguir una imagen a color, se debe filtrar la imagen mediante un prisma. Esto permite descomponer la luz en sus componentes. Tras realizar el barrido entrelazado de cada componente, se mezclan las señales de luminancia y crominancia. Dado que no es objeto de este documento, no vamos a entrar en detalle acerca de cómo se mezclan estas señales.

Estas señales analógicas son moduladas y transmitidas mediante ondas electromagnéticas, que son captadas por las antenas de los aparatos receptores de televisión. Se produce la demodulación y se muestran en la pantalla (Pérez & Zamanillo, 2003).

Como podemos ver, todo el proceso se produce de forma analógica, de modo que cualquier interferencia o ruido, afecta directamente a la imagen que recibimos. Por no decir que los servicios añadidos de los que el espectador podía disponer eran muy reducidos.

Cabe destacar el primer servicio de Teletexto de la BBC en el año 1972, que llego a España en el año 1988 de la mano de TVE. Este sistema utilizaba las denominadas "líneas de recuperación" (en inglés VBI, Vertical Blanking Interval) para transmitir las páginas del teletexto.

Para entenderlo mejor, la señal que se recibe se divide, como hemos descrito antes, en cuadros y campos. Cada cuadro es el equivalente en cine a un fotograma. Dependiendo del país en que nos encontremos, se recibe un número diferente de cuadros por segundo. En España el estándar marca 24 cuadros por segundo. Cada uno se compone de dos campos con las líneas pares por un lado y las impares por el otro. Las líneas correspondientes a los bordes de la pantalla no se muestran en los receptores de televisión, por lo que se utilizan para enviar la señal de sincronismo vertical, sincronismo horizontal, salva de color y líneas de recuperación.

El tiempo que emplean estas últimas líneas de recuperación para ser recibidas, se utiliza para "esperar" que los circuitos de deplexión vertical se estabilicen. Esto es debido al funcionamiento de los televisores basados en tubo de rayos catódicos, hoy ya en desuso, en los que un haz de electrones es desviado al atravesar un campo magnético generado por unas bobinas de cobre. La señal que marca la posición del haz se generaba con un circuito de válvulas, que son los antecedentes de los actuales transistores. Este haz va recorriendo la pantalla dibujando el recorrido de la Ilustración 3, de forma que el punto más crítico es el momento en que pasa de la última línea a la primera. Por ello, necesita un tiempo de guarda para estabilizar el haz en el principio de la primera línea.

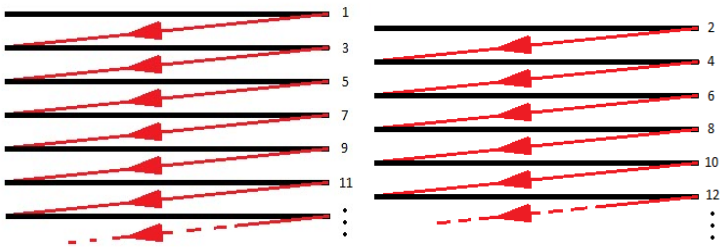


Ilustración 3: Barrido de líneas; Campo superior vs Campo inferior

Son estas últimas líneas de recuperación las que se aprovechan para enviar el servicio de teletexto. Los televisores preparados para recibir y mostrar el teletexto toman los datos de estas líneas, mientras que los televisores que no están preparados, simplemente las ignoran, pues no contienen información relevante sobre la imagen.

Cabe destacar, que el Teletexto es un servicio de naturaleza digital enmarcado en el ámbito de una emisión analógica. Las líneas VBI o líneas de recuperación, anteriormente descritas, contienen 45 bytes de información digital, que llevaron a los hogares de todo el mundo información actualizada a la carta, mucho antes de que Internet irrumpiera en nuestras vidas. Aún hoy, el teletexto sigue presente en nuestras vidas. Casi todos los canales digitales de hoy en día mantienen el Teletexto, aunque se le augura un final no muy lejano.

Hoy en día, la televisión analógica ha sido reemplazada casi por completo por la digital. Ha quedado desplazada para casos específicos, como puede ser la difusión de televisión en

complejos hoteleros, comunidades de vecinos, o incluso, dentro de nuestra vivienda. Aun así, la mayoría de receptores de televisión de hoy en día disponen de sintonizador de televisión analógica, aunque la tendencia es a desaparecer.

2.2.2 – Televisión digital

La digitalización de la televisión comenzó en los años 80, afectando inicialmente sólo a la producción, para luego extenderse poco a poco hacia la transmisión y recepción. Estas dos últimas fases han sido más difíciles de llevar a cabo, debido a que implicaban, no sólo una modernización de los medios transmisores, sino también de los aparatos receptores de televisión. Además responde a una demanda importante: una mejor calidad de imagen y mayor cantidad y calidad de servicios.

Mucho han evolucionado los televisores desde su invención. Las pantallas de apenas unas pocas pulgadas han ido dejándose atrás y, en su lugar, los televisores han ido ganando en tamaño y prestaciones. Además, dado que el espectro radioeléctrico es un recurso natural finito, el número de canales que pueden emitirse también lo es. Cada país soberano se hace cargo de la buena gestión de este recurso, que no solo ofrece servicios televisivos, sino que se comparte con otros como la radio y la telefonía, entre otros.

El objetivo es hacer un uso óptimo del espectro radioeléctrico, ofreciendo el mayor número de servicios, con la mejor calidad posible. Para conseguirlo, se promueve la migración de transmisiones analógicas a técnicas digitales que permiten un mayor número de servicios con una sustancial mejora de la calidad.

En el caso de la televisión, esto se traduce en:

- Un mayor número de canales disponibles.
- Una mejora de calidad de la imagen que permite incluso canales de alta definición.
- Nuevos servicios: varios canales de audio (multi-idioma), canales de radio, canales interactivos, **guía electrónica de programación** y un largo etcétera.

Así fue cómo, allá en el año 2000 y en los años sucesivos, comenzaron poco a poco y de forma muy dispersa por el mundo las primeras emisiones digitales terrestres de televisión. En España llegaron con poco éxito en el año 2000, de la mano de la plataforma de pago Quiero TV. Debido a la escasa audiencia y baja rentabilidad, tan sólo dos años después echó en cierre. No fue hasta el año 2005 cuando se retomó de nuevo el plan de digitalización, para adaptarse a las necesidades de la sociedad y a la tendencia en otros países. En Abril de 2010 la televisión analógica echó el cierre definitivo, dejando un ancho de banda disponible para nuevos canales y servicios de la TDT (Televisión Digital Terrestre).

Por otro lado, hay que señalar que las primeras emisiones digitales llegaron con anterioridad de la mano de la televisión por satélite y por cable, antes de que la TDT empezara a sonar con fuerza. Estas dos tecnologías quedaron casi monopolizadas por plataformas de televisión de pago, que sufragaron los gastos extra de la infraestructura.

En el caso de la televisión por satélite, como es obvio, se requiere de un satélite (o mejor dicho, una constelación de ellos) que haga de repetidor de la señal televisiva, la cual se manda desde una estación en tierra firme. Los receptores tienen que estar dentro del cono de cobertura de la

señal que devuelve el satélite. Además, se precisa de una antena parabólica orientada al satélite en cuestión con cierta precisión y de un decodificador externo a la televisión.

Dado que es más difícil acotar la cobertura de un satélite, resulta complejo que un gobierno de un país en concreto, promueva medidas en favor de esta tecnología sin llegar a acuerdos económicos con otros países fronterizos. Más sencillo es, por otro lado, que plataformas privadas de televisión de diferentes países se pongan de acuerdo para hacer llegar al espacio la costosa infraestructura espacial, necesaria para poner en marcha sus negocios televisivos. Esto justifica que la televisión satélite hoy en día, siga siendo territorio inexplorado para muchos de los espectadores de televisión, a pesar de disponer de canales FTA (Free To Air), que no suponen suscripción alguna a plataformas de televisión de pago.

Respecto a la difusión de televisión por cable, ocurre algo similar. Está copada por operadores privados que proveen de contenidos audiovisuales, de Internet y telefonía fija a clientes de pago. También es significativo señalar, que algunos de los proveedores de servicios de televisión digital (y otros servicios digitales) por cable en España, tienen muy localizado geográficamente su mercado.

Es difícil predecir las pautas para el futuro de la televisión. Lo que está claro es que las redes de fibra óptica están cambiando la forma de hacer llegar la televisión a los hogares. Este medio de comunicación ofrece una gran capacidad y ya se están ofertando servicios de televisión, Internet y teléfono a través de fibra óptica.

Los indicios apuntan a una unificación de varios medios (televisión e Internet) para ofrecer servicios interactivos a la carta, contenidos audiovisuales bajo demanda, retransmisiones en directo y cualquier cosa que se nos pueda pasar por la cabeza. Internet se caracteriza por tener una enorme versatilidad que ahora ha llegado a la televisión de la mano de las Smart TV, juntando todos los recursos disponibles en la red, con los contenidos audiovisuales provenientes de diferentes fuentes (satélite, cable, fibra óptica, etc). Poco queda ya de aquellas primeras emisiones únicas e irrepetibles que cambiaron el ocio de nuestros hogares para siempre.

2.3 – Primeros pasos: el Teletexto

El Teletexto es el primer servicio adherido a la señal de televisión convencional. Este sistema de naturaleza digital, nace a comienzos de los años 70, cuando aún las transmisiones televisivas se hacían de forma analógica. Permite hacer llegar información de diversa índole (típicamente información sobre el tiempo, programación, subtítulos, noticias, etc) a televisores equipados con decodificador VBI (Vertical Blanking Interval).

Este sistema fue inicialmente propuesto por John Adams, desde los laboratorios de Philips en California, y posteriormente lanzado en 1973 por la BBC en Inglaterra bajo el nombre de Ceefax. En España las transmisiones regulares de Teletexto llegaron en Mayo del 1988 de la mano de TVE. La principal funcionalidad era prestar un servicio de subtítulos a las personas con problemas de audición. Además, provee de noticias, programación y otros datos de interés que pueden ser consultados en televisores compatibles con este sistema.

Este sistema utiliza las denominadas líneas de recuperación (son líneas vacías de información hasta entonces) para transmitir los datos utilizando una codificación digital. Como suele ocurrir

en estos casos, existen varios sistemas de codificación que cada país adopto para sus transmisiones, pero todos ellos tienen pequeñas variaciones en cuanto a tasas binarias, número de caracteres por línea o número de líneas.

El Teletexto envía todas sus páginas, codificadas con número de tres cifras, de forma periódica. Algunas páginas (las más usadas), se envían varias veces durante un ciclo para poder acceder a ellas. El usuario selecciona con el mando el número de página que desea ver. El televisor pasa a modo de espera hasta que recibe la página solicitada. Algunos televisores disponen de un buffer que guarda un número de páginas para reducir los tiempos de espera. Cuando la página se recibe, se muestra al usuario. Algunas páginas van variando en cada ciclo. Esto permite enviar varios niveles de página. La Ilustración 4 muestra el servicio Ceefax de la BBC, frente al Teletexto de TVE en España:

Ilustración 4: Ceefax, de la BBC vs Teletexto, de TVE

No hay interacción alguna por parte del usuario. Es un sistema denominado simplex. La información solo fluye en un sentido y nunca en el otro. Todos los datos se envían al usuario de forma periódica, y es este quien decide que ver en cada momento. El mando permite navegar de una página a otra.

Lo más destacable de este sistema, en lo que respecta al presente documento, es el primer servicio de **guía de programación**, enmarcado en el contexto del Teletexto. El usuario puede consultar la parrilla de programación en cualquier momento mediante este sistema, lo cual era una gran novedad para la época.

2.4 – Guía Electrónica de Programación (EPG)

La Guía Electrónica de Programación, o en inglés, Electronic Program Guide, representa la evolución a la televisión digital del servicio de programación que ofrecía el Teletexto. Esta nueva prestación viene de la mano de la televisión digital y permite, además de ofrecer información ampliada sobre las emisiones en curso, programar la grabación (o configurar un aviso) de emisiones futuras que respondan a un determinado criterio.

La información de la EPG viene encapsulada dentro del llamado flujo de transporte bajo el estándar Europeo DVB (Digital Video Broadcasting). Estos datos son interpretados y se muestran al usuario. Este puede navegar entre la programación de los distintos canales, en diferentes franjas horarias.

La Ilustración 5 muestra un ejemplo de EPG en el modo “info”. Podemos ver que la información que se muestra corresponde con la emisión que el usuario está visualizando en ese momento. Algunos sintonizadores muestran esa información resumida, durante algunos segundos, cada vez que cambiamos de canal.

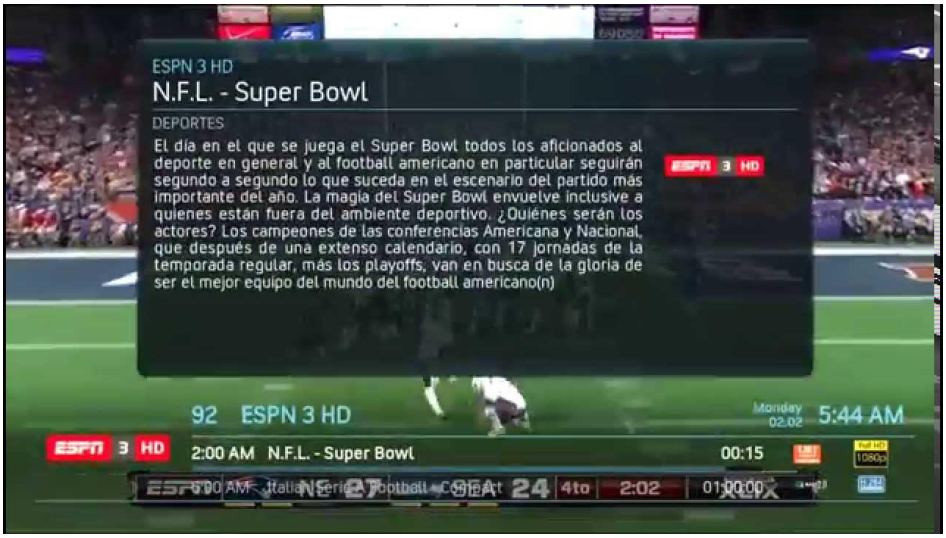


Ilustración 5: Ejemplo 1 de EPG

La Ilustración 6 muestra otro ejemplo de EPG en modo “guía”. En este caso se visualiza simultáneamente la programación de varios canales. Utilizando el mando a distancia, podemos navegar dentro de esta línea temporal. Esto nos permite ver las emisiones de cada canal en cada momento, pudiendo acceder a sus detalles, creando alertas o programando una grabación (si el decodificador lo permite).

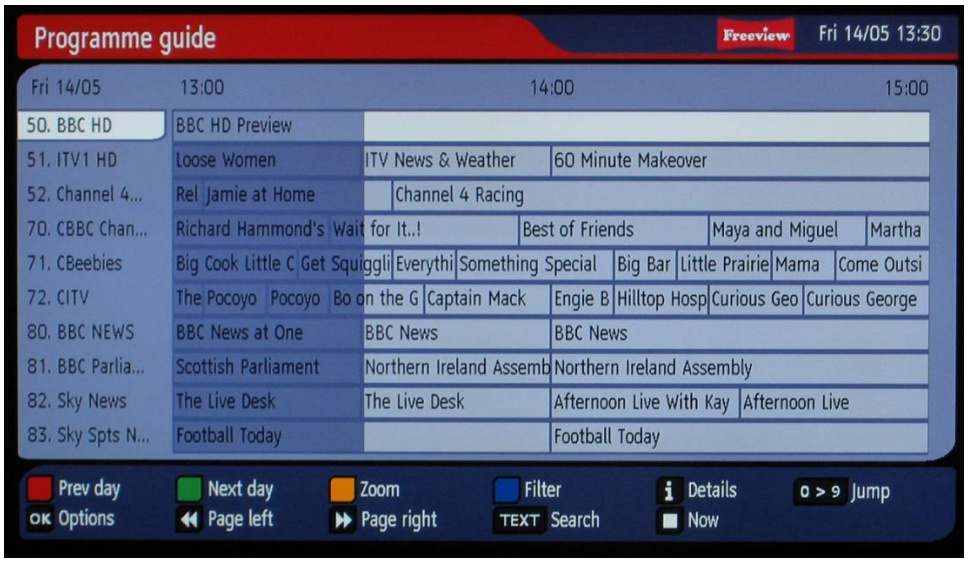


Ilustración 6: Ejemplo 2 de EPG

Por otro lado, en el contexto de las plataformas de televisión de pago, junto con la información de cada emisión, existen otros datos que, aunque no tienen significado directo para el usuario, permiten a estos acceder a servicios extra.

Estos datos no son estáticos. Pueden irse añadiendo etiquetas asociados a las emisiones según la necesidad. Más adelante en el documento veremos la estructura de un archivo de EPG y cómo este permite, mediante el uso de '*flags*' y '*values*', añadir nuevos datos que permitan identificar el tipo de codificación del audio, la calidad de la imagen, emisiones con restricciones geográficas, emisiones con coste extra, etc. Esto deja las puertas abiertas para necesidades que puedan ir surgiendo posteriormente.

Por poner un ejemplo, en este contexto se dispone de las categorías. Cada emisión tiene asociado un programa. Este contiene todos los datos de interés, como el año de producción, titulo original, intérpretes, director, productora, etc. Pero también tiene asignada una categoría dentro de una lista prediseñada. Estas categorías permiten conocer a las plataformas de pago las preferencias de sus clientes. De esta manera, se puede crear un perfil de cliente en base a los programas que visualiza. Estos datos pueden ser usados para recomendar nuevos contenidos, hacer estadísticas de uso, o adaptar la oferta a la demanda de los clientes.

Otro claro ejemplo son las imágenes. Los archivos de EPG sólo pueden transportar texto plano. Esto genera EPGs planas y simples, cuyo estilismo cambia en función de la marca y modelo de televisión. Pero la llegada de Internet de la mano de las Smart TV, hace posible EPGs mucho más vistosas e intuitivas. Sólo necesitamos incorporar nuevas etiquetas con la URL del logo de cada canal e imágenes del programa en cuestión. La Smart TV se encarga de leer estas etiquetas, acceder a la URL, descargar la imagen y almacenarla en memoria para mostrar dónde proceda y crear una EPG mucho más rica. Además, algunos televisores acceden a bases de datos como IMDb para extraer datos adicionales actualizados sobre películas o series.

Las siguientes imágenes muestran algunos ejemplos de EPGs en modo "guía". Podemos ver los diferentes estilos y la inclusión de imágenes para darle un aspecto visual más agradable.

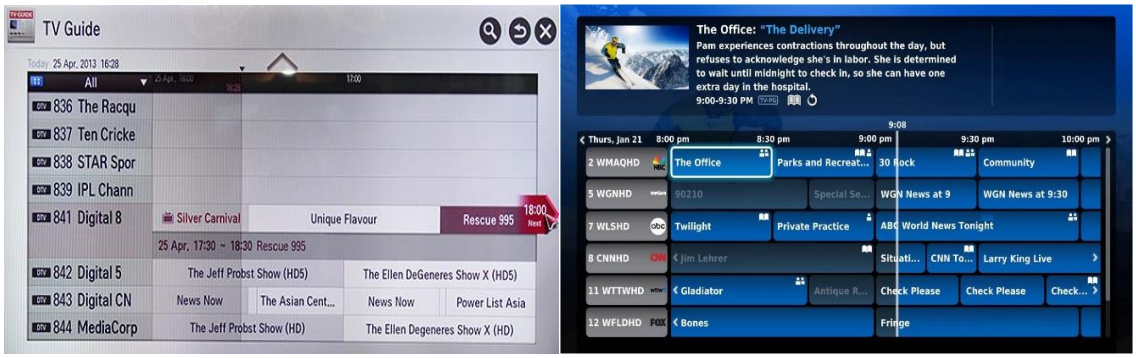


Ilustración 7: Ejemplo 3 EPG

Estos, y otros posibles ejemplos, ponen de manifiesto la constante evolución a la que está sometida este servicio. Diseñado para albergar una información básica que puede ser ampliada adaptándose a los nuevos tiempos. En Telefónica I+D trabajan a diario con este servicio, diseñando y mejorando funciones nuevas que hagan más interactiva y atractiva la oferta audiovisual.

2.5 – EPG VIEWER 1.0

EPG Viewer 1.0 es la primera versión de un software, basado en código Java, que se presenta en este TFG para facilitar el trabajo de ver y editar archivos EPG a los desarrolladores de Telefónica I+D. Está diseñado para perdurar en el tiempo, ya que contempla la posibilidad de añadir nuevos elementos a las listas de '*flags*' y '*values*', de forma que pueda adaptarse a necesidades futuras.

Además ofrece otras posibilidades, como por ejemplo, el diagnóstico de una EPG para encontrar defectos o información detallada sobre el número de elementos de cada lista. A estas funciones se le pueden sumar otras en el futuro. Utilizando la documentación aportada a la empresa, cualquier desarrollador puede tomar el código fuente como punto de partida, y añadir cualquier requerimiento que pueda surgir en el futuro.

EPG Viewer se encuentra accesible dentro de la Intranet de la empresa, y algunos desarrolladores empiezan a incorporarla dentro de sus herramientas software de uso diario.

En los próximos capítulos comenzaremos a adentrarnos poco a poco en el desarrollo de esta aplicación software, intentando aclarar todos los aspectos técnicos que hacen posible su funcionamiento. Describiremos y detallaremos todas sus características, incluyendo algunos diagramas que expliquen el funcionamiento de las partes más críticas del EPG Viewer.

Capítulo 3: Tecnologías para el desarrollo

3.1 – Introducción

Los primeros pasos cuando nos enfrentamos a la programación de un software, sea del tipo que sea, giran en torno a los aspectos técnicos básicos de este. Debemos lanzar, casi con estilo periodístico, unas cuantas preguntas al aire:

- ¿Qué debe hacer?
- ¿Qué requisitos debe cumplir?
- ¿Quién se va a usar?
- ¿Dónde se va a usar?
- ¿Cómo se va a usar?

Las respuestas a estas preguntas de forma independiente pueden, en ocasiones, diferir. El camino para fijar las tecnologías que se van a utilizar para el desarrollo de una aplicación software, pasa por evaluar cada una de las posibilidades y buscar el nexo en común a cada una de las respuestas. De esta forma conseguimos llegar a una solución de compromiso que ofrezca el mejor resultado posible al problema planteado.

En los siguientes apartados responderemos a estas y otras preguntas, justificando la respuesta, para obtener los requisitos necesarios y dar la mejor respuesta software que cumpla todos los requisitos.

3.2 – Lenguajes de programación

Una de las primeras cuestiones técnicas a la que hay que dar respuesta es el lenguaje de programación que se va a utilizar. Para ello se deben evaluar las necesidades de nuestra aplicación. Cuestiones como rendimiento, portabilidad, compatibilidad, librerías disponibles, soporte, futuro de la aplicación son clave para la toma de decisión.

Con EPG Viewer podemos evaluar los requerimientos según se indica en Ilustración 8.

	Dependencia				
Aspecto	Baja	Media-Baja	Media	Media-Alta	Alta
Rendimiento			X		
Compatibilidad con otros lenguajes				X	
Portabilidad					X
Compatibilidad con diferente SO					X
Disponibilidad de librerías				X	
Continuidad del lenguaje en el futuro			X		
Popularidad del lenguaje				X	

Ilustración 8: Tabla de requerimientos

En base a estas respuestas, las primeras ideas giraron en torno a dos tecnologías de desarrollo. La primera de ellas contemplaba la posibilidad de una aplicación web, utilizando un lenguaje de programación de propósito específico como es Javascript. La otra idea propone una aplicación de escritorio utilizando un lenguaje de propósito general como pueden ser Java o C++. Vamos a analizar cada una de las opciones por separado.

3.2.1 – Aplicación web basada en Javascript

JavaScript es un lenguaje de programación interpretado que se utiliza normalmente del lado del cliente permitiendo hacer páginas web dinámicas. Esto implica que el código no es compilado desde su descripción, sino que el intérprete, en este caso el navegador web, realiza la traducción a código máquina a medida que se ejecuta (normalmente instrucción a instrucción).

Dispone de un amplio número de *frameworks* y librerías de distinto propósito que hacen posible crear potentes aplicaciones web con diferentes finalidades. Tiene una sintaxis similar a C, aunque también hereda algunas convecciones de Java (Code school, 2016).

Uno de los frameworks más populares es AngularJS. Se mantiene gracias a Google y su comunidad, ya que es de código abierto. Dispone por tanto de un gran número de manuales disponibles en Internet, aunque no siempre están actualizados ya que está en continuo cambio. Su uso está muy extendido.

Las principales características de AngularJS son (Hernández, 2015):

- Permite la creación de aplicaciones web de una sola página.
- A diferencias de otras librerías como jQuery, Knockout o Handlebars, que sólo dan solución a problemas concretos, AngularJS abarca un espectro más amplio, ofreciendo una solución más completa.
- Es un framework fácil de testear.
- Al estar soportada por una gran comunidad, está en continuo cambio.

Node.js es una librería y entorno en tiempo de ejecución, dirigida por eventos (por tanto asíncrona) y de código abierto, que normalmente se utiliza del lado del servidor. Se ejecuta sobre el mismo intérprete que Javascript, utilizando un motor desarrollado por Google, denominado V8, a una velocidad muy superior a la habitual. Esto trae como novedad el uso de Javascript del lado del servidor, que hasta entonces había quedado relegado al lado del cliente para tareas de menor carga computacional.

El uso de Node.js en el lado del servidor facilita la tarea de los desarrolladores, ya que la base de la programación es similar de ambos lados (cliente y servidor). Esto hace posible crear proyectos de una forma más ágil, de gran potencia, y con un coste de infraestructura menor (Node.js Foundation, 2015).

Es muy popular utilizar el denominado MEAN para la creación de aplicaciones web. MEAN es el acrónimo de MongoDb, ExpressJS, AngularJS y NodeJS. Consiste en el uso de un conjunto de subsistemas, todos ellos basados en Javascript, para la integrar de forma conjunta una plataforma autosuficiente (Mean by Linnovate, 2016).

Cada uno de los componentes de MEAN son de código abierto. De algunos ya hemos hablado anteriormente. Este conjunto se distribuye como muestra la Ilustración 9.

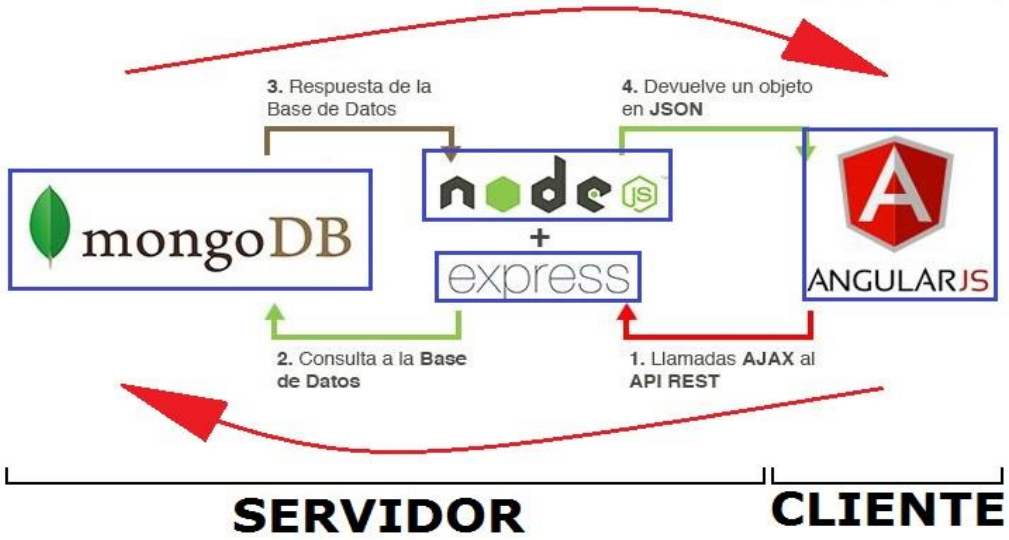


Ilustración 9: Diagrama MEAN

Como podemos ver, AngularJS trabaja en la parte cliente, mientras que NodeJS y ExpressJS hacen su trabajo en el servidor. MongoDb es una base de datos NoSQL que almacena los datos utilizando notación JSON, dotando al sistema de una rápida manipulación. Una de las características de esta base de datos es que es escalable. Al contrario de lo que ocurre en las bases de datos SQL, que pierden rendimiento cuanto mayor sea la cantidad de datos que contienen, en bases de datos de este tipo el funcionamiento no se ve apenas afectado.

Podemos ver que es posible dar solución perfectamente válida al problema de editar archivos EPG mediante una aplicación web. Javascript tiene infinidad de herramientas, librerías y frameworks que pueden asentar las bases del proyecto que este documento plantea.

Dado que uno de los requisitos de la aplicación es la portabilidad y disponibilidad, desecho esta opción por resultar más compleja. De esta forma elimino la necesidad de una parte servidora y una parte cliente, simplificando el esquema. Además, mis conocimientos iniciales sobre Javascript son básicos, y la curva de aprendizaje sería más lenta. Es mejor opción utilizar un lenguaje más familiar para tener un punto de partida más sólido.

3.2.2 – Aplicación de escritorio basada en Java o C++

Otra posible solución consiste en una aplicación de escritorio. Resulta lógico pensar que si vamos a trabajar con archivos de EPG locales, la aplicación corra en local. De esta forma evitamos la intervención de un servidor y todos los problemas derivados que eso conlleva.

C++ es un lenguaje de programación creado como evolución de C. Es considerado un lenguaje multiparadigma. Permite seguir patrones de programación estructurada, programación genérica y programación orientada a objetos. Estos dos últimos paradigmas son la principal diferencia con su antecesor C. De hecho, en sus inicios fue llamado "C con clases", hasta ser rebautizado como C++.

Las principales características de C++ son (Deitel, 2003):

- Es un lenguaje muy potente y robusto. Permite crear aplicaciones de gran complejidad.
- Está muy extendido aunque ahora tiene fuertes competidores. Por ello, existen una gran documentación en Internet.
- Es un lenguaje muy didáctico. La sintaxis de C (y de C++) es similar a cualquier otro lenguaje de uso común (Java, C#, Javascript, etc).
- El uso de librerías dinámicas (DLLs) es complejo. Existen lenguajes que mucho más evolucionados en ese sentido.

Por otro lado, y a diferencia de C++, Java es un lenguaje interpretado. A pesar de eso, el código de un programa en Java se compila a un punto intermedio, denominado bytecode. Esto hace posible que, aun siendo un lenguaje interpretado, la velocidad de ejecución sea relativamente rápida aunque puede ser limitante para ciertas aplicaciones. La Máquina Virutual Java (JVM, Java Virtual Machine) se encarga de mediar entre el bytecode y el código máquina. También cumple otras funciones relacionadas con la seguridad, portabilidad, y uso óptimo de recursos y memoria.

Las principales características de Java son (Garrido, 2015):

- Eliminación del complejo uso de punteros con respecto a otros lenguajes, entre ellos C++.
- Simplicidad en el uso de herencias mediante interfaces.
- Independencia de la plataforma software o hardware donde se ejecute gracias al uso de la JVM. Esto significa que un código Java puede ejecutarse en un Linux, en un Mac o en cualquier sistema operativo sin necesidad de hacer adaptación. La máquina virtual se encarga de las peculiaridades propias de cada sistema operativo.
- Es un lenguaje concurrente, permite el uso y gestión de diferentes hilos de ejecución de forma muy simple.
- Es más lento en comparación con C++.

Dado que el proyecto no requiere de una gran carga computacional, pero si ha de ser portable a cualquier sistema operativo, Java en su versión 8 es la mejor opción para dar solución al problema que se plantea. Además, Java es un lenguaje que me resulta más familiar y esas nociones básicas pueden ser de gran utilidad para obtener unos mejores resultados.

3.3 – AMBIENTE DE DESARROLLO INTEGRADO (IDE)

Una vez tomada la decisión de utilizar **Java** como lenguaje de programación, tenemos dar el siguiente paso: la elección de un IDE (Integrated Development Environment). Dado que Java es un lenguaje muy extendido y popular, existen gran variedad de IDEs con diferentes propósitos. Vamos a analizar brevemente algunos de ellos (Rodríguez, 2013):

- Eclipse: es un software de libre uso, muy utilizado a nivel profesional. Se pueden extender sus funcionalidades mediante la instalación de plugins.
- NetBeans: es también de libre uso. Otro IDE muy popular para uso profesional. También pueden añadirse plugins, llamados Beans, para añadir funciones extra.
- BlueJ: de libre uso y diseñado para entornos académicos. Sin uso profesional. Provee de herramientas que ayudan a la comprensión de la programación orientada a objetos.
- JBuilder: es un software comercial aunque existen versiones de prueba gratuitas con características reducidas.
- JCreator: es también una herramienta comercial.

Dado que el uso de la herramienta va a ser profesional, descartamos BlueJ. Vamos a centrar la atención en Eclipse y NetBeans, pues son herramientas que no necesitan licencia para ser utilizadas.

Eclipse es un IDE de libre uso, programado en Java, que proporciona herramientas de desarrollo software ampliable mediante el uso de plugins. Principalmente se utiliza como entorno de programación para desarrollos en Java pero puede ser utilizado para programar en otros lenguajes como C o C++.

Eclipse no proporciona en sí herramientas para desarrollos con ventanas, pero sí que dispone de plugins que añaden esta funcionalidad.

La otra opción posible que planteamos es el popular NetBeans. Este IDE es también de uso libre. Permite realizar, de forma bastante sencilla, aplicaciones con ventanas sin necesidad de ningún plugin. Dispone de diferentes packs en función para soportar proyectos C/C++. Muy interesante también para el desarrollo de código PHP, pues ofrece una depuración muy potente a los desarrolladores.

Dado que uno de los requerimientos es hacer un software sencillo de utilizar mediante la navegación mediante ventanas, NetBeans 8.1 es la opción más acertada para este caso. Ofrece una potente herramienta de gestión de ventanas y una potente depuración que resultaron muy útiles durante el desarrollo del proyecto.

3.4 – Otras tecnologías transversales

Una vez cerradas las cuestiones sobre el lenguaje de programación para desarrollar el proyecto, y del IDE para llevarlo a cabo, el siguiente paso consiste en realizar una búsqueda de tecnologías transversales que puedan ser de utilidad.

Como el software que me dispongo a desarrollar no está vinculado a ningún Sistema Operativo, gracias a que Java y su máquina virtual nos ofrecen compatibilidad multiplataforma, vamos a utilizar Windows por estar más habituado a su uso.

La empresa donde realicé las prácticas, Telefónica I+D, me facilita un ordenador Packard Bell de sobremesa equipado con Windows 8.1. Sus principales característica son:

- 1 GB de memoria RAM
- Procesador Core Duo a 2,33 GHz
- 500 Gb de Disco duro

Por otro lado, necesitamos un editor de imágenes sencillo. Este ha de ser de uso sencillo, pero con opciones avanzadas. Preferentemente gratuito, pero más potente que un editor básico. Tras una búsqueda en Internet, tomo la decisión de adoptar **paint.net** para este fin.

Paint.net es un software gratuito, disponible en http://www.getpaint.net/, disponible para Windows. Tiene una interface sencilla e intuitiva. Permite un tratamiento de las imágenes avanzado mediante el uso de capas (similar en este sentido a Photoshop). Esta soportado por una gran comunidad de usuarios, que provee de ayuda y tutoriales.

Ha resultado muy útil en nuestro proyecto para el diseño de los logos, iconos, y demás elementos gráficos, pues necesitaban un tratamiento previo con una herramienta software para imágenes. Permite hacer desde cosas sencillas, como reducir el tamaño o pequeños recortes, hasta la fusión de varias imágenes para formar un único logo.

También fue necesario crear un instalador del software en su versión para Windows. Existen numerosas herramientas software que permiten hacer esta tarea, pero la mayoría son de pago. En su versión de prueba, introducen incómoda publicidad. Finalmente opté por utilizar Inno Setup.

Inno Setup es una herramienta gratuita, disponible para Windows en http://www.jrsoftware.org/isinfo.php. Permite crear un único archivo .exe ejecutable, que facilita la instalación de un software. Esto facilita la distribución de este por Internet. A su vez, permite crear un desinstalador para eliminar todos los archivos pertenecientes al software. Soporta multilenguaje, archivos comprimidos y arquitecturas x64, entre otras cosas. En definitiva, es una herramienta muy completa, que dará un buen aspecto final a la presentación del proyecto.

En ocasiones ha sido necesario editar texto plano para generar archivos XML aptos para pruebas. Esta tarea le ha correspondido a Notepad++. Se trata de un editor de texto de código abierto y uso gratuito. Soporta una gran variedad de lenguajes de programación. Es de aspecto muy sencillo y similar al bloc de notas de Windows, pero en cambio ofrece herramientas muy interesantes como la búsqueda y conteo de palabras o el autocompletado, que lo hacen muy útil para desarrolladores de código. Es una herramienta de gran utilidad casi en cualquier contexto.

Para la crear y presentar la documentación en la empresa, he utilizado el editor de texto más popular para este fin: Word en su versión 2013. Partiendo de una plantilla preestablecida para crear manuales, he ido recogiendo todos los datos referentes al software de una manera ordenada, utilizando capturas y diagramas. Este manual está redactado en inglés, por requerimiento de la empresa, y ha sido traducido en español para ser incorporado en este documento. Puede encontrar el manual completo en el Capítulo 5. Posteriormente el manual fue presentado también en formato PDF.

Siguiendo las costumbres de Telefónica I+D para el manejo de toda la documentación, he creado una versión del manual en la Intranet, utilizando como herramienta MediaWiki. Esta herramienta ya está instalada en los servidores de la empresa y su uso es muy común para el intercambio de información.

Para el control de versiones voy a utilizar GitHub Enterprise. La empresa me ofrece una de las licencias de las que disponen para mi proyecto. De esta forma puedo llevar un control completo de las diferentes versiones. Habituarme a su uso es, al comienzo, una tarea dura pero los beneficios son numerosos. Puedo compartir el código con otros compañeros. También me permite mantener una copia de seguridad del proyecto, de forma que puedo trabajar desde diferentes ordenadores, teniendo disponible en todo momento la última versión de los archivos que modifico. Como cliente voy a utilizar la consola gratuita Git Bush (descargado de https://git-for-windows.github.io/)

3.5 – Resumen de tecnologías utilizadas

	Opción final
Sistema Operativo	Windows 8.1
Lenguaje de programación	Java 8
IDE	NetBeans 8.1
Editor de imágenes	Paint.net 4.0.9
Crear el instalador	Inno Setup 5.5.9
Editor de texto	Notepad++
Documentación estándar	Word 2013
Documentación para la Intranet	MediaWiki (www.mediawiki.org)
Control de versiones	GitHub Enterprise con Git Bush como cliente

Segunda Parte

Capítulo 4: Desarrollo de la aplicación

4.1 – Análisis de una EPG

Una de las primeras tareas que debemos llevar a cabo, antes de abordar el proyecto, es conocer y estudiar la naturaleza de los archivos EPG. Para ello, partimos de la base de que estos archivos que contienen la guía de programación, los llamados EPG, son ficheros XML bajo el esquema GLF. Por lo tanto, vamos a analizar primero el lenguaje XML.

4.1.1 – Lenguaje XML

XML es el acrónimo de *eXtensible Markup Language*, que en castellano quiere decir Lenguaje de Marcas eXtensible. Este lenguaje propone un estándar para el intercambio de información. Fue inventado por la compañía IBM, allá en los años setenta, con el objetivo de almacenar información y compartirla de forma ordenada y bien estructurada dentro de la empresa. Años más tarde fue normalizado por la ISO (International Organization for Standardization) y sirvió de base para otro lenguaje, hoy en día bien conocido, como el HTML.

Las piezas básicas de un archivo XML son los **elementos** y los **atributos**. Estos han de cumplir una serie de requisitos (Tutorials point, 2014):

- El nombre de los elementos puede formarse utilizando cualquier combinación de caracteres alfanuméricos, guiones (-), guiones bajos (_) y puntos (.). Como excepción, no puede empezar por xml, XML o cualquier variante. Tampoco puede empezar por guiones, guiones bajos o puntos.
- Es sensible a mayúsculas y minúsculas.
- Cada elemento se compone de dos etiquetas: la de apertura y la de cierre. Por ejemplo:

```
<elemento1> Contenido </elemento1>
```

- Los elementos pueden contener a su vez otros elementos. Han de estar entre la etiqueta de apertura y la de cierre, de forma anidada.
- Todas las etiquetas tienen que ser cerradas en el orden apropiado. Esto quiere decir que una etiqueta ha de ser cerrada antes que la etiqueta exterior lo haga. Por ejemplo:

```
<elemento-exterior>
  <elemento-interior>
      Contenido interior
  </elemento-interior>
</elemento-exterior>
```

- Un elemento vacío puede ser cerrado de dos formas:

```
<elemento-vacio /> ó <elemento-vacio></elemento-vacio>
```

- Los elementos pueden tener atributos. Los atributos son siempre un par nombre-valor. Se utilizan de la siguiente forma:

```
<elemento1 atributo1 = "Valor 1" atributo2='Valor 2'>
  Contenido elemento1
</elemento1>
```

Como podemos ver, el nombre del atributo se separa con un espacio del nombre de la etiqueta. Luego se coloca el signo igual (=) y después el valor del atributo, siempre dentro de comillas simples (‘ ’) o dobles (“ ”).

- Los atributos no pueden aparecer más de una vez en la etiqueta de inicio de un mismo elemento.

Además, los archivos XML suelen empezar, aunque no es obligatorio, con un prólogo. Este se encuentra situado en la primera línea, y describe los datos principales del documento. Por ejemplo:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Como podemos ver, se compone de tres elementos:

- *version* (obligatorio): La versión utilizada de XML. En este ejemplo, la versión es la 1.0.
- *encoding* (opcional): La codificación utilizada para caracteres del documento. Se puede usar US-ASCII, UTF-8, UCS-2, EUC-JP, Shift_JIS, Big5 o ISO-8859-1 hasta ISO-8859-7. El valor por defecto, en caso de omisión, es UTF-8.
- *standalone* (opcional): Documento autónomo. Indica si se trata de un documento independiente (standalone=”yes”), o si responde a un esquema prefijado proveniente de una fuente externa (standalone=”no”). Esta última, es la opción por defecto en caso de omitir el dato.

Este esquema prefijado que proviene de una fuente externa se denomina DTD (Document Type Definitión). Contiene la información necesaria para validar y definir los datos del documento XML. En sentido amplio, el DTD es una descripción detallada de la estructura que nos vamos a encontrar en el XML, o dicho coloquialmente, es el “molde” del documento XML.

El DTD es el formato nativo de XML para describir el esquema. Pero presenta ciertas limitaciones. No es capaz de soportar nuevas extensiones del XML ni permite describir ciertos aspectos formales a nivel expresivo de un documento XML. Esto, unido a una sintaxis que dista bastante de la de XML, son los detonantes para que haya caído en desuso.

Como evolución del DTD, nace el XSD (XML Schema Definition). Es un lenguaje de sintaxis más similar a XML, que ofrece una solución más completa y potente, para proporcionar una capacidad descriptiva mucho mayor. Además es capaz de soportar la extensión del documento XML.

Vamos a contrastar las diferencias de forma y sintaxis entre el DTD y el XSD para un mismo ejemplo de código XML. Este es el código:

```
<e-mail prioridad="alta" enviado="yes" servidor=”gmail.com”>
  <remitente>prueba-1@gmail.com</remitente>
  <destinatario>prueba-2@gmail.com</destinatario>
  <asunto>Esto es una prueba</asunto>
  <contenido>Este e-mail es de prueba</contenido>
</e-mail>
```

La siguiente tabla recoge ambos esquemas para el código XML superior utilizando DTD y XSD.

DTD

```
<!ELEMENT e-mail (remitente, destinatario, asunto, contenido?)>

<!ATTLIST e-mail prioridad (alta | normal) “normal”) #IMPLIED
enviado (yes | no) “no” #REQUIRED
servidor CDATA #REQUIRED)>

<!ELEMENT remitente (#PCDATA)>
<!ELEMENT destinatario (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT contenido (#PCDATA)>
```

XSD

```
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="e-mail">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="remitente"/>
        <xs:element type="xs:string" name="destinatario"/>
        <xs:element type="xs:string" name="asunto"/>
        <xs:element type="xs:string" name="contenido"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="prioridad"/>
      <xs:attribute type="xs:string" name="enviado"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Según refleja la tabla, existen grandes diferencias de notación entre ambos métodos de especificación. El DTD tiene una sintaxis nueva, que exige al desarrollador de su conocimiento. Por el contrario, el XSD tiene una sintaxis más similar a la de XML. También llama la atención la complejidad del XSD frente al DTD. Esto hace posible una mayor capacidad de detalle del XSD frente al DTD, que es más limitado (Pascual, 2015) (ASP tutor, 2016).

Asentadas las bases de XML y de los archivos de especificación (DTD o XSD), podemos plantearnos varias técnicas para generar los archivos XML que contienen información. Podemos evitar el uso de atributos, centrando la estructura en el uso de elementos anidados. O por el contrario, podemos crear elementos vacíos con varios atributos. Es importante fijar el esquema de nuestro documento, para luego seguir unas pautas.

Lo habitual es utilizar los atributos para especificar, con un poco más de detalle, el elemento. Suelen ser atributos que detallan un subtipo del elemento. En ocasiones detallan el formato de los datos que contienen o el identificador del elemento en cuestión. Pero no tiene por qué ser siempre así. Puede haber una gran diversidad en el uso de atributos.

Para comprender mejor la esencia de una estructura XML, vamos a poner varios ejemplos. Imaginemos en una estructura XML para almacenar los datos de los miembros de una escuela. Vamos a resolver el problema de varias formas, siguiendo en cada caso, una pauta diferente.

Este sería un posible esquema utilizando elementos anidados:

```
<escuela>
  <docente>
      <director>
          <d>
              <id>0001</id>
              <nombre>Julio Campo</nombre>
          </d>
      </director>
      <profesores>
          <p>
              <id>1001</id>
              <nombre>Pedro Vals</nombre>
          </p>
      </profesores>
      <alumnos>
          <a>
              <id>2001</id>
              <nombre>Luis Cerro</nombre>
          </a>
      </alumnos>
  </docente>
  <noDocente>
      <limpieza>
          <l>
              <id>3001</id>
              <nombre>Ana Jiménez</nombre>
          </l>
      </limpieza>
      <seguridad>
          <s>
              <id>4001</id>
              <nombre>Miguel Barros</nombre>
          </s>
      </seguridad>
      <bedel>
          <b>
              <id>5001</id>
              <nombre>Jose Luis Ramos</nombre>
          </b>
      </bedel>
  </noDocente>
</escuela>
```

Como podemos apreciar, el personal de la escuela se divide en varios grupos anidados siguiente esta estructura:

- Docente
    - Director
    - Profesores
    - Alumnos
- No docente
    - Limpieza
    - Seguridad
    - Bedel

Otra posible clasificación, utilizando los atributos, podría ser así:

```
<escuela>
  <miembro docente="yes" rol="director" id="0001"
  nombre="Julio Campo" />
  <miembro docente="yes" rol="profesor" id="1001"
  nombre=" Pedro Vals" />
  <miembro docente="yes" rol="alumno" id="2001"
  nombre=" Luis Cerro" />
  <miembro docente="no" rol="limpieza" id="3001"
  nombre=" Ana Jiménez" />
  <miembro docente="no" rol="seguridad" id="4001"
  nombre="Miguel Barros" />
  <miembro docente="no" rol="bedel" id="5001"
  nombre="Jose Luis Ramos" />
</escuela>
```

En este caso solo tenemos dos elementos, escuela y miembro, y son los atributos los que detallan la clase de miembro y rol que desempeñan dentro de la escuela. Ambas soluciones son perfectamente válidas. Lo importante es llegar a un acuerdo, mediante un esquema cerrado prefijado, que esté en conocimiento a disposición de todas las personas que vayan a estar involucradas con el manejo de estos datos.

4.1.2 – El esquema GLF

Como ya hemos visto en el apartado anterior, los ficheros XSD describen la estructura que han de cumplir los datos estructurados de un XML. Este es el caso de los datos de la guía de programación, también denominada EPG. Para este fin, existe un esquema, propiedad de Microsoft, denominado GLF (*Global Listings Format*). Se puede ver el esquema completo en el Anexo 2.

En este apartado vamos a desglosar el contenido de los archivos XML, encuadrados en el esquema GLF. Para ello, y como primera aproximación, la Ilustración 10 presenta un diagrama que muestra el anidamiento de los diferentes elementos.

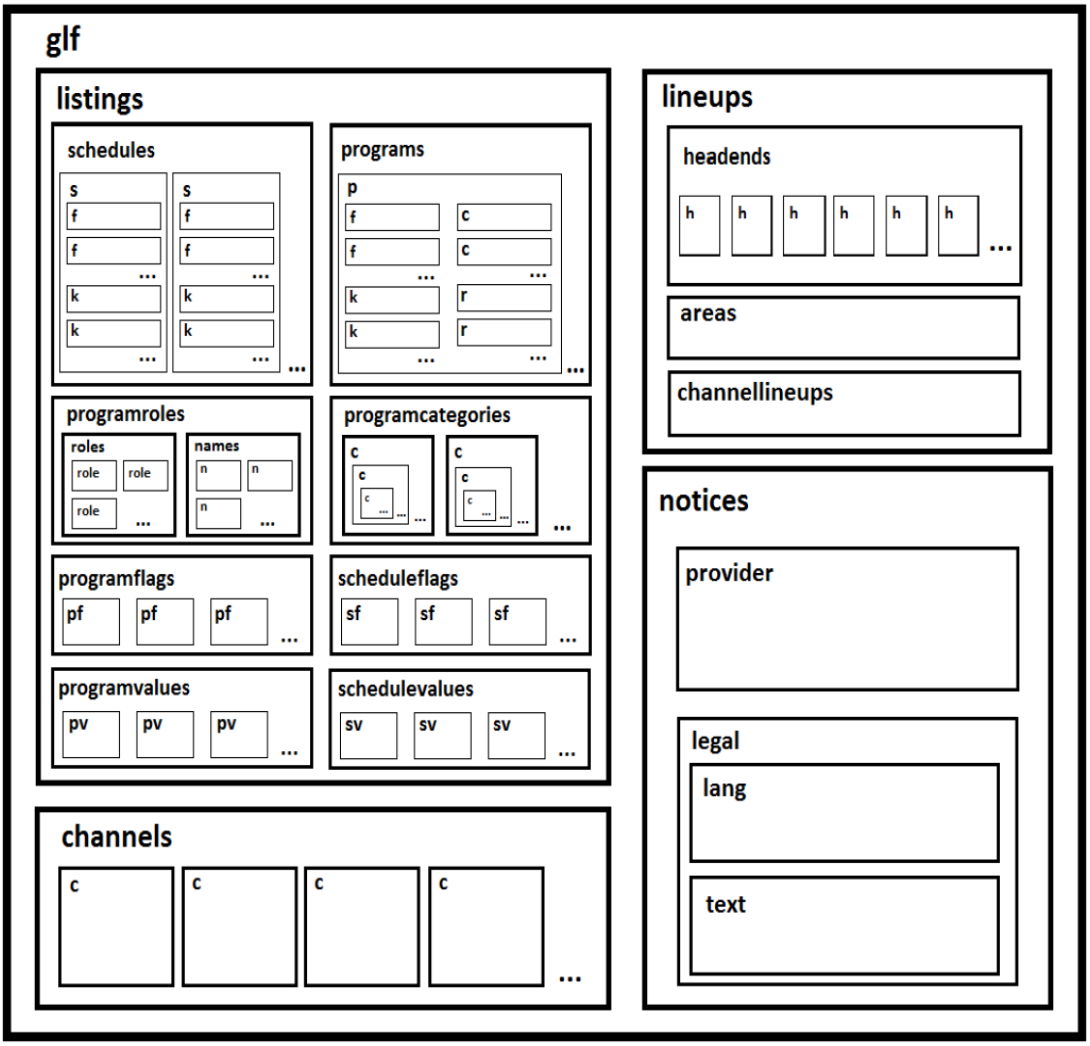


Ilustración 10: Esquema GLF

El esquema GLF establece cuatro elementos principales: *listings*, *lineups*, *channels*, *notices*. De estos cuatro elementos, centraremos la atención en sólo dos de ellos, *listings* y *channels*. Estos dos elementos son los que contienen toda la información relevante a la guía de programación, y por lo tanto son los elementos que podremos visualizar y editar mediante EPG Viewer.

4.1.2.1 & 4.1.2.2 – Lineups & Notices

Carecen de importancia en este documento, puesto que el EPG Viewer no puede ni visualizar ni editar estos dos elementos. El contenido de 'lineups' y 'notices' viene determinado por el proveedor del servicio de EPG, y no es relevante para el desarrollo del proyecto.

4.1.2.3 – Channels

Este elemento contiene, como era de esperar, la lista de canales. Cada canal genera un nuevo elemento, de nombre 'c', que dispone de una serie de atributos que lo describen. El siguiente ejemplo muestra una lista con dos canales:

```
<channels>
  <c id="2543" c="LA1HD" l="La 1 HD" iso3166="ES" iso639="ES"
  i="http://www.url.que/lleva_a_la_imagen1.png" />
  <c id="2544" c="LA2" l="La 2" iso3166="ES" iso639="ES"
  i="http://www.url.que/lleva_a_la_imagen2.png" />
</channels>
```

Cabe señalar que este elemento tiene tres atributos obligatorios:

- id: identifica de forma inequívoca el canal dentro de la EPG. Este id es que se usará para referenciar las emisiones con este canal.
- c: también llamado *Call Letter.* Este atributo identifica el canal mediante un nombre. No debe haber canales con el mismo *Call Letter*.
- l: es el nombre del canal. Suele ser, junto con el *Call Letter*, el nombre que se utiliza para mostrar al usuario en interfaces gráficas.

El resto atributos son opcionales. El más utilizado es el atributo 'i', que se usa para añadir la URL a la imagen del logo del canal en cuestión.

4.1.2.4 – Listings

Este elemento contiene ocho listas. Estas listas, junto con la lista de canales, se encuentran relacionadas entre sí mediante los identificadores únicos de cada elemento (atributo 'id' de cada elemento). Las listas son las siguientes:

- schedules: contiene la lista de las emisiones. Son el eje principal de la EPG. Interrelacionan elementos de la lista de *channels* con la lista *programs*.
- programs: esta lista contiene la descripción detallada de los programas.
- programroles: contiene a su vez dos listas: *roles* y *names*. Hablaremos más adelante en detalle.
- programflags: contiene la lista de posibles *flags* que pueden utilizarse en la lista *programs*.

- scheduleflags: contiene la lista de posibles *flags* que pueden utilizarse en la lista *schedules*.
- programcategories: contiene la lista de posibles categorias que pueden utilizarse en la lista *programs*.
- programvalues: contiene la lista de posibles *values* que pueden utilizarse en la lista *programs*.
- schedulevalues: contiene la lista de posibles *values* que pueden utilizarse en la lista *schedules*.

A – Lista 'schedules'

La primera de estas listas, y una de las más importantes, es la lista de emisiones, denominadas *schedules*. Representa el eje principal de la EPG, pues cada elemento de esta lista está ligado a un elemento de la lista *programs*, y a un elemento de la lista *channels*. También es, por lo general, la lista con mayor número de elementos. Recoge las emisiones de todos los canales a lo largo de una o dos semanas. Suele tener un valor medio de alrededor de 30,000 elementos, aunque en algunos casos puede superar los 100,000.

Esta lista se compone de elementos 's', cada uno de los cuales representa una emisión. Dispone de 4 atributos obligatorios: 's', 'p', 'c' y 'd'. Además puede llevar anidados dos tipos de elementos: *flags* y/o *values*. Los atributos son los siguientes:

- s: este atributo representa la fecha y hora del inicio de la emisión. El formato habitual es AAAA-MM-DDThh:mm:ss donde 'A' representa el año, 'M' el mes, 'D' el día, 'h' la hora (en formato 24, de 00 a 23), 'm' los minutos, y 's' los segundos. La hora de emisión siempre es UTC (*Universal Time Coordinate*), lo cual quiere decir que para España, siempre será de una hora menos (dos cuando usemos el horario de verano).
- d: representa la duración de la emisión en segundos. Este valor ha de ser múltiplo de 60 (un minuto completo). Tiene como valor mínimo 60 segundos (1 minuto) y como valor maximo 86400 segundos (24 horas).
- c: relaciona la emisión con el elemento 'c' de la lista *channels* donde se emite mediante el atributo id del canal. Una emisión sólo puede estar asociada a un canal. Un canal puede tener múltiples emisiones.
- P: relaciona la emisión con el elemento 'p' de la lista *programs* que contiene la información sobre dicha emisión, mediante el atributo id del programa. Esto permite que un programa pueda ser emitido varias veces utilizando la misma descripción.

Por otro lado, los elementos que puede llevar anidados son dos:

- f: representa una bandera o *flag*. Tiene un atributo obligatorio, denominado 'id', que referencia con el *scheduleflag* correspondiente. Una misma emisión puede llevar múltiples *flags*.
- k: representa un valor o *value*. Tiene dos atributos obligatorios, 'id' y 'v'. El primero, como es habitual, referencia al correspondiente *schedulevalue*. El segundo es el valor propiamente dicho. Una misma emisión puede llevar múltiples *values*.

Las siguientes líneas representan algunos ejemplos de la forma que pueden tomar cada uno de los elementos de la lista *schedules*:

```
<schedules>
  <s s="2015-06-20T10:30:00" d="3600" p="40360530" c="597"/>
  <s s="2015-06-20T11:30:00" d="5400" p="40360531"
  c="597"></s>
  <s s="2015-06-20T13:00:00" d="3600" p="40360532" c="597">
  </s>
  <s s="2015-06-20T14:00:00" d="3600" p="40360533" c="597">
      <f id=”2”>
  </s>
  <s s="2015-06-20T15:00:00" d="3600" p="40360534" c="597">
      <k id=”1” v=”euro”>
      <k id=”3” v=”10”>
      <f id=”16”>
  </s>
</schedules>
```

Todas las emisiones de la lista superior pertenecen al canal con id 597. Además, las emisiones no se solapan, ni tampoco dejan agujeros vacíos. Algunas emisiones tienen *flags* y *values* pertenecientes a la siguiente lista:

```
<scheduleflags>
  <sf id="2" name="STEREO" pname="Estereo" desc="Programa
  emitido en estereo" lang="es-ES"/>
  <sf id="16" name="MINISERIES" pname="Mini Serie" desc="El
  programa es una miniserie" lang="es-ES"/>
  <sf id="30" name="USER-NETWORK-DVR" pname="Grabable"
  desc="El programa es grabable" lang="es-ES"/>
  <sf id="100" name="BLACKOUT" pname="Blackout"
  desc="Blackout" lang="es-ES"/>
</scheduleflags>

<schedulevalues>
  <sv id="1" name="PPV-SCHEDULE-PRICE-CURRENCY"
  pname="Moneda" desc="Moneda"/>
  <sv id="3" name="PPV-SCHEDULE-PRICE" pname="Precio PPV"
  desc="Precio PPV"/>
</schedulevalues>
```

B – Lista ‘programs’

Otra de las listas principales es la lista ‘programs’. Contiene la descripción detallada de cada uno de los programas a los que se hace referencia desde la lista ‘schedules’. Los elementos de esta lista disponen de un atributo ‘id’ único, además de otros atributos. Además, pueden llevar cuatro tipos de elementos anidados:

- f: representa una bandera o *flag*. Tiene un atributo obligatorio, denominado ‘id’, que referencia con el *programflag* correspondiente. Un mismo programa puede llevar multiples *flags*.
- k: representa un valor o *value*. Tiene dos atributos obligatorios, ‘id’ y ‘v’. El primero referencia al correspondiente *programvalue*. El segundo es el valor propiamente dicho. Un mismo programa puede llevar múltiples *values*.
- r: representa un nombre. Este elemento tiene dos atributos que a su vez lo referencian con un elemento de la lista *names* y un elemento de la lista *roles* (que son elementos anidados de la lista *programroles*). El tercer atributo establece el orden en que debe mostrarse la lista de nombres y es opcional.
- c: representa una categoría. Tiene un atributo obligatorio que lo relaciona con un elemento de la lista *programcategories*, y sirve para categorizar los programas. Un mismo programa puede tener varias categorías asociadas.

Para entenderlo mejor, vamos a poner un ejemplo de elemento 'p', dentro de la lista *programs*:

```
<programs>
  <p id="40413321" t="Cine: 16 calles" d="El detective de la
  policía de Nueva York Jack Mosley recibe una misión
  aparentemente sencilla: el delincuente Eddie Bunker está
  citado para testificar y debe trasladarle de la celda al
  juzgado, a 16 calles de distancia. Al atravesar la ciudad
  de los rascacielos en hora punta de la mañana, no se da
  cuenta de que le sigue una furgoneta. El agente decide parar
  ante una tienda de licores para comprar el desayuno. De
  repente, Eddie, que espera en el interior del coche, se
  encuentra con una pistola...">
      <f id="29"/>
      <k id="4" v="2006"/>
      <k id="5" v="Estados Unidos"/>
      <k id="99" v="16 blocks"/>
      <k id="20"
      v="http://www.mediadata.tv/api_image/fotos/2014/04/293
      75295.jpg"/>
      <k id="21"
      v="http://www.mediadata.tv/api_image/MC4G_Caratula/201
      4/04/29375296.jpg"/>
      <k id="200" v="Cine de acción y suspense"/>
      <k id="100" v="12"/>
      <c id="501"/>
      <r r="1" n="29" o="1"/>
      <r r="2" n="30" o="2"/>
      <r r="2" n="31" o="3"/>
      <r r="3" n="32" o="4"/>
      <r r="7" n="33" o="5"/>
  </p>
</programs>
```

Podemos ver que lleva asociados varios elementos de otras listas. En las siguientes líneas vamos a ver qué elementos, y de qué listas son:

```
<programflags>
  <pf id="29" name="MOVIE" pname="Película" desc="Película"
  lang="es-ES"/>
</programflags>

<programvalues>
   <pv id="4" name="Creation_Year" pname="Año" desc="Año de
   creación"/>
   <pv id="5" name="Originating_Country" pname="País"
   desc="País de producción"/>
   <pv id="99" name="ORIGINAL_TITLE" pname="Título original"
   desc="Título original"/>
   <pv id="100" name="SR" pname="Sistema de clasificación
   moral" desc="Sistema de clasificación moral"/>
   <pv id="200" name="SHOWTYPE" pname="Theme"
   desc="Theme#SubTheme"/>
   <pv id="20" name="Image_Landscape" pname="Landscape"
   desc="URL Landscape"/>
   <pv id="21" name="Image_Portrait" pname="Portrait"
   desc="URL Portrait"/>
</programvalues>

<programcategories>
  <c id="501" mscname="ACTION MOVIE" value="Accion"/>
</programcategories>

<programroles>
  <roles>
      <role id="1" title="Director" desc="Director"/>
      <role id="2" title="Intérprete" desc="Intérprete"/>
      <role id="3" title="Guión" desc="Guión"/>
      <role id="7" title="Producción" desc="Producción"/>
  </roles>
  <names>
      <n id="29" lname="Richard Donner"/>
      <n id="30" lname="Bruce Willis"/>
      <n id="31" lname="Mos Def"/>
      <n id="32" lname="Richard Wenk"/>
      <n id="33" lname=" Millennium Films"/>
  </names>
</programroles>
```

Es fácil identificar en el código del programa los elementos de otras listas a los que se hace referencia. Esta forma de describir el programa, mediante el uso de referencias a otras listas, promueve la reutilización de la información. Así pues, por ejemplo, sólo es necesario escribir el nombre de un actor una vez, y llamarlo tantas veces como sea necesario.

C – Lista 'programroles'

Esta lista se compone de dos. La primera de ellas contiene los posibles roles que pueden darse en la creación de un programa, como la producción, el guion o la dirección. La segunda lista contiene todos los nombres de actores, productores, guionistas o productores. De esta forma, como ya hemos visto en el ejemplo anterior, es fácil relacionar un actor y su rol, con un programa determinado.

Esta separación permite a su vez que una misma persona (o entidad) pueda ser incluida en la creación de un programa, jugando roles distintos en cada uno. Este es el caso de algunos actores, que algunas veces juegan el rol de directores, guionistas o incluso productores.

Estas listas suelen ser fijas en diferentes archivos EPG, aunque muchos de los elementos no sean llamados nunca. De esta forma, los identificadores de cada miembro de la lista son fijos y resulta más cómodo a la hora crear un archivo EPG.

D,E,F,G – Lista 'programcategories'

La lista *programcategories* contiene la descripción de diferentes categorías que clasifican los programas. Estas categorías se pueden anidar entre sí. Lo habitual es anidar las categorías en dos niveles como máximo. Vamos a ver un ejemplo para entender mejor este anidamiento de categorías y subcategorías:

```
<c id="5" mscname="MOVIES" value="Películas">
    <c id="500" mscname="All-movies" value="Todo"/>
    <c id="501" mscname="Action-movie" value="Acción"/>
    <c id="502" mscname="Comedy-movie" value="Comedia"/>
    <c id="503" mscname="Drama-movie" value="Drama"/>
    <c id="510" mscname="Other-movie" value="Otros"/>
</c>

<c id="6" mscname="NEWS" value="Noticias">
    <c id="600" mscname="All news" value="Todo"/>
    <c id="601" mscname="CurrentEvts" value="Actualidad"/>
    <c id="602" mscname="Sports-news" value="Deportes"/>
    <c id="603" mscname="Weather-news" value="El Tiempo"/>
    <c id="610" mscname="Other-news" value="Otros"/>
</c>
```

El ejemplo anterior muestra dos niveles de anidamiento. Esto permite categorizar los programas de una forma más concreta. Esta lista de categorías, al igual que ocurría con la lista 'programroles', se mantiene constante para todos los archivos de EPG. Categorizar los programas tiene grandes ventajas para el usuario, ya que posibilita realizar búsquedas sencillas, o recomendar contenidos nuevos.

4.5 – Listas 'programflags', 'programvalues', 'scheduleflags' y schedulevalues'

Estas cuatro listas recogen los posibles *flags* y *values* que pueden ser utilizados en las listas 'schedules' y 'programs'. Los elementos de estas listas suelen ser también fijos, se usen o no se usen. De esta forma, los identificadores se mantienen constantes, facilitando el uso de los mismos. En las siguientes líneas mostramos algunos ejemplos de declaración de *flags* y *values*:

```
<programflags>
  <pf id="32" name="PAIDPROGRAMMING" pname="PPV" desc="Paid
  Programming" lang="es-ES"/>
  <pf id="34" name="SERIES" pname="Series" desc="Series"
  lang="es-ES"/>
</programflags>

<scheduleflags>
  <sf id="2" name="STEREO" pname="Estereo" desc="Programa
  emitido en estereo" lang="es-ES"/>
  <sf id="30" name="USERNETWORKDVR" pname="Grabable"
  desc="El programa es grabable" lang="es-ES"/>
</scheduleflags>

<programvalues>
  <pv id="3" name="Runtime_Minutes" pname="Duración min."
  desc="Duración en minutos"/>
  <pv id="99" name="ORIGINAL_TITLE" pname="Título original"
  desc="Título original"/>
</programvalues>

<schedulevalues>
  <sv id="1" name="PPV-SCHEDULE-PRICE-CURRENCY"
  pname="Moneda" desc="Moneda"/>
  <sv id="3" name="PPV-SCHEDULE-PRICE" pname="Precio PPV"
  desc="Precio PPV"/>
</schedulevalues>
```

4.1.3 –Contenido del fichero EPG

Como podemos ver, todas estas listas anteriormente descritas están interrelacionadas mediante identificadores únicos. El sistema puede resultar confuso y complejo, pero lo cierto es que la información está perfectamente estructurada y contenida en listas separadas bien definidas. La Ilustración 11 nos ayuda a comprender la relación entre los elementos de las principales lista de un archivo EPG.

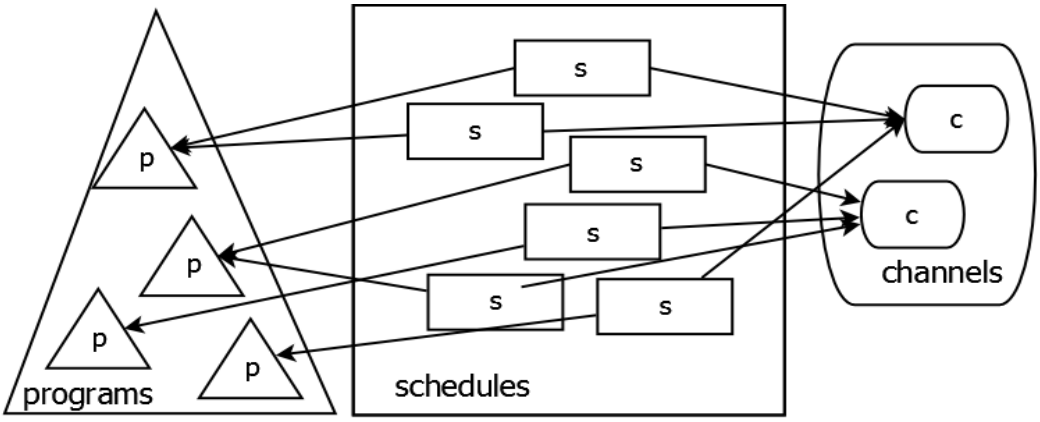


Ilustración 11: Relación entre principales listas

Podemos ver que el eje principal de la información de un fichero de EPG es la lista *schedules*. Los elementos de esta lista tienen una relación directa con los elementos de las listas *channels* y *programs*. Es evidente que el número de elementos de la lista *schedules* es directamente proporcional con el número de días de programación que abarque el fichero. En cambio el número de canales se mantiene constante y depende del servicio de televisión al que pertenezca el fichero. El número de elementos de la lista *programs* también depende del margen temporal que abarque el fichero, pero la relación no es tan fuerte. Existen otros factores como el factor de reutilización de programas (reposiciones del mismo programa a diferentes horas o en diferentes canales), duplicación de la programación para el mismo canal en SD (*Standard Definition*) y HD (*High Definition*) o la inclusión de programas "muertos" (que nunca se usen). Esto último se da, por ejemplo, cuando el proveedor de la EPG incluye la descripción de todos los capítulos de una temporada, pero solo se van a emitir algunos de ellos durante el periodo temporal que abarca el fichero.

Si analizamos con detenimiento los elementos 's' y 'p', podemos ver que otros elementos ayudan a definir estos (Ilustración 12 e Ilustración 13):

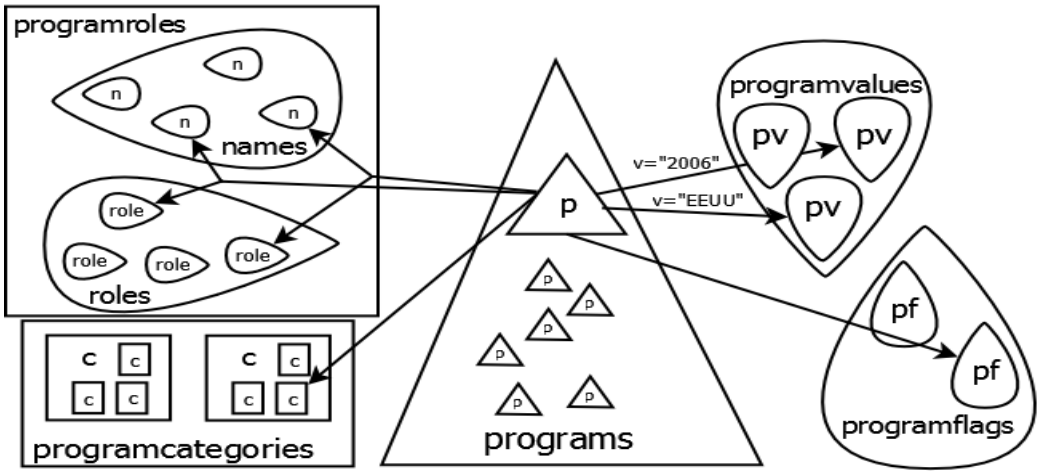


Ilustración 12: Detalle elemento 'p' de la lista programs

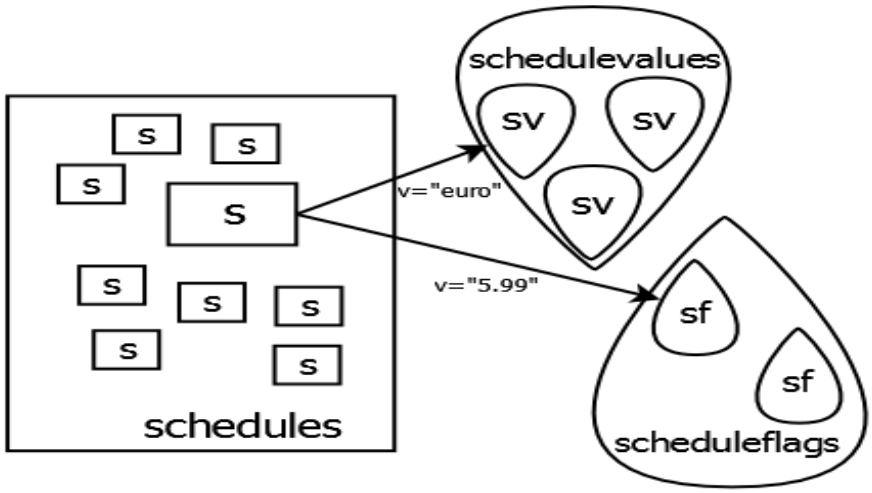


Ilustración 13: Detalle elemento 's' de la lista schedules

Poco queda que decir con respecto a los archivos EPG. La estructura del fichero ha quedado descrita por partes en líneas anteriores. Todos esos principios unidos y replicados cientos de veces, conforman el archivo que contiene la información de la guía de programación. Las listas de *flags* y *values* son las herramientas estrella para el diseño de nuevos servicios, basados en la estructura básica EPG.

Por último, y para finalizar este apartado, vamos a listar los requisitos (y recomendaciones) en cuanto a contenido que debe cumplir el fichero de EPG. Estos requerimientos han de darse de forma simultánea a los descritos con anterioridad con respecto a la arquitectura.

- Los identificadores de los elementos 'c' (atributo 'id') de la lista *channels* no han de repetirse. Han de ser números naturales (del 1 en adelante).
- El *Call Letter* de los elementos 'c' (atributo 'c') de la lista *channels* no ha de ser usado por más de un canal (recomendación).
- El valor del *Call Letter* y del identificador asociado a un canal ha de ser el mismo para ese mismo canal en distintos archivos de EPG que provengan del mismo proveedor de contenidos (recomendación). En caso de cambiar puede dar conflictos o errores.
- La duración de las emisiones ha de ser múltiplo de 60 segundos, es decir, minutos completos. Además la duración minimiza es de 1 minuto y la máxima de 24 horas (aunque para ese último caso se recomienda hacer dos emisiones de 12 horas).
- No debe haber agujeros (*gaps*) entre dos emisiones consecutivas en el mismo canal. En el caso de haber huecos en la programación, se debe rellenar con emisiones asociadas a programas de contenido vacío (recomendación).
- No se deben incluir emisiones solapadas (*overlapping*). Esto se da cuando la duración de una emisión supera en el tiempo la hora de inicio de la emisión posterior. Esto puede causar errores importantes.

4.2 – EPG VIEWER 1.0 COMO EDITOR DE EPG

Dada la complejidad de los ficheros EPG y la frecuencia con la que son editados para pruebas, el presente documento propone un software, llamado EPG Viewer, que posibilita la visualización y edición de estos de una forma mucho más cómoda que la habitual.

Una de las primeras cuestiones que hubo que resolver era la lectura del fichero XML. Dado el desconocimiento del lenguaje elegido para el desarrollo, durante los primeros meses el desarrollo se hizo de forma manual. Tan sólo utilicé una librería auxiliar, llamada XmlParser, que permite transformar el contenido de un fichero en una serie de nodos anidados. Este método consumía gran cantidad de recursos, dado que los ficheros que se maneja contienen gran cantidad de datos. Los errores eran comunes, y no resultaba cómodo manejar los datos de esta forma.

Dado que el método en cuestión, así como la librería XmlParser no dieron solución al problema, se descartó por completo su utilización. En su lugar, investigué acerca de otro método más eficiente, muy utilizado, y sin embargo mucho más simple: JAXB

4.2.1 – PRINCIPIOS DE JAXB

JAXB (*Java Architecture for XML Binding*) es un estándar Java que permite transformar un esquema XML en una representación de objetos java. Utilizando la API de JAXB, se puede mapear un objeto java en un documento XML (*marshal*) o la operación opuesta, es decir, transformar un documento XML en un objeto java asociado. Esto facilita enormemente el manejo de documentos XML mediante Java (java.net, 2016).

El primer paso consiste en compilar el fichero que contiene el esquema del documento XML que queremos capturar. En nuestro caso estamos hablando del fichero XSD que contiene la estructura GLF. Para esta tarea, JAXB dispone de un compilador, llamado `xjc` , que mediante un sencillo paso, permite crear todas las clases java necesarias para capturar la información de los documentos XML bajo el esquema GLF.

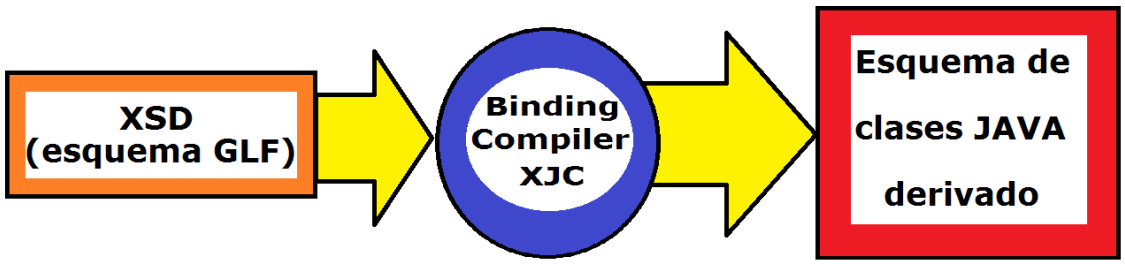


Ilustración 14: Compilación el esquema del documento XML

Para completar este paso, nos situamos en la carpeta dónde se encuentre el esquema y ejecutamos en la línea de comandos el siguiente código:

```
C:\Users\Public\Documents\GLF>xjc esquema_GLF.xsd
parsing a schema...
compiling a schema...
C:\Users\Public\Documents\GLF>
```

Como resultado de esta operación, se generan una serie de archivos .java necesarios para la aplicación. Estos archivos se encuentran organizados por carpetas, de forma que mantienen la jerarquía correspondiente al contenido del esquema GLF. La Ilustración 15 contiene una captura con los ficheros .java creados por el compilador:

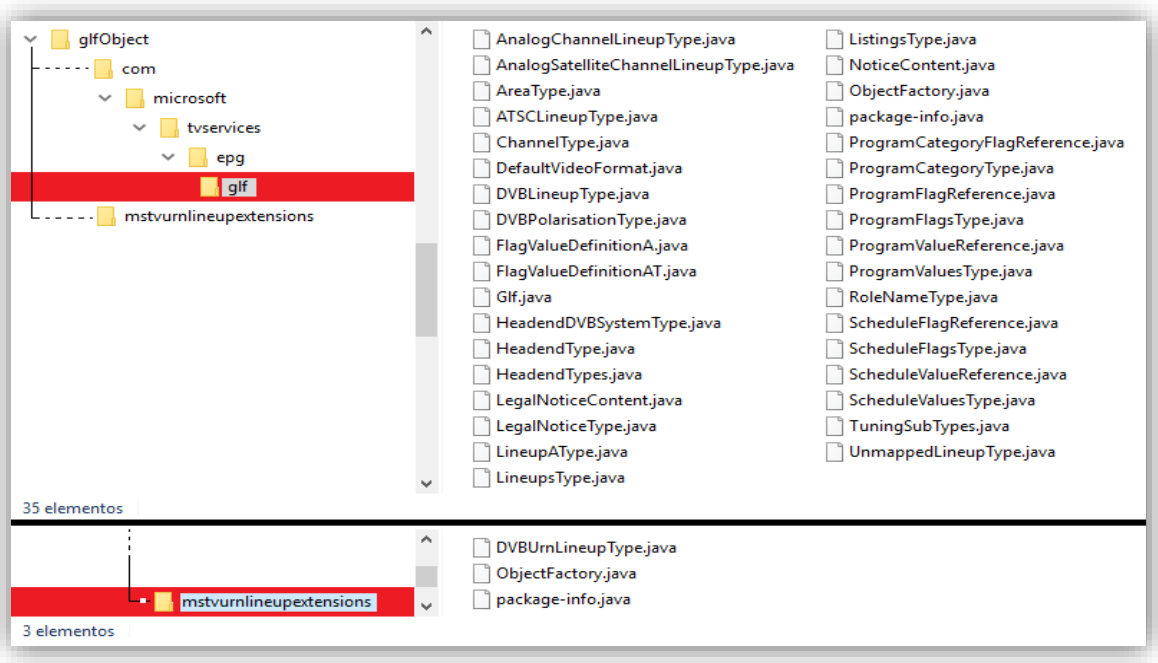


Ilustración 15: Ficheros creados por el compilador XJC

Todos estos archivos contienen el código necesario para manejar los datos contenidos en un documento XML bajo el esquema GLF. Esto incluye métodos *get* y *set* para extraer y modificar los datos, comentarios en formato *Javadock*, inicialización y tipos de variables, etcétera. De esta forma tan simple, no necesitamos comprender la naturaleza del documento XML. Tan sólo necesitamos disponer de su esquema y compilar mediante el comando xjc. La relación entre unas clases java y otras van determinadas tras la compilación. La Ilustración 16 contiene el diagrama de clases correspondiente al resultado de la compilación del esquema GLF:

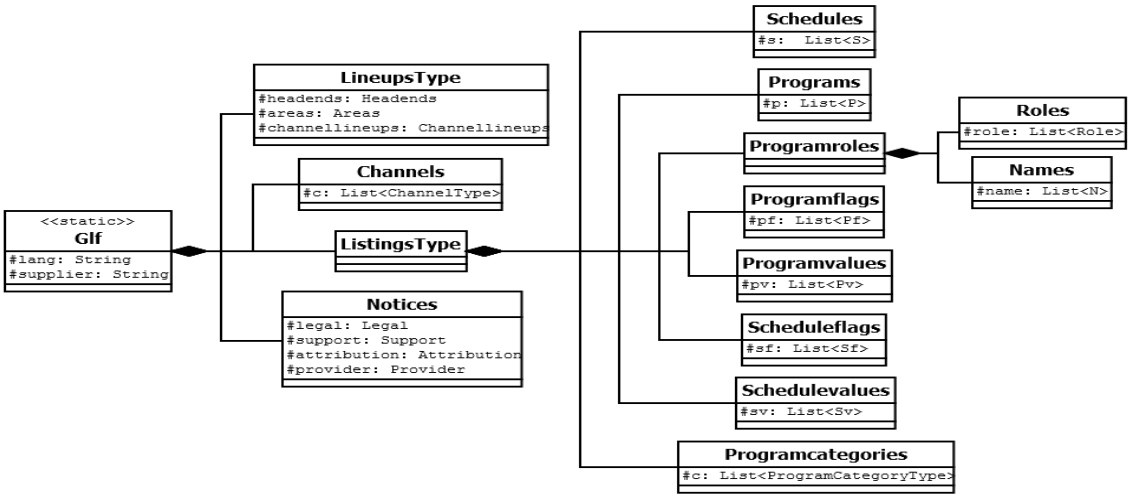


Ilustración 16: Diagrama de clases del objeto GLF

Toda esta estructura de archivos y de clases que genera el compilador de JAXB nos permite, de una forma muy sencilla, leer y crear documentos XML que atiendan a un determinado esquema. A estas operaciones se les denomina *marshal* y *unmarshal*. La primera de ellas se corresponde con la acción de codificar un objeto java en un documento XML. La segunda se corresponde con la acción contraria. Estos términos no solo se utilizan para el manejo de archivos XML, sino que también se aplica a otros formatos de similares características, como puede ser JSON.

En ocasiones se utiliza también el término "serializar" como sinónimo de *marshal*. Esto es muy común en aplicaciones distribuidas donde, de forma habitual, es necesario transmitir objetos en memoria utilizando una conexión de red. Para conseguirlo, se codifican estos objetos en un lenguaje intermedio, normalmente XML o JSON, para después ser transmitidos byte a byte por la red.

La Ilustración 17 muestra un esquema de los procesos de *marshalling* y *unmarshalling*:

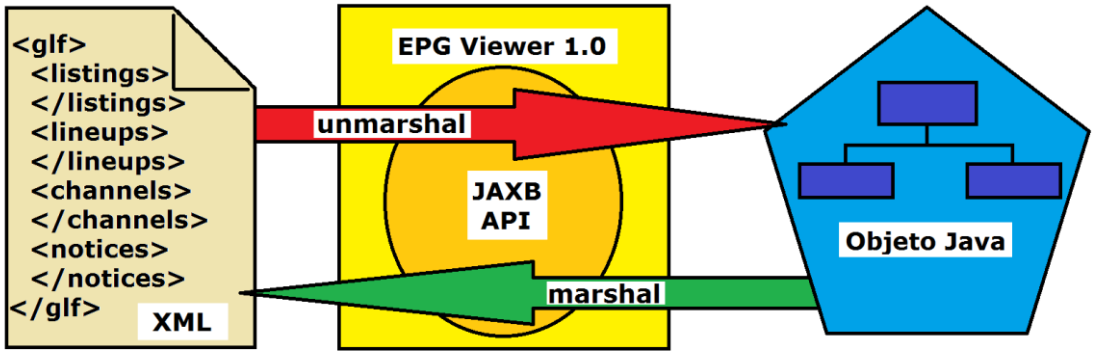


Ilustración 17: Marshal vs Unmarshal

Como podemos ver, el manejo del fichero XML mediante JAXB es relativamente sencillo. Sólo necesitamos utilizar los métodos `unmarshal` y `marshal`. Las siguientes líneas contienen el código necesario para realizar ambas **operaciones**:

Unmarshal

```
JAXBContext jaxbContext = JAXBContext.newInstance(Glf.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
Glf glfOBJ = (Glf) jaxbUnmarshaller.unmarshal(new File("ruta_del_XML_INPUT"));
```

Marshal

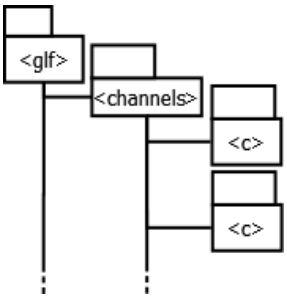
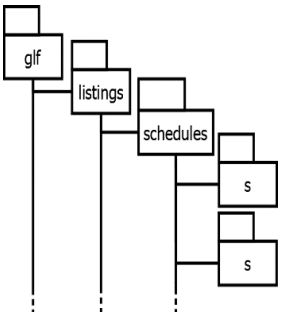
```
JAXBContext jaxbContext = JAXBContext.newInstance(Glf.class);
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
jaxbMarshaller.marshal(glfOBJ, new File("ruta_del_XML_OUTPUT"));
```

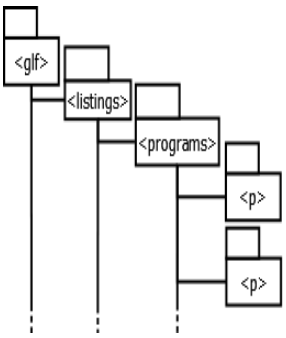
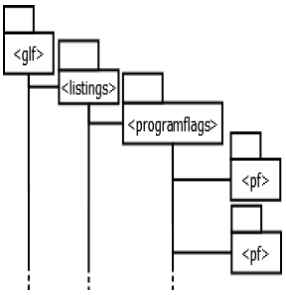
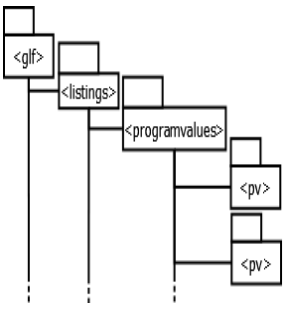
Como se puede apreciar, apenas es necesaria una línea de código para conseguir nuestro propósito de importar o exportar desde o hacia un documento XML. Resulta rápido y cómodo, y nos extrae del problema de manejar ficheros XML de grandes dimensiones.

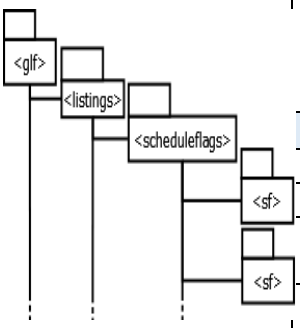
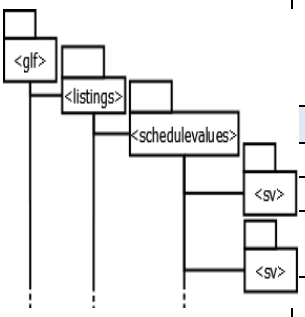
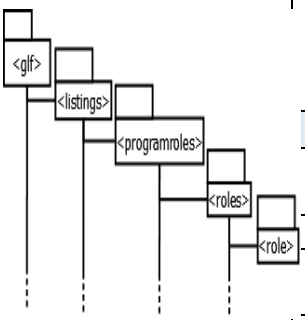
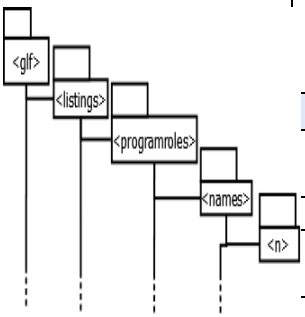
El siguiente paso en el desarrollo de EPG Viewer consiste centrarse en el desarrollo de sus funcionalidades. Estas se centran en la visualización y edición del objeto java obtenido mediante la operación de *unmarshal*.

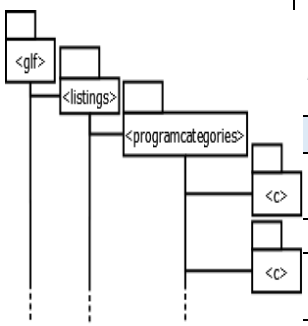
4.2.2 – Funcionalidades de EPG Viewer

Vamos a recoger todas las funcionalidades del software en una tabla. La información está organizada atendiendo al tipo de elemento que afectan, indicando su lugar en el documento XML:

<table>
<tr><th colspan="4">FUNCIONALIDADES DE EPG VIEWER 1.0</th></tr>
<tr><th>Elemento</th><th>Lugar en el XML</th><th colspan="2">Descripción</th></tr>
<tr><td rowspan="9">c</td><td rowspan="9"> 
</td><td colspan="2">La lista channels contiene la descripción de cada uno de los canales contenidos en el documento EPG. Cada elemento c se corresponde con un canal diferente</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos (de uno en uno o de forma múltiple)</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos del canal:
id, c, l, t, d, i, iso3166, iso639, ppv, a, b, cf, u, vbi, tz</td></tr>
<tr><td>Buscar</td><td>Permite buscar canales filtrando por los atributos id, c, ó l</td></tr>
<tr><td>Ver código</td><td>Muestra el código que generó el canal</td></tr>
<tr><td>Ver emisiones</td><td>Muestra las emisiones (elementos s de la lista schedules) asociadas al canal</td></tr>
<tr><td>Timeline</td><td>Muestra en una línea temporal el conjunto de todas las emisiones asociadas al canal en orden cronológico</td></tr>
<tr><td>Ver logo</td><td>Muestra el logo del canal (utilizando la URL contenida en el atributo i)</td></tr>
<tr><th>Elemento</th><th>Lugar en el XML</th><th colspan="2">Descripción</th></tr>
<tr><td rowspan="9">s</td><td rowspan="9"> 
</td><td colspan="2">La lista schedules contiene una entrada por cada emisión. Cada elemento de la lista referencia al canal y al programa asociado, junto con la duración y la fecha y hora de dicha emisión</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar las emisiones. Esta función se ejecuta cuando se elimina un canal</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos de la emisión:
d, p, c</td></tr>
<tr><td>Buscar</td><td>Permite buscar una emisión atendiendo a los siguientes criterios:
Canal asociado
Flag asociado
Value asociado
Duración
Programa asociado</td></tr>
<tr><td>Flags/Values</td><td>Permite añadir, eliminar y editar flags o values asociados a una emisión</td></tr>
<tr><td>Ver programa</td><td>Permite ver la descripción del programa asociado a la emisión</td></tr>
<tr><td>Ver código</td><td>Muestra el código que generó la emisión</td></tr>
</table>

<table>
<tr><th>Elemento</th><th>Lugar en el XML</th><th colspan="2">Descripción</th></tr>
<tr><td rowspan="10">p</td><td rowspan="10"> 
</td><td colspan="2">La lista programs contiene la descripción detallada de cada programa que se emite. Cada elemento de la lista es un programa diferente. Estos son referenciados desde la lista schedules mediante su identificador</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite añadir elementos nuevos</td></tr>
<tr><td>Eliminar</td><td>Esta función está disponible cuando utilizamos la herramienta de diagnóstico. Los programas “huerfanos”, que nunca son usados, pueden ser eliminados</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos del programa:
id, t, rt, et, d, rd, l</td></tr>
<tr><td>Buscar</td><td>Permite buscar programas utilizando el atributo t</td></tr>
<tr><td>Flags/Values</td><td>Permite añadir, eliminar y editar flags o values asociados a un programa</td></tr>
<tr><td>Names</td><td>Permite añadir, eliminar y editar names asociados a un programa</td></tr>
<tr><td>Categories</td><td>Permite añadir, eliminar y editar categorias asociados a un programa</td></tr>
<tr><td>Ver preview</td><td>Muestra una pre-visualización del programa. Esta opción se encuentra disponible cuando la descripción del programa contiene la URL de la imagen</td></tr>
<tr><td>Ver código</td><td>Muestra el código que generó el programa</td></tr>
<tr><td>Elemento</td><td>Lugar en el XML</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="5">pf</td><td rowspan="5"> 
</td><td colspan="2">La lista programflags contiene la declaración de los flags que pueden utilizarse como soporte para describir un programa. Estos son referenciados desde la lista programs mediante su identificador</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos:
id, name, lang, icon, pname, desc</td></tr>
<tr><td>Buscar</td><td>Permite filtrar elementos por el atributo name</td></tr>
<tr><td>Elemento</td><td>Lugar en el XML</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="5">pv</td><td rowspan="5"> 
</td><td colspan="2">La lista programvalues contiene la declaración de los values que pueden utilizarse como soporte para describir un programa. Estos son referenciados desde la lista programs mediante su identificador</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos en la lista</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos:
id, name, lang, icon, pname, desc</td></tr>
<tr><td>Buscar</td><td>Permite filtrar elementos por el atributo name</td></tr>
</table>

<table>
<tr><th>Elemento</th><th>Lugar en el XML</th><th colspan="2">Descripción</th></tr>
<tr><td rowspan="5">sf</td><td rowspan="5"> 
</td><td colspan="2">La lista scheduleflags contiene la declaración de los flags que pueden utilizarse como soporte para describir una emisión. Estos son referenciados desde la lista schedule mediante su identificador</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos: id, name, lang, icon, pname, desc</td></tr>
<tr><td></td><td></td><td>Buscar</td><td>Permite filtrar elementos por el atributo name</td></tr>
<tr><td>Elemento</td><td>Lugar en el XML</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="5">sv</td><td rowspan="5"> 
</td><td colspan="2">La lista schedulevalues contiene la declaración de los values que pueden utilizarse como soporte para describir una emisión. Estos son referenciados desde la lista schedule mediante su identificador</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos: id, name, lang, icon, pname, desc</td></tr>
<tr><td></td><td></td><td>Buscar</td><td>Permite filtrar elementos por el atributo name</td></tr>
<tr><td>Elemento</td><td>Lugar en el XML</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="5">role</td><td rowspan="5"> 
</td><td colspan="2">La lista roles contiene la declaración de los elementos role que pueden utilizarse como soporte para describir un programa. Estos son referenciados desde la lista programs mediante su identificador, el identificador del elemento n y opcionalmente, el orden que debe ocupar</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos en la lista</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos: id, title, desc</td></tr>
<tr><td></td><td></td><td>Buscar</td><td>Permite filtrar elementos por el atributo title</td></tr>
<tr><td>Elemento</td><td>Lugar en el XML</td><td colspan="2">Descripción</td></tr>
<tr><td rowspan="5">name</td><td rowspan="5"> 
</td><td colspan="2">La lista names contiene la declaración de los elementos n que pueden utilizarse como soporte para describir un programa. Estos son referenciados desde la lista programs mediante su identificador, el identificador del elemento role y opcionalmente, el orden que debe ocupar</td></tr>
<tr><td>Funcionalidad</td><td>Detalles</td></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos en la lista</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos: id, fname, mname, lname</td></tr>
<tr><td></td><td></td><td>Buscar</td><td>Permite filtrar elementos utilizando una concatenación de los atributos fname, mname y lname</td></tr>
</table>

<table>
<tr><th>Elemento</th><th>Lugar en el XML</th><th colspan="2">Descripción</th></tr>
<tr><td>c</td><td> 
</td><td colspan="2">La lista programcategories contiene la declaración de los elementos c que pueden utilizarse como soporte para describir un programa. Estos son referenciados desde la lista programs mediante su identificador. Es muy habitual anidar categorías
<table>
<tr><th>Funcionalidad</th><th>Detalles</th></tr>
<tr><td>Añadir</td><td>Permite crear elementos nuevos en la lista</td></tr>
<tr><td>Eliminar</td><td>Permite eliminar elementos</td></tr>
<tr><td>Editar</td><td>Permite editar los siguientes atributos: id, mscname, value</td></tr>
<tr><td>Buscar</td><td>Permite filtrar elementos utilizando el atributo mscname</td></tr>
</table></td></tr>
<tr><td colspan="4">FUNCIONALIDADES EXTRA</td></tr>
<tr><td rowspan="3">Menú de configuración</td><td>Idioma</td><td colspan="2">Permite seleccionar el idioma de preferencia del usuario. Idiomas disponibles:
Español
Inglés
Portugués</td></tr>
<tr><td>Ruta</td><td colspan="2">Permite seleccionar la ruta por defecto para la exportación de ficheros</td></tr>
<tr><td>Zona horaria</td><td colspan="2">Permite seleccionar la zona horaria. Esto permite mostrar las horas de las emisiones en la zona horaria deseada</td></tr>
<tr><td></td><td>Perfil de verificación</td><td colspan="2">Permite seleccionar el perfil de verificación de los campos deseado. Perfiles de verificación disponibles:
Mediaroom
Microsoft</td></tr>
<tr><td rowspan="2">Menú exportación</td><td>Pretty print</td><td colspan="2">Permite incluir (o no) saltos de líneas en el documento de salida</td></tr>
<tr><td>Exportar</td><td colspan="2">Exporta el objeto java a un documento XML, incluyendo las modificaciones que se hayan realizado</td></tr>
<tr><td rowspan="2">Actualizar emisiones</td><td>Offset</td><td colspan="2">Permite incluir un offset en las fechas de todas las emisiones</td></tr>
<tr><td>Actualizar fecha</td><td colspan="2">Actualiza las fechas de todas las emisiones sumando los días necesarios para que la primera emisión comience hoy. Toma la fecha y hora del sistema como referencia. Además, suma el valor del offset deseado</td></tr>
<tr><td rowspan="2">Info de EPG</td><td>Tamaños de las listas</td><td colspan="2">Muestra el número de elementos que tiene cada lista. Esto nos da una idea del tamaño del documento XML</td></tr>
<tr><td>Tamaño del fichero de salida</td><td colspan="2">Muestra el tamaño del fichero si se exportase. Hace diferencia entre el fichero con saltos de carro (Pretty print), y sin ellos</td></tr>
</table>

Diagnosticar EPG	Gaps & Overlapping	Encuentra saltos o solapamientos entre emisiones consecutivas de un mismo canal. Permite reparar
	Programas huérfanos	Localiza programas que no se utilizan. Permite eliminarlos para crear EPG ligeras
	Programas desaparecidos	Busca programas referenciados en la lista de *schedules* que no existen
	Canales desaparecidos	Busca canales referenciados en la lista de *schedules* que no existen
	Identificadores repetidos	Busca identificadores repetidos en las listas *channels* y *programs*
Añadir múltiples flags / values	Añadir *flag/value* a la lista *programs*	Permite añadir un mismo *flag* o *value* a toda la lista *programs*. Esto evita realizar tareas repetitivas
	Añadir *flag/value* a la lista *schedules*	Permite añadir un mismo *flag* o *value* a toda la lista *schedules*. Esto evita realizar tareas repetitivas
	Eliminar flag/value de la lista *programs*	Permite eliminar un mismo *flag* o *value* a toda la lista *programs*. Esto evita realizar tareas repetitivas
	Eliminar flag/value de la lista *schedules*	Permite eliminar un mismo *flag* o *value* a toda la lista *schedules*. Esto evita realizar tareas repetitivas
ABOUT	Versión	Muestra la fecha de la última compilación del software
	URL a la Wiki	Muestra la URL a la wiki (sólo disponible dentro de la Intranet de Telefónica I+D)

Capítulo 5: Manual de Usuario

5.1 – Introducción

Una de las partes más importantes de un proyecto, sea de la naturaleza que sea, es la documentación. Es prioritario generar una buena colección de diagramas, esquemas, capturas y todo tipo de explicaciones necesarias para hacer entender a personas ajenas el proyecto todo lo relacionado con él.

Este capítulo está dedicado al Manual de Usuario de EPG Viewer 1.0. Esta documentación fue generada de forma inicial en inglés, por requerimiento de la empresa. Ha sido traducido en español para formar parte de este Trabajo Fin de Grado.

Aunque existe una versión imprimible en PDF (similar a la que aquí se presenta), el manual fue generado con el objetivo de ser expuesto en la Wiki de Telefónica I+D. Esto es una práctica habitual en muchas empresas, ya que posibilita el desarrollo de una documentación de forma colaborativa y escalable. El apartado **5.3 – Documentación en la Wiki de TID** contiene algunas capturas relacionadas con la presentación real del Manual de Usuario de EPG Viewer 1.0

EPG Viewer 1.0

MANUAL SOFTWARE

Telefónica Investigación y Desarrollo

Tabla de contenido

INTRODUCCIÓN

EPG Viewer 1.0 es un software multiplataforma que simplifica la forma en que exploramos los archivos XML bajo el esquema GLF (*Global Listings Format*). Este tipo de archivo es conocido como EPG (*Electronic Program Guide*) y contiene, entre otra información, la información de la parrilla televisiva. El usuario final puede acceder a esta información usando el control remoto de su televisor, explorando el menú EPG. Pero no es todo tan simple. Los archivos EPG también contienen otra información relevante, no accesible al usuario final, pero determinante para su experiencia final cuando accede a contenidos audiovisuales digitales.

La intención de este manual no es describir la naturaleza de los archivos EPG ni su implicación en la experiencia de usuario cuando accede a contenidos digitales de televisión en directo. El objetivo de este manual es proveer de información estructurada relevante sobre cómo usar EPG Viewer 1.0 para explorar y editar el contenido de los archivos EPG de una forma elegante y simple, ahorrando tiempo y abstrayendo a los desarrolladores de la edición de archivos de texto XML enormes.

EPG VIEWER 1.0, CARACTERÍSTICAS PRINCIPALES:

- Explorar el contenido de archivos EPG.
- Editar canales, emisiones y programas.
- Añadir, editar o borrar *flags* / *values* / *names* / *categories*.
- Buscar errores y puntos de fallo.
- Testear URL a las imágenes (logos de canal, imágenes de programas, etc).
- Inserción intencionada de errores para analizar el comportamiento de diferentes sistemas.
- Actualizar las fechas de las emisiones de un archivo EPG antiguo para testeo.
- Encontrar *gaps* y *overlapping* en la lista de emisiones.

Requerimientos del sistema

EPG Viewer 1.0 funciona sobre la máquina virtual Java versión 8 (JVM, *Java Virtual Machine*). Se adapta al sistema operativo en donde se ejecuta, eliminando la necesidad de una configuración manual. No existen unos requerimientos mínimos muy marcados, pero si al menos unas recomendaciones definidas debajo de este epígrafe, para un funcionamiento correcto. De otra manera, puede experimentar un comportamiento lento o indebido del software.

Requerimientos mínimos recomendados:

- Intel Pentium Core Duo o un procesador equivalente de 2.33 GHz o superior.
- Al menos 1 Gb de memoria RAM. Recomendado 2 Gb.
- Al menos 150 Mb libres de espacio en el disco duro. Recomendado 500 Mb. Depende del tamaño del fichero de EPG en caso de querer exportar.
- Se recomienda conexión a Internet (para descargar logos de canales e imágenes de programa).

SOBRE EL ESQUEMA GLF

GLF (*Global Listing Format*) es un esquema definido por Simms Andrew M. y Scott III Samuel Thomas y registrado por Microsoft Corporation en el año 2011 (número de patente US 7913279 B2). A continuación se muestra un extracto del documento:

ABSTRACT

"An exemplary global listings format (GLF) is metadata for electronically transferring multimedia programming content and electronic program guide information. The GLF metadata specifies a self-referential data structure having a self-consistency mechanism comprising interlocking and crosslocking data elements. The self-consistency mechanism ensures completeness and validity of transferred programming data. In one implementation, the exemplary GLF is expressed in an extensible markup language (XML) schema definition (XSD) specification."

Esto significa que el esquema GLF especifica el contenido de los archivos XML, para contener la información de la guía de programación de forma estructurada. En la siguiente página mostraremos un diagrama de la estructura de un archivo XML bajo el esquema GLF.

Diagrama de la estructura GLF

El siguiente diagrama (Ilustración 18) representa la estructura de un archivo XML bajo la especificación GLF. Cabe destacar que el esquema es una representación a alto nivel. Algunos atributos no están detallados en el diagrama. Para más información, consulte otra documentación relacionada con la especificación GLF (no incluida en este manual).

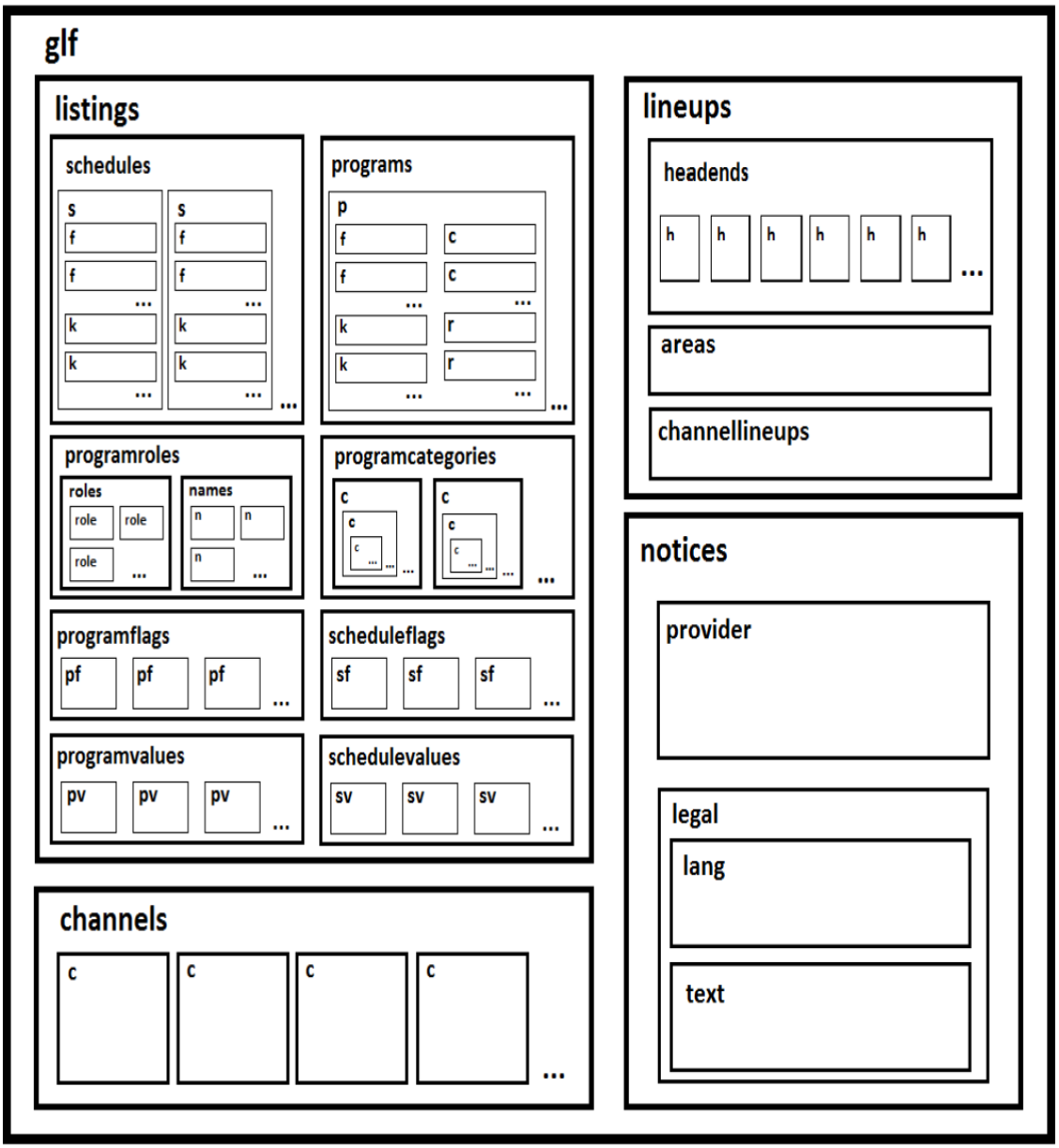


Ilustración 18: Esquema GLF

PRIMEROS PASOS

Cuando EPG Viewer 1.0 es lanzado la primera vez, se crea un archivo de configuración (`EPG_Viewer_config.ini`). Este archivo se genera con los siguientes valores por defecto:

- Idioma: 0 – Español.
- Ruta por defecto: "Ruta de instalación de EPG Viewer".
- Perfil de verificación: 0 – Mediaroom.
- Zona horaria: 12 – GMT.

Las siguientes líneas representan un ejemplo de este este archivo `EPG_Viewer_config.ini` de configuración con los valores por defecto:

```
lang=0
filepath=C:\Users\Public\Desktop
verificationprofileID=0
timezoneID=12
```

La próxima vez que EPG Viewer es lanzado, este archivo se utiliza para cargar las preferencias de uso. Si el usuario trata de manipular este archivo con valores o un esquema no válido, o si el archivo está corrupto, EPG Viewer regenera este archivo con los valores por defecto. Para más información, lea el apartado "**Especificación del archivo de configuración**".

1 – VENTANA PRINCIPAL

La Ventana Principal se muestra cuando EPG Viewer 1.0 es lanzado. Esta ventana incluye algunos botones rápidos para las principales funciones. Algunos de estos botones no están habilitados mientras no hay un archivo abierto.

Lo primero que hay que hacer es abrir un archivo XML válido. Hay dos formas de hacerlo:

- Siga la ruta "**Archivo >> Abrir**" en el menú superior (**¡Error! La autoreferencia al marcador no es válida.**).

- Click en el botón **Abrir** de la barra de herramientas (Ilustración 20).

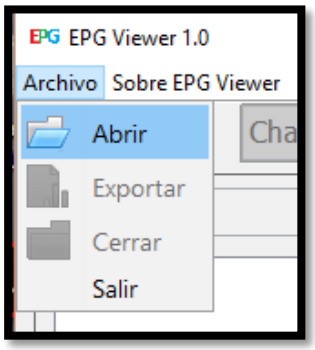


Ilustración 19: Abrir archivo (método 1)

Ilustración 20: Abrir archivo (método 2)

2 – Archivo abierto

Si abrimos un archivo válido (un archivo XML bajo el esquema GLF descrito en **"Sobre el esquema GLF**" y **"Diagrama de la estructura GLF"**), todas las tablas se rellenan automáticamente con información perteneciente al archivo. La ventana principal luce como en la Ilustración 21:

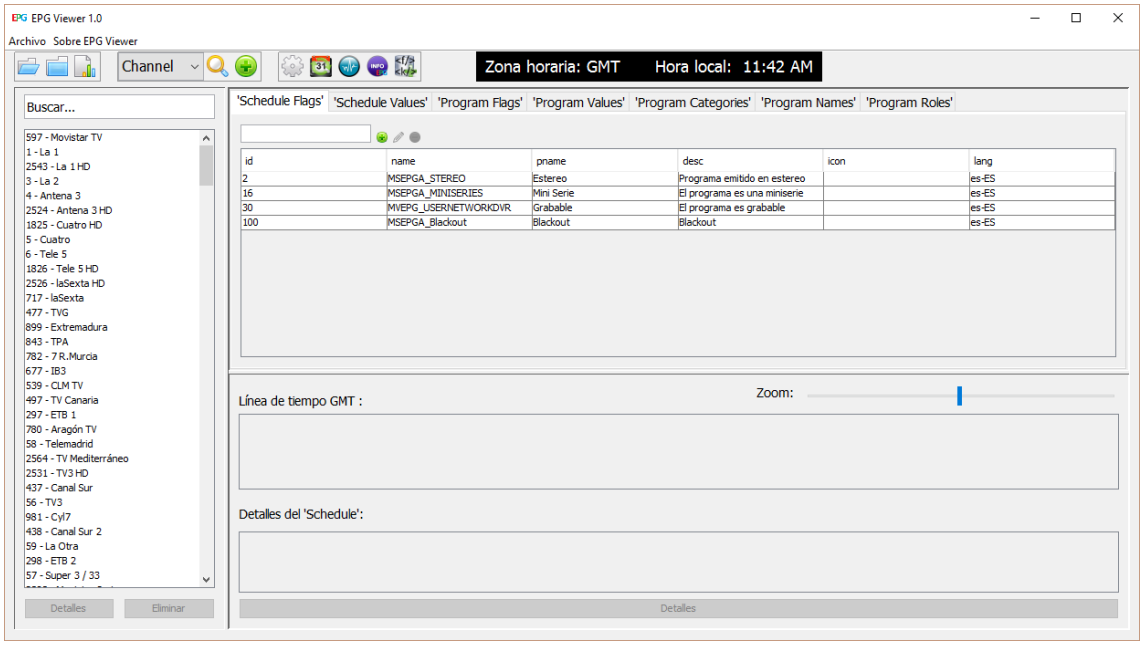


Ilustración 21: Ventana principal

- La parte izquierda de la ventana está dedicada a la lista de canales extraída del archivo XML bajo las etiquetas "**glf >> channels**". Cada canal se muestra siguiendo el siguiente patrón:

```
'id' + " - " + 'l'
```

Por ejemplo, el canal definido en el archivo XML como:

```
<c id="2" c="Movistar TV" l="Movistar+" iso3166="ES"
iso639="ES"/>
```

Se mostrará como:

2 - Movistar+

Hay una caja justo encima con la leyenda "**Buscar…**". Esta herramienta permite una búsqueda rápida de canales. Encuentra coincidencias entre el texto introducido en el campo de búsqueda y el atributo 'l' de cada canal. No es sensible a mayúsculas y minúsculas. Si no hay resultados, la lista de canales se muestra completa.

Hay dos botones debajo de la lista de canales:

- **Detalles**: este botón está habilitado cuando un único canal está seleccionado. Cuando presionamos el botón, una nueva ventana se abre con información acerca del canal.
- **Eliminar**: este botón está habilitado cuando al menos un canal esta seleccionado (admite selección múltiple). Cuando el botón es presionado, elimina los canales seleccionados junto con las emisiones asociadas a dichos canales. Antes de realizar la acción, una ventana pregunta al usuario para confirmar la ejecución antes de proceder. Cuando la acción finaliza, EPG Viewer muestra al usuario el resultado final de la operación. La Ilustración 22 muestra estas ventanas:

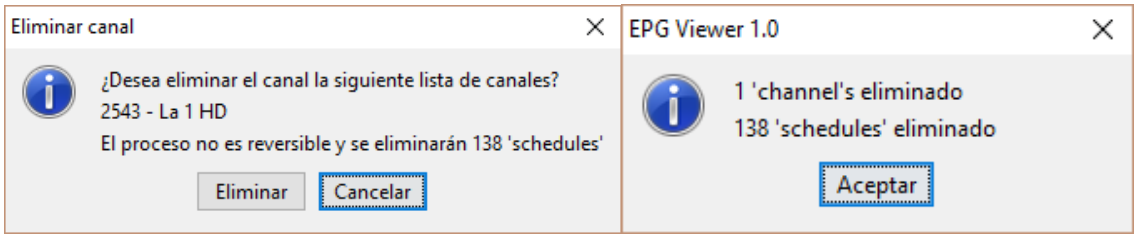


Ilustración 22: Eliminar canal

- La parte derecha de la ventana está dividida en tres partes:
  - Un panel con diferentes pestañas que contienen las siguientes listas:
    - **Schedule Flags** bajo las etiquetas "**glf >> listings >> scheduleflags**".
    - **Schedule Values** bajo las etiquetas "**glf >> listings >> schedulevalues**".
    - **Program Flags** bajo las etiquetas "**glf >> listings >> programflags**".
    - **Program Values** bajo las etiquetas "**glf >> listings >> programvalues**".
    - **Program Categories** bajo las etiquetas "**glf >> listings >> programcategories**".
    - **Program Names** bajo las etiquetas "**glf >> listings >> programroles >> names**".
    - **Program Roles** bajo las etiquetas "**glf >> listings >> programroles >> roles**".

    Para cambiar de una lista a otra, sólo necesitamos hacer click en la pestaña de la lista correspondiente. Cada pestaña se compone de los siguientes componentes:

    - Un **Cuadro de texto** para una búsqueda rápida del elemento deseado.
    - Un **Botón añadir** permite añadir un nuevo elemento a la lista seleccionada.
    - Un **Botón editar** permite editar un elemento de la lista (sólo disponible cuando un elemento de la lista está seleccionado)
    - Un **Botón eliminar** permite eliminar un elemento de la lista (sólo disponible cuando un elemento de la lista está seleccionado). Este botón lanza una ventana modal para preguntar por confirmación antes de eliminar
    - Una **Tabla** con la lista de elementos y los valores de sus atributos. Cada tabla se refresca cada vez que la ventana es seleccionada. Esto asegura al usuario disponer de la tabla siempre actualizada.

- Una **Línea temporal (Time line)** muestra todas las emisiones pertenecientes al canal seleccionado (cuando un único canal esta seleccionado). Las emisiones están ordenadas por fecha y el tamaño de la celda es proporcional a la duración de cada una. '*Gaps*' y '*Overlapping*' son ignorados. Dispone de un slider etiquetado como **Zoom** para modificar el tamaño de las celdas. Las emisiones pueden ser seleccionadas. Los detalles de estas se muestran debajo. La fecha de cada emisión se muestra encima de cada celda y usa la **Zona horaria (Time zone)** seleccionada en la configuración del programa. El usuario puede configurar esta zona horaria. Para información más detallada, vaya al apartado **"Configurar EPG Viewer 1.0"** de este manual.
- Los **Detalles de emisión** muestran los valores de los atributos cuando una emisión es seleccionada en la línea temporal. Se compone de una tabla con dos filas y cuatro columnas (una columna por cada atributo). La primera fila contiene los valores del correspondiente atributo. La segunda fila contiene la traducción a lenguaje natural de cada atributo.

- La parte de arriba de la ventana está reservada para las barras de herramientas y menús. Dispone de tres barras y dos menús:

- La barra **Archivo** contiene tres botones con las funciones básicas:
  - **Abrir** permite abrir un archivo XML. Si hay otro archivo abierto, y además ha sido modificado, pregunta por confirmación. En cualquier otro caso, cierra el archivo y abre el nuevo fichero.
  - **Cerrar** permite cerrar el actual XML (solo habilitado cuando hay un fichero abierto). En caso de que el fichero haya sido modificado, pregunta antes de cerrar.
  - **Exportar** permite exportar el actual fichero a un nuevo fichero. Todos los cambios efectuados serán incluidos en el nuevo archivo.
- La barra **Búsqueda** (sólo disponible cuando hay un fichero abierto) contiene los siguientes componentes:
  - Un **Combo** para seleccionar el elemento objeto de la búsqueda. Puede seleccionar entre 'Channel', 'Schedule' y 'Program'.
  - Un **Botón de búsqueda** para abrir la ventana de búsqueda correspondiente al elemento del combo seleccionado.
  - Un **Botón de añadir** para crear una nueva instancia del elemento seleccionado en el combo.
- La barra "**Herramientas EPG**" contiene las siguientes funciones:
  - **Botón de configuración:** Vaya a la sección "**Configurar EPG Viewer 1.0**" para obtener más información.
  - **Actualizar emisiones**: Vaya a la sección "**10 – Actualizar emisiones**" para obtener más información.
  - **Herramientas de diagnóstico**: Vaya a la sección "**Diagnosticar EPG**" para obtener más información.
  - **Añadir flags / values**: Vaya a la sección "**11 – Añadir *flags* / *values* a todas las emisiones**" para obtener más información.

- **Info EPG** : Vaya a la sección "**12 – Info de EPG**" para obtener más información.

- El menú **Archivo** contiene las mismas opciones que la barra Archivo descrita líneas más arriba. Dispone además de la opción Salir. Este botón finaliza la ejecución del EPG Viewer cuando es presionado. Si hay un archivo abierto que ha sido modificado, pregunta por confirmación antes de finalizar el software.
- El menú "**Sobre EPG Viewer**" abre en una nueva ventana la información referente al software (versión y fecha de la última compilación) y la URL de la Wiki (sólo disponible dentro de la intranet de Telefónica I+D).

3 – Ver detalles de canal

Seleccione un canal de la lista situada a la izquierda de la ventana. Cuando un canal es seleccionado, el botón Detalles se muestra habilitado. Para ver los detalles, presione en el botón o haga doble click sobre el canal. Una nueva ventana mostrará los detalles del canal seleccionado. Esta ventana tiene tres pestañas:

- **PESTAÑA 1, Detalles**: esta pestaña contiene la información detallada del canal. La ventana muestra esta pestaña por defecto cuando se abre. Podemos ver una captura de la pestaña en la Ilustración 23:

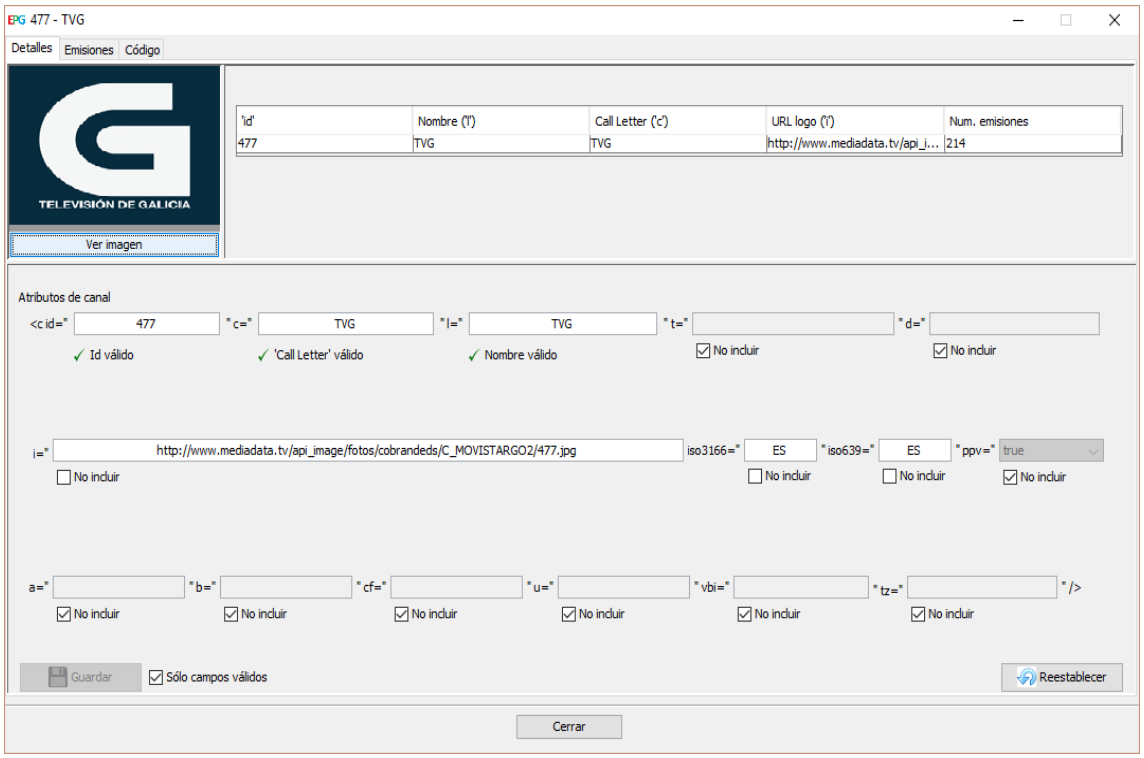


Ilustración 23: Ventana de detalles del canal

Esta pestaña tiene dos partes:

- La parte superior contiene:
  - La imagen del logo del canal. Esta imagen se descarga de internet utilizando el valor del atributo 'i'. Este atributo es opcional y contiene la URL del logo del canal. En caso de error se muestran las imágenes de la siguiente tabla (Ilustración 24):

• **"Not found 404". Se muestra cuando el servidor lanza el error 404. Esto ocurre cuando el recurso solicitado no ha sido encontrado.**	
• **"Time out". Se muestra cuando el servidor toma más de 500 ms para responder.**	
• **"No internet". Se muestra cuando no hay una conexion activa a Internet.**	
• **"No pic". Se muestra cuando no existe atributo 'i' o el atributo encontrado no contiene una URL válida.**	

Ilustración 24: Tabla imágenes alternativas al logo de canal

- Un botón debajo del logo del canal. Cuando se pulsa, abre en una nueva ventana el logo del canal junto a la URL (Ilustración 25):

Ilustración 25: Ventana logo de canal

Esta ventana contiene un botón junto a la URL. Cuando se presiona, la URL es copiada al portapapeles. La ventana se cierra pulsando el botón **Cerrar** o haciendo click fuera de la ventana (perdiendo el foco).

- La información resumida sobre el canal se muestra en una tabla con las siguientes columnas:
  - **'id'**: atributo "id" del canal.
  - **Nombre ('l')**: atributo "l" del canal.
  - **Call Letter ('c')**: atributo "c" del canal.
  - **URL Logo ('i')**: atributo "i" del canal.
  - **Num. emisiones**: muestra el número de emisiones asociadas a dicho canal.

- El resto de la pestaña muestra toda la información detallada acerca del canal. Todos los campos son editables. Para más información, vaya a "**7- Editar canal**".

  Hay tres componentes en la parte baja de la pestaña:

  - **Restablecer** : este botón reestablece todos los valores de los campos al último valor salvado.
  - **Guardar** : este botón está habilitado cuando al menos un campo ha sido modificado con un valor válido.
  - **Sólo campos válidos**: este campo habilita o deshabilita la herramienta de verificación. Esta herramienta aplica algunas limitaciones a ciertos campos, como exclusividad o limitación de caracteres. Por ejemplo, el campo de *Call Letter* debe ser exclusivo para cada canal. Si el usuario trata de introducir un valor de *Call Letter* existente en otro canal, deshabilita el botón Guardar para asegurar exclusividad.

- **PESTAÑA 2, Emisiones**: esta pestaña contiene la lista de emisiones asociadas al identificador del canal. Las emisiones están situadas en el fichero XML bajo las etiquetas "**glf >> listings >> schedules**". La Ilustración 26 muestra un ejemplo de esta pestaña:

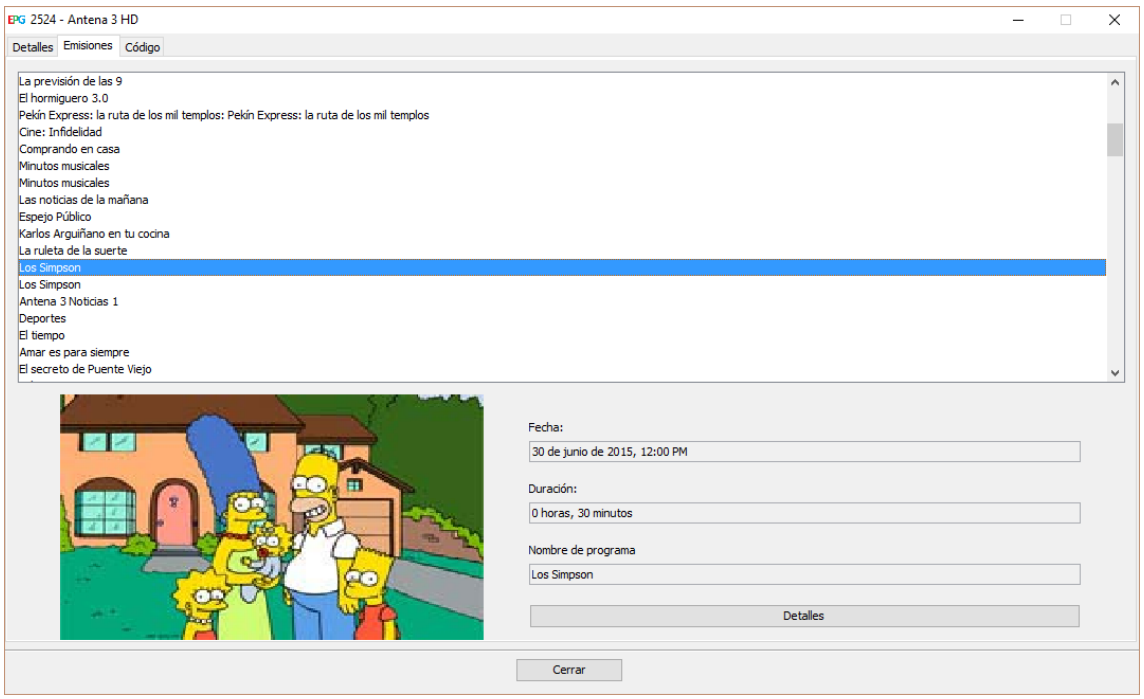


Ilustración 26: Pestaña 2, Emisiones

La lista muestra cada emisión siguiendo el siguiente patrón:

```
Título de programa (atributo 't' del programa asociado)
```

En caso de no encontrar el programa asociado, se muestra **'Programa' no encontrado**. Por ejemplo, la emisión definida en el XML como:

```
<s s="2015-08-06T13:00:00" d="3600" p="22" c="5"/>
```

Se mostrará como:

```
Moda reto: Episodio 30
```

Cuando un elemento de la lista es seleccionado, la parte de abajo de la pestaña se actualiza con la información asociada a la emisión. Tiene dos partes:

- Una **imagen** asociada al programa se descargar de Internet usando el valor del *value* llamado '**MSEPG_Image_Landscape**', normalmente con id 20. EPG Viewer encuentra el identificador de este *value* para extraer la URL de la imagen, utilizando la información del programa. En caso de no encontrar el programa, o que la información del programa sea incompleta, muestra una imagen vacía.
- Tres campos: **Fecha**, **Duración** y **Nombre del programa** usando los atributos 's', 'd' y 't', respectivamente.

El botón **Detalles** está habilitado cuando una emisión esta seleccionada. Cuando se pulsa este botón, una nueva ventana se muestra con la información de la emisión seleccionada. Para más información, vaya al apartado **4 – Ver detalles de emisión**.

- **PESTAÑA 3, Código**: esta pestaña muestra al usuario el código del archivo XML. Es muy útil cuando se necesita verificar el código que ha generado el canal en EPG Viewer. Esta pestaña no funciona cuando el canal ha sido creado por el usuario utilizando EPG Viewer. Cuando se presiona el botón **Buscar**, el área de texto inferior muestra el código de dicho canal extraído del fichero XML. Si el fichero XML no tiene saltos de línea válidos, no se puede mostrar el código. La Ilustración 27 muestra un ejemplo de esta pestaña:

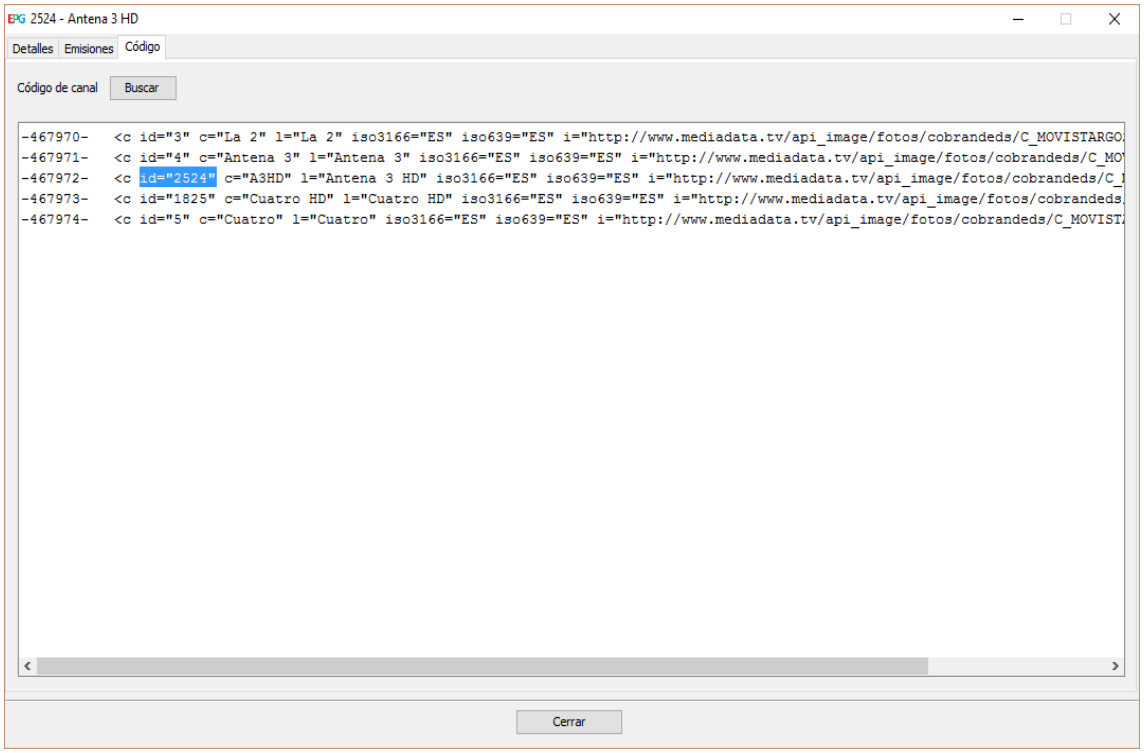


Ilustración 27: Pestana 3, Código

4 – VER DETALLES DE EMISIÓN

Desde la **PESTAÑA 2, Emisiones**, seleccione una emisión y pulse el botón Detalles (o doble click en la emisión). Una nueva ventana aparece como muestra la Ilustración 28:

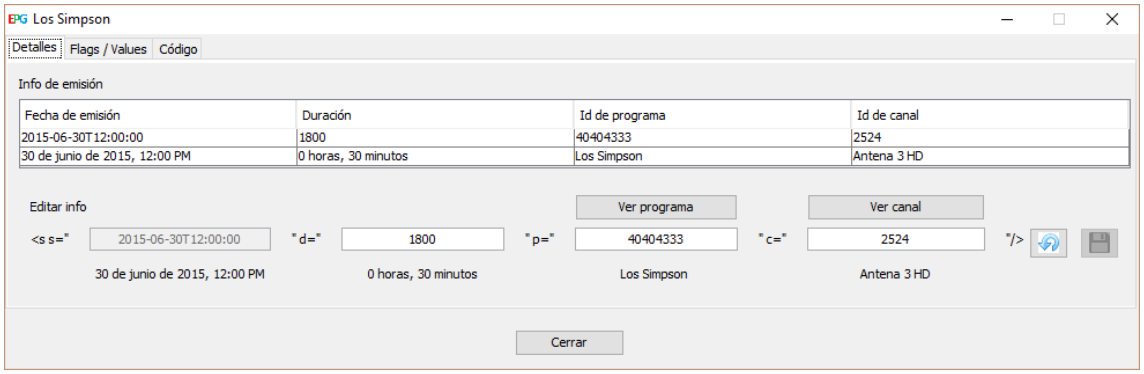


Ilustración 28: Ventana detalles de emisión

Esta ventana tiene tres pestañas:

- **PESTAÑA 1, Detalles**: esta pestaña se muestra por defecto cuando la ventana se abre. Tiene dos partes:

  - La parte superior muestra la información resumida:

    - **Fecha de emisión**, valor del atributo 's' de la emisión.
    - **Duración**, valor del atributo 'd' de la emisión.
    - **ID de programa**, valor del atributo 'p' de la emisión.
    - **ID de programa**, valor del atributo 'c' de la emisión.

    Esta tabla tiene dos filas. La primera de ellas muestra los valores de dichos atributos. La segunda muestra la traducción a lenguaje natural de esos atributos. En el caso de los atributos 'c' y 'p', se busca el nombre del canal y del programa respectivamente. Por ejemplo:

Info de emisión

Fecha de emisión	Duración	Id de programa	Id de canal
2015-06-30T12:00:00	1800	40404333	2524
30 de junio de 2015, 12:00 PM	0 horas, 30 minutos	Los Simpson	Antena 3 HD

Ilustración 29: Tabla detalles de emisión

Como podemos ver en el ejemplo superior (Ilustración 29), la primera fila contiene los valores del atributo tal y como aparecen en el fichero XML. La segunda fila contiene la fecha, la duración en horas y minutos, el título del programa y el nombre del canal.

- La parte inferior de la pestaña tiene los mismos atributos, pero en este caso editables por el usuario. Todos los campos son editables, excepto el atributo 's' perteneciente a la fecha. Para más información, vaya a "**8 – Editar emisión de este manual**".

  Hay cuatro botones en esta pestaña:

  - **Guardar**: este botón se muestra habilitado cuando al menos un campo ha sido modificado con un valor válido. Para salvar los cambios, pulse el botón y la ventana se refrescara con los nuevos valores.
  - **Reestablecer**: este botón reestablece todos los campos a el último valor salvado.
  - **Ver programa**: este botón abre en una nueva ventana los detalles del programa. En caso de no encontrar un programa asociado, el botón muestra la etiqueta "Nuevo programa" que permite crear un nuevo programa cuando es pulsado.
  - **Ver canal**: este botón abre en una nueva ventana los detalles del canal asociado a la emisión. En caso de no encontrar canal asociado, el botón muestra la etiqueta "Nuevo canal" que permite crear un nuevo canal.

- **PESTAÑA 2, Flags / Values**: Esta pestaña tiene dos partes (Ilustración 30):

  - La parte de la izquierda está reservada para los *flags*. Si hay *flags* asociados a la emisión, una tabla de dos columnas muestra la identificador del *flag* (atributo 'id') y el valor del atributo 'name' del *flag* asociado de la lista 'scheduleflags' ("**glf >> listings >> scheduleflags**"). En caso de no encontrar el *flag* asociado, se muestra "**'Flag no encontrado'**" en esta columna.

  - La parte derecha está reservada para los *values*. Si hay *values* asociados a esta emisión, se muestra una tabla de tres columnas con el identificador del *value* (atributo 'id'), el valor (atributo 'v'), y el nombre del *value* asociado al identificador (atributo 'name') de la lista '*schedulevalue*' ("**glf >> listings >> schedulevalues**"). En caso de no encontrar un *value* asociado, se muestra "**'Value' no encontrado**" en esta columna.

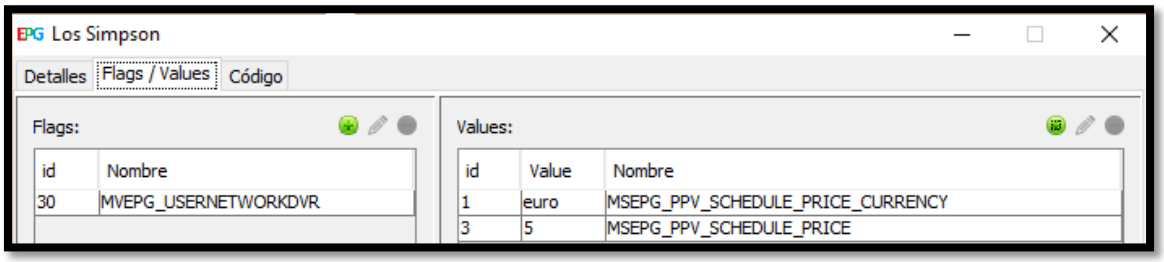


Ilustración 30: Pestaña 2, Flags / Values

El usuario puede añadir, editar, o eliminar elementos de estas listas (*flags* o *values*) usando las herramientas junto a las tablas. Los botones Editar y Eliminar están habilitados cuando un elemento de la tabla esta seleccionado. Se pueden añadir *flags* y/o *values* a todas las emisiones utilizando la herramienta descrita en el apartado "**11 – Añadir *flags* / *values***" a todas las emisiones de este manual.

- **PESTAÑA 3, Código**: esta pestaña muestra el código asociado a la emisión, extraído del archivo XML. Pulse el botón **Buscar**, junto al área de texto para iniciar la búsqueda.

5 – Ver detalles de programa

Desde la ventana Detalles de emisión (**PESTAÑA 1, Detalles)**, presione **Ver programa** para abrir en una nueva ventana los detalles del programa. La Ilustración 31 muestra un ejemplo de esta ventana:

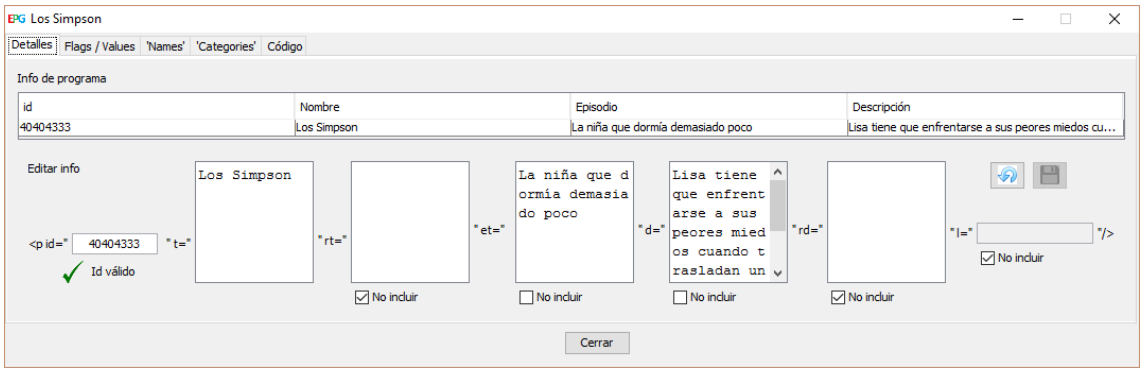


Ilustración 31: Ventana de detalles de programa

Esta ventana tiene cinco pestañas:

- **PESTAÑA 1, Detalles**: esta pestaña se muestra por defecto cuando la ventana se abre. Tiene dos partes:
    - La parte superior muestra la información resumida del programa. Tiene cuatro columnas:
        - **id**: contiene el valor del atributo 'id' del programa.
        - **Nombre**: contiene el valor del atributo 't' del programa.
        - **Episodio**: contiene el valor del atributo 'et' del programa.
        - **Descripción**: contiene el valor del atributo 'd' del programa.
    - La parte inferior de la pestaña se compone, como es habitual, de todos los atributos del programa. Estos son editables por el usuario. Para más información, vaya a "**9 – Editar programa**" de este manual. Hay dos botones en este área:
        - **Reestablecer**: este botón reestablece los campos a los últimos valores salvados.
        - **Guardar**: este botón se muestra habilitado cuando al menos un campo se ha modificado con un valor válido. Para salvar, pulsar el botón.

- **PESTAÑA 2, Flags / Values**: se divide a su vez en dos partes:
  - La parte izquierda de esta ventana contiene los *flags* asociados al programa. Contiene una tabla con dos columnas con el identificador del *flag* y el nombre del correspondiente *flag* asociado de la lista 'programflags' (extraidos de "**glf >> listings >> programflags**")
  - La parte derecha muestra una tabla similar de tres columnas con los *values* asociados al programa.

    Ambas tablas son editables mediante los botones  , similares en la ventana de **Detalles de emisión**, en la parte superior de cada tabla. Editar y eliminar sólo están disponibles cuando un elemento de la tabla esta seleccionado. Todos los campos son requeridos.

- **PESTAÑA 3, 'Names'**: esta pestaña se divide a su vez en dos partes (Ilustración 32):

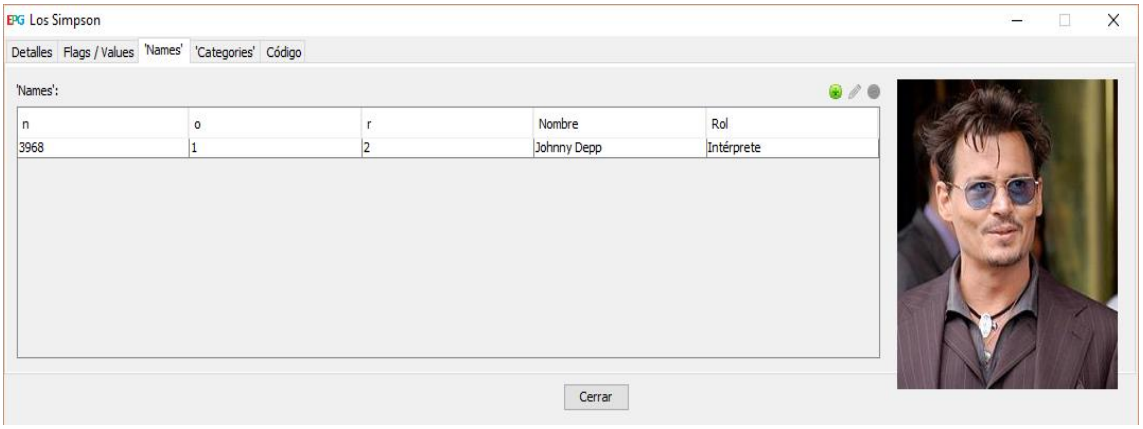


Ilustración 32: Pestaña 3, 'Names'

  - La parte de la izquierda contiene una tabla con cinco columnas:
    - **n** contiene el valor del atributo 'n' del nombre.
    - **n** contiene el valor del atributo 'o' del nombre.
    - **r** contiene el valor del atributo 'r' del nombre.
    - **Nombre** contiene la concatenación de los atributos 'fname', 'mname' y 'lname' del elemento n de la lista extraída de "**glf >> listings >> programroles >> names >> n**".
    - **Rol** contiene el valor del atributo 'title' del elemento correspondiente de la lista extraída de "**glf >> listings >> programroles >> roles >> r**".

    En caso de no encontrar *name* o *role* asociado a los identificadores, la leyenda "**No encontrado**" se muestra en estas columnas.

- La parte derecha muestra una foto del elemento seleccionado mediante una búsqueda en `wikipedia.org` usando el siguiente método:
  - EPG Viewer envía una petición al servidor utilizando "GET" a `https://en.wikipedia.org/wiki/` añadiendo al final de la URL la concatenación de los valores 'fname', 'mname' y 'lname' (sin espacios, sustituyendo barra baja '_' en su lugar)
  - En caso de éxito, se descargar la imagen de la Wikipedia.
  - En caso de error, se intenta de nuevo, esta vez utilizando `https://es.wikipedia.org/wiki/` (Wikipedia en español).

  Names y ***Roles*** pueden ser editados, añadidos o eliminados, utilizando los botones en la parte superior de la tabla. Los campos 'r' y 'n' son obligatorios. El campo 'o' es opcional.

- **PESTAÑA 4, 'Categories'**: esta pestaña contiene una tabla con tres columnas (Ilustración 33):

  - **'id'** contiene el valor del atributo 'id' de la categoría asociada al canal.
  - **'mscname'** contiene el valor del atributo 'mscname'' del elemento asociado a la categoria del programa de la lista "**glf >> listings >> programcategories**". Si no hay coincidencia, se muestra "**No encontrado**".
  - **'value'** contiene el valor del atributo *value* del elemento asociado a la categoría del programa de la lista "**glf >> listings >> programcategories**". Si no hay coincidencia, se muestra No encontrado.

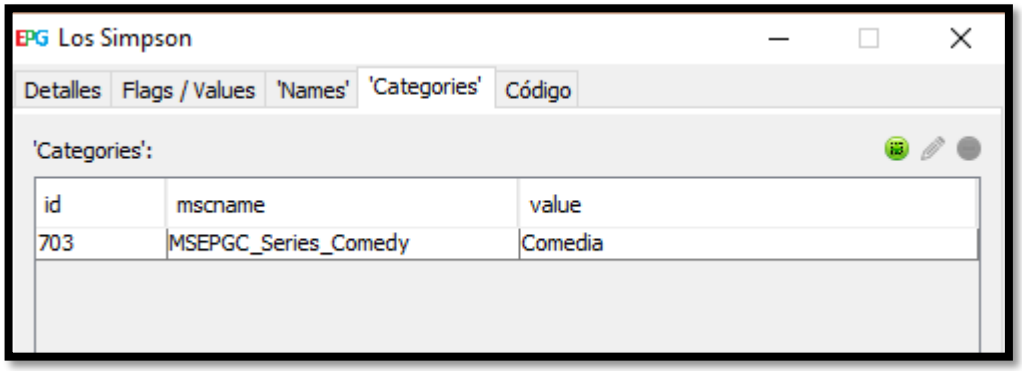


Ilustración 33: Pestaña 4, 'Categories':

El usuario puede editar, añadir, o eliminar categorías mediante los botones correspondientes en la parte superior de la tabla.

- **PESTAÑA 5, Código**: esta pestaña contiene un área de texto y un botón de búsqueda. Permite extraer el código que genero dicha descripción de programa del fichero XML. Presione el botón Buscar para comenzar la búsqueda.

6 – Herramienta de búsqueda

Desde la ventana principal, seleccione en el combo el objeto de la búsqueda y pulse el **Botón de Búsqueda** (Ilustración 34)

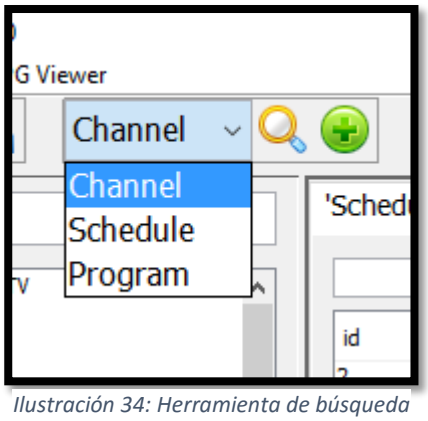


Ilustración 34: Herramienta de búsqueda

Dependiendo del objeto de la búsqueda, la nueva ventana que aparece es diferente:

- **Búsqueda de canal**: La búsqueda de canal ofrece tres modos (Ilustración 35):
    - **Búsqueda por 'id'**: seleccione la opción etiquetada con 'id' y rellene el campo correspondiente a esta opción. Sólo se admiten números en este campo. Cuando se introduce un valor válido, el botón de **Buscar** se muestra habilitado. Presione ENTER o click en el botón para empezar la búsqueda.
    - **Búsqueda por 'c'**: seleccione la opción etiquetada con "'c' (Call Letter)" y rellene el campo correspondiente a esta opción. Cuando se introduce un valor válido, el botón de Buscar se muestra habilitado. Presione ENTER o click en el botón para empezar la búsqueda.
    - **Búsqueda por 'l'**: seleccione la opción etiquetada con "'l' (name)" y rellene el campo correspondiente a esta opción. Cuando se introduce un valor válido, el botón de Buscar se muestra habilitado. Presione ENTER o click en el botón para empezar la búsqueda.

El botón **Ver todos** muestra todos los canales encontrados en el fichero EPG. Los resultados de cada búsqueda aparecen en la lista inferior. Cuando no ha resultados de la búsqueda, se muestra "**No hay resultados**" en la lista. Cuando un elemento de la lista es seleccionado, el botón **Detalles** se habilita. Presione en el botón o doble click en el elemento de la lista para abrir la ventana con los detalles del canal. La ventana de búsqueda permanece abierta aunque se abra la nueva ventana que muestra los detalles del canal. Ciérrela manualmente cuando haya terminado.

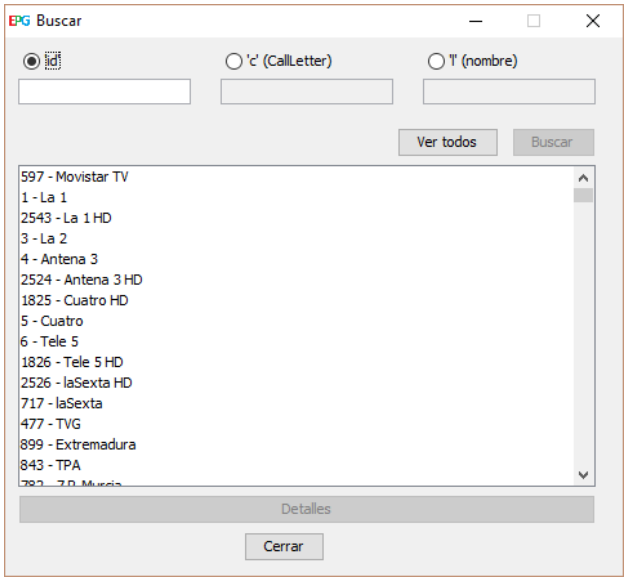


Ilustración 35: Buscar canal

- **Búsqueda de emisiones**: esta ventana se compone de cinco pestañas, diseñadas para distintos tipos de búsqueda:

  - **PESTAÑA 1, Canal**: esta pestaña permite hacer una búsqueda de emisiones asociados a un determinado canal. Para empezar la búsqueda, seleccione en el canal deseado y presione el botón **Mostrar emisiones**. La parte derecha de la pestaña muestra las emisiones asociadas al canal seleccionado. Hay dos campos encima de cada lista para buscar elementos de forma más rápida. Para mostrar los detalles de la emisión, seleccione una de la lista y presione el botón **Detalles** o doble click sobre el elemento. La Ilustración 36 muestra un ejemplo de esta pestaña:

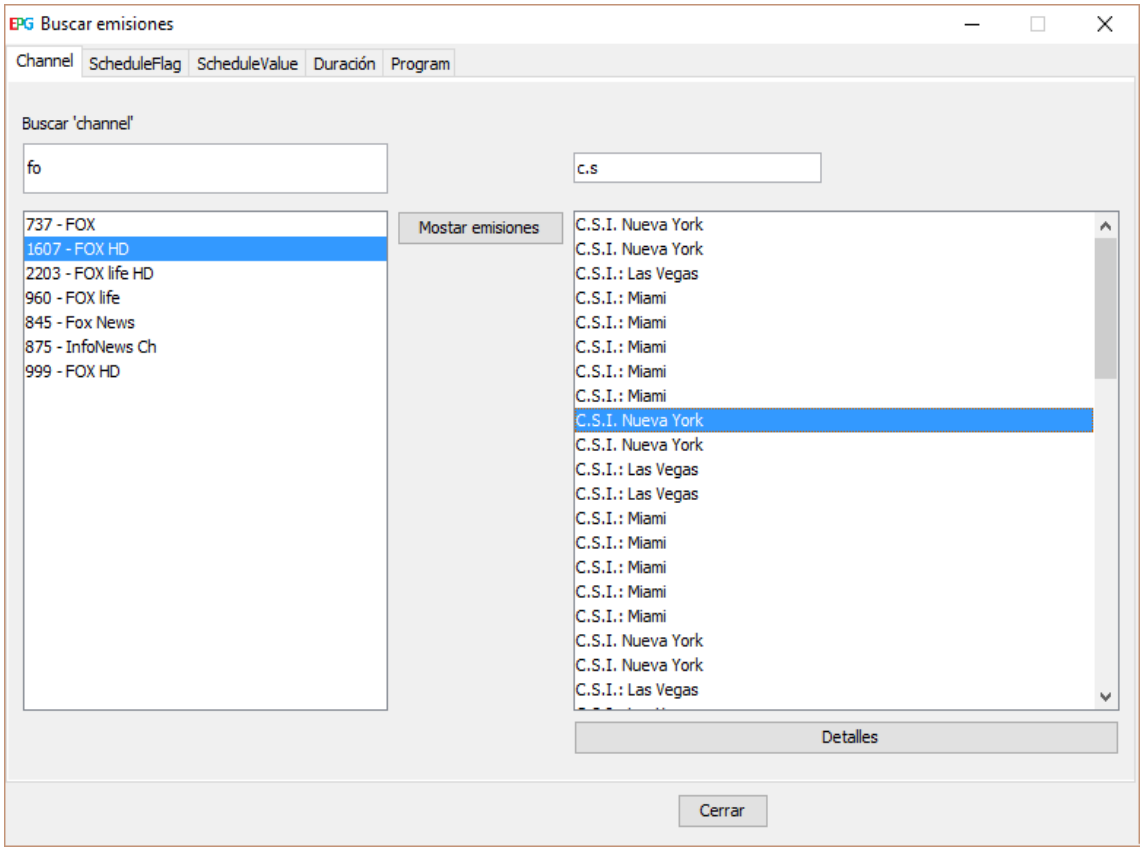


Ilustración 36: Buscar emisión (schedule)

  - **PESTAÑA 2 y PESTAÑA 3, Schedule *Flag* y Schedule *Value***: estas pestañas permiten hacer búsquedas asociadas a los *flags* o *values* que puedan llevar asociados las emisiones. Para empezar la búsqueda, seleccione un *flag* o un *value* de la lista de la izquierda y presione el botón **Mostrar emisiones**. Las emisiones que contengan dicho *flag* o *value* se mostraran en la lista de la derecha. En la parte superior de cada lista hay un campo de búsqueda para localizar elementos más rápidamente. Seleccione el elemento de la lista de la derecha, y haga doble click sobre él o presione sobre el botón **Detalles**. Una nueva ventana mostrará los detalles de la emisión seleccionada.

- **PESTAÑA 4, Duración**: esta pestaña permite filtrar emisiones por duración. Hay dos campos a rellenar antes de comenzar la búsqueda: **Duración min** y **Duración max**. Cada campo representa el máximo y el mínimo tiempo en minutos objeto de la búsqueda. Ambos límites están incluidos en la búsqueda. Cuando ambos campos están rellenos con valores válidos, el botón **Mostrar emisiones** aparece habilitado. Pulse el botón para iniciar la búsqueda. Los resultados se muestran en la parte derecha de la pestaña. Dispone de una herramienta de búsqueda encima de cada lista para una búsqueda rápida. Seleccione una emisión de la lista y pulse el botón **Detalles** o doble click en el elemento.
- **PESTAÑA 5, Programa**: esta pestaña ofrece la opción de buscar emisiones asociadas a un mismo programa. Esto ocurre cuando existen canales duplicados en calidades diferentes (SD y HD), o cuando hay reposiciones de programas a diferentes horas o en diferentes canales. Introduzca el nombre de programa en el cuadro de búsqueda de la izquierda (al menos tres letras) y presione ENTER o utilice el botón de búsqueda . Esta búsqueda no es sensible a mayúsculas y minúsculas. El resultado aparece en la lista inferior. Seleccione el programa de la lista y pulse en **Mostrar emisiones**. La lista de emisiones asociadas al programa seleccionado de la lista de la izquierda aparece en la lista de la derecha. Doble click en la emisión o pulse el botón **Detalles** para mostrar los detalles de la emisión.

- **Búsqueda de programas**: esta ventana ofrece la búsqueda de un programa determinado (Ilustración 37). Sólo tiene que rellenar el campo con el título del programa deseado (atributo 't' del programa) con al menos tres caracteres. La búsqueda no es sensible a mayúsculas y minúsculas. Presione en el botón de búsqueda (o ENTER) para iniciar la búsqueda. Los resultados se muestran en la lista inferior siguiendo el siguiente patrón:

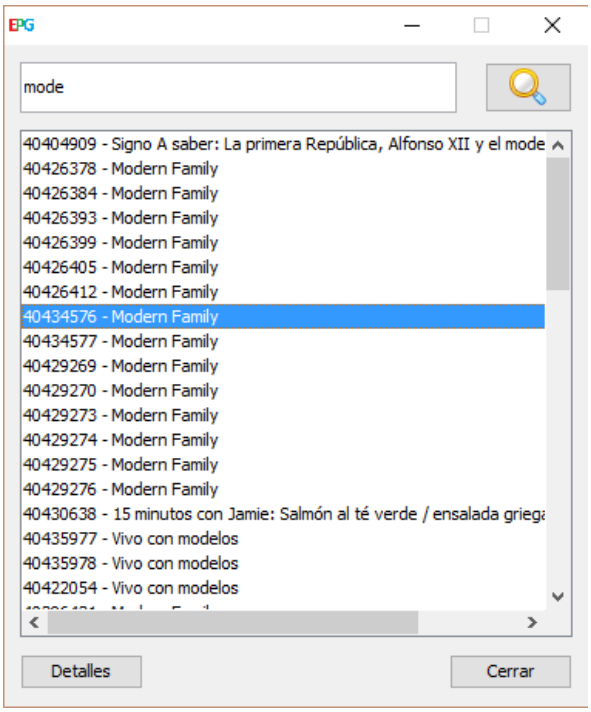


Ilustración 37: Buscar programa

```
Atributo 'id' + '-' +
    atributo 't'
```

Por ejemplo, un programa definido con 'id'="40399153" y t="Noticias 24H" se mostrará así:

```
40399153 - Noticias 24H
```

Seleccione un elemento de la lista y presione en el botón **Detalles**, o doble click sobre el elemento, para mostrar los detalles del programa en una nueva ventana.

7 – Editar canal

Para editar los atributos de un canal, siga primero los pasos descritos en el apartado "**3 – Ver detalles de canal**" para abrir la ventana con los detalles del canal. También puede utilizar la herramienta de búsqueda de canal descrita en el apartado "**6 – Herramienta de Búsqueda**". Luego siga los siguientes pasos:

1. Seleccione la **PESTAÑA 1, Detalles** de la ventana **Detalles de canal**.
2. Edite los campos debajo de la etiqueta Atributos de canal siguiendo las siguientes indicaciones:
   - Hay tres campos obligatorios: 'id', 'c' y 'l'.
   - El resto de los campos son opcionales. Pueden ser incluidos o excluidos mediante el *checkbox* etiquetado con la leyenda No incluir.
   - El usuario puede editar cualquier campo, incluyendo los requeridos ('id', 'c' y 'l').
   - Los campos 'id' y 'cf' han de ser números:
     - Enteros positivos.
     - Dentro del intervalo [1 , 9.223.372.036.854.775.807] (ambos extremos incluidos).
     - 'id' tiene que ser exclusivo de un canal (no puede haber dos canales con mismo id).
   - El campo 'b' tiene que ser un número entero (positivo o negativo, cualquier rango).
3. El botón **Guardar** aparece habilitado si todos los campos son válidos y se ha producido algún cambio con respecto a los últimos valores salvados. Si no es así, asegúrese de haber introducido parámetros válidos en todos los campos. Recuerde que EPG Viewer, bajo la especificación de GVP, tiene algunas limitaciones:
   - Los campos 'l', 'd', 'u' y 'vbi' tienen una longitud máxima de 150 caracteres (*).
   - El campo 'c' tiene una longitud máxima de 150 caracteres y no debe estar en uso por otro canal (*).
   - El campo 'i' tiene una longitud máxima de 150 caracteres y tiene que contener una URL con caracteres válidos (*).
   - Los campos 't', 'iso3166', 'iso639', 'tz' y 'a' tienen una longitud máxima de 50 caracteres (*).

   (*) Estas restricciones pueden ser ignoradas desactivando el *checkbox* rotulado con **Sólo campos válidos**.
4. Presione el botón **Guardar** para salvar los cambios.
5. La ventana se refresca con la información salvada de forma automática. Si ha modificado el campo 'i', el nuevo logo aparecerá pasados unos segundos.

8 – Editar emisión

Para editar los atributos de una emisión, siga los pasos descritos en el apartado "**4- Ver los detalles de emisión**" en este manual. También puede utilizar la herramienta de búsqueda descrita en el apartado "**6 – Herramienta de búsqueda**". Después siga los siguientes pasos:

1. Seleccione la **PESTAÑA 1, Detalles**, de la ventana Detalles de emisión.
2. Edite los campos situados debajo de la etiqueta Detalles de emisión siguiendo las siguientes indicaciones:

   - El campo 's' no puede ser modificado. Podría causar *overlapping* (solapamiento).
   - El campo 'd' representa la duración en segundos. Por lo tanto debe ser divisible entre 60, para asegurar minutos completos. Por ejemplo, 30 minutos equivale a 1800 en este atributo.
   - El campo 'd' tiene un valor máximo de 24 horas (86400 segundos)
   - El campo 'd' tiene un valor mínimo de un minuto (60 segundos)
   - **Si modifica el atributo 'd', puede causar un gap o solapamiento. Recuerde editar este campo con cuidado.**
   - Los atributos 'p' y 'c' deben ser enteros dentro del intervalo [1 , 9.223.372.036.854.775.807] (ambos límites incluidos).

   Cuando un atributo es modificado, la etiqueta bajo cada campo se actualiza con datos extraídos del fichero XML. Por ejemplo, si el usuario introduce un identificador de programa existente, EPG Viewer muestra el título del programa. Lo mismo ocurre con el identificador del canal.

3. Cuando al menos un atributo es modificado con datos válidos, el botón **Guardar** 💾. Si no es así, revise los pasos anteriores.
4. Presione el botón **Guardar** para salvar los cambios efectuados.

9 – Editar programa

Para editar los datos de un programa, siga las instrucciones del apartado "**5 – Ver detalles de programa**". También puede utilizar la búsqueda descrita en el apartado "**6 – Herramienta de búsqueda**". Después siga los siguientes pasos:

1. Seleccione la **PESTAÑA 1, Detalles** de la ventana **Detalles de programa**.
2. Edite los campos disponibles bajo la etiqueta Detalles de programa siguiendo las siguientes indicaciones:

   - Los campos 'id' y 't' son requeridos.
   - El resto de campos son opcionales, y pueden ser incluidos o excluidos activando o desactivando el *checkbox* correspondiente.
   - El campo 'id' es exclusivo. No puede ser asignado a otro programa.
   - El atributo 'id' ha de estar dentro del intervalo [1 , 9.223.372.036.854.775.807] (ambos límites incluidos).
   - El campo 'rd' tiene una longitud máxima de 500 o 1000 caracteres (*).

- El resto de los campos tienen una longitud de 150 o 1000 caracteres (*).

(*) El primer valor es para el perfil de verificación de Mediaroom. El segundo para el de Microsoft. El usuario puede cambiar este perfil de verificación en el menú de configuración, haciendo click en el botón **Configuración** en la ventana principal.

3. Cuando al menos un campo ha sido modificado con respecto a los últimos valores salvados, el botón **Guardar** aparece habilitado. Si no es así, revise los pasos anteriores.
4. Presione el botón **Guardar** para conservar los cambios.

10 – Actualizar emisiones

Desde la ventana principal, haga click en el icono **Actualizar fechas** (Ilustración 38). Una nueva ventana aparece con la configuración disponible (Ilustración 39).

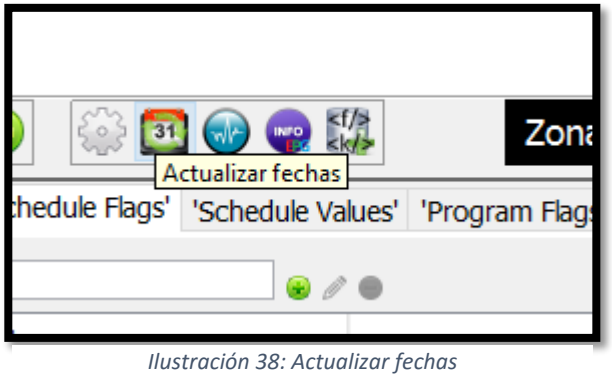


Ilustración 38: Actualizar fechas

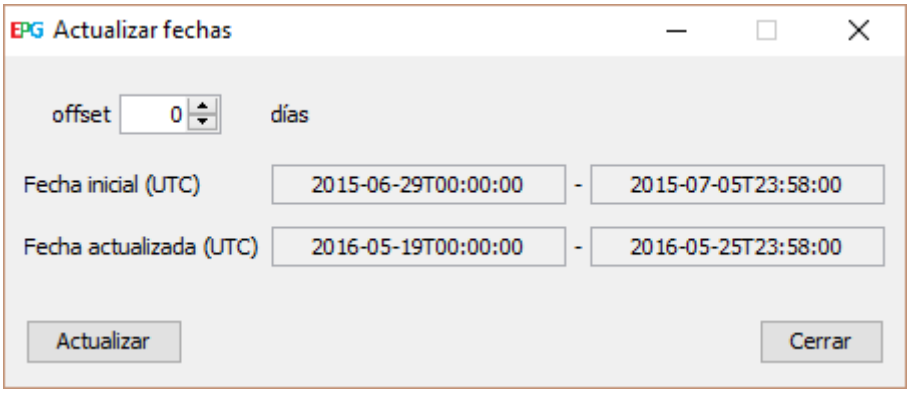


Ilustración 39: Ventana Actualizar emisiones

La ventana muestra la hora y fecha de la primera emisión y de la última. También muestra el resultado tras aplicar la actualización. EPG Viewer utiliza la fecha local del ordenador donde está ejecutando EPG Viewer para realizar el cálculo de la nueva fecha. Puede configurar un offset para mayor precisión. Las horas son UTC (GMT). Cuando se haya configurado la herramienta, presione el botón **Actualizar** para aplicar los cambios. Puede utilizar esta herramienta tantas veces como necesite.

De ahora en adelante, todas las emisiones quedan actualizadas con las nuevas fechas. Estos cambios solo afectan localmente. Recuerde exportar a un fichero XML para aplicar los cambios.

11 – AÑADIR *FLAGS / VALUES* A TODAS LAS EMISIONES

Desde la ventana principal, haga click en el icono **Añadir *flags / values*** a todas las emisiones (Ilustración 40). Una nueva ventana aparecerá con la configuración disponible que permite añadir un mismo *flag* o *value* a todas las emisiones (Ilustración 41).

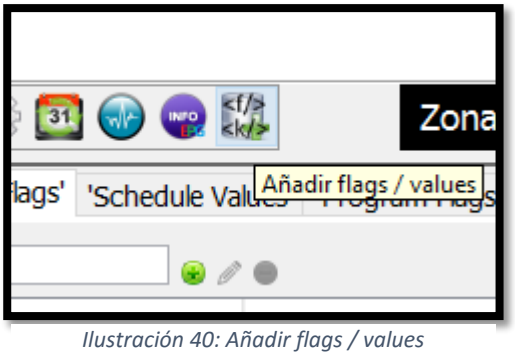


Ilustración 40: Añadir flags / values

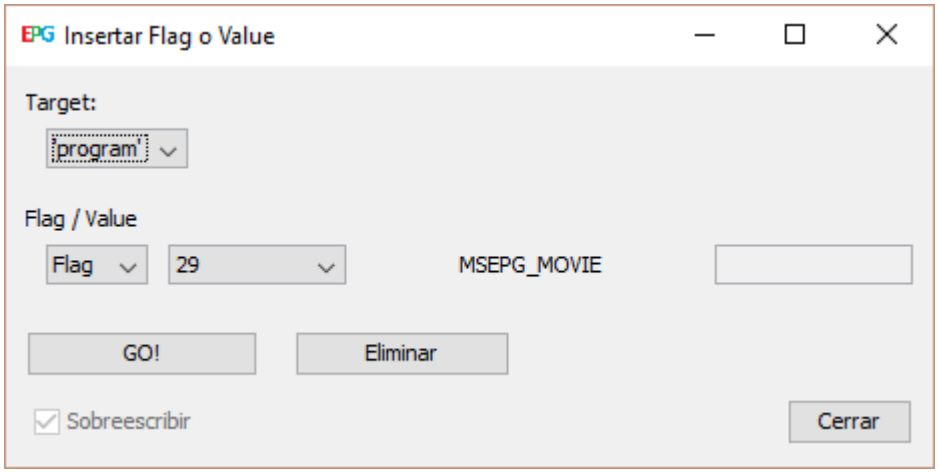


Ilustración 41: Ventana Insertar Flag o Value

Seleccione el elemento objetivo de la operación entre 'program' y 'schedules'. Seleccione el elemento deseado (*flag* o *value*) para añadir a la objetivo. Si ha seleccionado *value*, se habilitara un campo que permite introducir el valor deseado. Un combo muestra los *flags* o *values* disponibles. Cuando un elemento es seleccionado, la etiqueta se actualiza con el nombre del elemento.

Dos botones se habilitan cuando la operación está lista para empezar:

- **GO!**: presione el botón para añadir el elemento seleccionado a toda la lista de 'programs' o 'schedules'. Utilice el *checkbox* etiquetado con el texto **Sobrescribir** si desea sobrescribir los elementos ya existentes (esto no se aplica a *flags*). En caso contrario, los *values* existentes serán ignorados.
- **Eliminar**: este botón permite, del mismo modo, eliminar todos los elementos de la lista de una sola vez.

Cuando la operación ha finalizado, una ventana muestra el resultado de la operación.

12 – INFO DE EPG

Desde la ventana principal, haga click en el icono **Info** como muestra laIlustración 42. Una nueva ventana aparece con la información principal del fichero EPG abierto (Ilustración 43):

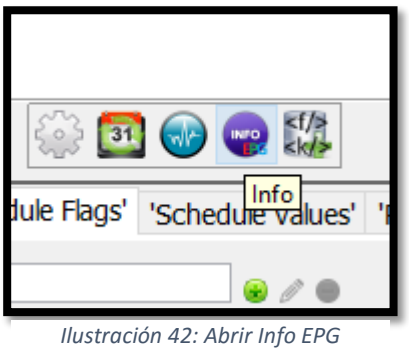


Ilustración 42: Abrir Info EPG

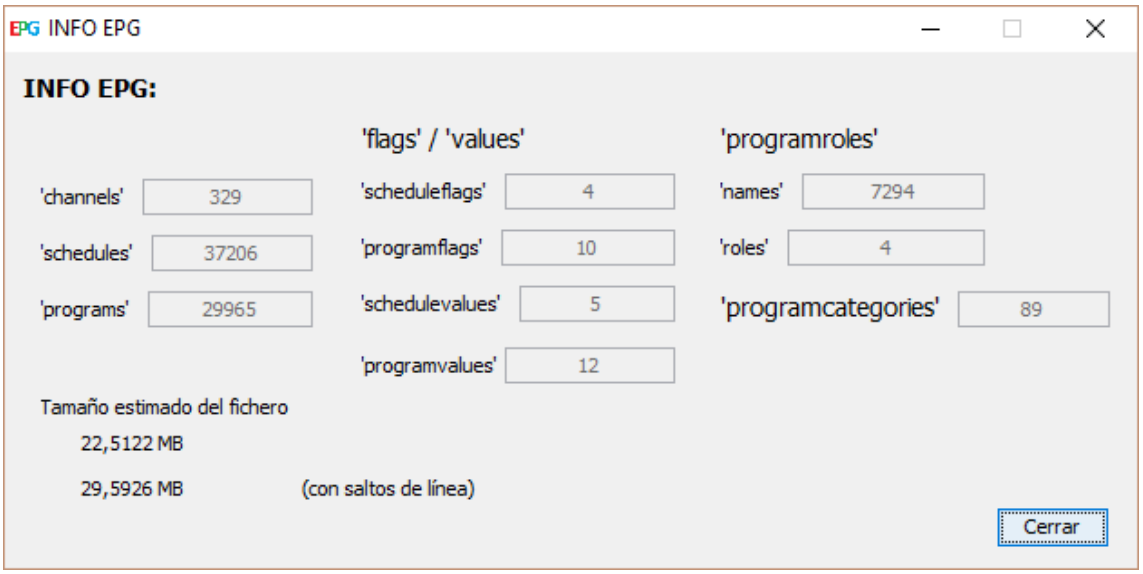


Ilustración 43: Ventana Info EPG

Esta ventana contiene el número de elementos de cada lista. En la parte inferior de la ventana se muestra un cálculo estimado del tamaño del fichero si se exportase (con y sin saltos de línea). Esto es muy útil cuando se desea crear un fichero EPG ligero. La información de esta ventana se actualiza cada vez que gana el foco (excepto la estimación del tamaño del fichero). Para volver a calcular la estimación, debe cerrar la ventana y volver a abrirla de nuevo.

Puede cerrar la ventana cuando termine pulsando el botón Cerrar.

DIAGNOSTICAR EPG

Desde la ventana principal, haga click en el icono **Diagnóstico de EPG** (Ilustración 44). Una nueva ventana aparece con las siguientes opciones disponibles (Ilustración 45):

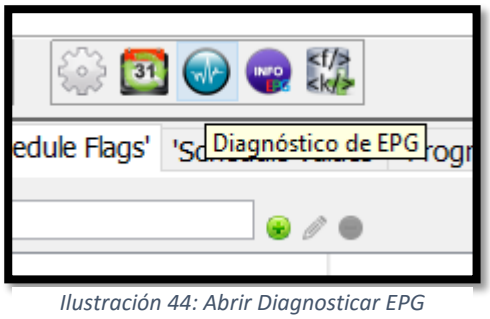


Ilustración 44: Abrir Diagnosticar EPG

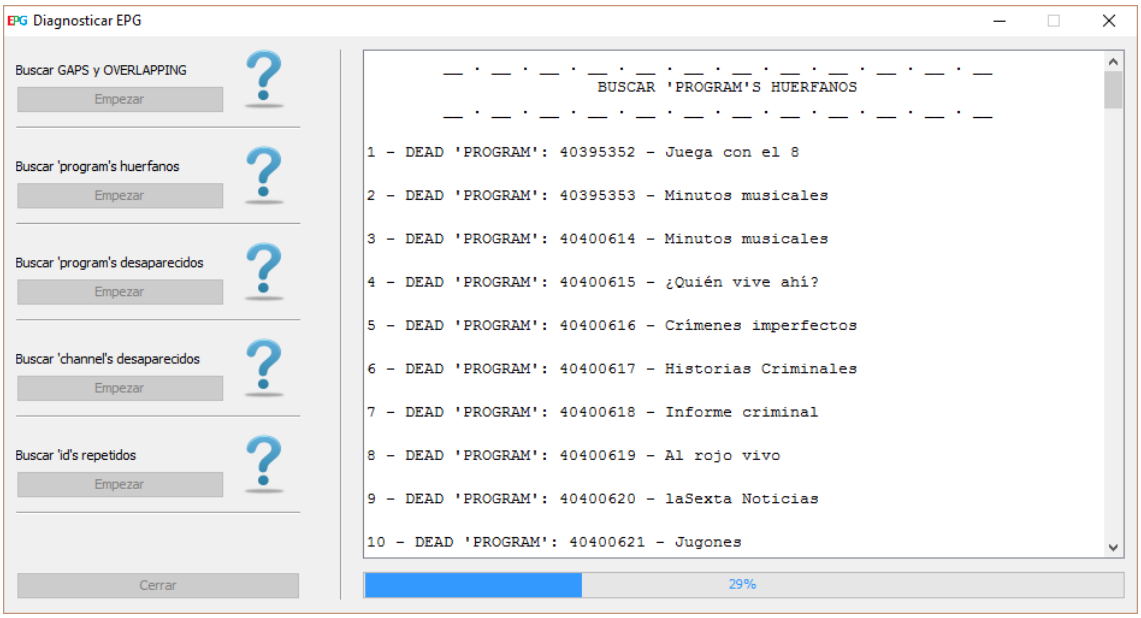


Ilustración 45: Ventana Diagnosticar EPG

- **Buscar GAP y OVERLAPPING**: esta opción revisa toda la lista de emisiones para detectar posibles saltos temporales (*gaps*) o solapamientos (*overlapping*). Los resultados se muestran en el área de texto. En la parte baja de la ventana puede seguir el progreso de la operación.

Cuando finaliza la operación, EPG Viewer puede reparar los gaps rellenando los huecos con emisiones y programas vacíos de contenido. Para ellos crea un único programa y una emisión por cada hueco reparado con el fin de crear un fichero EPG válido.

EPG Viewer también puede reparar los solapamientos (solamente recomendado para realizar test y pruebas). Para ello, reduce la duración para ajustarse en el tiempo con la emisión posterior.

- **Buscar *'programs'* muertos**: esta opción detecta programas que no están en uso (nunca son llamados desde la lista de emisiones). Estos programas provocan EPG pesadas y pueden eliminarse. Al finalizar la búsqueda, EPG Viewer ofrece la opción de eliminar los programas 'muertos' encontrados.

- **Buscar *'programs'* desaparecidos**: EPG Viewer busca en la lista de emisiones que las referencias a programas pertenecen a elementos de la lista 'programs'. Cuando la operación finaliza, se ofrece la opción de agregar los programas necesarios a la lista. Esto ayuda a crear archivos EPG válidos.

- **Buscar *'channels'* desaparecidos**: EPG Viewer busca en la lista de emisiones que las referencias a canales pertenecen a elementos de la lista 'channels'. Cuando la operación finaliza, se ofrece la opción de agregar los canales necesarios a la lista. Esto ayuda a crear archivos EPG válidos.

- **Búsqueda de *'ids'* repetidos**: EPG Viewer revisa la lista de 'programs' y de 'channels' para evitar que dos elementos tengan asociado el mismo identificador. El resultado de la búsqueda se muestra en el área de texto.

EXPORTAR A UN ARCHIVO

EPG Viewer 1.0 permite exportar en un solo click todos los cambios a un nuevo fichero. Sólo tiene que seguir los siguientes pasos:

1. Cuando haya terminado de editar el fichero EPG, presione el botón Exportar , que se encuentra en la barra de herramientas de la ventana principal. Una nueva ventana se muestra con las opciones de exportación (Ilustración 46):

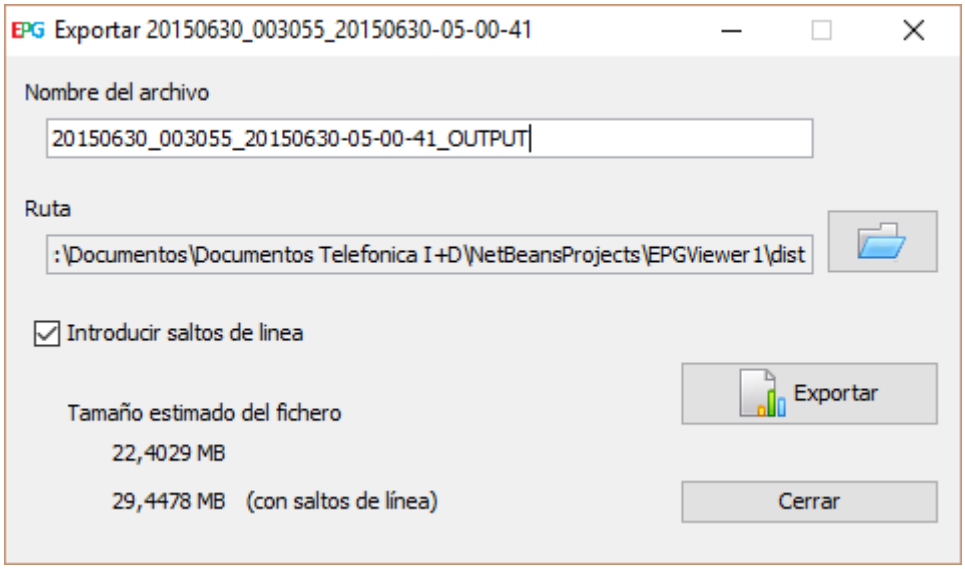


Ilustración 46: Ventana de exportación

Esta ventana contiene los siguientes valores por defecto:

- **Nombre del archivo** aparece relleno con el nombre del fichero XML abierto más "`_OUTPUT`" al final (sin extensión del fichero).
- **Ruta** utiliza la ruta por defecto configurada por el usuario. En caso de no haber definido una, utiliza la ruta de instalación de EPG Viewer. Puede ser editada desde el menú **Configuración**.
- *Pretty print* esta seleccionado por defecto.

Modifique los campos si es necesario. Recuerde que:

- El campo **Nombre del archivo** no puede estar vacío. Introduzca sólo el nombre, sin extensión.
- La **Ruta** de exportación puede ser editada pulsando el botón . Seleccione una ruta válida de su sistema de ficheros.
- **Introducir saltos de línea** introduce tabulaciones y saltos de línea y tabulaciones al fichero de salida. Si el usuario desea explorar el fichero de texto plano posteriormente, seleccione esta opción. Recuerde que el tamaño del fichero creado será mucho mayor con esta opción.

2. Si todos los campos son válidos, el botón **Exportar** aparece habilitado. Presione para exportar.

3. Si existe un fichero con el mismo nombre en la ruta seleccionada, EPG Viewer preguntara por confirmación antes de continuar. Si desea continuar, haga click en sobrescribir para empezar con la exportación.
4. Después de la exportación, EPG Viewer mostrara un mensaje para informar que la operación se ha llevado a cabo con éxito. El proceso de exportación puede tardar algunos segundos.

CONFIGURAR EPG VIEWER 1.0

En la ventana principal, haga click en el botón Configurar para abrir la ventana de **Configuración** (Ilustración 47). Esta ventana contiene todas las opciones configurables de EPG Viewer (Ilustración 48):

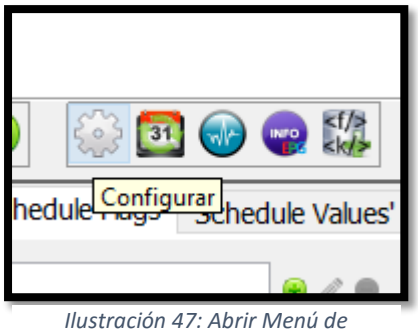


Ilustración 47: Abrir Menú de configuración

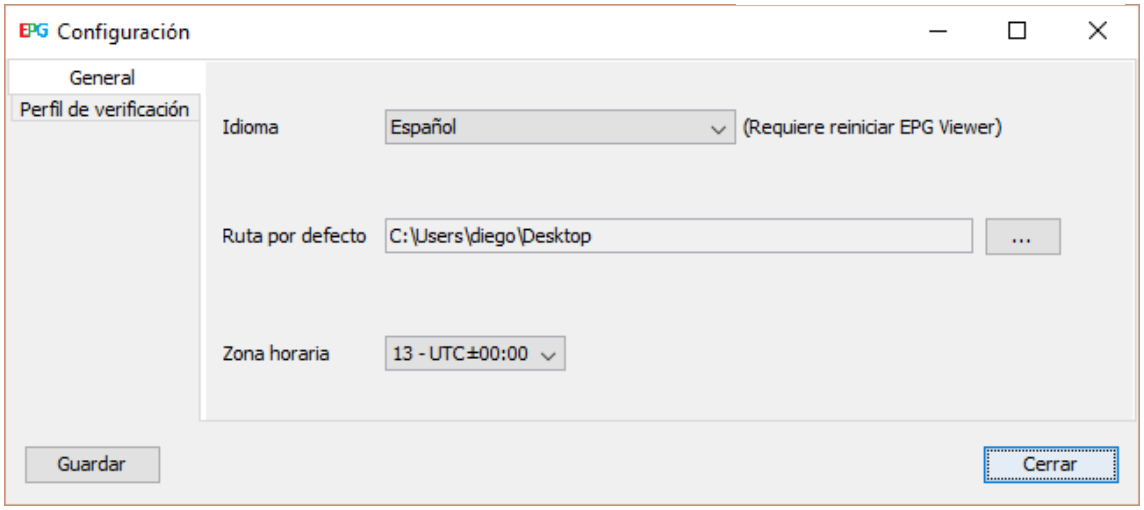


Ilustración 48: Ventana de Configuración de EPG Viewer

Esta ventana tiene dos pestañas:

- **PESTAÑA 1, General**: esta ventana ofrece tres parámetros configurables:
    - **Idioma**: seleccione uno de la lista. Los idiomas disponibles son español, inglés y portugués. Requiere reiniciar EPG Viewer para aplicar los cambios.
    - **Ruta por defecto**: utiliza por defecto la ruta donde se encuentra instalado EPG Viewer. El usuario puede configurar este parámetro como desee. Esta ruta se utilizará cuando desee exportar un fichero.

- **Zona horaria**: por defecto se utiliza GMT. Seleccione la zona que desee para explorar el fichero EPG con la hora correcta.

- **PESTAÑA 2, Perfil de verificación**: en esta pestaña puede seleccionar el perfil de verificación que desee.

Cuando presione el botón **Guardar**, la configuración será exportada a un fichero externo situado en la misma ubicación donde se encuentra instalado el software. Esto permite conservar la configuración deseada del usuario para futuras ocasiones. La próxima vez que EPG Viewer es lanzado, este fichero de configuración recupera su configuración preferida. En caso de manipulación de este fichero, EPG Viewer regenera la información con los parámetros por defecto. Vaya a Especificación del archivo de configuración para más detalles.

ESPECIFICACIÓN DEL ARCHIVO DE CONFIGURACIÓN

El archivo de configuración contiene las preferencias del usuario. Este fichero se regenera cada vez que el usuario realiza un cambio en la configuración del software o cuando se detecta que el archivo ha sido manipulado con valores no válidos. Este fichero se almacena en el mismo directorio donde se encuentra instalado EPG Viewer. No se recomienda manipular el contenido del mismo.

Las siguientes líneas representan un ejemplo del contenido de `EPG_Viewer_config.ini`:

```
lang=0
filepath=C:\Users\Public\Desktop
verificationprofileID=0
timezoneID=12
```

- **Idioma** (lang): contiene el identificador del idioma configurado por el usuario. Por defecto, este idioma es Español (ID = 0). Los idiomas disponibles son:

  - Español. ID = 0.
  - Inglés. ID = 1.
  - Portugués. ID = 2.

- **Ruta** (filepath): representa la ruta por defecto para exportación de ficheros. Por defecto, EPG Viewer 1.0 utiliza la ruta de instalación.

- **Perfil de verificación** (verificationprofileID): contiene el identificador del perfil de verificación seleccionado. Por defecto, EPG Viewer 1.0 utiliza Mediaroom (ID = 0). Perfiles de verificación disponibles:

  - Mediaroom. ID = 0.
  - Microsoft. ID = 1.

- **Zona horaria** (timezoneID): representa el identificador de la zona horaria seleccionada por el usuario. Por defecto, EPG Viewer 1.0 utiliza GMT (ID=12). El resto de zonas horarias e identificadores se recogen en la siguiente tabla (Ilustración 49):

TZ	ID	TZ	ID	TZ	ID	TZ	ID	TZ	ID
UTC - 12:00	0	UTC - 07:00	5	UTC - 02:00	10	UTC + 03:00	15	UTC + 08:00	20
UTC - 11:00	1	UTC - 06:00	6	UTC - 01:00	11	UTC + 04:00	16	UTC + 09:00	21
UTC - 10:00	2	UTC - 05:00	7	UTC ± 00:00 (GMT)	12	UTC + 05:00	17	UTC + 10:00	22
UTC - 09:00	3	UTC - 04:00	8	UTC + 01:00	13	UTC + 06:00	18	UTC + 11:00	23
UTC - 08:00	4	UTC - 03:00	9	UTC + 02:00	14	UTC + 07:00	19	UTC + 12:00	24

Ilustración 49: Tabla de Zonas horarias e identificadores

LIMITACIONES

EPG Viewer 1.0 tiene algunas restricciones de uso, que deberá tener en cuanta cuando maneje el software:

- Cuando utilice la herramienta de búsqueda ("**6 – Herramienta de búsqueda**") para encontrar programas, el campo de búsqueda tiene que contener al menos tres caracteres antes de poder iniciar la búsqueda. Esta limitación evita un número muy grande de resultados, pudiendo provocar que el software se ralentice.
- Cuando edite elementos, el identificador no puede ser cero (atributo 'id'). Java utiliza este valor para inicializar los identificadores de un nuevo elemento.
- No es posible crear nuevas emisiones.
- Las categorías pueden contener otras categorías. EPG Viewer explora hasta tres niveles de categorías. Si el usuario trata de abrir un fichero con más de tres niveles, sólo se mostrarán los niveles primero, segundo y tercero. Los demás serán ignorados.
- EPG Viewer no puede crear categorías anidadas. Todas las categorías que se creen serán del nivel primero.
- Del mismo modo, si se elimina una categoría del primer nivel que contenga otras categorías, estas también serán eliminadas.
- Cuando exporte a un fichero externo, el tamaño del puede diferir del original. Esto no sólo depende de las modificaciones que realice, sino de la tabulación que se aplique en cada caso. Para obtener el fichero más ligero posible, deseleccione la opción **Introducir saltos de línea** antes de exportar.

INFORMACIÓN DEL DOCUMENTO

Este manual ha sido desarrollado para Telefónica I+D por Diego Velayos San Juan durante el desarrollo de sus prácticas curriculares entre los meses de Agosto 2015 y Abril 2016. El idioma original del manual es inglés. Ha sido creada una versión del manual en español para incluir en la memoria del Trabajo Fin de Grado.

DATOS DE LA EMPRESA

Telefónica Investigación y Desarrollo

C/ Abraham Zacuto, 118-120, C.P. 47151

Parque Tecnológico de Boecillo, Valladolid

www.tid.es

Desarrollado por Diego Velayos San Juan

diego.velayossanjuan@telefonica.com

diegovsj@hotmail.com

5.3 – Documentación en la Wiki de TID

Como ya se adelantaba en el punto **5.1 – Introducción**, al inicio de este capítulo, la distribución de este Manual de Usuario se hizo utilizando un servicio de Wiki perteneciente a la Intranet de Telefónica I+D. Por este motivo, sólo se pueden mostrar algunas capturas de pantalla de dicho manual.

La **¡Error! No se encuentra el origen de la referencia.** muestra la página principal en la Wiki. Se compone de enlaces al manual, al área de descargas, el repositorio de código y una página que recoge futuras funciones que se iban necesitando y que estaban en desarrollo (Ilustración 50).

Ilustración 50: Página principal y de futuras funciones

La Ilustración 51 muestra una captura de la página del manual.

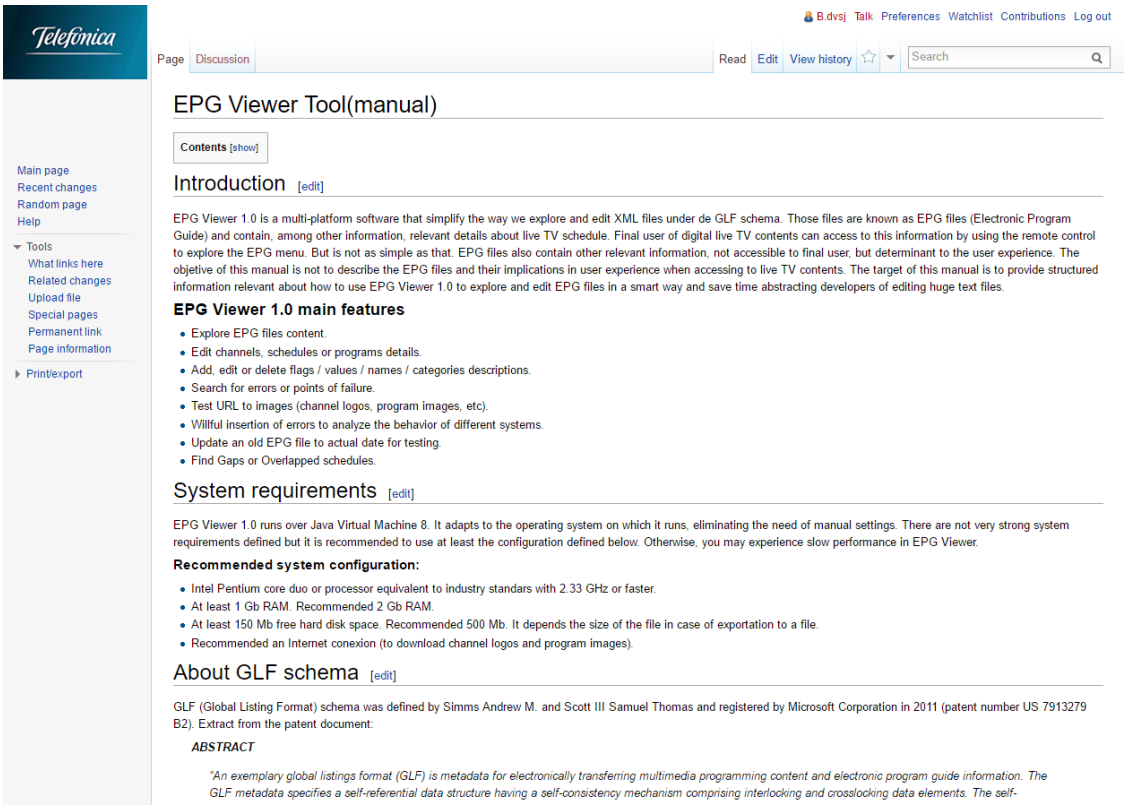


Ilustración 51: Captura de la página del manual

La Ilustración 52 muestra una captura de los apartados que contienen los enlaces a documentos y archivos externos y los enlaces de descarga del programa y manual de usuario.

Ilustración 52: Ddocumentación externa y Enlaces de descargas

Toda esta información está distribuida y gestionada mediante un software libre, programado en lenguaje PHP, conocido como MediaWiki.

Ilustración 53: Logo de MediaWiki

MediaWiki da vida a uno de los proyectos web más populares de Internet. Se trata de la Wikipedia. Esta enciclopedia digital contiene más de 37 millones de artículos en casi 300 idiomas. Este proyecto se sustenta en la colaboración de personas de todo el mundo, que de forma desinteresada, van creado y dando forma a los artículos de los que se compone. Ocupa los primeros puestos en el ranking de páginas webs más visitadas del mundo (sexto puesto en Mayo de 2016).

Tercera Parte

Capítulo 6: Presupuesto

6.1 – Introducción

A lo largo de este capítulo estimaremos el presupuesto económico para la realización de este proyecto. Se incluirán, de forma detallada, tanto los gastos referentes a software, como a hardware, así como las horas de trabajo dedicadas al desarrollo en cada una de las fases. Dado que en alguna de estas fases ha sido necesaria la dedicación de horas a la formación y aprendizaje de alguna tecnología concreta, necesaria para el desarrollo del proyecto, se estimará que parte de este coste asume la empresa, y que parte asume al cliente.

Lo mismo ocurre con el coste de las licencias de algunas herramientas informáticas utilizadas. Sería impensable incluir el total del coste de la licencia, dado que estas herramientas pueden ser reutilizadas para otros proyectos. Para estos casos, se incluirá un coste proporcional, basando la cantidad económica en el número de días que se ha utilizado dicho software, en comparación con el coste anual de la licencia.

Por otro lado, cabe destacar que existen diferentes soluciones software para el mismo problema. La elección de una u otra opción depende de diferentes criterios que dependen la empresa, del proyecto o de las necesidades del cliente. Estos criterios giran en torno a calidad de producto frente al precio final.

En este proyecto ha primado el uso de herramientas ya disponibles en la empresa (con licencias de pago ya abonadas) junto con otras gratuitas. Esto no ha sido un factor determinante en la calidad del producto final pues las herramientas gratuitas utilizadas han cubierto las necesidades del proyecto. Es el caso de NetBeans que, junto otras utilidades de pago, como el control de versiones de GitHub, han facilitado las tareas de codificación y depuración del software EPG Viewer.

Se han empleado un total de 4 meses de trabajo, sin incluir los meses previos dedicados a formación, pruebas e investigación en programación en Java. Para el cálculo de este presupuesto, se ha estimado una jornada de 5 horas de dedicación diaria, es decir, 25 horas semanales. Esto hace un total de 400 horas de trabajo repartidas a lo largo de 4 meses.

Algunas partidas del presupuesto se calculan en base a 4 meses de duración. Otras se calcularan en base a 80 días de trabajo.

A continuación se desglosa el presupuesto estimado asociado a este proyecto.

PRESUPUESTO ECONÓMICO (80 días de trabajo, 5 horas al día)		
Cuota de autónomo	267,04 €/mes 8.9 €/día * 80 días	**712.10 €**
Licencia Microsoft Word	100 € con amortización anual del 25% 25 €/año	**8.3 €**
PC con Windows 10	Asus F556U Series i7-6500U, 1TB HDD, 8 GB RAM, NVIDIA 2GB, W10 650 € con amortización anual del 25% 162.5 €/año	**54.17 €**
Control de versiones: **GitHub Enterprise**	(21 $ / month) 18.76 €/mes (10 usuarios)	**7.5 €**
Software libre	**NetBeans** **NotePad++** **paint.net** **innoSETUP** **MediaWiki** **Java 8 JDK**	**0 €**
Gastos de oficina	Alquiler Luz Internet Telefonía fija y móvil Climatización Limpieza	**600 €**
Formación en programación en Java	2 meses de formación y pruebas	**50 €**
Mano de obra: Programador Junior	13 €/hora * 400 horas	**5200 €**
TOTAL		**6623 €**

Ilustración 54: Tabla estimada del presupuesto económico

Algunas aclaraciones necesarias acerca del presupuesto:

- **Cuota de autónomo**: esta partida corresponde con la aportación mensual a la seguridad social en régimen de autónomos. La cantidad asciende en el año 2016 a 267.04 € al mes, con una base mínima de cotización de 893.10 €. Se ha calculado la parte proporcional a 80 días de trabajo cotizados.
- **Licencia Microsoft Word**: para elaborar la documentación ha sido necesario el uso de un procesador de textos. La licencia de Word se estima en 100 € para toda la vida.
- **PC con Windows 10**: para la realización del proyecto ha sido necesario el uso de un ordenador portátil. Se estima la amortización anual en un 25%. En el presupuesto se han considerado 4 meses de disposición del ordenador.
- **Licencia GitHub Enterprise**: la empresa disponía de licencias ya pagadas para este gestor y controlador de versiones. El coste de la licencia es de 21 $ mensuales, con 10 usuarios. Esta licencia se paga anualmente.

- **Software libre**: se han utilizado una serie de herramientas software de libre uso. No repercuten por tanto en el precio del presupuesto.
- **Gastos de oficina**: se han estimado estos gastos en 150 € al mes por el alquiler de una oficina pequeña en un vivero de empresas con gastos incluidos. El coste real es 300 €, pero se supone que se comparte la oficina con otra persona para dividir este gasto entre dos. Esta cantidad podría oscilar dependiendo de la ciudad.
- **Formación**: dado que se han invertido varias semanas de formación y pruebas para la realización del proyecto, este coste se refleja con una partida de 50 €. No se ha incluido el alquiler de equipo, oficina, ni demás medios técnicos usados en este tiempo.
- **Mano de obra**: esta partida es la más cuantiosa del presupuesto. Representa el número de horas dedicadas con un coste por hora estimado de 13 € como programador Junior.

El presupuesto tiene un coste total de unos **6600 euros**.

El beneficio neto se corresponde con la partida de "Mano de obra", aunque habría que tener en cuenta el coste de los traslados a la oficina. Se puede cifrar en unos 5000 euros, que dividido entre 4 meses, nos da un total de **1250 euros/mes**.

Capítulo 7: Conclusiones y líneas futuras

7.1 – Conclusiones

En este capítulo vamos a realizar una reflexión final sobre lo que ha supuesto la realización de este Trabajo Fin de Grado a diferentes niveles.

A nivel profesional, la experiencia ha sido muy positiva. Considero que una de las mejores formas de llevar a cabo un proyecto de estas características es hacerlo en el ámbito más profesional posible. Esto es, diseñar y orientar el TFG a las necesidades concretas de una empresa.

En mi caso, y como me encontraba realizando unas prácticas en Telefónica I+D, se me ofreció la posibilidad de enfocar mi proyecto hacia una necesidad concreta de la empresa, consensuada con mi tutor de prácticas por parte de la empresa. Se me ofreció libertad casi total para organizar, diseñar y desarrollar el software EPG Viewer. Esto fue una gran ventaja, dado que todas las horas que pasé en la empresa fueron dedicadas en exclusiva a mi proyecto, pero al mismo tiempo suponen un beneficio propio para la empresa.

Por otro lado, y dentro de la dinámica diaria de la empresa, se realizaban con cierta regularidad reuniones informativas, cursos, ponencias y dinámicas grupales que hicieron mi estancia allí mucho más agradable. Estas actividades eran de temática tan variada que incluso en alguna ocasión, sirvieron de gran ayuda para el desarrollo de mi idea.

También he de agradecer la ayuda y apoyo prestado por algunas personas que conocí durante el desarrollo de estas dinámicas.

No obstante, a nivel técnico, la experiencia no fue sencilla. El proyecto se basaba en una tecnología de la que poco conocía. Nunca antes había programado un software de tal complejidad. Durante las primeras semanas, la mayor parte del tiempo lo dediqué a realizar tutoriales de programación orientada a objetos, y principios básicos de Java. Realicé muchas pruebas para capturar archivo XML en un objeto de Java, con el objetivo de manejarlo con más comodidad.

Dado que desconocía la utilidad XJC de Java, que permite crear las clases Java necesarias para capturar en un objeto el contenido de un fichero XML (en base a su esquema XSD), dedique los primeros meses a realizar esta tarea de forma manual. Utilizando una librería de Java que permite leer este tipo de ficheros, fui desmenuzando todas las listas XML hasta conseguir mi objetivo. Este modo de operar no fue acertado, dado que consumía una gran cantidad de los recursos del ordenador, y no cumplía su objetivo al 100%. Producía errores frecuentemente.

Lo mismo ocurre con NetBeans y su gestión de ventanas. Fue todo un descubrimiento utilizar este entorno de programación, que no sólo facilita enormemente las tareas del programador, sino que también ofrece herramientas para la gestión del entorno gráfico (mediante la librería JFrame), importación de librerías, control de versiones, depuración y detección de errores y puntos de fallo, etc.

También he aprendido a utilizar y compartir el código mediante GitHub. Al principio del proyecto esta herramienta resultaba un lastre, dado que no había formado parte nunca de mi metodología de trabajo. Pero con el tiempo fui aprendiendo a explotar el potencial de un control de versiones. Además, muchas empresas utilizan este u otro tipo de plataforma para compartir

y colaborar en proyectos, y creo que me mi experiencia con GitHub va a servir de gran ayuda para mí en el futuro.

El manejo de la documentación en la empresa es también una cuestión relevante. Cuando se trabaja en proyectos de forma conjunta, organizados en equipos, es importante llevar un seguimiento de las aportaciones de cada miembro. Cada uno genera documentación referente a sus avances, que ha de estar disponible para todos los demás integrantes. De esta forma se evita duplicar esfuerzos, y una mejor comunicación. Esto se lleva a cabo de diferentes formas. En el caso particular de Telefónica, este seguimiento se realiza mediante una herramienta de gestión de proyectos, llamada Jira, y MediaWiki para compartir la documentación. Todo esto integrado a la vez con GitHub. De esta forma, todo el proyecto se distribuye de forma organizada.

Al principio, todo este entramado parecía un tanto complejo, debido al uso de diversas herramientas, con fines tan diferentes. Pero con el tiempo resultó muy cómodo llevar el control y el avance de los objetivos del proyecto. La mayoría de empresas implementan sistemas similares, que de una u otra forma, integran todas las facetas que abarca un proyecto. Creo que en el futuro, el conocimiento de este "*modus operandi*" va a serme muy útil.

No puedo dejar pasar esta oportunidad sin mencionar los conocimientos adquiridos en programación orientada a objetos, utilizando Java como lenguaje. Para mi supone un primer paso importante en mi carrera. El mundo de la programación está ligado prácticamente a cualquier área científico-técnica de conocimiento, y es importante aprender, en una u otra medida, a manejarse de forma habitual con diversos lenguajes de programación. Por ello, creo que uno de puntos más positivos que puedo extraer es, el haber aprendido a programar de forma autónoma y autodidacta, enfrentándome a un problema real. Esta situación ha sido en ocasiones incómoda, por tener que lidiar con un problema real, tratando de dar una solución factible. Para ello he tenido que ojear cientos de páginas webs, con manuales, cursos, y foros de debate, hasta dar con la solución que mejor se ajustaba con mi necesidad.

A nivel personal, la experiencia ha sido muy enriquecedora. El desarrollo de una aplicación software desde cero ha sido un reto. Tomar parte en todas y cada una de las fases, hasta llegar al producto final, ha sido muy instructivo. Me ha servido para darme cuenta de la gran importancia de trabajar en equipo. Esto permite compartir el conocimiento, llegando a la mejor solución de forma conjunta, creando proyectos que integren diversas tecnologías. También me ha servido para aprender a documentar de forma correcta, compartiendo todo la información relevante con el resto de personas del equipo. En ocasiones esto ha de hacerse utilizando un idioma de conveniencia, generalmente en inglés, cuando las personas que forman el equipo son de diferente nacionalidad. Esto es algo frecuente en empresas grandes, más aún, en empresas que se dedican a temas de ingeniería, dónde la documentación de componentes se distribuye en este idioma.

Una de las conclusiones más útiles que puedo sacar para futuros proyectos ha sido el cómo organizar un proyecto. Cuando te enfrentas a un problema relativamente grande, es importante realizar un diseño previo, documentado mediante esquemas y diagramas. Esto facilita y orienta la fase de desarrollo y codificación del software. En mi caso particular, y por falta de conocimiento del lenguaje Java, estos esquemas que se realizaron al inicio no fueron de gran utilidad, y tuvieron que ir adaptándose sobre la marcha. Esto ha provocado cierta confusión en algunos momentos.

Para futuros proyectos he de considerar, mucho más en serio, el diseño previo a la codificación. Esto facilitará la localización de errores, depuración, y asegurará un futuro más próspero a la aplicación. Los diseños complicados suponen problemas futuros de soporte y mantenimiento, ya que no tiene una estructura tan clara y resulta complicado cambiar algo de una funcionalidad, sin que afecte al comportamiento de otras funcionalidades.

7.2 – LÍNEAS FUTURAS

Como ya se ha comentado anteriormente, la aplicación software que propone este TFG responde a una necesidad puntual de la empresa. Las funcionalidades se han ido adaptando y diseñando a medida, para adaptarse a las metodologías de trabajo de los miembros del grupo.

Ocurre de forma casi constante que, las necesidades que motivaron un cierto proyecto se transforman, mutan, o incluso desaparecen para dar lugar a otras nuevas. Esto desemboca en nuevos proyectos o en evoluciones de los existentes para abarcar un uso más diverso.

EPG Viewer en su versión 1.0 abarca el ámbito personal. Puede ser utilizado para editar archivos de EPG de forma local en un ordenador. La finalidad de este software es servir de herramienta a los desarrolladores de Telefónica I+D para facilitar la edición de un determinado tipo de archivo. Existen posibles mejoras para futuras versiones que podrían ser desarrollados como continuación de este proyecto. Algunos de las posibles líneas futuras se enumeran a continuación:

Añadir emisiones (*schedules*)

Una posible mejora seria la posibilidad de crear y añadir elementos ‘s’ a la lista ‘schedules’. Esta funcionalidad no se contemplaba, debido a que podría provocar EPG no válidas. Resulta complejo diseñar una herramienta que contemple y limite esta función, para evitar la inclusión de errores. Del mismo modo, podría aprovecharse la ocasión para desbloquear la edición del atributo ‘s’ de las emisiones, que se corresponde con el campo de fecha y hora de la emisión. El manejo en Java de las fechas (tanto como para añadir emisiones como para editar la fecha y hora de las existentes) resulta tedioso y conlleva diversos problemas.

Por el contrario, se permite la edición del atributo ‘d’ que se corresponde con la duración de la emisión. Este campo también es sensible a provocar un error, ya que podría provocar *overlapping* o *gaps* en la línea temporal de las emisiones. Pero se decidió que se dejaba libre, para poder introducir este tipo de error de forma intencionada, si se desea. En el manual se recomienda y se advierte editar este atributo con cabeza, sabiendo de la repercusión de introducir un valor no válido.

Extraer EPG de la base de datos

El siguiente paso, después de que el desarrollador edita el archivo EPG, consiste en ingestar este fichero en el sistema que se desea testear. Este proceso implica, de nuevo, extraer las listas que contiene el fichero XML e introducirlas en una base de datos.

Una posible mejora podría consistir en hacer el proceso inverso. Mediante unos parámetros de configuración (IP del servidor de la base de datos, tipo de base de datos, puerto, usuario, contraseña, etc.), extraer los elementos de las listas para crear de nuevo un fichero XML. También podría permitirse editar directamente el contenido de la base de datos, utilizando EPG Viewer como interfaz gráfica. Algunos desarrolladores realizan de esta forma sus pruebas y test. La inclusión de esta nueva mejora podría suponer un paso importante, abarcando también a este tipo de metodología. Se podría llegar a un número mayor de posibles usuarios del software, asegurando el futuro y permanencia del mismo.

Mejora visual del *Timeline*

Uno de los aspectos que puede mejorarse es el visual. Dado que los usuarios del software no necesitan una interfaz gráfica particular, el acabado final es austero y funcional. Cumple con su función sin ser especialmente atractivo visualmente. Esta cuestión no resulta preocupante pero puede ser mejorada. Para ser más preciso, podría trabajarse de forma más artística en lo que se refiere al *Timeline*. Esta funcionalidad se basa en una tabla de la librería JFrame, cuya anchura de celda es proporcional a la duración de la emisión (valor del atributo 'd'). Existen librerías de Java, con aspecto visual más logrado, que podrían mejorar el acabado final de esta función. Esto puede hacer más intuitivo la búsqueda de emisiones utilizando la línea temporal como herramienta de búsqueda.

Como no determina el funcionamiento de EPG Viewer, esta mejora no se considera una prioridad, pero podría ser una posible mejora sustancial en cuanto al aspecto gráfico.

Incorporación de tecnologías WEB

Esta mejora supone un cambio radical en el software. Implica reinventar el EPG Viewer desde cero. Propongo transformar el software, diseñado para trabajar de forma local, en una aplicación distribuida. Esto divide en dos EPG Viewer: por un lado el cliente y por otro el servidor. La lógica de EPG Viewer funciona del lado del servidor. Se mantienen las funciones presentes, y se pueden incluir las que este capítulo propone.

El usuario accede vía Web, mediante un explorador de Internet. Este hace las veces de interfaz de usuario, realizando las peticiones al servidor dónde esté ejecutándose EPG Viewer. Esto permite el uso simultáneo de varios usuarios al servicio. También podría crearse un sistema remoto de archivos, dónde se puedan alojar diferentes archivos EPG editados de una u otra forma para diferentes fines. Incluso compartir archivos entre diferentes desarrolladores. Para ellos también debería implementarse un filtro de usuarios, protegidos con contraseñas.

Esta solución es de gran complejidad, ya que implica la integración de diferentes tecnologías, trabajando al unísono. Por otro lado, abre las puertas para muchísimas posibles mejoras basadas en tecnologías Web. Una de las grandes novedades sería la

posibilidad de utilizar la herramienta varios usuarios de forma simultánea. Además se elimina la necesidad de disponer de un equipo de cierta potencia, ya que el código pesado se ejecuta en el servidor, que ya dispone de los recursos necesarios.

Estas últimas novedades, son sólo parte de un gran número de ventajas asociadas a esta variante sobre el software EPG Viewer. Podrían incluso coexistir la versión de escritorio, para trabajar en local, junto con la versión que se propone en este punto.

Externalizar los perfiles de verificación

Una de las características de EPG Viewer es los perfiles de verificación. Existe la posibilidad de seleccionar uno de los dos disponibles, contenidos en el código fuente del software. Una posible mejora podría ser externalizarlos. De esta forma se podrían crear nuevos perfiles, siguiendo algún tipo de esquema prefijado. Así podríamos conseguir una herramienta más abierta, dando la posibilidad al desarrollador de crear sus propios perfiles, en función de su necesidad. Esto podría extenderse a los idiomas, dando la posibilidad de traducir a diferentes lenguas los diferentes menús del software.

Estas son sólo algunas de las posibles líneas futuras de desarrollo para EPG Viewer. Podríamos pensar en otras mejoras relacionadas con la evolución de los sistemas de televisión. Pongamos, por ejemplo, que en el futuro se decide utilizar otro esquema diferente a GLF, uno diseñado por la empresa, basado en sus propios estándares. Según esta codificada la herramienta software, esto implicaría reescribir prácticamente todo el código.

Sin embargo, considero que la herramienta actual cumple su cometido con creces, ya que facilita el trabajo y representa un ahorro significativo de tiempo en el manejo de los archivos EPG. Las funcionalidades básicas permiten editar prácticamente casi cualquier atributo de todas las listas que forman la guía de programación. Las funcionalidades extra ayudan a evaluar y corregir errores, para asegurar la calidad de los archivos generados. En conjunto satisfacen una necesidad REAL de la empresa, y creo que siento así, puedo calificar este TFG de satisfactorio.

Capítulo 8: Referencias y Glosario de términos

8.1 – Referencias

ASP tutor. (2016). *ASP tutor*. Recuperado el Abril de 2016, de http://www.asptutor.com/xml/index1.asp

Code school. (2016). *javascript.com*. Recuperado el Abril de 2016, de https://www.javascript.com/resources

Deitel, P. (2003). *Cómo programar en C++.* Pearson Educación.

Garrido, P. (2015). *Comenzando a programar con JAVA.* Universidad Miguel Hernández.

Hernández, J. M. (2015). *AngularJS: Lo bueno y lo malo*. Recuperado el Mayo de 2016, de Koalite: http://blog.koalite.com/2013/06/angularjs-lo-bueno-y-lo-malo/

java.net. (2016). *Proyect JAXB*. Recuperado el Mayo de 2016, de https://jaxb.java.net/

Mean by Linnovate. (2016). *mean.io*. Recuperado el Mayo de 2016, de http://learn.mean.io/

Microsoft Coporation. (2011). *Patent No. US 7913279 B2*. Recuperado el Mayo de 2016, de Lens: https://www.lens.org/images/patent/US/7913279/B2/US_7913279_B2.pdf

Node.js Foundation. (2015). *nodejs.org*. Recuperado el Abril de 2016, de ECMAScript 2015 (ES6) and beyond: https://nodejs.org/en/docs/es6/

Pascual, J. (2015). *Un muerciano en el polo*. Recuperado el Mayo de 2016, de http://jpascu.blogspot.com.es/2011/11/jaxb-leer-y-escribir-ficheros-xml.html

Pérez, C., & Zamanillo, J. M. (2003). *Fundamentos de Televisión Analógica y Digital.* Servicio de Publicaciones de la Universidad de Cantabria.

Rodríguez, A. (2013). *aprendeaprogramar.com*. Recuperado el Abril de 2016, de http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=398:netbeans-eclipse-jcreator-jbuilder-icual-es-el-mejor-entorno-de-desarrollo-ide-para-java-cu00613b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

Tutorials point. (2014). *XML - Tutorial*. Recuperado el Mayo de 2016, de Aprender XML: http://www.tutorialspoint.com/es/xml/xml_tags.htm

8.2 – Glosario de términos

GVP: Global Video Platform.

QA: Quality Assurance.

EPG: Electronic Program Guide.

TFG: Trabajo Fin de Grado.

TDT: Televisión Digital Terrestre.

XML: eXtensible Markup Language.

XSD: XML Schema Definition Language.

TID: Telefónica Investigación y Desarrollo.

VBI: Vertical Blanking Interval.

CRT: Cathode Ray Tube.

FTA: Free To Air.

DVB: Digital Video Broadcasting.

JVM: Java Virtual Machine.

IDE: Integrated Development Environment.

DTD: Document Type Definition.

XSD: XML Schema Definition.

UTC: Universal Time Coordinate.

SD: Standard Definition.

HD: High Definition.

ANEXOS

Anexo I:

Documento original de la patente US 7913279 B2 perteneciente al esquema GLF (Microsoft Coporation, 2011)

US007913279B2

(12) **United States Patent**
Simms et al.

(10) **Patent No.: US 7,913,279 B2**
(45) **Date of Patent: Mar. 22, 2011**

(54) **GLOBAL LISTINGS FORMAT (GLF) FOR MULTIMEDIA PROGRAMMING CONTENT AND ELECTRONIC PROGRAM GUIDE (EPG) INFORMATION**

(75) Inventors: **Andrew M. Simms**, Fountain Hills, AZ (US); **Samuel Thomas Scott, III**, Los Gatos, CA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1574 days.

(21) Appl. No.: **10/356,694**

(22) Filed: **Jan. 31, 2003**

(65) **Prior Publication Data**

US 2004/0154039 A1 Aug. 5, 2004

(51) **Int. Cl.**

G06F 3/00	(2006.01)
G06F 13/00	(2006.01)
H04N 5/445	(2011.01)
H04N 7/173	(2011.01)

(52) **U.S. Cl.** **725/39**; 725/48; 725/50

(58) **Field of Classification Search** 725/39
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,890,321	A *	12/1989	Seth-Smith et al.	380/231
5,576,755	A *	11/1996	Davis et al.	725/48
5,666,645	A *	9/1997	Thomas et al.	725/47
6,002,394	A *	12/1999	Schein et al.	725/39
6,040,829	A *	3/2000	Croy et al.	715/864
6,184,877	B1	2/2001	Dodson et al.	
6,209,129	B1 *	3/2001	Carr et al.	725/42
6,401,242	B1 *	6/2002	Eyer et al.	725/35
6,442,755	B1 *	8/2002	Lemmons et al.	725/47
7,073,193	B2 *	7/2006	Marsh	725/114
7,197,715	B1 *	3/2007	Valeria	715/747
7,213,256	B1	5/2007	Kikinis	
7,536,705	B1 *	5/2009	Boucher et al.	725/112
7,627,881	B1 *	12/2009	Gonno et al.	725/39
7,712,113	B2 *	5/2010	Yoon et al.	725/9
2002/0038358	A1 *	3/2002	Sweatt et al.	709/218
2002/0049971	A1 *	4/2002	Augenbraun et al.	725/39
2002/0166120	A1 *	11/2002	Boylan et al.	725/35
2002/0170060	A1 *	11/2002	Lyman	725/73
2002/0199204	A1 *	12/2002	Mory et al.	725/113
2003/0004955	A1 *	1/2003	Cedola et al.	707/100
2003/0028884	A1 *	2/2003	Swart et al.	725/51

(Continued)

FOREIGN PATENT DOCUMENTS

WO	WO 96/13935	5/1996

(Continued)

OTHER PUBLICATIONS

"XML Schema Part 0: Primer", W3C Proposed Recommendation, Mar. 30, 2001, XP-002305611, 64 pages.

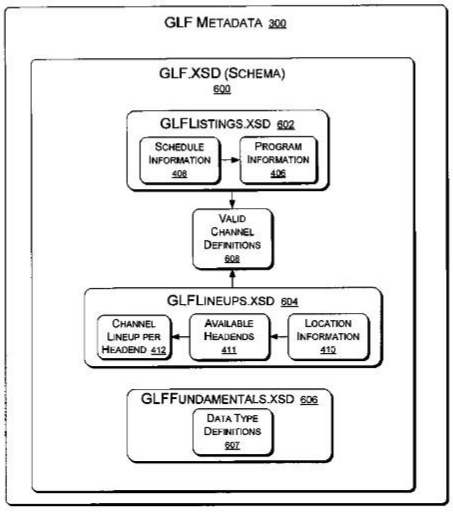
(Continued)

Primary Examiner — Brian T Pendleton
Assistant Examiner — Jonathan Lewis
(74) *Attorney, Agent, or Firm* — Lee & Hayes, PLLC

(57) **ABSTRACT**

An exemplary global listings format (GLF) is metadata for electronically transferring multimedia programming content and electronic program guide information. The GLF metadata specifies a self-referential data structure having a self-consistency mechanism comprising interlocking and cross-locking data elements. The self-consistency mechanism ensures completeness and validity of transferred programming data. In one implementation, the exemplary GLF is expressed in an extensible markup language (XML) schema definition (XSD) specification.

12 Claims, 9 Drawing Sheets



U.S. PATENT DOCUMENTS

2003/0165324	A1	9/2003	O'Connor et al.	
2003/0233241	A1 *	12/2003	Marsh	705/1
2003/0236708	A1 *	12/2003	Marsh	705/26
2004/0001106	A1 *	1/2004	Deutscher et al.	345/838
2004/0117831	A1	6/2004	Ellis et al.	
2004/0221310	A1	11/2004	Herrington et al.	
2004/0221311	A1	11/2004	Dow et al.	
2005/0188402	A1 *	8/2005	de Andrade et al.	725/46
2005/0235319	A1	10/2005	Carpenter et al.	
2006/0064716	A1 *	3/2006	Sull et al.	725/37
2007/0033533	A1 *	2/2007	Sull	715/752
2007/0299685	A1 *	12/2007	Marsh	705/1
2008/0115169	A1	5/2008	Ellis et al.	
2008/0147650	A1 *	6/2008	Marsh	707/5

FOREIGN PATENT DOCUMENTS

WO	WO9613935	A1	5/1996
WO	WO 03/007596		1/2003

OTHER PUBLICATIONS

"XML schema mappings for heterogeneous database access", Collins et al., Information and Software Technology 44, 2002, pp. 251-257.

* cited by examiner

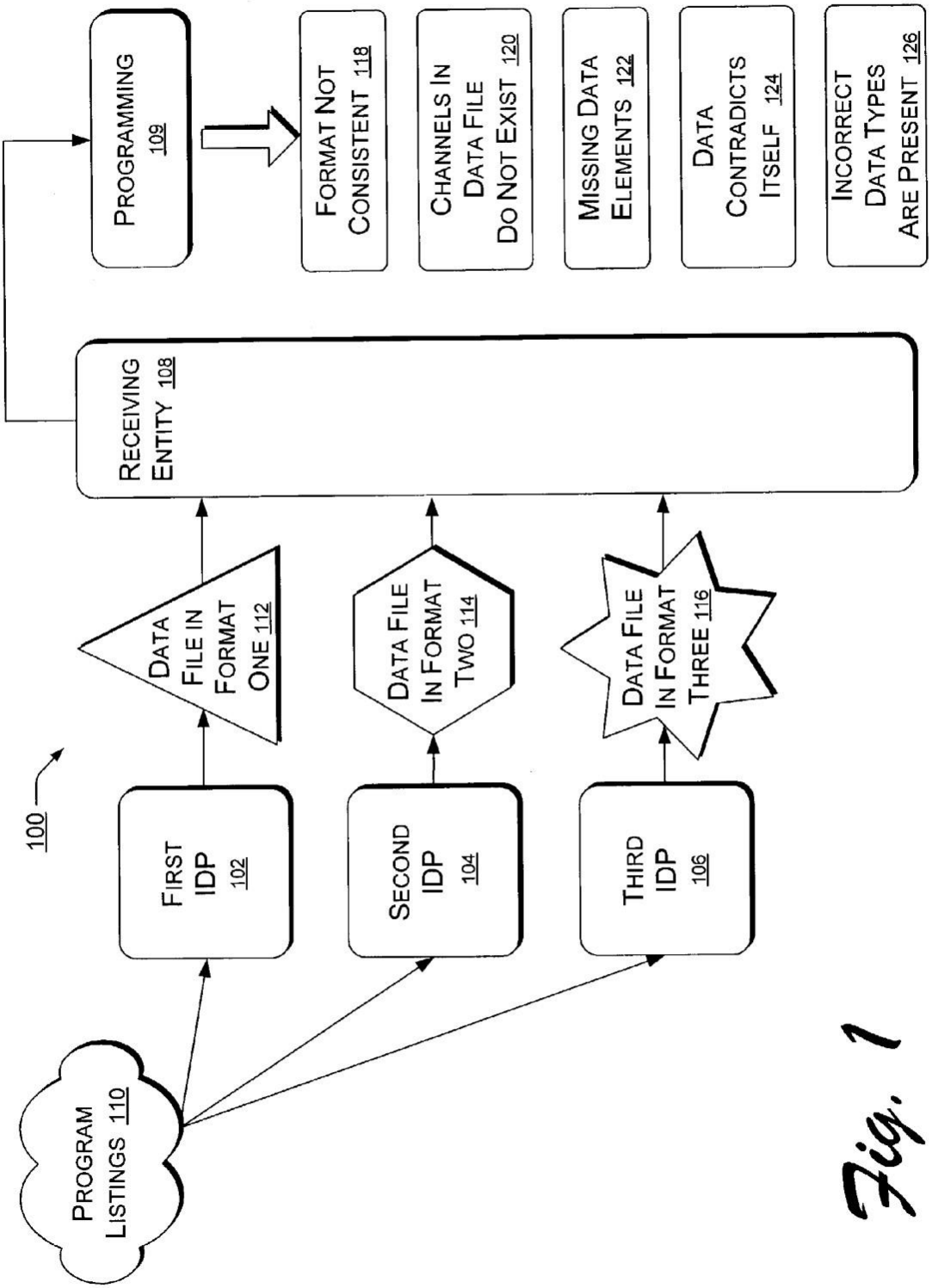


Fig. 1
(PRIOR ART)

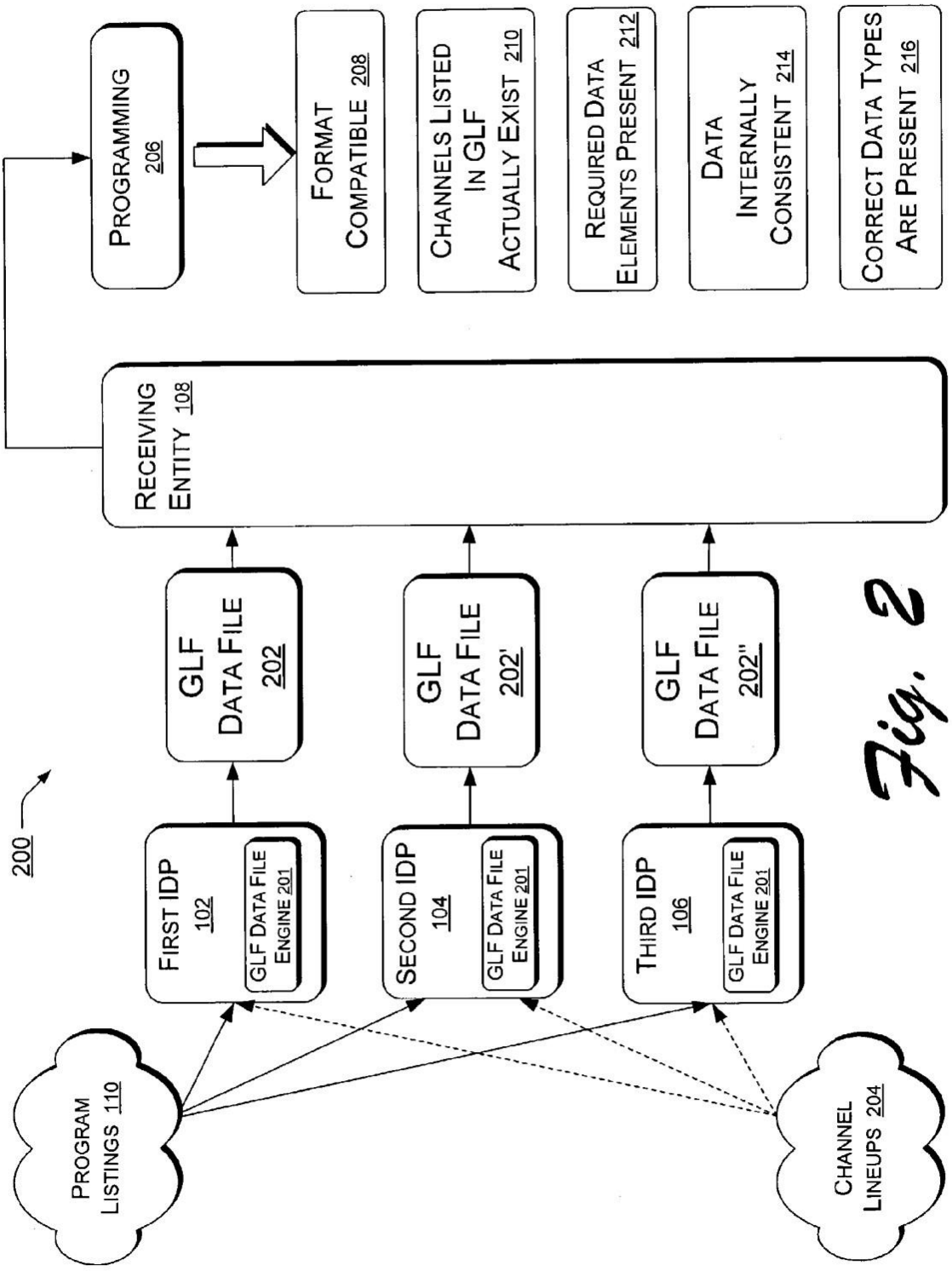

200
PROGRAM LISTINGS 110
CHANNEL LINEUPS 204
FIRST IDP 102
SECOND IDP 104
THIRD IDP 106
GLF DATA FILE ENGINE 201
GLF DATA FILE 202
GLF DATA FILE 202'
GLF DATA FILE 202"
RECEIVING ENTITY 108
PROGRAMMING 206
FORMAT COMPATIBLE 208
CHANNELS LISTED IN GLF ACTUALLY EXIST 210
REQUIRED DATA ELEMENTS PRESENT 212
DATA INTERNALLY CONSISTENT 214
CORRECT DATA TYPES ARE PRESENT 216

Fig. 2

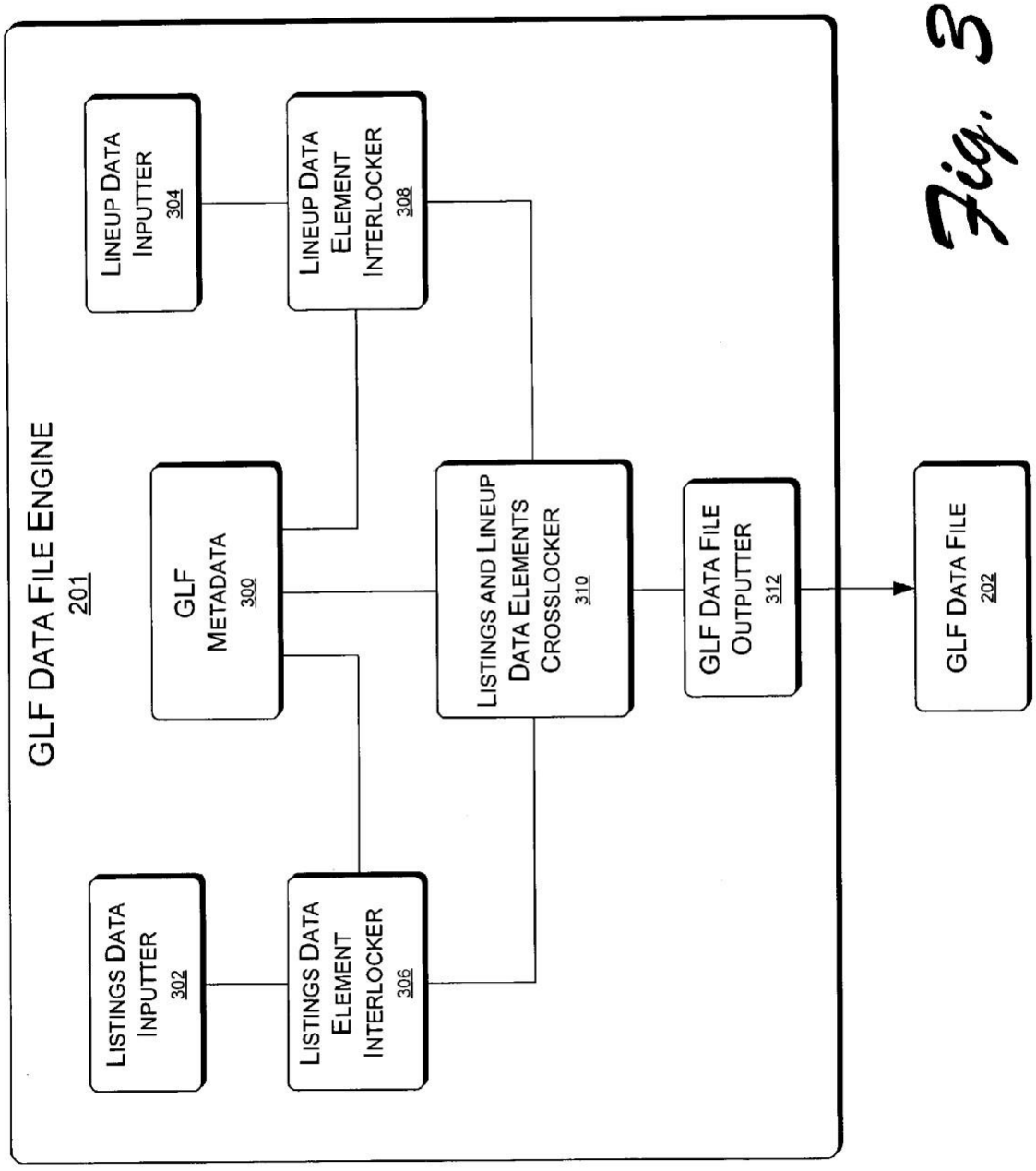
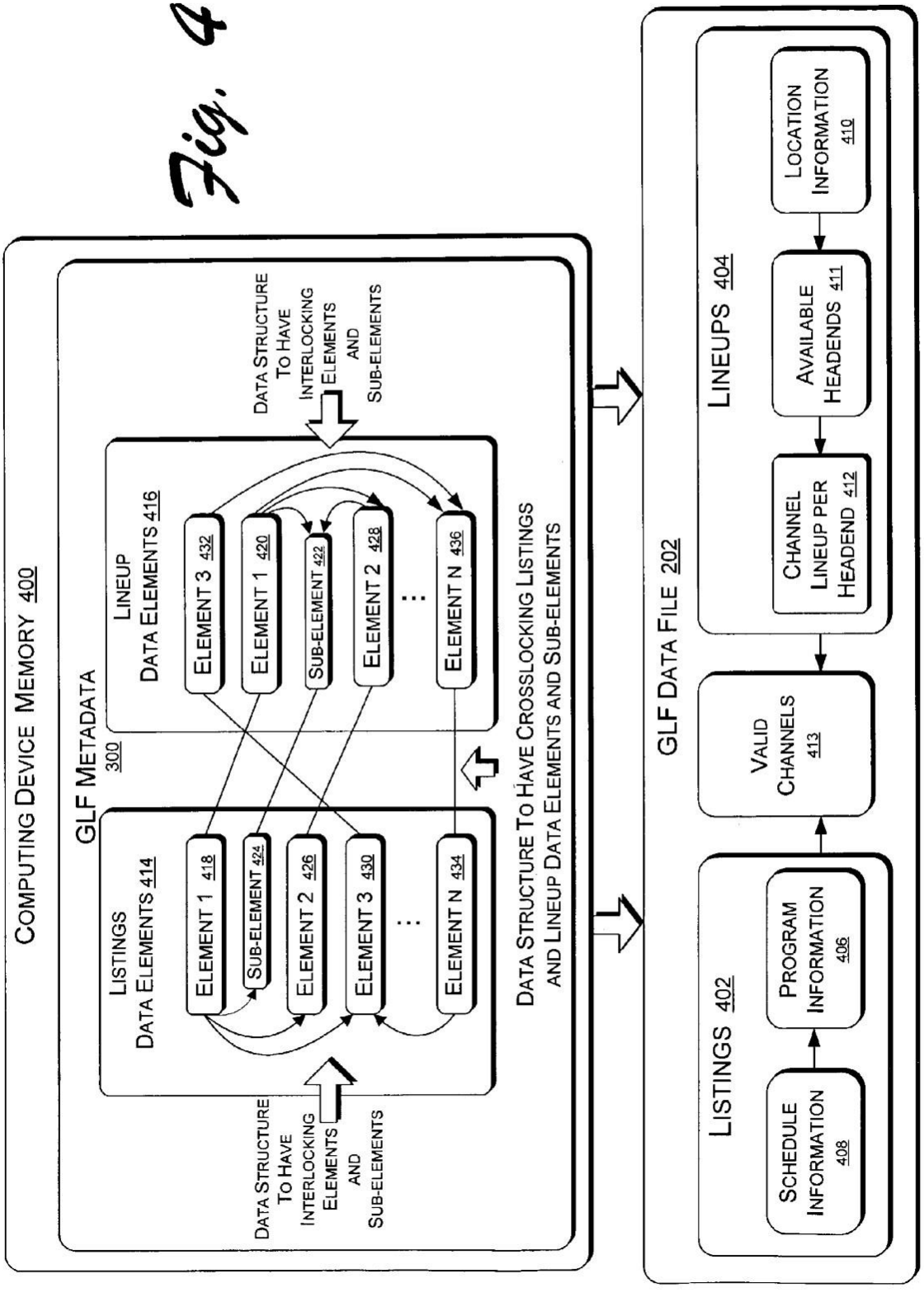

GLF DATA FILE ENGINE
201
LINEUP DATA INPUTTER
304
LINEUP DATA ELEMENT INTERLOCKER
308
GLF METADATA
300
LISTINGS AND LINEUP DATA ELEMENTS CROSSLOCKER
310
GLF DATA FILE OUTPUTTER
312
GLF DATA FILE
202
LISTINGS DATA INPUTTER
302
LISTINGS DATA ELEMENT INTERLOCKER
306

Fig. 3

Fig. 4


COMPUTING DEVICE MEMORY 400
GLF METADATA 300
LISTINGS DATA ELEMENTS 414
ELEMENT 1 418
SUB-ELEMENT 424
ELEMENT 2 426
ELEMENT 3 430
ELEMENT N 434
LINEUP DATA ELEMENTS 416
ELEMENT 3 432
ELEMENT 1 420
SUB-ELEMENT 422
ELEMENT 2 428
ELEMENT N 436
DATA STRUCTURE TO HAVE INTERLOCKING ELEMENTS AND SUB-ELEMENTS
DATA STRUCTURE TO HAVE INTERLOCKING ELEMENTS AND SUB-ELEMENTS
DATA STRUCTURE TO HAVE CROSSLOCKING LISTINGS AND LINEUP DATA ELEMENTS AND SUB-ELEMENTS
GLF DATA FILE 202
LISTINGS 402
SCHEDULE INFORMATION 408
PROGRAM INFORMATION 406
VALID CHANNELS 413
LINEUPS 404
CHANNEL LINEUP PER HEADEND 412
AVAILABLE HEADENDS 411
LOCATION INFORMATION 410

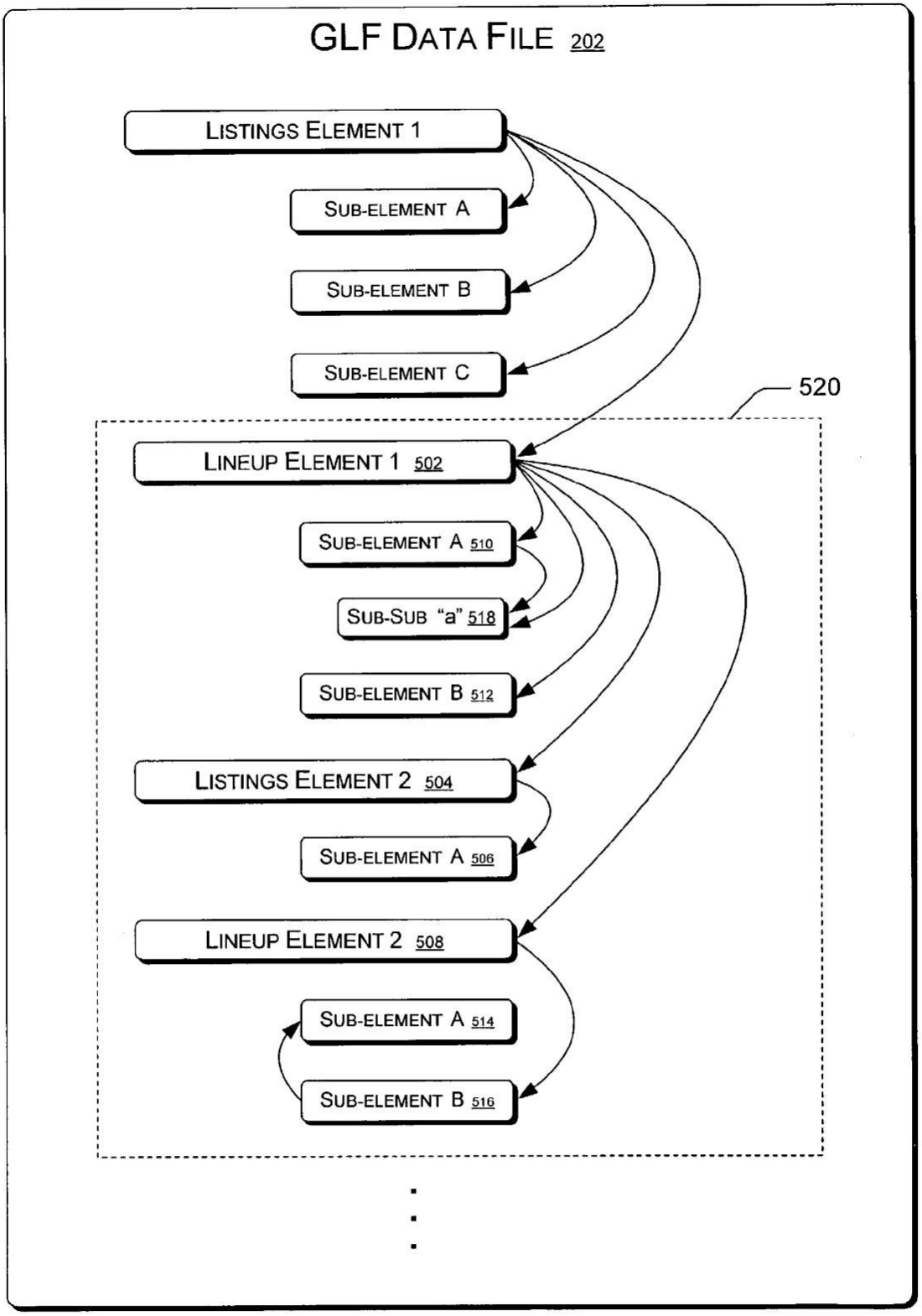

GLF DATA FILE 202
LISTINGS ELEMENT 1
SUB-ELEMENT A
SUB-ELEMENT B
SUB-ELEMENT C
520
LINEUP ELEMENT 1 502
SUB-ELEMENT A 510
SUB-SUB "a" 518
SUB-ELEMENT B 512
LISTINGS ELEMENT 2 504
SUB-ELEMENT A 506
LINEUP ELEMENT 2 508
SUB-ELEMENT A 514
SUB-ELEMENT B 516

Fig. 5

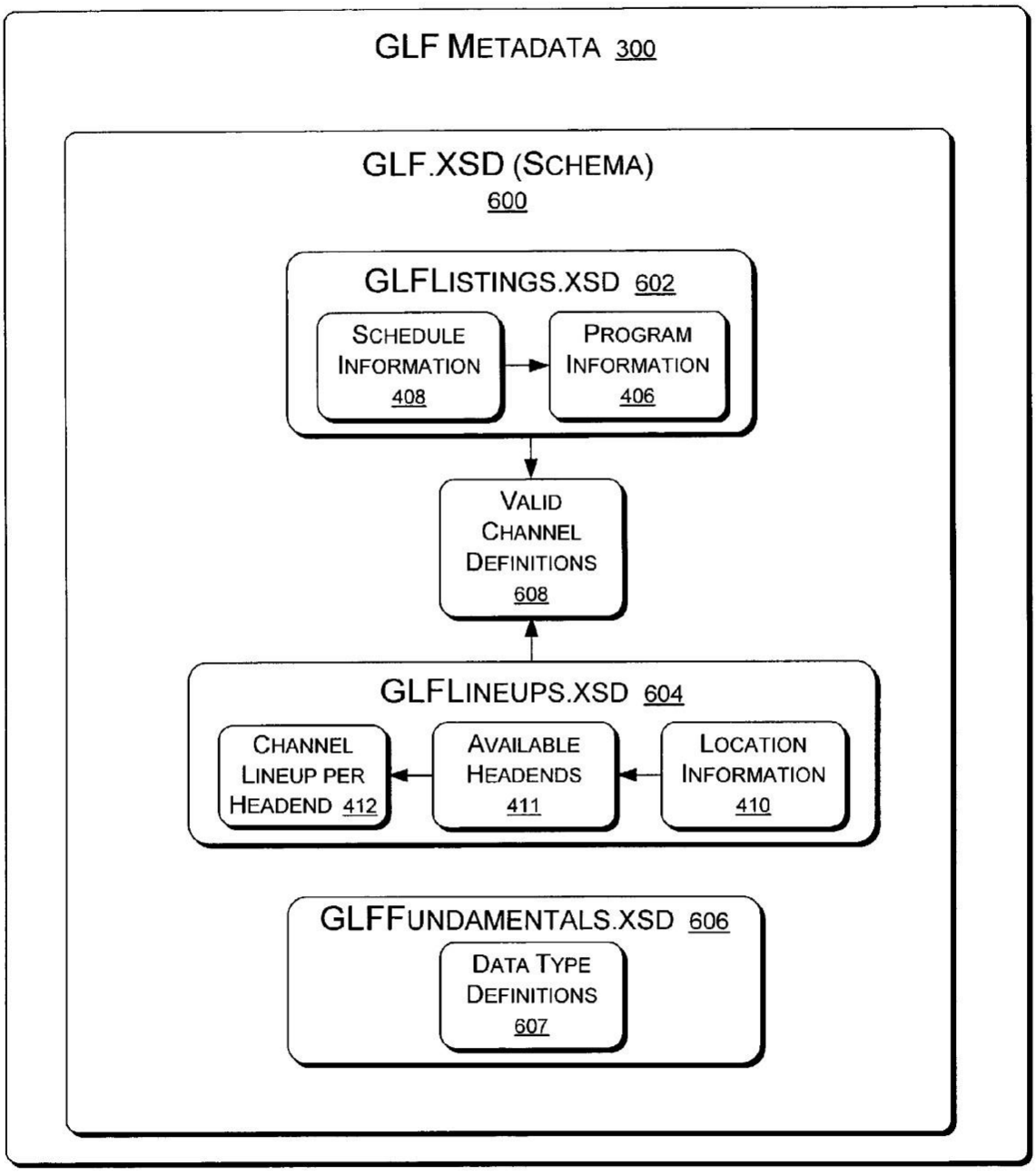

GLF METADATA 300
GLF.XSD (SCHEMA)
600
GLFLISTINGS.XSD 602
SCHEDULE INFORMATION 408
PROGRAM INFORMATION 406
VALID CHANNEL DEFINITIONS 608
GLFLINEUPS.XSD 604
CHANNEL LINEUP PER HEADEND 412
AVAILABLE HEADENDS 411
LOCATION INFORMATION 410
GLFFUNDAMENTALS.XSD 606
DATA TYPE DEFINITIONS 607

Fig. 6

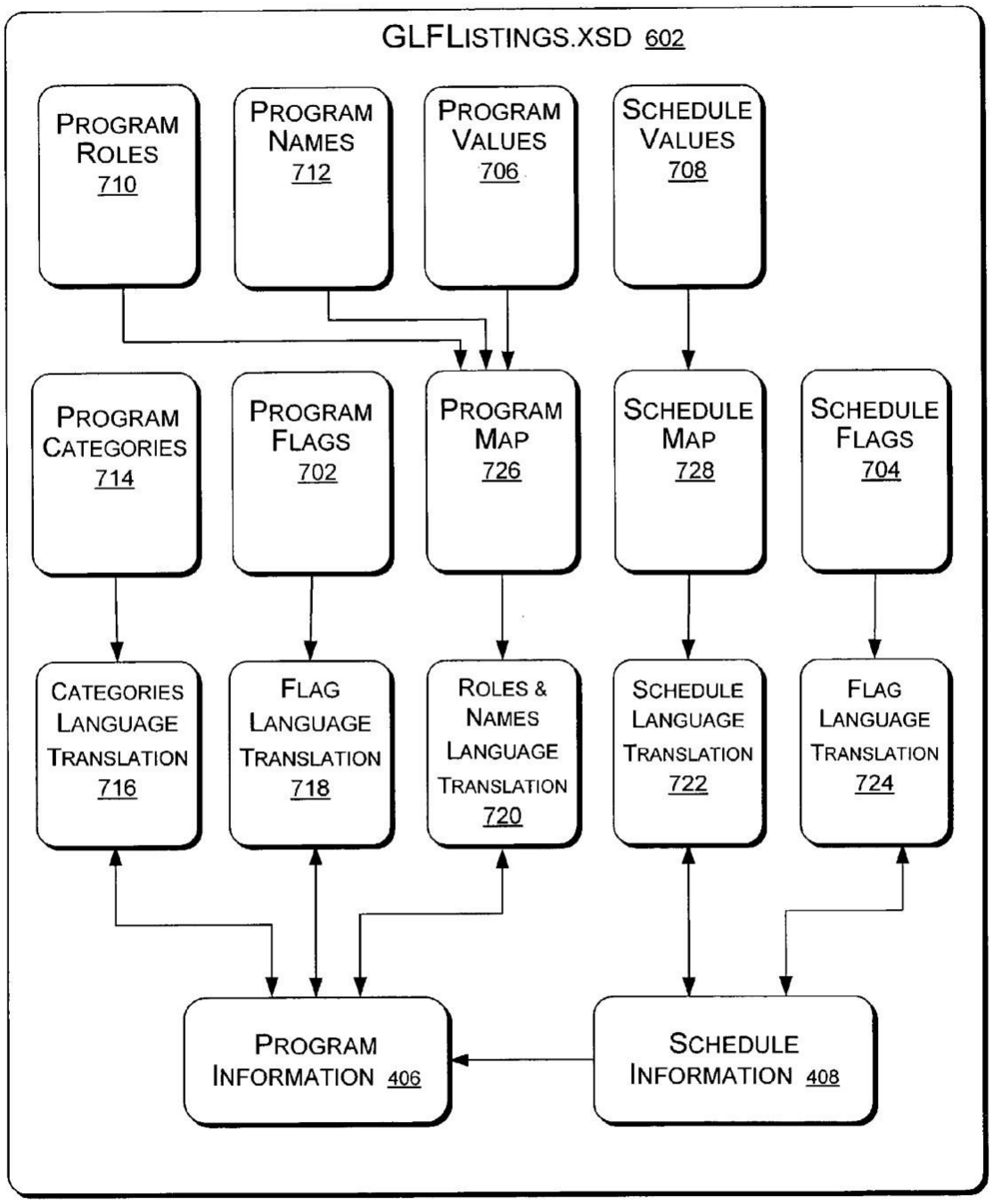

GLFLISTINGS.XSD 602
PROGRAM ROLES 710
PROGRAM NAMES 712
PROGRAM VALUES 706
SCHEDULE VALUES 708
PROGRAM CATEGORIES 714
PROGRAM FLAGS 702
PROGRAM MAP 726
SCHEDULE MAP 728
SCHEDULE FLAGS 704
CATEGORIES LANGUAGE TRANSLATION 716
FLAG LANGUAGE TRANSLATION 718
ROLES & NAMES LANGUAGE TRANSLATION 720
SCHEDULE LANGUAGE TRANSLATION 722
FLAG LANGUAGE TRANSLATION 724
PROGRAM INFORMATION 406
SCHEDULE INFORMATION 408

Fig. 7

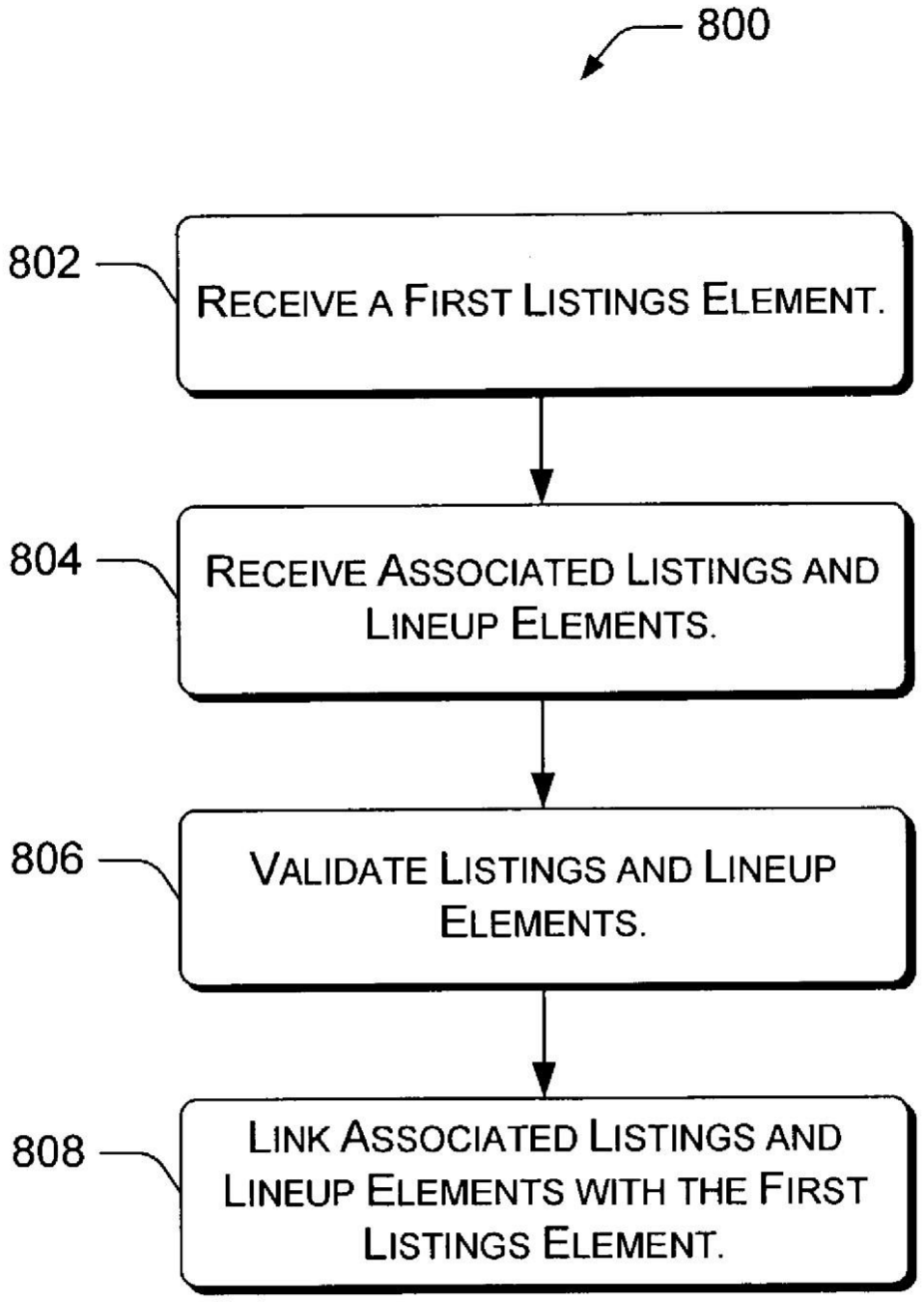

800
802
RECEIVE A FIRST LISTINGS ELEMENT.
804
RECEIVE ASSOCIATED LISTINGS AND LINEUP ELEMENTS.
806
VALIDATE LISTINGS AND LINEUP ELEMENTS.
808
LINK ASSOCIATED LISTINGS AND LINEUP ELEMENTS WITH THE FIRST LISTINGS ELEMENT.

Fig. 8

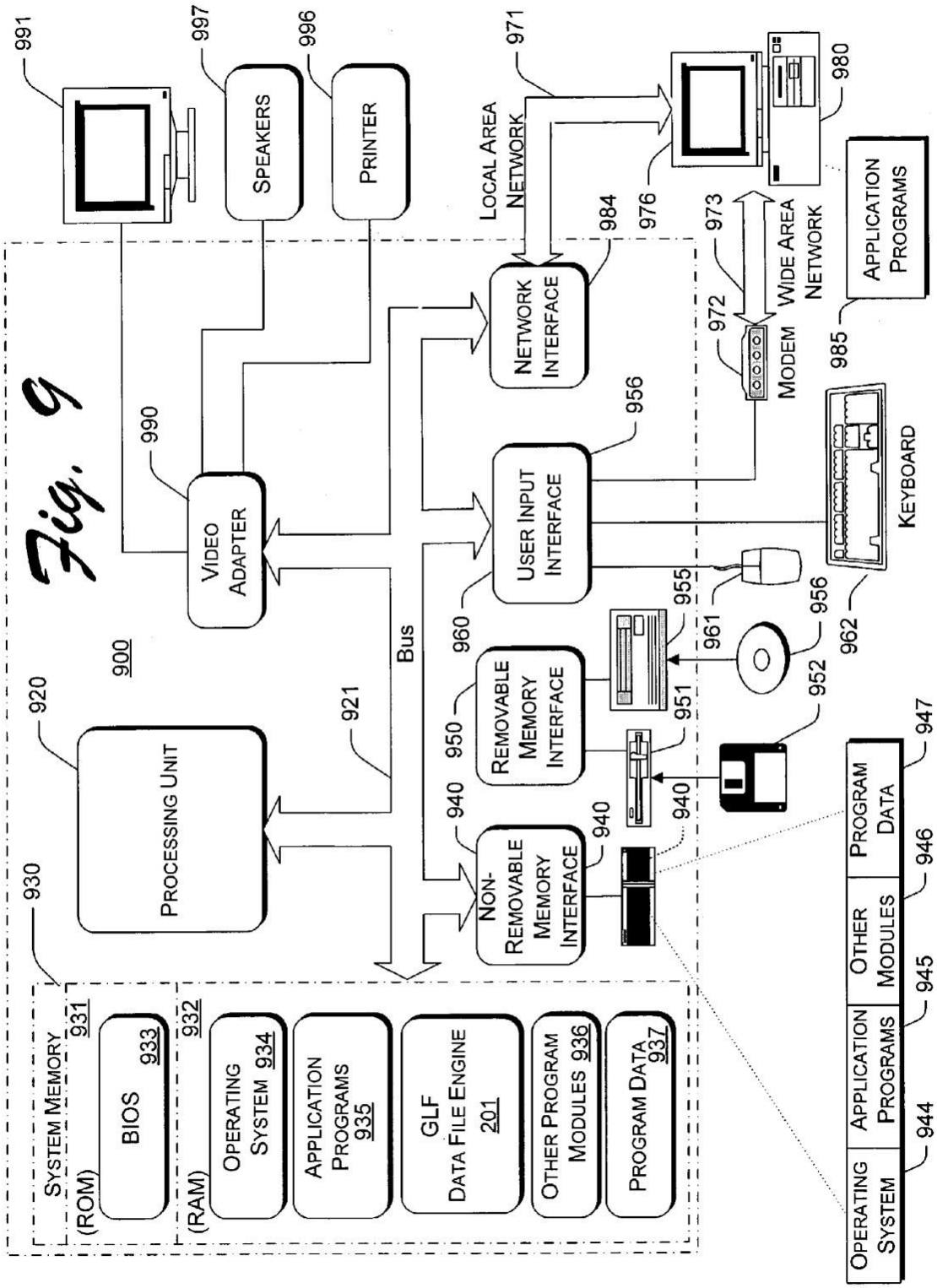

991
997
996
971
980
SPEAKERS
PRINTER
LOCAL AREA NETWORK
APPLICATION PROGRAMS
984
976
973
WIDE AREA NETWORK
NETWORK INTERFACE
972
MODEM
985
990
956
VIDEO ADAPTER
USER INPUT INTERFACE
KEYBOARD
900
920
921
Bus
960
955
961
956
962
952
950
REMOVABLE MEMORY INTERFACE
951
940
940
940
PROCESSING UNIT
NON-REMOVABLE MEMORY INTERFACE
930
947
946
945
944
SYSTEM MEMORY
931
(ROM)
BIOS 933
932
(RAM)
OPERATING SYSTEM 934
APPLICATION PROGRAMS 935
GLF DATA FILE ENGINE 201
OTHER PROGRAM MODULES 936
PROGRAM DATA 937
OPERATING SYSTEM
APPLICATION PROGRAMS
OTHER MODULES
PROGRAM DATA

Fig. 9

GLOBAL LISTINGS FORMAT (GLF) FOR MULTIMEDIA PROGRAMMING CONTENT AND ELECTRONIC PROGRAM GUIDE (EPG) INFORMATION

One set of XML text file listings used in accordance with the subject matter are provided in an appendix after the abstract on 7 sheets of paper and incorporated by reference into the specification. The XML text file listing is an exemplary sample global listings format data file.

TECHNICAL FIELD

This invention relates generally to multimedia data communications and specifically to a global listings format (GLF) for multimedia programming content and electronic program guide (EPG) information.

BACKGROUND

As computerized products for enjoying television and other multimedia forms expand across international markets, originators and distributors of multimedia programming known as Independent Data Providers (IDPs) have multiplied to provide programming content from many countries. Programming content usually consists of the program listings, which are transferred in a "data feed," that is, a "programming data file" that supplies local programming distributors with enough programming content to fill locally available channels for a specified duration, usually measured in days. The amount of programming data to be delivered by an IDP is usually defined in a listings data agreement.

Each IDP uses a proprietary listings format to create a programming data file that can be received and processed into local programming by a receiving entity. These proprietary listings formats are sometimes similar to each other but more often are different from each other and/or incompatible with each other, and more importantly, are incompatible with the receiving entity's needs. Specifically, the insertion of channel information needed to achieve a channel "lineup" that really works in a given locality for the supplied programming content is haphazard, due to the varying proprietary listings formats. In some instances the proprietary formats vary because collections of listings (i.e., programs and schedules) and lineups (i.e., sets of channels on which the programs and schedules will be implemented) are created by different types of IDPs. IDPs that produce listings usually predominate over the IDPs that produce lineups resulting in proprietary formats that favor listings information but neglect complete and accurate lineup information.

Additionally, the varying proprietary listings formats do not reliably convey information that has been translated across languages spoken in different countries. It is not always the translation itself that is problematic. A correct translation may be linked incorrectly to the wrong program number, for instance, or a new program number may be unnecessarily adopted because of the new language translation, resulting in a loss of referential consistency and, in this instance, a plethora of program numbers for one program.

FIG. **1** shows a conventional programming data file delivery environment **100**, in which delivery proceeds from three example IDPs **102**, **104**, **106** to a receiving entity **108**. IDPs typically select programming from a relatively unlimited universe of program listings **110** to create a programming data file that is marketed to the receiving entity **108**, i.e., a local programming distributor, such as local broadcast station or a television cable company. The programming data files are

characteristically large and therefore require, for example, approximately twenty hours of processing time to convert the data file into usable form.

In creating a programming data file, IDPs typically emphasize program listings **110** information, such as the names of movies and programs, to the exclusion or haphazard inclusion of supporting "lineup" logistical information that would be necessary to present programs at given times on specific channels available in the receiving entity's locale or catchment area. Although most IDPs add schedule information, attempts to include consistent and correct lineup information usually prove insufficient, as discussed above, especially when programming data files are transferred between different countries.

The blend of information formulated in a proprietary programming data file and the proprietary formats used may vary widely between IDPs. Thus, as shown in FIG. **1**, a first IDP **102** uses a first proprietary data file format **112**, a second IDP **104** uses a second proprietary data file format **114**, and a third IDP **106** uses a third proprietary data file format **116**. A receiving entity **108**, such as a local programming distributor may even subscribe to several IDPs and must process programming data files from each into programming **109**. Often, after spending, for example, twenty hours of processing time to convert a programming data file into usable programming **109**, the receiving entity **108** discovers that one or more aspects of a proprietary format are not consistent **118**. Perhaps channels listed in the programming data file do not exist **120** or do not exist in the local supplier's geographical and/or geopolitical area. Since the data file formats **112**, **114**, **116** from the various IDPs are arbitrary, data elements may be missing **122**, the data may contradict itself **124**, and/or the data may be present but may be unusable because incorrect and/or unrecognizable data types have been used **126**. Because the data file is partially or completely defective, part or all of the processing time investment may be lost, and troubleshooting may be needed.

SUMMARY

Subject matter includes an exemplary global listings format (GLF) for electronically transferring multimedia programming content and electronic program guide (EPG) information. The GLF can be a type of metadata that specifies a data structure having a self-consistency mechanism comprising linked, that is, interlocking and crosslocking, data elements. The self-consistency mechanism ensures consistency, completeness, and validity of a multimedia programming data file to be electronically transferred.

In one implementation, the exemplary GLF is cast in an extensible markup language (XML) schema definition (XSD) specification and IDPs are supplied with a set of editorial instructions as a guide for producing and delivering programming data. A receiving entity having XML database (XDB) capabilities can import the GLF standardized programming data file for producing substantially complete and error-free programming for a variety of clients, using standard XML tools and resources. The strong data typing inherent in XML provides one aspect of GLF data validation.

An exemplary GLF standardizes key relationships in programming data, particularly relationships between listings data and lineup data. Various implementations of the GLF have sufficient richness and versatility to accommodate listings and lineups for programming content and EPG guide information that can be used in many types of multimedia

products. Exemplary GLFs are expandable and extensible to accommodate detailed attributes related to the listings and the lineups.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a conventional environment in which programming data files are transferred.

FIG. **2** is a block diagram of an exemplary environment in which global listings format (GLF) programming data files are transferred.

FIG. **3** is a block diagram of an exemplary GLF data file engine.

FIG. **4** is a graphic representation showing formation of an exemplary GLF data file using exemplary GLF metadata residing as a data structure in the memory of a computing device.

FIG. **5** is a graphic representation showing formation of an exemplary GLF data structure in an exemplary GLF data file.

FIG. **6** is a block diagram of a GLF.xsd schema and components.

FIG. **7** is a block diagram of an exemplary GLFListings.xsd schema component.

FIG. **8** is a flow diagram of an exemplary method for producing a GLF programming data file.

FIG. **9** is a block diagram of an exemplary computing device environment in which to practice the subject matter.

DETAILED DESCRIPTION

Overview

It is rather frustrating to invest perhaps twenty hours of processing time to decode a programming data file **112**, **114**, **116** into usable form, only to find that channels listed in the data do not really exist, that a foreign language translation is attached to the wrong program, or that the same program has multiple unique ID numbers.

Exemplary subject matter includes a global listings format (GLF) and related methods and data structures for electronically transferring programming content and electronic programming guide (EPG) information, for example, from an IDP to a local distributor client, or between countries in which different languages are spoken. The GLF for multimedia programming content and EPG information also forms a consistent foundation for operating systems that support multimedia processing, such as Windows XP Media Center Edition (Microsoft Corporation, Redmond, Wash.).

A listings element may be any data or information related to programming content or to the scheduling thereof. A lineup element may be any data or information related to sets of channels on which the programming content and scheduling will be implemented. The exemplary GLF can be thought of as metadata for consistently structuring the listings and lineup information. Insofar as the listings and lineup information is itself metadata about the programming content, the exemplary GLF is arrangement metadata for the listings and lineup metadata. The GLF metadata aims to provide consistently comprehensive and correct programming information, so that a given local distributor receiving a GLF programming data file has all the information at hand that might be needed to provide programming and EPG information in any degree of desired detail. Since the GLF metadata specifies a consistent listings format, whoever uses the GLF metadata can realize the downstream benefits of consistency, completeness, and correctness of the programming data.

Receiving a GLF programming data file, the local distributor may be freed from the tedious tasks of error correction and gathering of additional information.

In other words, the exemplary GLF has a self-referential structure and built-in self-consistency mechanism that includes various interlocking mechanisms that establish and enforce completeness and validity of the programming information. Each data element included in a GLF data structure is referred to, so that there are no "left-over," unidentified, or unaccounted for data elements. If completeness is not satisfied, then various specific actions may follow. The various interlocking mechanisms not only detect that a data file is invalid, but can also report on what specifically is incorrect. An error can be logged and a full description of the error sent to the IDP. Since the IDP can also be in possession of the GLF, the IDP can validate their GLF data file before sending it to the local distributor.

The self-referential structure of an exemplary GLF has interlocking and crosslocking (collectively: "linked") fields and/or data elements that create a data structure wherein important or desirable programming information is not forgotten or inserted incorrectly during programming data file creation. The term "crosslocking" is used for establishing linkages between listings and lineup data. This distinguishes "crosslocking" from "interlocking," which refers to links between two listings data elements or between two lineup data elements.

In one implementation, the exemplary GLF is cast in an extensible markup language (XML) schema definition (XSD) specification and IDPs are supplied with a set of editorial instructions as a guide for producing and delivering programming data. A receiving entity **108** having XML database (XDB) capabilities can import the GLF standardized programming data file for producing substantially complete and error-free programming for a variety of clients, using standard XML tools and resources.

An exemplary GLF standardizes key relationships in programming data, particularly relationships between listings data and lineup data. "Listings" is a term used to include program and schedule information, for example, the program title, unique program ID, episode title, episode number, description, year of creation, cast, acting roles, crew, ratings, category, length, start time, frequency, etc. "Lineup" is a term used for channeling data, such as channels available in a given area, for example on cable, over-the-air, and satellite services. Thus, lineup data is used to partition the set of all available channels into a smaller set that is relevant to a given receiving entity **108** in a particular locale.

Implementations of the GLF have sufficient richness and versatility to accommodate listings and lineups for many types of multimedia products. Exemplary GLFs are expandable and extensible to accommodate detailed attributes about the listings and the lineups.

Programming Data File Transfer Environment

FIG. **2** shows a programming data file delivery environment **200**, in which delivery proceeds from the three example IDPs **102**, **104**, **106** to the receiving entity **108** using exemplary GLF data files **202**, **202'**, **202"**. Example IDPs **102**, **104**, **106** each having an exemplary GLF data file engine **201**, select programming content from a relatively unlimited universe of program listings **110** to create a data file(s) that can be electronically transferred to a receiving entity **108**, such as a local programming distributor. Each exemplary GLF data file engine **201** contains or has access to GLF metadata for creating a GLF data file **202** in an exemplary GLF format. In each exemplary GLF data file **202**, **202'**, **202"** the programming data is structured so that each selected element from the

program listings **110** to be transferred is accompanied, if appropriate, by one or more elements from a universe of channel lineups **204**. The various listings elements and lineup elements are interlocked and crosslocked, where appropriate, to provide an exemplary GLF data structure, including a self-consistency mechanism (i.e., the self-referential interlocks and crosslocks to be discussed more fully below). Thus, an exemplary GLF data file **202** formatted according to exemplary GLF metadata has substantially complete and error-free programming information regardless of which IDP **102**, **104**, **106** creates the programming data file.

The self-consistency mechanism(s) imbued in exemplary GLF data structures ensure substantially complete and error-free programming information, and are especially useful when the programming data is being transferred between international sources. Built-in language translation fields can optionally be used to ensure that language translations are inserted into or interlocked with the proper data elements. The receiving entity **108** processes each received GLF data file **202** to obtain programming **109** that is true to an expected format (**208**), that is, the usable programming **109** conforms to GLF standards of completeness and validity. Further, channels listed in a GLF data file **202** actually exist in the receiving entity's location **210**. The exemplary GLF data structure also assures data completeness and integrity, namely, that required data elements are present **212** (e.g., if a program listing is supplied, then lineup information is also supplied), that the data is internally consistent **214** (i.e., no contradictions are present), and that correct data types have been applied to each data element supplied **216**.

Exemplary GLF System Components

FIG. **3** shows the exemplary GLF data file engine **201** of FIG. **2** in greater detail. Of course, other means, such as other engines, routines, rules, etc. can yield an exemplary GLF data structure result. The exemplary GLF data file engine **201** uses GLF metadata **300** to link data elements and attributes into the GLF data structure that comprises at least part of a GLF data file **202**. In one implementation, a listings data inputter **302** and a lineup data inputter **304** are communicatively coupled with a listings data element interlocker **306** and a lineup data element interlocker **308**, respectively. The listings data element interlocker **306** and the lineup data element interlocker **308** are communicatively coupled with a listings and lineup data elements crosslocker **310**. The two interlockers **306**, **308** and the crosslocker **310** are communicatively coupled with and/or have access to the GLF metadata **300**. A GLF data file outputter **312** produces a GLF data file **202** or a combination of GLF data file component files that can be processed by a receiving entity (**108** of FIG. **2**) into usable programming **109**. The GLF data file outputter **312** may also perform validation of interlocked and crosslocked elements and attributes, including data type validation. The inputters **302**, **304** may also perform the validation of data that is input into the exemplary GLF data file engine **201**.

The purpose of the interlockers **306**, **308** and the crosslocker **310** is to provide completeness and validity to the GLF data file **202** or, in other words, to create a data structure in which expected data elements and attributes for each program are present, complete, and in proper form. The format for providing this completeness and validity is supplied by the GLF metadata **300**. In some implementations of the subject matter, the GLF metadata **300** is a set of rules, a set of tags, one or more templates, and/or one or more markup language schemas for gathering complete information about a program, particularly complete associated lineup information. Specifically, the GLF metadata **300** aims to effect construction of valid channel definitions for a given geographical or geopolitical area where the program will be "broadcast." When the listings data inputter **302** is presented with a program title or unique ID number, therefore, the listings data element interlocker **306** may link other required fields to the prospective program and prompt the listings data inputter **302** and the lineup data inputter **304** to "fill in the blanks" presented by these other required fields before the GLF data file **202** can be considered valid (e.g., pass validity tests, etc.). These consistent and strictly enforced metadata rules with their insistence on completeness enable the GLF data file engine **201** to avoid haphazard insertion of lineup data, which causes incorrect presentation of information, program corruption, and frozen systems.

In some implementations, the GLF metadata **300** takes the form of markup information in a language vehicle, such as extensible markup language (XML). In the case of XML, the elements and attributes of the markup language may be logically linked, for example, by key and keyref constraints that establish and enforce GLF completeness and validity.

It should be noted that if a markup language such as XML is selected as the vehicle for an exemplary GLF, then the GLF data file **202** may validate against one or more GLF.xsd files, as will be discussed more fully below. Also, when the GLF is cast in such a markup language, then all the tools and resources associated with the language may be used to conveniently form and validate GLF data structures and data types.

The listings data element interlocker **306** links listings data elements and attributes with other listings data elements and attributes. For example, if a listings data element is the program name, e.g., and episode of "Star Trek," then the listings data element interlocker **306** may require a second listings data element, such as the duration of the Star Trek episode, to be supplied to the GLF data file engine **201** and correctly entered as content in a listings data element.

Likewise, the lineup data element interlocker **308** links lineup data elements and attributes with other lineup data elements and attributes. For example, if a lineup data element is a geographical area, e.g., a postal zip code area in San Jose, Calif., then the lineup data element interlocker **308** may require a second lineup data element, such as the name of a headend available in the given zip code area, to be supplied to the GLF data file engine **201** and correctly entered as content into a lineup data element.

The listings and lineup data elements crosslocker **310** links a listings data element or attribute with one or more lineup data elements or attributes. For example, a unit of schedule information (a listings datum) may be relevant only with regard to a certain channel (a lineup datum). The crosslocker **310** requires the channel lineup datum to be supplied and correctly entered once the unit of schedule listings datum is present. Since the crosslocker **310** as well as the interlockers **306**, **308** establish and strictly enforce the GLF described by the GLF metadata **300**, an entire GLF data file **202** (or set of GLF data files) contains consistently applied interlocks and crosslocks providing programming information that is complete and valid for an intended geographical or geopolitical area in which specific language(s) are spoken and in which only some channels but not others are available.

FIG. **4** shows formation of an exemplary GLF data file **202** in which listings **402** and lineups **404** are formed from GLF metadata **300** residing as a data structure in a computing device memory **400**. A computing device having memory **400** and suitable as an environment for practicing the subject matter will be described in detail in relation to FIG. **9** below. In the implementation illustrated in FIG. **4**, listings **402** may contain subcategories of program information **406** and sched-

ule information **408**. Lineups **404**, in this implementation, may contain subcategories of areas **410** (location information), headends **411**, and channel lineup(s) **412** (per headend). Headends define a set of channels available at one source, such as channels on a particular cable service. Examples of headends are: a DirecTV national feed, over-the-air channels for a specific area, and channel offerings of a cable company. Channel lineups (per headend) form the relation between headends and channels. Areas refer to a geographical or geopolitical region in which certain headends exist.

The presence of both the listings **402** and lineups **404** allow the creation of valid channels **413** for stated locations. Incoming data to be formatted according to the exemplary GLF metadata **300** may be selected or formed from listings **402**, lineups **404**, and/or any of the subcategories **406**, **408**, **410**, **411**, **412**.

Within an exemplary GLF data structure, listings data **414** are crosslocked with lineup data **416**, that is, linkages are comprehensively established between fields for listings data elements and relevant and/or supporting fields for lineup data elements. Furthermore, the fields for the elements and sub-elements within the listings data **414** may be interlocked with each other, in addition to being crosslocked with the fields for the elements and sub-elements of the lineup data **416**. Likewise, within the lineup data **416**, fields for elements and sub-elements may be interlocked with each other, as well as crosslocked with fields for the elements and sub-elements of the listings data **414**.

For example, if the listings data "element **1**" **418** is a television episode, for example "Star Trek," then this episode may only be available via a satellite programming distributor, and so the listings data "element **1**" **418** is crosslocked with lineup data "element **1**" **420**, in this case representing the satellite programming distributor. Lineup data "element **1**," in turn, is interlocked to a lineup data sub-element **422** that represents, for example, the specific channel on which the Star Trek episode will be broadcast via satellite. Likewise, returning to listings data "element **1**" **418**, the Star Trek episode is interlocked with a listings data sub-element **424** that represents the time of day that the Star Trek episode will be broadcast. The particular illustrated implementation of interlocking and crosslocking fields for data elements is only meant to illustrate subject matter, in other exemplary GLFs the interlocking and crosslocking may vary.

As information is added, interlocking and crosslocking linkages between fields for data elements are applied throughout the growth of an exemplary GLF data structure as specified by the exemplary GLF metadata **300**. Thus, for example, listings data "element **2**" **426** is crosslocked with lineup data "element **2**" **428** and interlocked with listings data "element **1**" **418**; listings data "element **3**" **430** is crosslocked with lineup data "element **3**" **432** and interlocked with listings data "element **1**" **418** and listings data "element N" **434**; and listings data "element N" **434** is crosslocked with lineup data "element N" **436** and interlocked with listings data "element **3**" **430**. Likewise, fields for the lineup data elements and sub-elements may be interlocked with each other, in addition to being crosslocked with the listings data elements.

FIG. **5** shows one implementation of a data structure in an exemplary GLF data file **202** in which the crosslocked and interlocked linkages between fields for data elements include "required field" logical relationships. If a first data element is present, such as the illustrated lineup "element **1**" **502** then data fields linked to lineup "element **1**" **502** must also be included and supplied with correctly entered data in order for data file being created to be valid. "Required fields" linked to lineup "element **1**" **502** may include listings elements, such as listings "element **2**" **504**; listings sub-elements, such as "sub-element A" **506**; other lineup elements, such as lineup "element **2**" **508**; and/or lineup sub-elements, such as sub-elements **510**, **512**, **514**, **516** and sub-sub-element **518**, etc. Thus, because the data structure of the exemplary GLF **202** comprises at least some interlocking fields, various sections of the exemplary GLF **202** data structure will be present or absent in an "all-or-nothing" manner. When lineup "element **1**" **502** is added to the GLF **202** data structure then data structure section **520** must also be present in its entirety, in this implementation. An IDP creating the GLF data file **202** may be prompted to add information to complete the data structure section **520** that accompanies or surrounds a new data element, such as lineup "element **1**" **502**.

Exemplary XSD Implementation of the GLF

As mentioned above, in one implementation the exemplary GLF metadata **300** includes an XML schema definition (XSD) specification and a set of editorial instructions. A data file whose structure conforms to the GLF.xsd schema offers many features, which will now be described. In XML, data is hierarchically structured. This hierarchical structure is an excellent vehicle for programming content and EPG information, which is also hierarchically structured. An exemplary data file conforming to the GLF.xsd schema facilitates correct generation of data files that a receiving entity **108** having XML database (XDB) capability can import and process to obtain consistent, standardized, reliably complete, and substantially error-free programming data **109**. For example, the programming data **109** is complete relative to schedule information **408** because the GLF.xsd schema directs that schedule data represent a contiguous block of time, wherein all channels that are included exist in the relevant geographical or geopolitical area and are supplied with programming for the entire time range included in the data file, with no gaps.

An exemplary GLF data file **202** conforming to the GLF.xsd schema requires no additional import work or query tuning and has a rich enough structure to encode currently known programming attributes as well as new attributes as they arise. Further, the GLF.xsd schema is designed to handle international listing data (i.e., in various languages) via XML's XSD facilities. The GLF.xsd schema is also operating system independent: however, XML and XSD validation may be required in some implementations.

In an exemplary GLF.xsd schema data structure, basic listings relationships are expressed according to strict form. Strict validation is provided to ensure correct data generation and delivery. Although strict form is used, the GLF.xsd schema is relatively compact and efficient, but not to the point where data generation and data validation suffer. A receiving entity **108** using an exemplary GLF schema parses and validates XML documents against world wide web consortium (W3C) XSD compliant schemas. Use of Microsoft's XML 4 (MSXML4) is recommended in some implementations (Microsoft Corporation, Redmond, Wash.). The exemplary GLF.xsd schema enforces referential integrity constraints and required fields on the underlying data and defines additional structure for features that may not be supported by every IDP but must still be rigorously defined. GLF data files **202** conforming to an exemplary GLF.xsd schema may be stored in more than one programming data file to facilitate electronic transfer.

FIG. **6** shows exemplary GLF metadata **300**, which includes and/or comprises an exemplary GLF.xsd schema **600**. The illustrated exemplary GLF.xsd schema **600** is an aggregate that can be broken up into components: a GLFListings.xsd component **602**, a GLFLineups.xsd component **604**, and a GLFFundamentals.xsd component **606**.

The GLFFundamentals.xsd component **606** contains basic data type definitions **607** used in the aggregate GLF.xsd schema **600**. The GLFListings.xsd component **602** defines the portion of the data related to listings **402**, and the GLFLineups.xsd component **604** defines the portion of the data related to lineups **404**. The GLFListings.xsd component **602** and the GLFLineups.xsd component **604** are joined by a common data entity, channels, i.e., valid channel definitions **608**.

In one implementation, the GLFListings.xsd component **602**, the GLFLineups.xsd component **604**, and the GLFFundamentals.xsd component **606** can be used as separate modules. Because listings and lineups are often produced by different groups of people, that is, different types of IDPs, the division of the aggregate GLF.xsd schema **600** into a GLFListings.xsd component **602** and a GLFLineups.xsd component **604** may be useful in allowing data files conforming to the components **602**, **604** of an aggregate GLF.xsd schema **600** to be produced in different places, by different entities, and/or at different times, and then combined later by the receiving entity **108** to form the aggregate programming data file conforming to the GLF.xsd schema **600**. This is possible because the GLF data structure is consistent and the component parts intra-compatible regardless of where the component piece originate. An IDP that provides only listings information may use only the GLFListings.xsd component **602** (and the GLFFundamentals.xsd component **606** from which definitions are imported). Likewise, an IDP that provides only lineup information may use only the GLFListings.xsd component **604** (and the GLFFundamentals.xsd component **606**).

The various .xsd schema definition file components **602**, **604**, **606** are self-contained, i.e., they do not need to refer to outside files for schema definition. Thus, the aggregate GLF.xsd schema **600** can combine listings, lineup, channel definition, and data type definition components into a single file representation suitable for electronic transfer or alternatively, as discussed above, the components **602**, **604**, **606** can be electronically transferred independently as separate files.

Referential Structure of an Exemplary XSD Implementation of GLF

In one exemplary XSD implementation, the GLF metadata **300** defines three basic entities with primary keys (also called primary "IDs"). These are "program," i.e., the definition of television programming, including fields such as title and description; "channel," i.e., the definition of television programming source, e.g. a broadcaster; and "headend," i.e., the definition of a group of television programming sources. Program, channel, and headend keys (or IDs) are intended to be persisted indefinitely for each IDP **102**, **104**, **106** and not reused, i.e., each provider is responsible for managing their own ID space. This helps to ensure consistency across language translation, and prevents multiplication of IDs for a single program. Such consistency is important for capabilities such as automatic recording of programs. If a program has already been recorded, it would not be desirable to record it again because an additional program ID number has been unnecessarily generated.

The three basic entities defined above are further related by three additional entities that tie the data together. These are "schedule," i.e., the definition of the program start time and duration on a channel; "lineup," i.e., the definition of sets of channels associated with a specific headend; and "areas," i.e., the definition of headend mapping according to locale, for example by postal code.

The structured relationships between the three primary entities and the three additional entities comprise metadata for the GLF self-consistency mechanism. The self-consistency metadata expressed as a referential structure establishes and enforces referential integrity of data between the three primary entities and the three additional entities by means of key and keyref constraints placed at appropriate levels in the XSD. These constraints ensure that all data elements referenced in a relational element (e.g. a schedule element) must be present in the programming data file or the file will not be considered valid.

A representation of an exemplary XML programming data file conforming to the GLF.xsd schema **600** appears in Appendix A: "Exemplary GLF Data File Sample Which Conforms to the GLF Specification." This representation contains only an abbreviated selection of elements, attributes, and corresponding values. Accordingly, a typical GLF data file **202** conforming to the GLF.xsd schema **600** can contain more entries than those shown, and/or different elements, attributes and/or corresponding values.

Expandability of the Exemplary XSD Implementation of the GLF

The self-referential data structure imparted by an exemplary GLF.xsd schema **600** is expandable with regard to both listings and lineups. In an exemplary GLF.xsd schema **600**, listings **602** may consist of program information **406** and schedule information **408**, as discussed above with reference to FIG. **4**.

FIG. **7** shows another implementation of an exemplary GLFListings.xsd component **602** in which the structure of program information **406** can be enriched by linking optional sub-elements and/or attributes, such as program flags **702**, schedule flags **704**, program values **706**, schedule values **708**, program roles **710**, program names **712**, program categories **714**, and various language translations **716**, **718**, **720**, **722**, **724**.

A flag **702** is a Boolean mechanism used to express general attributes of a program when the domain of possible values that the attribute can assume is small, e.g., television ratings, such as MPAA and Star ratings in the United States, can be represented by a small set of flags corresponding to each possible value.

The program values **706** and schedule values **708** are general-purpose key-value mechanisms used to express attributes of a program or schedule when the domain of possible values that the attribute can assume is relatively large, e.g., a program's year of release, or a schedule attribute of "primetime Monday." The extensibility of key-value pairs is applicable for both program and schedule types. Any number of key-value pairs may be assigned to a program information entry (**406**) or a schedule information entry (**408**). The schedule values **708** are similar to the program values **706** except that they are for specific airings of a program, e.g., "sponsored by" and "company name."

Program roles **710** and program names **712** associate cast, crewmembers, and some functions with a program. For example, the names of the writer, producer, and director may be stored in roles as well as casting information, such as "William Shatner as Captain James T. Kirk." Optionally, an explicit attribute for ordering the names and roles is also included. There may also be optional provisions for supporting multiple translations of roles for presentation. Program roles **710** are expressed in this manner to provide strict schema validation and to provide a compact representation without arbitrary restrictions on the number of program roles **710** associated with a program or person. A program map **726** and/or a schedule map **728** may be included in the GLF

structure to link program roles **710** and program names **712** to the various program information (**406**) data elements, such as program ID.

Program categories **714** are a set of references to a hierarchical category structure used to group related programs. For example, it may be useful to categorize all news programs, all sports programs etc. for EPG menuing. Program categories **714** are important attributes for searching. Instead of having to search for a program by title, a user can search a set or tree of hierarchies and find the program without knowing the title.

The various translations **716**, **718**, **720**, **722**, **724** of text fields (e.g., program title) optional sub-elements, and/or attributes may also be associated with program information **406** and schedule information **408**. If information about a program is available in more than one language, additional program translations can be assigned to a program's unique "program ID" number to accommodate the various languages.

Schedule information **408** can include the program key (ID) reference, the channel key (ID) reference, a start time, and a duration in seconds, and can map programs to channels by means of the program ID reference, the channel ID reference, and the start time. Like program information **406**, schedule information **408** can be expanded with even more optional sub-elements and/or attributes than illustrated.

Schedule flags **704** are similar to program flags **702** but exist for specific airings of a program. Thus, a schedule flag **704** is a Boolean mechanism used to express general attributes of an airing when the domain of possible values that the attribute can assume is small, e.g., "closed captioned" which is either true or false. Any number of Boolean schedule flags **704** may be assigned to a schedule entry in the schedule information **408**. Some examples include: U.S. "Vchip" Ratings, U.S. Content Rating, Canadian TV Ratings, Dolby Digital, HDTV, and "Letterbox." A schedule flag **704** may be bound to another schedule flag to indicate, for example, that a given program is in Dolby on Tuesday, but only in regular stereo on Wednesday. Schedule flags **704** may also show the number of parts of a multipart program, or indicate that the program is edited for television. Flag language translation(s) **724** may accompany the schedule flags **704**.

Like listings **402**, lineups **404** are also expandable via optional sub-elements and attributes, such as lineup type, tuner position, channels per tuner position, and geographical and/or geopolitical areas, which will now be described.

Lineup types include analog lineup (used for terrestrial broadcast, digital and analog cable), analog satellite (used for large format analog satellite), DVB, ATSC, and "unmapped" (used for any of the above types when the provider cannot provide tuning information). Areas define the availability of lineups (headends, viz sets of channels) in specific geographical and/or geopolitical areas.

Tuner positions are optional data elements assigned to channels in a lineup. Tuner position data elements represent information used to tune the receiver to the specified channel. This data is generally applicable only in situations where the receiver supports tuning by number (frequency). In many countries televisions are equipped with radio button style channel selectors that must be configured by the user prior to use (this is similar to old styles of video cassette recorders with "per channel" tuning). An expansion of tuner position is "channels per tuner position" data elements, i.e., more than one channel source occupying a tuner position. These are usually of two types: scheduled and non-scheduled. Scheduled scenarios include programming during primetime followed by a switch to adult programming after a specified time. Non-scheduled means that two or more channels occupy the same tuner position and either could be on at a given time, for example, in the case of PPV channels running free previews on top of paid programming.

GLF Methods

FIG. **8** shows an exemplary method **800** for formatting programming data according to the teachings of the subject matter. This method **800** can be performed by a module, such as the exemplary GLF data file engine **201** shown in FIGS. **2-3**. In the flow diagram, the operations are summarized in individual blocks. The operations may be performed in hardware and/or as machine-readable instructions (software or firmware) that can be executed by a processor.

At block **802**, a first listings element is received. A listings element may be any data or information related to programming content or the scheduling thereof. For example, a listing element can include the program name, episode number, year of creation, cast, acting roles, crew, ratings, category, length, start time, frequency, etc. The GLF, which may be in the form of GLF metadata **300**, aims to provide comprehensive programming information, so that a given local distributor receiving a GLF data file **202** has all the information at hand that might be needed to provide programming and EPG information in any degree of desired detail.

At block **804**, listings elements and lineup elements associated with the first listings element are received. The GLF, because of its self-referential structure, ensures completeness of data inclusion. If a data element is required but not entered into a linked field, then the GLF data file **202** is considered invalid. Usually, IDPs have the desired or required information needed to "fill out" the GLF on hand, but proprietary file formatting results in missing parts and/or errors.

At block **806**, listings and lineup elements are validated. At some point in the process of creating a GLF data file **202**, the various elements and attributes being added to a GLF data structure are verified and/or validated with regard to proper form, including use of proper data types. In some implementations, the validation of block **806** could occur late in the process, but most likely implementation check for valid data types and proper form, including data integrity, checksum matching, etc. upon input. In an exemplary manual input implementation, in which listings and lineup elements are input "by hand" or with human supervision via a user interface or a programming environment, an incorrect data type, for example, can be identified and rejected immediately upon attempted insertion into the GLF data structure. In other implementations, incorrectly formatted data fields or content are identified and/or rectified automatically and/or electronically.

At block **808**, the associated listings and lineup elements are linked with the first listings element. In other words, a section of a GLF data structure, such as section **520** shown in FIG. **5**, is thoroughly filled out with elements and attributes, or at least "required field" sections of the data structure section are filled out.

It should be noted that the interlocks and crosslocks established between various elements of the GLF data structure are logical, not necessarily physical, in nature and so size does not prevent an exemplary large GLF data file **202** from being electronically shipped in parts, for example, in the GLFListings.xsd file **602**, the GLFLineup.xsd file **604**, and the GLFFundamentals.xsd file **606** pieces shown in FIG. **6**.

Exemplary Computing Device

With reference to FIG. **9**, the components of computer **900** may include, but are not limited to, a processing unit **920**, a system memory **930**, and a system bus **921** that couples various system components including the system memory to the processing unit **920**. The system bus **921** may be any of

several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISAA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as the Mezzanine bus.

Computer **900** typically includes a variety of computer-readable media. Computer-readable media can be any available media that can be accessed by computer **900** and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer **900**. Communication media typically embodies computer-readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

The system memory **930** includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) **931** and random access memory (RAM) **400**. A basic input/output system **933** (BIOS), containing the basic routines that help to transfer information between elements within computer **900**, such as during start-up, is typically stored in ROM **931**. RAM **400** typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit **920**. By way of example, and not limitation, FIG. **9** illustrates operating system **934**, application programs **935**, an exemplary GLF data file engine **201**, other program modules **936**, and program data **937**. Although the exemplary GLF data file engine **201** is depicted as software in memory **400**, other implementations of an exemplary GLF data file engine **201** can be hardware or combinations of software and hardware.

The computer **900** may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. **9** illustrates a hard disk drive **941** that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive **951** that reads from or writes to a removable, nonvolatile magnetic disk **952**, and an optical disk drive **955** that reads from or writes to a removable, nonvolatile optical disk **956** such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive **941** is typically connected to the system bus **921** through a non-removable memory interface such as interface **940**, and magnetic disk drive **951** and optical disk drive **955** are typically connected to the system bus **921** by a removable memory interface such as interface **950**.

The drives and their associated computer storage media discussed above and illustrated in FIG. **9** provide storage of computer-readable instructions, data structures, program modules, and other data for computer **900**. In FIG. **9**, for example, hard disk drive **941** is illustrated as storing operating system **944**, application programs **945**, other program modules **946**, and program data **947**. Note that these components can either be the same as or different from operating system **934**, application programs **935**, other program modules **936**, and program data **937**. Operating system **944**, application programs **945**, other program modules **946**, and program data **947** are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer **900** through input devices such as a keyboard **962** and pointing device **961**, commonly referred to as a mouse, trackball, or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **920** through a user input interface **960** that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port, or a universal serial bus (USB). A monitor **991** or other type of display device is also connected to the system bus **921** via an interface, such as a video interface **990**. In addition to the monitor, computers may also include other peripheral output devices such as speakers **997** and printer **996**, which may be connected through an output peripheral interface **995**.

The computer may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **980**. The remote computer **980** may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer **900**, although only a memory storage device **981** has been illustrated in FIG. **9**. The logical connections depicted in FIG. **9** include a local area network (LAN) **971** and a wide area network (WAN) **973**, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets, and the Internet.

When used in a LAN networking environment, the computer **900** is connected to the LAN **971** through a network interface or adapter **970**. When used in a WAN networking environment, the computer **900** typically includes a modem **972** or other means for establishing communications over the WAN **973**, such as the Internet. The modem **972**, which may be internal or external, may be connected to the system bus **921** via the user input interface **960**, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer **900**, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. **9** illustrates remote application programs **985** as residing on memory device **981**. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

CONCLUSION

It should be noted that the subject matter described above can be implemented in hardware, in software, or in both

hardware and software. In certain implementations, the exemplary system and related methods may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The subject matter can also be practiced in distributed communications environments where tasks are performed over wireless communication by remote processing devices that are linked through a communications network. In a wireless network, program modules may be located in both local and remote communications device storage media including memory storage devices.

The foregoing discussion describes exemplary systems and methods for a GLF for programming content and EPG information. Although the invention has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the claimed invention.

The invention claimed is:

1. A method implemented on a computing device by a processor configured to execute instructions that, when executed by the processor, direct the computing device to perform acts comprising:

formatting multimedia programming information having listing elements and lineup elements according to a global listings format (GLF), the GLF causing each listing element to be validated and linked with associated listings elements and linked with associated validated lineup elements to provide complete and valid programming information for an area;

receiving a listings element from a first Independent Data Provider;

receiving associated listings elements and associated lineup elements to link to the listings element from one or more second Independent Data Providers;

prompting the one or more second Independent Data Providers for the associated listings elements and the associated lineup elements when the associated listings elements and the associated lineup elements are missing, according to the GLF;

validating the listings element, the associated listings elements, and the associated lineup elements, by the processor, against a predefined schema that enforces referential integrity constraints and required fields on the data representing the listings elements and the lineup elements, the predefined schema including:

a listings component including:

schedule information comprising a channel identifier, a start time, and a duration; and

program information comprising a program identifier and a program name;

a lineups component including the area, available headends, and a channel lineup per one of the headends, the headends defining a set of channels available at a source, and the area being a region in which a one of the headends exists;

a common data entity including channels, the common data entity joining the listings component and the lineups component;

a fundamentals component, including basic data type definitions, that defines a structure of the data in the listings component and the lineups component;

linking the associated listings elements and the associated lineup elements with the listings element according to the GLF; and

transferring the formatted multimedia programming information to a local programming distributor;

wherein the linking includes creating a logical relationship between one of the associated listings elements and one of the associated lineup elements such that presence of the one of the associated listings elements requires presence of the one of the associated lineup elements.

2. The method as recited in claim **1**, wherein the area is a geographical and/or geopolitical region.

3. The method as recited in claim **1**, wherein the area is a region characterized by specific languages spoken in the region.

4. The method as recited in claim **1**, wherein the area is a region characterized by specific multimedia programming channels available in the region.

5. A method for formatting multimedia programming information having listings elements and lineup elements, the method implemented on a computing device by a processor configured to execute instructions that, when executed by the processor, direct the computing device to perform acts comprising:

creating a global listings format (GLF), wherein the GLF:

includes a set of editorial instructions for producing and delivering the listings elements and the lineup elements; and

requires each of the listings elements to be validated and linked with associated validated listings elements and linked with associated validated lineup elements to provide complete and valid programming information for an area;

receiving a listings element from a data provider;

receiving, from the data provider, associated listings elements and associated lineup elements to link to the listings element;

prompting the data provider for the associated listings elements and the associated lineup elements when the associated listings elements and the associated lineup elements are missing, according to the GLF;

validating the listings element, the associated listings elements, and the associated lineup elements, by the processor, against a predefined schema comprising:

a listings component including:

schedule information including a channel identifier, a start time, and a duration; and

program information including a program identifier and a program name;

a lineups component including the area, available headends, and a channel lineup per one of the headends, the headends defining a set of channels available at a source, and the area being a region in which one of the headends exists;

a common data entity including channels, the common data entity joining the listings component and the lineups component; and

a fundamentals component, including basic data type definitions, that defines a structure of the data in the listings component and the lineups component; and

linking the associated listings elements and the associated lineup elements with the listings element according to the GLF, the linking including creating a logical relationship between one of the associated listings elements and one of the associated lineup elements such that

presence of the one of the associated listings elements requires presence of the one of the associated lineup elements;

wherein each listings element and lineup element is validated if data conforming to the editorial instructions exists for all the linked listings elements and for all the linked lineup elements, the listings elements are data related to a programming content, the lineup elements are data related to sets of channels on which the programming content is implemented, and the lineup elements indicate the area.

6. The method as recited in claim **5**, wherein the area is a geographical and/or geopolitical region.

7. The method as recited in claim **5**, wherein the area is a region characterized by specific languages spoken in the region.

8. The method as recited in claim **5**, wherein the area is a region characterized by specific multimedia programming channels available in the region.

9. A method for formatting multimedia programming information having listings elements and lineup elements to provide complete and valid programming information for an area, the method implemented on a computing device by a processor configured to execute instructions that, when executed by the processor, direct the computing device to perform acts comprising:

defining a global listings format (GLF) in an extensible markup language (XML) that includes (i) an XML schema definition (XSD) for producing the listings elements and the lineup elements compliant with the XML XSD, (ii) one or more keys and keyref constraints, and (iii) a requirement that data conforming to the schema to exist for all listings elements and for all lineup elements;

receiving a listings element from a data provider;

receiving, from the data provider, associated listings elements and associated lineup elements;

prompting the data provider for the associated listings elements and the associated lineup elements to receive a complete set of all the listings elements and the lineup elements associated with the received listings element when the associated listings elements and the associated lineup elements are missing, according to the GLF;

parsing and validating the listings element, the associated listings elements, and the associated lineup elements, by the processor, against the XML XSD, wherein the XML XSD includes:

a listings component including:

schedule information including a channel identifier, a start time, and a duration; and

program information including a program identifier and a program name;

a lineups component including the area, available headends, and a channel lineup per one of the headends, the headends defining a set of channels available at a source, and the area being a region in which one of the headends exists;

a common data entity including channels, the common data entity joining the listings component and the lineups component; and

a fundamentals component including basic data type definitions that defines a structure of the data in the listings component and the lineups component;

linking the associated listings elements and the associated lineup elements with the listings element according to the GLF by using XML XSD keys and keyref constraints, the linking including creating a logical relationship between one of the associated listings elements and one of the associated lineup elements such that presence of the one of the associated listings elements requires presence of the one of the associated lineup elements; and

prompting the data provider for the associated listings elements and the associated lineup elements to enable parsing and validation showing conformance with the XML XSD when the associated listings elements and the associated lineup elements are not parsed and validated, according to the GLF;

prompting the data provider for the associated listings elements and the associated lineup elements to complete the linking when the associated listings elements and the associated lineup elements are not linked, according to the GLF;

wherein the listings elements include at least one of a program title, a unique program ID, an episode title, an episode number, a description, a year of creation, a cast, an acting role, a crew, a rating, a category, a length, a start time, or a frequency, and the lineup elements include at least a channel and a location information for the area.

10. The method as recited in claim **9**, wherein the area is a geographical and/or geopolitical region.

11. The method as recited in claim **9**, wherein the area is a region characterized by specific languages spoken in the region.

12. The method as recited in claim **9**, wherein the area is a region characterized by specific multimedia programming channels available in the region.

* * * * *

Anexo II:

Esquema GLF Global Listings Format

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- © 2002  Microsoft Corporation.  All rights reserved. -->
<xs:schema targetNamespace="http://tvservices.microsoft.com/epg/glf"
xmlns:glf="http://tvservices.microsoft.com/epg/glf"
xmlns="http://tvservices.microsoft.com/epg/glf"
xmlns:mstvUrnLineupExtensions="urn:mstvUrnLineupExtensions"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.13">
<xs:import namespace="urn:mstvUrnLineupExtensions"
schemaLocation="./GLFUrnLineupExtensions.xsd"/>
  <xs:element name="glf">
    <xs:annotation>
      <xs:documentation>
        © 2002-2006  Microsoft Corporation.  All rights reserved.
        Global Listings Format (GLF)
        PROVIDER CANDIDATE
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="listings" type="ListingsType">
          <xs:annotation>
            <xs:documentation>Listings Data.</xs:documentation>
          </xs:annotation>
          <xs:key name="GLFProgramKey">
            <xs:annotation>
              <xs:documentation>Program ID Key Definition.  Enforces the
requirement that programs must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:programs/glf:p"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFSchedulesProgramKeyRef" refer="GLFProgramKey">
            <xs:selector xpath="glf:schedules/glf:s"/>
            <xs:field xpath="@p"/>
          </xs:keyref>
          <xs:keyref name="GLFRolesProgramKeyRef" refer="GLFProgramKey">
            <xs:selector xpath="glf:programroles/glf:rpn/glf:r"/>
            <xs:field xpath="@p"/>
          </xs:keyref>
          <xs:key name="GLFProgramFlagsKey">
            <xs:annotation>
              <xs:documentation>Program Flags  Key Definition.  Enforces the
requirement that programflags must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:programflags/glf:pf"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFProgramFlagKeyRef" refer="GLFProgramFlagsKey">
            <xs:selector xpath="glf:programs/glf:p/glf:f"/>
            <xs:field xpath="@id"/>
          </xs:keyref>
          <xs:key name="GLFProgramValuesKey">
            <xs:annotation>
              <xs:documentation>Program Flags  Key Definition.  Enforces the
requirement  that programflags must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:programvalues/glf:pv"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFProgramValuesKeyRef"
refer="GLFProgramValuesKey">
            <xs:selector xpath="glf:programs/glf:p/glf:k"/>
            <xs:field xpath="@id"/>
          </xs:keyref>
          <xs:key name="GLFProgramCategoryKey">
            <xs:annotation>
              <xs:documentation>Program Categories Key Definition.
Establishes that program can be associated with categories up to two levels
deep.</xs:documentation>
            </xs:annotation>
            <xs:selector
xpath="glf:programcategories/glf:c|glf:programcategories/glf:c/glf:c"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFProgramCategoryKeyRef"
refer="GLFProgramCategoryKey">
            <xs:selector xpath="glf:programs/glf:p/glf:c"/>
            <xs:field xpath="@id"/>
          </xs:keyref>
```

```
          <xs:unique name="GLFMSCNameUniqueTopLevel">
            <xs:annotation>
              <xs:documentation>The MSCName optional attribute can be
applied to any category, but each MSCName can only be applied to one top
level category (for backwards compatability).</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:programcategories/glf:c"/>
            <xs:field xpath="@mscname"/>
          </xs:unique>
          <xs:key name="GLFScheduleFlagKey">
            <xs:annotation>
              <xs:documentation>Schedule Flags  Key Definition.  Enforces
the requirement that scheduleflags must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:scheduleflags/glf:sf"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFScheduleFlagKeyRef"
refer="GLFScheduleFlagKey">
            <xs:selector xpath="glf:schedules/glf:s/glf:f"/>
            <xs:field xpath="@id"/>
          </xs:keyref>
          <xs:key name="GLFScheduleValuesKey">
            <xs:selector xpath="glf:schedulevalues/glf:sv"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFScheduleValuesKeyRef"
refer="GLFScheduleValuesKey">
            <xs:selector xpath="glf:schedules/glf:s/glf:k"/>
            <xs:field xpath="@id"/>
          </xs:keyref>
          <xs:key name="GLFRolesKey">
            <xs:annotation>
              <xs:documentation>Program Roles Key Definition.  Enforces the
requiremetn that programroles must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:programroles/glf:roles/glf:role"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFProgramRolesKeyRef" refer="GLFRolesKey">
            <xs:selector xpath="glf:programs/glf:p/glf:r"/>
            <xs:field xpath="@r"/>
          </xs:keyref>
          <xs:key name="GLFNamesKey">
            <xs:selector xpath="glf:programroles/glf:names/glf:n"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFProgramNamesKeyRef" refer="GLFNamesKey">
            <xs:selector xpath="glf:programs/glf:p/glf:r"/>
            <xs:field xpath="@n"/>
          </xs:keyref>
          <xs:unique name="GLFScheduleEntryUnique">
            <xs:annotation>
              <xs:documentation>Enforce schedule entry uniqueness--only one
program can be scheduled on a channel at a specified
time.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:schedules/glf:s"/>
            <xs:field xpath="@s"/>
            <xs:field xpath="@p"/>
            <xs:field xpath="@c"/>
          </xs:unique>
        </xs:element>
        <xs:element name="lineups" type="LineupsType">
          <xs:annotation>
            <xs:documentation>Lineup Data.</xs:documentation>
          </xs:annotation>
          <xs:key name="GLFHeadendKey">
            <xs:annotation>
              <xs:documentation>Headend Key Definition.  Establishes that
headends must have unique ids which are referenced elsewhere in the
schema.</xs:documentation>
            </xs:annotation>
            <xs:selector xpath="glf:headends/glf:h"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="GLFAreasHeadendRef" refer="GLFHeadendKey">
            <xs:selector xpath="glf:areas/glf:a"/>
            <xs:field xpath="@h"/>
          </xs:keyref>
          <xs:keyref name="GLFLineupsHeadendRef" refer="GLFHeadendKey">
```

```
            <xs:selector xpath="glf:channellineups/*"/>
            <xs:field xpath="@h"/>
          </xs:keyref>
        </xs:element>
        <xs:element name="channels">
          <xs:annotation>
            <xs:documentation>Programming source data (e.g. broadcast
stations, cable channels, etc.).</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="c" type="channelType" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Channel entry.</xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="notices" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Optional provider notices</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="provider">
                <xs:annotation>
                  <xs:documentation>Official Provider Name and Optional
Website.  Any attribution, support or legal notices require that this
element be set.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:attribute name="website" type="restrictedURIType"
use="optional">
                    <xs:annotation>
                      <xs:documentation>Official provider
website.</xs:documentation>
                    </xs:annotation>
                  </xs:attribute>
                  <xs:attribute name="name" type="noticeTextType"
use="required">
                    <xs:annotation>
                      <xs:documentation>Official provider company
name.</xs:documentation>
                    </xs:annotation>
                  </xs:attribute>
                </xs:complexType>
              </xs:element>
              <xs:element name="attribution" nillable="true" minOccurs="0"
maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Attribution notices, e.g. "Listings data
provided by ProviderName".  Actual notice text should be wrapped in the
required text element.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:complexContent>
                    <xs:restriction base="noticeContent">
                      <xs:sequence>
                        <xs:element name="lang" type="xs:language">
                          <xs:annotation>
                            <xs:documentation>Language of attribution
notice.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                        <xs:element name="text" type="noticeTextType">
                          <xs:annotation>
                            <xs:documentation>Full attribution text,
localized to language specified.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                      </xs:sequence>
                    </xs:restriction>
                  </xs:complexContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="support" minOccurs="0"
maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Support notices, e.g. "Go to
www.providersupportsite.com to report problems with listings"   Actual
notice text should be wrapped in the required text
element.</xs:documentation>
```

```
                </xs:annotation>
                <xs:complexType>
                  <xs:complexContent>
                    <xs:restriction base="noticeContent">
                      <xs:sequence>
                        <xs:element name="lang" type="xs:language">
                          <xs:annotation>
                            <xs:documentation>Language of support
notice.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                        <xs:element name="text" type="noticeTextType">
                          <xs:annotation>
                            <xs:documentation>Full text of support notice,
localized to language specified.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                      </xs:sequence>
                    </xs:restriction>
                  </xs:complexContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="legal" minOccurs="0" maxOccurs="unbounded">
                <xs:annotation>
                  <xs:documentation>Legal notices, e.g. "Listings data
copyright 2002"   Actual notice text should be wrapped in the required text
element.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:complexContent>
                    <xs:restriction base="legalNoticeContent">
                      <xs:sequence>
                        <xs:element name="type" type="legalNoticeType"
minOccurs="0">
                          <xs:annotation>
                            <xs:documentation>Legal Notice Type.  Currently
limited to be one of tos (terms of service), copyright, or
disclaimer.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                        <xs:element name="pname" type="shortNoticeTextType"
minOccurs="0">
                          <xs:annotation>
                            <xs:documentation>Presentation name of notice,
e.g. "Terms of Service" localized to the language specified in
lang.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                        <xs:element name="url" type="restrictedURIType"
minOccurs="0">
                          <xs:annotation>
                            <xs:documentation>URL of notice (in addition to
or insted of the full text field). </xs:documentation>
                          </xs:annotation>
                        </xs:element>
                        <xs:element name="lang" type="xs:language">
                          <xs:annotation>
                            <xs:documentation>Language of
notice.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                        <xs:element name="text" type="noticeTextType">
                          <xs:annotation>
                            <xs:documentation>Full text of
notice.</xs:documentation>
                          </xs:annotation>
                        </xs:element>
                      </xs:sequence>
                    </xs:restriction>
                  </xs:complexContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="lang" type="xs:language" use="required"/>
      <xs:attribute name="supplier" type="SupplierNameType" use="required"/>
    </xs:complexType>
    <xs:keyref name="AnalogLineupChannelKeyRef" refer="ChannelKey">
      <xs:selector xpath="glf:lineups/glf:channellineups/glf:a"/>
      <xs:field xpath="@c"/>
    </xs:keyref>
```

```
    <xs:keyref name="AnalogSatelliteChannelKeyRef" refer="ChannelKey">
      <xs:selector xpath="glf:lineups/glf:channellineups/glf:s"/>
      <xs:field xpath="@c"/>
    </xs:keyref>
    <xs:keyref name="DVBLineupChannelKeyRef" refer="ChannelKey">
      <xs:selector xpath="glf:lineups/glf:channellineups/glf:d"/>
      <xs:field xpath="@c"/>
    </xs:keyref>
    <xs:keyref name="ATSCLineupChannelKeyRef" refer="ChannelKey">
      <xs:selector xpath="glf:lineups/glf:channellineups/glf:t"/>
      <xs:field xpath="@c"/>
    </xs:keyref>
    <xs:keyref name="UnmappedLineupChannelKeyRef" refer="ChannelKey">
      <xs:selector xpath="glf:lineups/glf:channellineups/glf:u"/>
      <xs:field xpath="@c"/>
    </xs:keyref>
    <xs:keyref name="ScheduleChannelKeyRef" refer="ChannelKey">
      <xs:selector xpath="glf:listings/glf:schedules/glf:s"/>
      <xs:field xpath="@c"/>
    </xs:keyref>
    <xs:key name="ChannelKey">
      <xs:selector xpath="glf:channels/glf:c"/>
      <xs:field xpath="@id"/>
    </xs:key>
  </xs:element>
  <xs:annotation>
    <xs:documentation>
      Â© 2002-2006  Microsoft Corporation.  All rights reserved.
      Global Listings Format (GLF)
      Listing Data Definition
      PROVIDER CANDIDATE
    </xs:documentation>
  </xs:annotation>
  <xs:element name="listings" type="ListingsType">
    <xs:annotation>
      <xs:documentation>Represents schedule information and program
information.  References channel information (from
lineups).</xs:documentation>
    </xs:annotation>
    <xs:key name="ProgramKey">
      <xs:annotation>
        <xs:documentation>Program ID Key Definition.  Enforces the
requirement that programs must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:programs/glf:p"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="SchedulesProgramKeyRef" refer="ProgramKey">
      <xs:selector xpath="glf:schedules/glf:s"/>
      <xs:field xpath="@p"/>
    </xs:keyref>
    <xs:keyref name="RolesProgramKeyRef" refer="ProgramKey">
      <xs:selector xpath="glf:programroles/glf:rpn/glf:r"/>
      <xs:field xpath="@p"/>
    </xs:keyref>
    <xs:key name="ProgramFlagsKey">
      <xs:annotation>
        <xs:documentation>Program Flags  Key Definition.  Enforces the
requirement that programflags must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:programflags/glf:pf"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="ProgramFlagKeyRef" refer="ProgramFlagsKey">
      <xs:selector xpath="glf:programs/glf:p/glf:f"/>
      <xs:field xpath="@id"/>
    </xs:keyref>
    <xs:key name="ProgramValuesKey">
      <xs:annotation>
        <xs:documentation>Program Values  Key Definition.  Enforces the
requirement  that programflags must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:programvalues/glf:pv"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="ProgramValuesKeyRef" refer="ProgramValuesKey">
      <xs:selector xpath="glf:programs/glf:p/glf:k"/>
      <xs:field xpath="@id"/>
    </xs:keyref>
    <xs:key name="ProgramCategoryKey">
```

```
      <xs:annotation>
        <xs:documentation>Program Categories Key Definition.  Establishes
that program can be associated with categories up to two levels
deep.</xs:documentation>
      </xs:annotation>
      <xs:selector
xpath="glf:programcategories/glf:c|glf:programcategories/glf:c/glf:c"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="ProgramCategoryKeyRef" refer="ProgramCategoryKey">
      <xs:selector xpath="glf:programs/glf:p/glf:c"/>
      <xs:field xpath="@id"/>
    </xs:keyref>
    <xs:unique name="MSCNameUniqueTopLevel">
      <xs:annotation>
        <xs:documentation>The MSCName optional attribute can be applied to
any category, but each MSCName can only be applied to one top level category
(for backwards compatability).</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:programcategories/glf:c"/>
      <xs:field xpath="@mscname"/>
    </xs:unique>
    <xs:key name="SchedulesFlagKey">
      <xs:annotation>
        <xs:documentation>Schedule Flags  Key Definition.  Enforces the
requirement that scheduleflags must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:scheduleflags/glf:sf"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="SchedulesFlagKeyRef" refer="SchedulesFlagKey">
      <xs:selector xpath="glf:schedules/glf:s/glf:f"/>
      <xs:field xpath="@id"/>
    </xs:keyref>
    <xs:key name="ScheduleValuesKey">
      <xs:selector xpath="glf:schedulevalues/glf:sv"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="ScheduleValuesKeyRef" refer="ScheduleValuesKey">
      <xs:selector xpath="glf:schedules/glf:s/glf:k"/>
      <xs:field xpath="@id"/>
    </xs:keyref>
    <xs:key name="RolesKey">
      <xs:annotation>
        <xs:documentation>Program Roles Key Definition.  Enforces the
requiremetn that programroles must have unique ids which are referenced
elsewhere in the schema.</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:programroles/glf:roles/glf:role"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="ProgramRolesKeyRef" refer="RolesKey">
      <xs:selector xpath="glf:programs/glf:p/glf:r"/>
      <xs:field xpath="@r"/>
    </xs:keyref>
    <xs:key name="NamesKey">
      <xs:selector xpath="glf:programroles/glf:names/glf:n"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="ProgramNamesKeyRef" refer="NamesKey">
      <xs:selector xpath="glf:programs/glf:p/glf:r"/>
      <xs:field xpath="@n"/>
    </xs:keyref>
    <xs:unique name="ScheduleEntryUnique">
      <xs:annotation>
        <xs:documentation>Enforce schedule entry uniqueness--only one
program can be scheduled on a channel at a specified
time.</xs:documentation>
      </xs:annotation>
      <xs:selector xpath="glf:schedules/glf:s"/>
      <xs:field xpath="@s"/>
      <xs:field xpath="@p"/>
      <xs:field xpath="@c"/>
    </xs:unique>
  </xs:element>
  <xs:complexType name="ListingsType">
    <xs:sequence>
      <xs:element name="schedules">
        <xs:annotation>
          <xs:documentation>Set of schedule entries.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
```

```
          <xs:sequence>
            <xs:element name="s" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Schedule entry.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:annotation>
                  <xs:documentation>
                    scheduleType.  Fundamental schedule entry type.  Defines
relationship between programs and channels
                  </xs:documentation>
                </xs:annotation>
                <xs:sequence>
                  <xs:element name="f" minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                      <xs:documentation>Schedule Flag entry.  This element
associates a binary attribute with this schedule entry.</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                      <xs:complexContent>
                        <xs:restriction base="glf:scheduleFlagReference"/>
                      </xs:complexContent>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="k" minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                      <xs:documentation>Schedule Value entry.  This element
makes a key/value association with this schedule entry.</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                      <xs:complexContent>
                        <xs:restriction base="glf:scheduleValueReference"/>
                      </xs:complexContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
                <xs:attribute name="s" type="xs:dateTime" use="required">
                  <xs:annotation>
                    <xs:documentation>UTC Start Time.</xs:documentation>
                  </xs:annotation>
                </xs:attribute>
                <xs:attribute name="d" type="programDuration"
use="required">
                  <xs:annotation>
                    <xs:documentation>Schedule Duration in
seconds.</xs:documentation>
                  </xs:annotation>
                </xs:attribute>
                <xs:attribute name="p" type="programIDType" use="required">
                  <xs:annotation>
                    <xs:documentation>Reference to program
id.</xs:documentation>
                  </xs:annotation>
                </xs:attribute>
                <xs:attribute name="c" type="channelIDType" use="required">
                  <xs:annotation>
                    <xs:documentation>Channel ID
Reference.</xs:documentation>
                  </xs:annotation>
                </xs:attribute>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="programs">
        <xs:annotation>
          <xs:documentation>Set of scheduled programs.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="p" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Program entry.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:annotation>
                  <xs:documentation>programType. Fundamental program entry
type.  Defines primary program level information.</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                  <xs:element name="f" type="programFlagReference"
minOccurs="0" maxOccurs="unbounded">
```

```
                    <xs:annotation>
                      <xs:documentation>Program Flag entry. This is a
mechanism for associating specific binary attribues with programs, e.g.
"repeat"  </xs:documentation>
                    </xs:annotation>
                  </xs:element>
                  <xs:element name="k" type="programValueReference"
minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                      <xs:documentation>Program Value entry. This is a
key/value pair mechanism used to associate additional per program values
with each program entry.  Example: a movie's release
year.</xs:documentation>
                    </xs:annotation>
                  </xs:element>
                  <xs:element name="c" type="programCategoryFlagReference"
minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                      <xs:documentation>Optional categories associated with
this program.</xs:documentation>
                    </xs:annotation>
                  </xs:element>
                  <xs:element name="r" minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                      <xs:documentation>Program roles reference (links to
people and their titles).</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                      <xs:attribute name="o" type="roleOrderType"
use="required">
                        <xs:annotation>
                          <xs:documentation>Specifies the order of cast/crew
members.</xs:documentation>
                        </xs:annotation>
                      </xs:attribute>
                      <xs:attribute name="r" type="roleIDType"
use="required">
                        <xs:annotation>
                          <xs:documentation>RoleID reference
key.</xs:documentation>
                        </xs:annotation>
                      </xs:attribute>
                      <xs:attribute name="n" type="nameIDType"
use="required">
                        <xs:annotation>
                          <xs:documentation>NameID
reference.</xs:documentation>
                        </xs:annotation>
                      </xs:attribute>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="pt" minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                      <xs:documentation>Program translation
entry.</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                      <xs:annotation>
                        <xs:documentation>programTrType. Alternate language
descriptions, titles etc. are defined here.</xs:documentation>
                      </xs:annotation>
                      <xs:attribute name="p" type="programIDType"
use="optional"/>
                      <xs:attribute name="t" type="programTitleType"
use="required">
                        <xs:annotation>
                          <xs:documentation>Translated Primary Title. This
is the full title.</xs:documentation>
                        </xs:annotation>
                      </xs:attribute>
                      <xs:attribute name="rt" type="programTitleType"
use="optional">
                        <xs:annotation>
                          <xs:documentation>Translated Reduced Title.
Optional string for use in limited space situations.</xs:documentation>
                        </xs:annotation>
                      </xs:attribute>
                      <xs:attribute name="d" type="programDescriptionType"
use="optional">
                        <xs:annotation>
                          <xs:documentation>Translated Description. Full
program description.</xs:documentation>
                        </xs:annotation>
```

```
                    </xs:attribute>
                    <xs:attribute name="rd" type="programDescriptionType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>Translated Reduced Description.
Shorter version of full description for limited space
applications.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="et" type="programDescriptionType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>Translated Episode Title. Used
for episodic programs (e.g. Friends).</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="l" type="xs:language"
use="required">
                      <xs:annotation>
                        <xs:documentation>Language
specification.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                  </xs:complexType>
                </xs:element>
                <xs:element name="cr" minOccurs="0" maxOccurs="unbounded">
                  <xs:annotation>
                    <xs:documentation>Program Character Role reference.
Links people to the roles they play in the program, e.g.
'Oedipus'.</xs:documentation>
                  </xs:annotation>
                  <xs:complexType>
                    <xs:attribute name="n" type="nameIDType"
use="required"/>
                    <xs:attribute name="c" type="characterRoleType"
use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="id" type="programIDType" use="required">
                <xs:annotation>
                  <xs:documentation>Primary Key for Program (p) entity.
This attribute (id) is REQUIRED to persist from feed to feed.  Example:  If
a program "CNN Headline News" is assigned id 34, this same id must be used
for "CNN Headline News" in all future data.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="t" type="programTitleType"
use="required">
                <xs:annotation>
                  <xs:documentation>Primary Title. This is the full
title.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="rt" type="programTitleType"
use="optional">
                <xs:annotation>
                  <xs:documentation>Reduced Title. Optional string for use
in limited space situations.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="et" type="programTitleType"
use="optional">
                <xs:annotation>
                  <xs:documentation>Episode Title. Used for episodic
programs (e.g. Friends)</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="d" type="programDescriptionType"
use="optional">
                <xs:annotation>
                  <xs:documentation>Description. Full program
description.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="rd" type="programDescriptionType"
use="optional">
                <xs:annotation>
                  <xs:documentation>Reduced Description. Shorter version
of full description for limited space applications</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="l" type="xs:language" use="optional">
```

```
                    <xs:annotation>
                      <xs:documentation>If primary title, desc, etc. are
provided in a language other than the default of the file, specify the
language here.</xs:documentation>
                    </xs:annotation>
                  </xs:attribute>
                </xs:complexType>
                <xs:unique name="roleOrderUnique">
                  <xs:selector xpath="r"/>
                  <xs:field xpath="@o"/>
                </xs:unique>
                <xs:unique name="programLanguageUnique">
                  <xs:selector xpath=". | pt"/>
                  <xs:field xpath="@l"/>
                </xs:unique>
                <xs:unique name="categoryUnique">
                  <xs:selector xpath="c"/>
                  <xs:field xpath="@id"/>
                </xs:unique>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="programroles" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Set of links from names to roles to programs,
set of names, set of roles.</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="roles">
                <xs:annotation>
                  <xs:documentation>Set of role definitions with optional per
language translations.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="role" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:annotation>
                          <xs:documentation>roleType. Basic encoding unit for
program roles (e.g. actor, director, writer).</xs:documentation>
                        </xs:annotation>
                        <xs:sequence>
                          <xs:element name="rolet" minOccurs="0"
maxOccurs="unbounded">
                            <xs:annotation>
                              <xs:documentation>Optional language specific
role translations.</xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                              <xs:attribute name="lang" type="xs:language"
use="required">
                                <xs:annotation>
                                  <xs:documentation>Language for this role
title/description translation.</xs:documentation>
                                </xs:annotation>
                              </xs:attribute>
                              <xs:attribute name="title" type="rolePNameType"
use="required">
                                <xs:annotation>
                                  <xs:documentation>Alternate language title
for this translation.</xs:documentation>
                                </xs:annotation>
                              </xs:attribute>
                              <xs:attribute name="desc"
type="roleDescriptionType" use="optional">
                                <xs:annotation>
                                  <xs:documentation>Alternate language
description for this role translation.</xs:documentation>
                                </xs:annotation>
                              </xs:attribute>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute name="id" type="roleIDType">
                          <xs:annotation>
                            <xs:documentation>RoleID.  Key for role relation.
This key should be consistently assigned from feed ot
feed.</xs:documentation>
                          </xs:annotation>
                        </xs:attribute>
```

```
                    <xs:attribute name="name" type="roleNameType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>Standardized name for role, this
tokenized name is used translation independent
processing.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="title" type="rolePNameType"
use="required">
                      <xs:annotation>
                        <xs:documentation>Presentation title for role.
Examples are "Director", "Lead Actor", "Writer", "Executive Producer".  This
string should be in the default language for the feed.  See Rolet for
specifying alternative translations.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="desc" type="roleDescriptionType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>Detailed role description
(optional).  This field is additional (optional) text space to augment the
title.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="msroleid" type="roleNameType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>Microsoft RoleID. Only reserved
Microsoft names may be used here, e.g. MSEPGR_DIRECTOR.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                  </xs:complexType>
                  <xs:unique name="roleLanguageUnique">
                    <xs:selector xpath="rolet"/>
                    <xs:field xpath="@l"/>
                  </xs:unique>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="names">
            <xs:annotation>
              <xs:documentation>Role name list.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:sequence>
                <xs:element name="n" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:annotation>
                      <xs:documentation>nameType. Basic name encoding unit
(e.g. actor name).</xs:documentation>
                    </xs:annotation>
                    <xs:attribute name="id" type="nameIDType"
use="required">
                      <xs:annotation>
                        <xs:documentation>Name ID (key) field.  This field
should be assigned consistently from feed to feed.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="fname" type="nameComponentType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>First name field.  This field
should be left blank for single names like "Prince", "Cher" or
"Charo".</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="mname" type="nameComponentType"
use="optional">
                      <xs:annotation>
                        <xs:documentation>Middle name or
initial.</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="lname" type="nameComponentType"
use="required">
                      <xs:annotation>
                        <xs:documentation>Lastname (surname).  This field
is required and should be used for people using single names like "Charo" or
"Madonna".</xs:documentation>
                      </xs:annotation>
                    </xs:attribute>
```

```
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="programflags" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Set of Program Flags.  Program Flags are binary
attributes associated with programs.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="pf" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Program Flag
Definition.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="glf:programFlagsType">
                    <xs:sequence>
                      <xs:element name="ft" minOccurs="0"
maxOccurs="unbounded">
                        <xs:annotation>
                          <xs:documentation>Optional Flag
Translations.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                          <xs:complexContent>
                            <xs:extension base="FlagValueDefinitionAT"/>
                          </xs:complexContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="id" type="flagIDType"
use="required"/>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="scheduleflags" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Set of schedule flag definitions.  Flag
attribute in program entreis will map to these
definitions.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="sf" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Schedule Flag
Definition.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="glf:scheduleFlagsType">
                    <xs:sequence>
                      <xs:element name="ft" minOccurs="0"
maxOccurs="unbounded">
                        <xs:annotation>
                          <xs:documentation>Optional Flag
Translations.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                          <xs:complexContent>
                            <xs:extension base="FlagValueDefinitionAT"/>
                          </xs:complexContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="id" type="flagIDType"
use="required"/>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
```

```
        </xs:complexType>
      </xs:element>
      <xs:element name="programcategories" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Defintions of program
categories.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="c" type="programCategoryType"
maxOccurs="unbounded">
              <xs:unique name="ProgramCategoryUnique">
                <xs:annotation>
                  <xs:documentation>Program category ids must be unique for
all levels of the category hierarchy.</xs:documentation>
                </xs:annotation>
                <xs:selector xpath=".//c"/>
                <xs:field xpath="@id"/>
              </xs:unique>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="programvalues" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Set of Program Values.  Program Values are
generic attributes which can be associated with programs.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="pv" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Program Value
Definition.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="glf:programValuesType">
                    <xs:sequence>
                      <xs:element name="vt" minOccurs="0"
maxOccurs="unbounded">
                        <xs:annotation>
                          <xs:documentation>Optional Value
Translations.</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                          <xs:complexContent>
                            <xs:extension base="FlagValueDefinitionAT"/>
                          </xs:complexContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="id" type="flagIDType"
use="required"/>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="schedulevalues" minOccurs="0">
        <xs:annotation>
          <xs:documentation>Schedule Values are generic attributes which can
be associated with specific schedule entries.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="sv" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Schedule Values
Definition.</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="glf:scheduleValuesType">
                    <xs:sequence>
                      <xs:element name="vt" minOccurs="0"
maxOccurs="unbounded">
                        <xs:annotation>
                          <xs:documentation>Optional Value
Translations.</xs:documentation>
                        </xs:annotation>
```

```
                        <xs:complexType>
                          <xs:complexContent>
                            <xs:extension base="FlagValueDefinitionAT"/>
                          </xs:complexContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                    <xs:attribute name="id" type="flagIDType"
use="required"/>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="programCategoryType">
    <xs:annotation>
      <xs:documentation>Basic Category Structure.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="ct" minOccurs="0" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Category Translation.  Provides a mechanism to
store translations of category names.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:attribute name="value" type="catValueType" use="required">
            <xs:annotation>
              <xs:documentation>Translation of parent category
value.</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="lang" type="xs:language" use="required">
            <xs:annotation>
              <xs:documentation>Language specification for
category.</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
      <xs:element name="c" type="programCategoryType" minOccurs="0"
maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>Sub-category definition.</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="id" type="flagIDType" use="required">
      <xs:annotation>
        <xs:documentation>Category ID.  This key value must persist
indefinitely once it has been assigned.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="value" type="catValueType" use="required">
      <xs:annotation>
        <xs:documentation>Presentation name for category, e.g.
"Sports."</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="mscname" type="msCatIDType" use="optional">
      <xs:annotation>
        <xs:documentation>Standard Microsoft Category Name, e.g.
MSEPG_SPORTS.  This field is used to implement filtered guide
capabilities.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:annotation>
    <xs:documentation>
      Â© 2002-2006  Microsoft Corporation.  All rights reserved.
      Global Listings Format (GLF)
      Lineup Data Definition
      PROVIDER CANDIDATE
    </xs:documentation>
  </xs:annotation>
  <xs:element name="lineups" type="LineupsType">
    <xs:annotation>
      <xs:documentation>Represents carriage structure (lineup, area served,
channels)</xs:documentation>
    </xs:annotation>
```

```
  </xs:element>
  <xs:complexType name="LineupsType">
    <xs:annotation>
      <xs:documentation>Grouping element above headends, areas and
channellineups.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="headends">
        <xs:annotation>
          <xs:documentation>Set of headend (lineups), e.g. Echostar, ATT
Cable, etc.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="h" type="headendType" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Lineup (Headend) entry</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="areas">
        <xs:annotation>
          <xs:documentation>Links areas to headends</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="a" type="areaType" minOccurs="0"
maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Area to headend link</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        <xs:key name="glfAreasKey">
          <xs:selector xpath="glf:a"/>
          <xs:field xpath="@a"/>
          <xs:field xpath="@h"/>
        </xs:key>
      </xs:element>
      <xs:element name="channellineups">
        <xs:annotation>
          <xs:documentation>Links channels to headends</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="a" type="analogChannelLineupType"
minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Any Channel Lineup (used for terrestrial
broadcast, digital or analog cable.</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="s" type="analogSatelliteChannelLineupType"
minOccurs="0" maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Analog Satellite Lineup (used for analog
satellite only).</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="d" type="DVBLineupType" minOccurs="0"
maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>DVB Channel Lineup Entry.  This entry
should be used for DVB lineups where digital tuning information will be used
by the client.  If a set top box is used which effectively abstracts this
information (i.e. the client sends IR codes only, use the a
type).</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="r"
type="mstvUrnLineupExtensions:DVBUrnLineupType" minOccurs="0"
maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Used for services which are specified
using the URN format.</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="t" type="ATSCLineupType" minOccurs="0"
maxOccurs="unbounded">
              <xs:annotation>
```

```
                <xs:documentation>ATSC Channel Lineup
Entry.</xs:documentation>
              </xs:annotation>
            </xs:element>
            <xs:element name="u" type="unmappedLineupType" minOccurs="0"
maxOccurs="unbounded">
              <xs:annotation>
                <xs:documentation>Unmapped Channel Lineup
Entry.</xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        <xs:key name="GLFAnalogLineupKey">
          <xs:selector xpath="glf:a"/>
          <xs:field xpath="@h"/>
          <xs:field xpath="@t"/>
        </xs:key>
        <xs:key name="GLFAnalogSatLineupKey">
          <xs:selector xpath="glf:s"/>
          <xs:field xpath="@h"/>
          <xs:field xpath="@t"/>
        </xs:key>
        <xs:key name="GLFATSCLineupKey">
          <xs:selector xpath="glf:t"/>
          <xs:field xpath="@h"/>
          <xs:field xpath="@t"/>
        </xs:key>
        <xs:key name="GLFUnmappedLineupKey">
          <xs:selector xpath="glf:u"/>
          <xs:field xpath="@h"/>
          <xs:field xpath="@c"/>
        </xs:key>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LineupAType" abstract="true">
    <xs:annotation>
      <xs:documentation>Abstract base type for lineup
entries.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="h" type="headendIDType" use="required">
      <xs:annotation>
        <xs:documentation>Headend reference for lineup
entry.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ef" type="xs:date" use="optional">
      <xs:annotation>
        <xs:documentation>Effective date of this entry.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ex" type="xs:date" use="optional">
      <xs:annotation>
        <xs:documentation>Expiration date of this entry.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="st" type="serviceTierType" use="optional">
      <xs:annotation>
        <xs:documentation>Service tier for this lineup entry (e.g. Premium,
PPV, etc.)</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="c" type="channelIDType" use="required">
      <xs:annotation>
        <xs:documentation>Channel reference.  This should be the primary
channel referenced on this tuner position.  Shared channel entries will
override this channel on the days and times specified in the optional "sc"
element sequence.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="analogChannelLineupType">
    <xs:annotation>
      <xs:documentation>Any Channel Lineup Type.  Channel lineup for analog
cable, terrestrial broadcast, and set-top box reception.  This type also can
also express shared channels.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="LineupAType">
        <xs:sequence minOccurs="0">
          <xs:element name="sc" maxOccurs="unbounded">
            <xs:annotation>
```

```
              <xs:documentation>Shared channel element
sequence.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="c" type="channelIDType" use="required">
                <xs:annotation>
                  <xs:documentation>Channel reference for shared
channel.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="st" type="xs:time" use="optional">
                <xs:annotation>
                  <xs:documentation>Start time for this
entry.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="et" type="xs:time" use="optional">
                <xs:annotation>
                  <xs:documentation>End time for this
entry.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
              <xs:attribute name="d" type="effectiveDaysType" use="optional"
default="1234567">
                <xs:annotation>
                  <xs:documentation>Effective days string.  Days are defined
as follows: 1 = Sunday, 2 = Monday, 3 = Tuesday, 4 = Wednesday, 5 =
Thursday, 6 = Friday, 7 = Saturday.  A channel active only on weekends would
be specified as 17.</xs:documentation>
                </xs:annotation>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="t" type="analogTunerPositionType"
use="required">
          <xs:annotation>
            <xs:documentation>Analog tuner position.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="cf" type="carrierFrequencyType" use="optional">
          <xs:annotation>
            <xs:documentation>If channel is being carried on a different
frequency than the base specified at the channel level, enter it
here.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="p" type="presetStringType" use="optional">
          <xs:annotation>
            <xs:documentation>Channel Preset. If a different remote tuning
sequence is needed to tune to this channel, enter it
here.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="DVBLineupType">
    <xs:annotation>
      <xs:documentation>General Channel lineup for DVB (cable, terrestrial,
satellite).</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="LineupAType">
        <xs:sequence minOccurs="0">
          <xs:element name="dsc">
            <xs:annotation>
              <xs:documentation>Shared channel support for DVB/Digital
lineups.  Use this only when an additional channel must be associated with
this tune request.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="c" type="channelIDType" use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="t" type="DVBTSIDType" use="required">
          <xs:annotation>
            <xs:documentation>Integer tuner position.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="s" type="DVBSIDType" use="optional">
          <xs:annotation>
```

```
            <xs:documentation>Source id.  This will be required in a future
release.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="o" type="DVBONIDType" use="optional">
          <xs:annotation>
            <xs:documentation>ONID.  This will be required in a future
release.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="pl" type="DVBPolarisationType" use="optional">
          <xs:annotation>
            <xs:documentation>Polarisation.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="op" type="DVBOrbitalPositionType"
use="optional">
          <xs:annotation>
            <xs:documentation>Orbital position.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="cf" type="carrierFrequencyType" use="optional">
          <xs:annotation>
            <xs:documentation>If channel is being carried on a different
frequency than the base specified at the channel level, enter it
here.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="p" type="presetStringType" use="optional">
          <xs:annotation>
            <xs:documentation>Channel Preset. If a different remote tuning
sequence is needed to tune to this channel, enter it
here.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="m" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>Obsolete field, will be prohibited in a future
release.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="r" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>Obsolete field, will be prohibited in a future
release.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="ATSCLineupType">
    <xs:annotation>
      <xs:documentation>Channel lineup for ATSC. This lineup type should
only be used if the client will directly tune to an ATSC programming source
(i.e. not through a set-top box).</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="LineupAType">
        <xs:sequence minOccurs="0">
          <xs:element name="asc" maxOccurs="unbounded">
            <xs:annotation>
              <xs:documentation>Shared channel support for ATSC Digital
lineups.  Use this only when an additional channel must be associated with
this tune request.</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="c" type="channelIDType" use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        <xs:attribute name="t" type="ATSCTSIDType" use="optional">
          <xs:annotation>
            <xs:documentation>ATSC Channel TSID.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ph" type="ATSCPhysicalChannelType"
use="optional">
          <xs:annotation>
            <xs:documentation>ATSC Physical Channel.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="m" type="ATSCMajorMinorChannelType"
use="optional">
```

```
          <xs:annotation>
            <xs:documentation>Major</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="r" type="ATSCMajorMinorChannelType"
use="optional">
          <xs:annotation>
            <xs:documentation>Minor</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="cf" type="carrierFrequencyType" use="optional">
          <xs:annotation>
            <xs:documentation>If channel is being carried on a different
frequency than the base specified at the channel level, enter it
here.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="p" type="presetStringType" use="optional">
          <xs:annotation>
            <xs:documentation>Channel Preset. If a different remote tuning
sequence is needed to tune to this channel, enter it
here.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
        <xs:attribute name="s" type="xs:nonNegativeInteger" use="optional">
          <xs:annotation>
            <xs:documentation>Source id. This field is obsolete and will be
deleted.</xs:documentation>
          </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="analogSatelliteChannelLineupType">
    <xs:annotation>
      <xs:documentation>Channel lineup for analog
satellite.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="LineupAType">
        <xs:attribute name="t" type="analogSatTunerPositionType"
use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="unmappedLineupType">
    <xs:annotation>
      <xs:documentation>Channel lineup for unmapped headend (no tuner
positions).</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="LineupAType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:annotation>
    <xs:documentation>
      Â© 2002-2006  Microsoft Corporation.  All rights reserved.
      Global Listings Format (GLF)
      Fundamental EPG data types (this schema is intended to be imported by
other schemas).
      PROVIDER CANDIDATE
    </xs:documentation>
  </xs:annotation>
  <!--simpleTypes-->
  <!--Fundamental Identifier simpleTypes-->
  <xs:simpleType name="iso639Type">
    <xs:annotation>
      <xs:documentation>Basic language type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:language">
      <xs:pattern value="\c\c"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="iso3166Type">
    <xs:annotation>
      <xs:documentation>Basic country type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="\c\c"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendIDType">
    <xs:annotation>
```

```
      <xs:documentation>Basic headendID type. This id is required to persist
from feed to feed.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelIDType">
    <xs:annotation>
      <xs:documentation>Basic channelID type. This id is required to persist
from feed to feed.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="programIDType">
    <xs:annotation>
      <xs:documentation>Basic programID type. This id is required to persist
from feed to feed.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="areaIDType">
    <xs:annotation>
      <xs:documentation>Local area reference type, e.g. postal
code.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <!--simpleTypes associated with channels-->
  <xs:simpleType name="channelVPSType">
    <xs:annotation>
      <xs:documentation>Identifier type for VPS
namespace.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:hexBinary">
      <xs:minLength value="2"/>
      <xs:maxLength value="2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelTSIDType">
    <xs:annotation>
      <xs:documentation>Identifier type fo TSID
namespace.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:hexBinary">
      <xs:minLength value="2"/>
      <xs:maxLength value="2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelCNIType">
    <xs:annotation>
      <xs:documentation>Identifier type for CNI
namespace.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:hexBinary">
      <xs:minLength value="2"/>
      <xs:maxLength value="2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelCallLettersType">
    <xs:annotation>
      <xs:documentation>Basic call letters type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:minLength value="1"/>
      <xs:maxLength value="20"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelDisplayNameType">
    <xs:annotation>
      <xs:documentation>Basic type for channel display
name.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
```

```
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelDescriptionType">
    <xs:annotation>
      <xs:documentation>Basic type for channel description display
name.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelAffiliationType">
    <xs:annotation>
      <xs:documentation>Basic type for channel
affiliations.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelProgrammingType">
    <xs:annotation>
      <xs:documentation>Basic type fo channel
programming.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelBroadcastTunerNumberType">
    <xs:annotation>
      <xs:documentation>Basic type of broadcast channel tuner
number.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelVBIType">
    <xs:annotation>
      <xs:documentation>String used to match channels with vbi inband
data.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="channelPPVType">
    <xs:annotation>
      <xs:documentation>Indicator for PPV programming.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:boolean"/>
  </xs:simpleType>
  <!--Schedule / Program Flag/Value simpleTypes-->
  <xs:simpleType name="flagValueIDType">
    <xs:annotation>
      <xs:documentation>Internal ID for schedule flag or schedule
value.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="flagValueNameType">
    <xs:annotation>
      <xs:documentation>Name of flag or value.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="flagValuePNameType">
    <xs:annotation>
      <xs:documentation>Presentation name of flag or
value.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
```

```
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="flagValueDescriptionType">
    <xs:annotation>
      <xs:documentation>Description of flag or value.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="valueType">
    <xs:annotation>
      <xs:documentation>Restricted string for value associated with program
or schedule.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="programFlagNameType">
    <xs:annotation>
      <xs:documentation>Official values for program flag
names.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="glf:flagValueNameType">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="programValueNameType">
    <xs:annotation>
      <xs:documentation>Official values for program value
names.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="glf:flagValueNameType">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="scheduleFlagNameType">
    <xs:annotation>
      <xs:documentation>Official values for schedule flag
names.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="glf:flagValueNameType">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="scheduleValueNameType">
    <xs:annotation>
      <xs:documentation>Official values for schedule value
names.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="glf:flagValueNameType">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <!--simpleTypes associated with programs -->
  <!--Category simpleTypes-->
  <xs:simpleType name="programTitleType">
    <xs:annotation>
      <xs:documentation>Basic Program Title type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="150"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="programDescriptionType">
    <xs:annotation>
      <xs:documentation>Basic Program Description type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="500"/>
    </xs:restriction>
  </xs:simpleType>
```

```
  <xs:simpleType name="catIDType">
    <xs:annotation>
      <xs:documentation>Internal ID for categorys.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="200"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="catValueType">
    <xs:annotation>
      <xs:documentation>Category value (name).</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="msCatIDType">
    <xs:annotation>
      <xs:documentation>Reserved category ID type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
    </xs:restriction>
  </xs:simpleType>
  <!--simpleTypes associated with roles-->
  <xs:simpleType name="roleIDType">
    <xs:annotation>
      <xs:documentation>Basic RoleID type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="roleNameType">
    <xs:annotation>
      <xs:documentation>Standardized name for role, not
displayed.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
      <xs:enumeration value="MSEPGR_ACTOR"/>
      <xs:enumeration value="MSEPGR_DIRECTOR"/>
      <xs:enumeration value="MSEPGR_HOST"/>
      <xs:enumeration value="MSEPGR_GUEST_STAR"/>
      <xs:enumeration value="MSEPGR_PRODUCER"/>
      <xs:enumeration value="MSEPGR_COHOST"/>
      <xs:enumeration value="MSEPGR_SCRIPTWRITER"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="rolePNameType">
    <xs:annotation>
      <xs:documentation>Presentation name for role.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="roleDescriptionType">
    <xs:annotation>
      <xs:documentation>Role description.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="roleOrderType">
    <xs:annotation>
      <xs:documentation>Role ordering type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:positiveInteger"/>
  </xs:simpleType>
  <xs:simpleType name="characterRoleType">
    <xs:annotation>
      <xs:documentation>Character Role type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="100"/>
    </xs:restriction>
```

```
  </xs:simpleType>
  <!--simpleTypes for people-->
  <xs:simpleType name="nameIDType">
    <xs:annotation>
      <xs:documentation>Basic NameID type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="nameComponentType">
    <xs:annotation>
      <xs:documentation>Person name component, viz. first name, last name,
etc.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="programDuration">
    <xs:annotation>
      <xs:documentation>Program running time data type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:positiveInteger">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="86400"/>
    </xs:restriction>
  </xs:simpleType>
  <!--simpleTypes associated with lineup entries -->
  <xs:simpleType name="effectiveDaysType">
    <xs:annotation>
      <xs:documentation>Specification for shared channel effective
days.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="7"/>
      <xs:pattern value="1{0,1}2{0,1}3{0,1}4{0,1}5{0,1}6{0,1}7{0,1}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="analogSatTunerPositionType">
    <xs:annotation>
      <xs:documentation>Specification of analog satellite tuning
string.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z]\d-\d\d"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="carrierFrequencyType">
    <xs:annotation>
      <xs:documentation>Specification of carrier
frequency.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
  <xs:simpleType name="serviceTierType">
    <xs:annotation>
      <xs:documentation>Specification of service tier.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="presetStringType">
    <xs:annotation>
      <xs:documentation>Specification of preset string.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="analogTunerPositionType">
    <xs:annotation>
      <xs:documentation>Specification of analog tuner
position.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
```

```
  </xs:simpleType>
  <xs:simpleType name="ATSCMajorMinorChannelType">
    <xs:annotation>
      <xs:documentation>Specification of major/minor channel
number.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="1023"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ATSCPhysicalChannelType">
    <xs:annotation>
      <xs:documentation>Specification of physical channel
number.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:integer">
      <xs:maxInclusive value="255"/>
      <xs:minInclusive value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ATSCTSIDType">
    <xs:annotation>
      <xs:documentation>Specification of ATSC TSID for
locator.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="DVBONIDType">
    <xs:annotation>
      <xs:documentation>Specification of DVB ONID type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="DVBTSIDType">
    <xs:annotation>
      <xs:documentation>Specification of DVB TSID type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="DVBSIDType">
    <xs:annotation>
      <xs:documentation>Specification of DVB SID type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:unsignedShort"/>
  </xs:simpleType>
  <xs:simpleType name="DVBPolarisationType">
    <xs:annotation>
      <xs:documentation>Specification of DVB Polarization</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="LINEAR_H"/>
      <xs:enumeration value="LINEAR_V"/>
      <xs:enumeration value="CIRCULAR_L"/>
      <xs:enumeration value="CIRCULAR_R"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="DVBOrbitalPositionType">
    <xs:annotation>
      <xs:documentation>Specification of DVB Orbital
Position</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:int">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="1800"/>
    </xs:restriction>
  </xs:simpleType>
  <!--simpleTypes associated with headends -->
  <xs:simpleType name="tuningSubTypes">
    <xs:annotation>
      <xs:documentation>Enumeration of client tuning
types.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="1"/>
      <xs:enumeration value="a">
        <xs:annotation>
          <xs:documentation>Analog.  Headend supports Analog
tuning.</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="d">
```

```
        <xs:annotation>
          <xs:documentation>Digital.  Headend supports digital
tuning.</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="m">
        <xs:annotation>
          <xs:documentation>Mixed.  Indicates headend supports a mix of
analog and digital tuning.</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendDescriptionType">
    <xs:annotation>
      <xs:documentation>Headend description basic type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:maxLength value="100"/>
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendDMAType">
    <xs:annotation>
      <xs:documentation>Headend DMA basic type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:maxLength value="100"/>
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendCommunityType">
    <xs:annotation>
      <xs:documentation>Headend community basic type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendLocationType">
    <xs:annotation>
      <xs:documentation>Headend location basic type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendNameType">
    <xs:annotation>
      <xs:documentation>Headend name basic type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="100"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendTypes">
    <xs:annotation>
      <xs:documentation>Enumeration of headend types (e.g.
Cable).</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="DBS"/>
      <xs:enumeration value="ATSC"/>
      <xs:enumeration value="DVB"/>
      <xs:enumeration value="DVBT"/>
      <xs:enumeration value="DVBS"/>
      <xs:enumeration value="CAB"/>
      <xs:enumeration value="SAT"/>
      <xs:enumeration value="TER"/>
      <xs:enumeration value="IPTV"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="headendDVBSystemType">
    <xs:annotation>
      <xs:documentation>Enumeration of DVB system types.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Cable"/>
      <xs:enumeration value="Terrestrial"/>
      <xs:enumeration value="Satellite"/>
```

```
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="defaultVideoFormat">
    <xs:annotation>
      <xs:documentation>Type for describing default video format for a
headend.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="NTSC_M"/>
      <xs:enumeration value="NTSC_M_J"/>
      <xs:enumeration value="NTSC_433"/>
      <xs:enumeration value="PAL_B"/>
      <xs:enumeration value="PAL_D"/>
      <xs:enumeration value="PAL_G"/>
      <xs:enumeration value="PAL_H"/>
      <xs:enumeration value="PAL_I"/>
      <xs:enumeration value="PAL_M"/>
      <xs:enumeration value="PAL_N"/>
      <xs:enumeration value="PAL_60"/>
      <xs:enumeration value="SECAM_B"/>
      <xs:enumeration value="SECAM_D"/>
      <xs:enumeration value="SECAM_G"/>
      <xs:enumeration value="SECAM_H"/>
      <xs:enumeration value="SECAM_K"/>
      <xs:enumeration value="SECAM_K1"/>
      <xs:enumeration value="SECAM_L"/>
      <xs:enumeration value="SECAM_L1"/>
      <xs:enumeration value="PAL_N_COMBO"/>
      <xs:enumeration value="IPTV"7>
    </xs:restriction>
  </xs:simpleType>
  <!--simpleTypes associated with DVB Lineup Entries-->
  <xs:simpleType name="dvbNetworkID">
    <xs:annotation>
      <xs:documentation>Identifier for DVB Network.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:nonNegativeInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="networkAffiliationType">
    <xs:annotation>
      <xs:documentation>String used to indicate network
affiliation.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- ServiceID simpleTypes -->
  <xs:simpleType name="serviceIDTypes">
    <xs:annotation>
      <xs:documentation>Enumeration of recognized in-band service (channel)
data identifiers.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
    </xs:restriction>
  </xs:simpleType>
  <!--Simple Miscellaneous Types-->
  <xs:simpleType name="timeZoneType">
    <xs:annotation>
      <xs:documentation>Simple time zone type
specification.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value="(\+|\-)\d\d:\d\d"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="restrictedURIType">
    <xs:annotation>
      <xs:documentation>URI string with limited length.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:anyURI">
      <xs:minLength value="1"/>
      <xs:maxLength value="500"/>
    </xs:restriction>
  </xs:simpleType>
   <xs:simpleType name="restrictedURITypeBis">
    <xs:annotation>
      <xs:documentation>URI string with limited length.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:anyURI">
```

```
      <xs:minLength value="1"/>
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="noticeTextType">
    <xs:annotation>
      <xs:documentation>Text notice type definition.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="1000"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="shortNoticeTextType">
    <xs:annotation>
      <xs:documentation>Text short notice type
definition.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="200"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="legalNoticeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="copyright"/>
      <xs:enumeration value="tos"/>
      <xs:enumeration value="disclaimer"/>
    </xs:restriction>
  </xs:simpleType>
  <!--Flag Related Items-->
  <xs:simpleType name="flagIDType">
    <xs:annotation>
      <xs:documentation>Basic ID Type for flags.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="2147483648"/>
    </xs:restriction>
  </xs:simpleType>
  <!--Supplier Information-->
  <xs:simpleType name="SupplierNameType">
    <xs:annotation>
      <xs:documentation>Supplier Name Type.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="1000"/>
    </xs:restriction>
  </xs:simpleType>
  <!--complexTypes-->
  <!--Lineup Data Fundamental Types -->
  <xs:complexType name="headendType">
    <xs:annotation>
      <xs:documentation>headendType.  Basic channel grouping data, contains
information required for lineup selection.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="headendIDType" use="required">
      <xs:annotation>
        <xs:documentation>Primary Key for Headend entity.  This attribute
(id) is REQUIRED to persist from feed to feed.  Example:  If a headend "Cox
Phoenix Metro" is assigned id 34, this same id must be used for "Cox Phoenix
Metro" in all future data.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="n" type="headendNameType" use="required">
      <xs:annotation>
        <xs:documentation>Lineup (Headend) Name, used in UI for headend
selection.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="t" type="headendTypes" use="required"/>
    <xs:attribute name="s" type="tuningSubTypes" use="optional">
      <xs:annotation>
        <xs:documentation>Subtype.  Used by client to provide a tuning
context, possible values are defined by the tuningSubTypes
simpleType.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="d" type="headendDescriptionType" use="optional">
      <xs:annotation>
        <xs:documentation>Lineup (Headend) Description, optionally used in
UI</xs:documentation>
      </xs:annotation>
```

```
    </xs:attribute>
    <xs:attribute name="i" type="restrictedURIType" use="optional">
      <xs:annotation>
        <xs:documentation>Local icon file reference to an icon for this
headend.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="hu" type="restrictedURIType" use="optional">
      <xs:annotation>
        <xs:documentation>External URL for this headend.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="dma" type="headendDMAType" use="optional">
      <xs:annotation>
        <xs:documentation>Market DMA for this headend.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="iso3166" type="iso3166Type" use="required">
      <xs:annotation>
        <xs:documentation>Headend country of origin.  This is specified
using ISO 3166 country codes.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="iso639" type="iso639Type" use="required">
      <xs:annotation>
        <xs:documentation>Primary language used on this headend.  This is
specified using ISO 639 language codes.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="l" type="headendLocationType" use="optional">
      <xs:annotation>
        <xs:documentation>Location.  Optional location of headend, typically
this is a city or county name.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="c" type="headendCommunityType" use="optional">
      <xs:annotation>
        <xs:documentation>Community.  Optional field describing the
community served by the headend, this is typically a city or area
name.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="df" type="defaultVideoFormat" use="optional">
      <xs:annotation>
        <xs:documentation>Default video format of this
headend.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="st" type="headendDVBSystemType" use="optional">
      <xs:annotation>
        <xs:documentation>For DVB headends only, specify the system
type.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="channelType">
    <xs:annotation>
      <xs:documentation>channelType.  Basic channel information including
items needed to draw basic grid elements.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="channelIDType" use="required">
      <xs:annotation>
        <xs:documentation>Primary Key for Channel entity.  This attribute
(id) is REQUIRED to persist from feed to feed.  Example:  If a station KCBS
is assigned id 34, this same id must be used for KCBS in all future
data.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="c" type="channelCallLettersType" use="required">
      <xs:annotation>
        <xs:documentation>Call letters.  This the short grid presentation
name for this channel.  It is very important that this string match the call
letter string provided in in-band data--the match can be case insensitive
but spacing is important.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="l" type="channelDisplayNameType" use="required">
      <xs:annotation>
        <xs:documentation>Display name for channel which will be displayed
in the UI for channel/lineup selection.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="d" type="channelDescriptionType" use="optional">
```

```
      <xs:annotation>
        <xs:documentation>Extended description.  Used to further identify
this programming source (in addition to the long name
attribute.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="t" type="channelProgrammingType" use="optional">
      <xs:annotation>
        <xs:documentation>Channel programming type (e.g. News, Sports,
etc.).</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="a" type="channelAffiliationType" use="optional">
      <xs:annotation>
        <xs:documentation>Network affiliation. Some examples are: "ABC",
"CNN" etc.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="i" type="restrictedURITypeBis" use="optional">
      <xs:annotation>
        <xs:documentation>Station logo icon.  This is a reference to a local
icon file to be used in display situations on the client where a logo is
appropriate.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="u" type="restrictedURIType" use="optional">
      <xs:annotation>
        <xs:documentation>Station URL, this is an external URL that can be
used to direct clients to the this station.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="b" type="xs:positiveInteger" use="optional">
      <xs:annotation>
        <xs:documentation>Optional broadcast tuner position (this applies
only to FPB stations).</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="iso3166" type="iso3166Type" use="optional">
      <xs:annotation>
        <xs:documentation>Programming country of origin.  This is specified
using ISO 3166 country codes.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="iso639" type="iso639Type" use="optional">
      <xs:annotation>
        <xs:documentation>Primary language used on this channel.  This is
specified using ISO 639 language codes.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="tz" type="timeZoneType" use="optional">
      <xs:annotation>
        <xs:documentation>Programming timezone.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="vbi" type="channelVBIType" use="optional">
      <xs:annotation>
        <xs:documentation>Unique ID used by the broadcaster to identify
themselves through VBI.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="cf" type="carrierFrequencyType" use="optional">
      <xs:annotation>
        <xs:documentation>For broadcast televison stations, enter the
default carrier frequency.  Not required if channel follows standard
regulatory carrier frequency (i.e. is a function of the broadcast tuner
position).</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="tsid" type="channelTSIDType" use="optional">
      <xs:annotation>
        <xs:documentation>TSID identifier for this channel (service).  This
value, if supplied, must match the in-band TSID used for this
channel.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ntsid" type="channelTSIDType" use="optional">
      <xs:annotation>
        <xs:documentation>NTSC TSID identifier for this channel (service).
This value, if supplied, must match the in-band TSID used for this
channel.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="cni" type="channelCNIType" use="optional">
```

```
      <xs:annotation>
        <xs:documentation>CNI identifier for this channel (service).  This
value, if supplied, must match the in-band CNI used for this
channel.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="vps" type="channelVPSType" use="optional">
      <xs:annotation>
        <xs:documentation>VPS identifier for this channel (service).  This
value, if supplied, must match the in-band VPS used for this
channel.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ppv" type="channelPPVType" use="optional">
      <xs:annotation>
        <xs:documentation>Channel carries pay-per-view data
exclusively.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="areaType">
    <xs:annotation>
      <xs:documentation>areaType.  Basic unit of lineup grouping, used for
lineup selection.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="a" type="areaIDType" use="required"/>
    <xs:attribute name="h" type="headendIDType" use="required"/>
    <xs:attribute name="ef" type="xs:dateTime" use="optional">
      <xs:annotation>
        <xs:documentation>Effective date for this entry.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="ex" type="xs:dateTime" use="optional">
      <xs:annotation>
        <xs:documentation>Expiration date for this entry.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="iso3166" type="iso3166Type" use="required">
      <xs:annotation>
        <xs:documentation>Country for this area.  This is specified using
ISO 3166 country codes.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <!--Listing Data Fundamental Types -->
  <xs:complexType name="FlagValueDefinitionA" abstract="true">
    <xs:annotation>
      <xs:documentation>Abstract type which defines attributes for flag and
value defintions.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="name" type="flagValueNameType" use="required">
      <xs:annotation>
        <xs:documentation>Flag name.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="pname" type="flagValuePNameType" use="required">
      <xs:annotation>
        <xs:documentation>Presentation name (used in UI), may be the same as
name.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="desc" type="flagValueDescriptionType"
use="required">
      <xs:annotation>
        <xs:documentation>Description of flag.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="icon" type="restrictedURIType" use="optional">
      <xs:annotation>
        <xs:documentation>icon representation of flag.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="lang" type="xs:language" use="optional"/>
  </xs:complexType>
  <xs:complexType name="FlagValueDefinitionAT" abstract="true">
    <xs:annotation>
      <xs:documentation>Abstract type which defines translations for a flag
or value defintion.  This abstract type is identical to the
FlagValueDefinitionA but makes the lang attribute
required.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="pname" type="flagValuePNameType" use="required">
      <xs:annotation>
```

```
        <xs:documentation>Presentation name (used in UI), may be the same as
name.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="desc" type="flagValueDescriptionType"
use="required">
      <xs:annotation>
        <xs:documentation>Description of flag.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="icon" type="restrictedURIType" use="optional">
      <xs:annotation>
        <xs:documentation>icon representation of flag.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="lang" type="xs:language" use="required"/>
  </xs:complexType>
  <xs:complexType name="programFlagsType">
    <xs:annotation>
      <xs:documentation>programFlagType.  Definition of
ProgramFlags.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:restriction base="glf:FlagValueDefinitionA">
        <xs:attribute name="name" type="glf:programFlagNameType"
use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="programFlagReference" abstract="false">
    <xs:annotation>
      <xs:documentation>programFlagReference.  Used in a schedule entry to
reference a program flag.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="flagIDType" use="required">
      <xs:annotation>
        <xs:documentation>Program flag key.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="programValuesType">
    <xs:annotation>
      <xs:documentation>programValues. Definitions of Program
Values.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:restriction base="glf:FlagValueDefinitionA">
        <xs:attribute name="name" type="glf:programValueNameType"
use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="programValueReference">
    <xs:annotation>
      <xs:documentation>programValueReference.  Used in a program to
reference a Program Value.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="flagIDType" use="required">
      <xs:annotation>
        <xs:documentation>Reference to flag ID.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="v" type="valueType" use="required">
      <xs:annotation>
        <xs:documentation>Value of flag, for example
"Color".</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="i" type="restrictedURITypeBis" use="optional">
      <xs:annotation>
        <xs:documentation>Icon corresponding to the value of this
flag.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="scheduleFlagsType">
    <xs:annotation>
      <xs:documentation>scheduleFlags.  Definitions of Schedule
Flags.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:restriction base="glf:FlagValueDefinitionA">
```

```
        <xs:attribute name="name" type="glf:scheduleFlagNameType"
use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="scheduleValuesType">
    <xs:annotation>
      <xs:documentation>scheduleValues.  Definitions of Schedule
Values.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:restriction base="glf:FlagValueDefinitionA">
        <xs:attribute name="name" type="glf:scheduleValueNameType"
use="required"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="scheduleFlagReference" abstract="false">
    <xs:annotation>
      <xs:documentation>scheduleFlagReference.  Used in a schedule entry to
reference a schedule flag.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="flagIDType" use="required">
      <xs:annotation>
        <xs:documentation>Schedule flag key.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="scheduleValueReference" abstract="false">
    <xs:annotation>
      <xs:documentation>scheduleFlagReference.  Used in a schedule entry to
reference a schedule flag.</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="flagIDType" use="required">
      <xs:annotation>
        <xs:documentation>Schedule flag key.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="v" type="valueType" use="required">
      <xs:annotation>
        <xs:documentation>Optional value of schedule flag (e.g. DirecTV
Rating "TV-MA")</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="i" type="restrictedURIType" use="optional">
      <xs:annotation>
        <xs:documentation>Icon URI which applies to the value of this
flag.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <!--Category Fundamental Types-->
  <xs:complexType name="programCategoryFlagReference">
    <xs:annotation>
      <xs:documentation>programCategoryFlagReference.  Used in a program to
indicate category (via standard flag mechanism).</xs:documentation>
    </xs:annotation>
    <xs:attribute name="id" type="flagIDType" use="required"/>
  </xs:complexType>
  <!--Misc. types-->
  <xs:complexType name="noticeContent" mixed="false">
    <xs:annotation>
      <xs:documentation>Notice with language element.  </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="lang" type="xs:language"/>
      <xs:element name="text" type="noticeTextType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="legalNoticeContent" mixed="false">
    <xs:annotation>
      <xs:documentation>Notice with language element.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="type" type="legalNoticeType" minOccurs="0"/>
      <xs:element name="pname" type="shortNoticeTextType" minOccurs="0"/>
      <xs:element name="url" type="restrictedURIType" minOccurs="0"/>
      <xs:element name="lang" type="xs:language"/>
      <xs:element name="text" type="noticeTextType"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```