Unión Internacional de Telecomunicaciones

UIT-R

Sector de Radiocomunicaciones de la UIT

Recomendación UIT-R BS.2088-1
(10/2019)

Formato de fichero largo para el intercambio internacional de material de programas de audio con metadatos

Serie BS

Servicio de radiodifusión (sonora)

Unión Internacional de Telecomunicaciones

Prólogo

El Sector de Radiocomunicaciones tiene como cometido garantizar la utilización racional, equitativa, eficaz y económica del espectro de frecuencias radioeléctricas por todos los servicios de radiocomunicaciones, incluidos los servicios por satélite, y realizar, sin limitación de gamas de frecuencias, estudios que sirvan de base para la adopción de las Recomendaciones UIT-R.

Las Conferencias Mundiales y Regionales de Radiocomunicaciones y las Asambleas de Radiocomunicaciones, con la colaboración de las Comisiones de Estudio, cumplen las funciones reglamentarias y políticas del Sector de Radiocomunicaciones.

Política sobre Derechos de Propiedad Intelectual (IPR)

La política del UIT-R sobre Derechos de Propiedad Intelectual se describe en la Política Común de Patentes UIT-T/UIT-R/ISO/CEI a la que se hace referencia en la Resolución UIT-R 1. Los formularios que deben utilizarse en la declaración sobre patentes y utilización de patentes por los titulares de las mismas figuran en la dirección web https://www.itu.int/ITU-R/go/patents/es, donde también aparecen las Directrices para la implementación de la Política Común de Patentes UIT-T/UIT-R/ISO/CEI y la base de datos sobre información de patentes del UIT-R sobre este asunto.

Series de las Recomendaciones UIT-R

(También disponible en línea en http://www.itu.int/publ/R-REC/es)

Series	Título
BO	Distribución por satélite
BR	Registro para producción, archivo y reproducción; películas en televisión
BS	**Servicio de radiodifusión (sonora)**
BT	Servicio de radiodifusión (televisión)
F	Servicio fijo
M	Servicios móviles, de radiodeterminación, de aficionados y otros servicios por satélite conexos
P	Propagación de las ondas radioeléctricas
RA	Radio astronomía
RS	Sistemas de detección a distancia
S	Servicio fijo por satélite
SA	Aplicaciones espaciales y meteorología
SF	Compartición de frecuencias y coordinación entre los sistemas del servicio fijo por satélite y del servicio fijo
SM	Gestión del espectro
SNG	Periodismo electrónico por satélite
TF	Emisiones de frecuencias patrón y señales horarias
V	Vocabulario y cuestiones afines

***Nota**: Esta Recomendación UIT-R fue aprobada en inglés conforme al procedimiento detallado en la Resolución UIT-R 1.*

Publicación electrónica
Ginebra, 2020

RECOMENDACIÓN UIT-R BS.2088-1*

Formato de fichero largo para el intercambio internacional de material de programas de audio con metadatos

(2015-2019)

Cometido

En esta Recomendación se especifica el formato de fichero de audio Broadcast Wave 64 Bit (BW64), en particular los nuevos segmentos <ds64>, <axml>, <bxml>, <sxml> y <chna>, que permiten transportar grandes ficheros multicanal y metadatos, incluido el Modelo de Definición de Audio (ADM) especificado en la Recomendación UIT-R BS.2076.

Palabras clave

ADM en serie (S-ADM), audio inmersivo, BW64, BWF, fichero, fichero de ondas, formato de fichero, intercambio, metadatos, modelo de definición de audio (ADM), onda, programa de audio, RF64, RIFF, WAV

La Asamblea de Radiocomunicaciones de la UIT,

considerando

a) que los medios de almacenamiento basados en la tecnología de la información, incluidos los discos y cintas de datos, serán utilizados en todos los campos de la producción de audio para la radiodifusión, a saber, edición no lineal, reproducción a partir de la emisión y archivos;

b) que esta tecnología ofrece ventajas importantes desde el punto de vista de la flexibilidad de funcionamiento, flujo de producción y automatización de la estación y que, en consecuencia, es interesante para la mejora de los estudios existentes y el diseño de nuevas instalaciones de estudios;

c) que la adopción de un solo formato de fichero para el intercambio de señales simplificaría considerablemente la interoperabilidad de los equipos y estudios distantes, y facilitaría la integración deseable de la edición, la reproducción a partir de la emisión y el archivo;

d) que se debe incluir un conjunto mínimo de información relacionada con la radiodifusión en el fichero para documentar la señal audio;

e) que, con miras a asegurar la compatibilidad entre aplicaciones con complejidades diferentes, se debe acordar un conjunto mínimo de funciones, comunes a todas las aplicaciones capaces de tratar el formato de fichero recomendado;

f) que la Recomendación UIT-R BS.646 define el formato de audio digital utilizado en la producción de audio para radiodifusión sonora y de televisión;

g) que la compatibilidad con los formatos de ficheros comerciales actualmente disponibles podría minimizar los esfuerzos de la industria para aplicar este formato en los equipos;

h) que un formato normalizado para la información de historial de codificación y para otros metadatos conexos simplificaría el empleo de la información tras el intercambio de programas;

---

* La Comisión de Estudio 6 de Radiocomunicaciones introdujo modificaciones formalesen esta Recomendación en febrero de 2020 y en septiembre de 2023, de conformidad con la Resolución UIT-R 1.

i) que la calidad de la señal de audio está influida por el tratamiento que haya tenido la señal, especialmente por la utilización decodificación y decodificación no lineales durante los procesos de reducción binaria,

j) que los sistemas de audio avanzados necesitan metadatos asociados con el audio que se transporte en el fichero;

k) que los sistemas de audio avanzados utilizan diversas configuraciones multicanal en particular audio basado en canales, objetos o escenas tal como especifica la Recomendación UIT-R BS.2051;

l) que en la Recomendación UIT-R BS.2076 se especifican los metadatos de audio, utilizados en sistemas de sonido avanzados, en la Recomendación UIT-R BS.2094 se especifican sus definiciones comunes y en la Recomendación UIT-R BS.2125 se especifican las representaciones serializadas de los metadatos (S-ADM);

m) que la Recomendación UIT-R BS.1352 tiene limitaciones en lo que respecta al tamaño de los ficheros y a su capacidad de transportar metadatos adicionales;

n) que los ficheros de audio multicanal podrían tener un tamaño superior a 4 Gbytes,

recomienda

1 que, para el intercambio de programas de audio, los parámetros de la señal de audio, la frecuencia de muestreo (parte 1), la profundidad de bits (partes 4 y 5) y la preacentuación (parte 6) se fijen de acuerdo con las partes pertinentes de la Recomendación UIT-R BS.646;

2 que se utilice el formato de fichero especificado en el Anexo 1 para el intercambio de programas de audio en los siguientes casos:

- en entornos basados en ficheros WAVE, en los que se desee mejorar las aplicaciones de radiodifusión con ficheros WAVE para tratar contenido inmersivo, manteniendo al mismo tiempo la compatibilidad hacia adelante;
- en flujos de trabajo basados en ficheros en los que exista una biblioteca mixta con contenido de ficheros WAVE históricos y contenido inmersivo;
- en flujos de trabajo basados en ficheros en los que se prefiera un único contenedor de datos y metadatos de paquetes.

NOTA − En el Anexo 4 se muestran las modificaciones aportadas a las especificaciones del Anexo 1 con respecto a la versión anterior de esta Recomendación. El Anexo 4 es sólo informativo.

Anexo 1
(normativo)

Especificación del formato de fichero BW64

1 Introducción

El formato BW64 está basado en el formato de fichero de audio WAVE (descrito en el Anexo 2), que es un tipo de fichero especificado en el formato de fichero de intercambio de recursos (RIFF, *Resource Interchange File Format*). Los ficheros WAVE contienen concretamente datos de audio. El componente básico del formato de fichero RIFF, denominado segmento, contiene un grupo de

piezas de información estrechamente relacionado. Consiste en un identificador de segmento, un valor entero que representa la longitud del segmento en bytes y la información. Un fichero RIFF se compone de una colección de segmentos. Este formato BW64 utiliza los elementos fundamentales del formato descrito en la especificación Tech 3306 de la UER.

El formato de fichero BWF de la Recomendación UIT-R BS.1352 tiene ciertas limitaciones, en particular:

- un tamaño de fichero máximo inferior a 4 Gbytes;
- no soporta audio multicanal avanzado debido a los metadatos de audio limitados;
- no soporta adecuadamente metadatos técnicos.

El formato BW64 descrito en la presente Recomendación pretende superar estas limitaciones y mantener la mayor compatibilidad posible con el formato de la Recomendación UIT-R BS.1352, compartiendo muchos de los elementos básicos.

Existe una demanda creciente para la transferencia de metadatos, en particular para la transferencia de metadatos del Modelo de Definición de Audio (ADM, *Audio Definition Model*) de conformidad con la Recomendación UIT-R BS.2076. La presente Recomendación incluye una definición de los segmentos <axml>, <bxml> y <sxml> para almacenar y transferir metadatos como XML escritos en UTF-8, en formato comprimido y serializado, respectivamente.

El objetivo fundamental del segmento <chna> descrito en la presente Recomendación es facilitar referencias de cada pista de ficheros BW64 a los identificadores en los metadatos ADM definidos en la Recomendación UIT-R BS.2076.

Además del objetivo principal de vincular cada pista del fichero con sus metadatos ADM asociados, el segmento <chna> también permite un acceso más rápido a los ID del ADM sin tener que acceder a los metadatos XML (cuando los ID se encuentran en la gama de valores predefinida para configuraciones ADM normalizadas). Puesto que es posible determinar el tamaño del segmento <chna> y está situado antes de los segmentos <data>, <axml>, <bxml> y <sxml> resulta más sencillo acceder, generar o modificar su contenido sobre la marcha.

En todo este documento, los tipos de datos se utilizan de conformidad con el Anexo 3.

2 Descripción del formato BW64

2.1 Contenido de un fichero de formato BW64

Un fichero de formato BW64 comenzará con el encabezamiento obligatorio «WAVE» y por lo menos con los segmentos siguientes:

```
<WAVE-form> ->
    BW64('WAVE'
        <ds64-ck>       // ds64 chunk for 64-bit addressing
        <fmt-ck>        // Format of the audio signal: PCM/non-PCM
        <chna-ck>       // chna chunk for ADM look-up
        <axml-ck>       // axml chunk for carrying XML metadata
        <bxml-ck>       // bxml chunk for carrying compressed XML metadata
        <sxml-ck>       // sxml chunk for carrying XML metadata associated with
                        // sub-chunk or sound data
        <wave-data>)    // sound data
```

NOTA 1 – Es posible que en el fichero estén presentes otros segmentos. Algunos de ellos pueden estar fuera del alcance de la presente Recomendación. Las aplicaciones pueden o no interpretar o utilizar estos segmentos, de modo que es imposible garantizar la integridad de los datos contenidos en ese tipo de segmento desconocidos. Sin embargo, las aplicaciones homologadas transfieren de manera transparente los segmentos desconocidos.

NOTA 2 – Se podría permitir situar un segmento <axml>, <bxml> o <sxml> después del segmento <data>, puesto que los metadatos XML probablemente tendrán una longitud desconocida y podría resultar más práctico conocer la posición inicial de las muestras de audio en el fichero.

2.2 Segmentos existentes definidos como parte de la norma RIFF/WAVE

La norma RIFF/WAVE utiliza algunos segmentos que ya están definidos. A saber:

- <RIFF>
- <fmt>
- <data>

Estos segmentos se describen en los § 2.6.1 a 2.6.3.

RIFF/WAVE es un subconjunto del formato de la Recomendación UIT-R BS.1352 que incluye los segmentos adicionales siguientes:

- <bext>
- <ubxt>

Estos segmentos no se incluirán en el formato BW64, que proporciona una solución más flexible al transportar metadatos de radiodifusión.

2.3 Nuevos segmentos y estructuras en el formato BW64

Los nuevos segmentos introducidos para BW64 son:

- <BW64>
- <ds64>
- <JUNK>
- <axml>, <bxml> o <sxml>
- <chna>

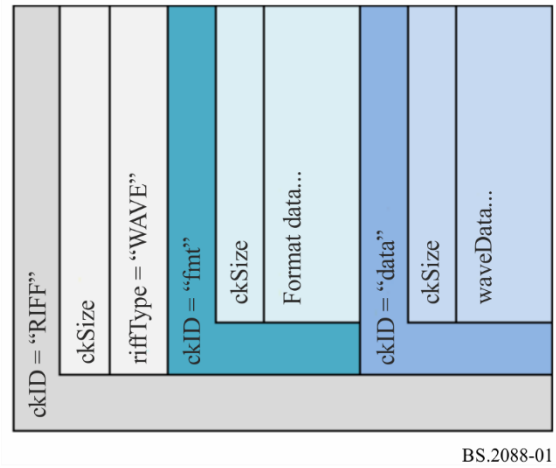
Estos segmentos se describen en los § 3 a 8.

2.4 Uso del segmento <ds64> para permitir la utilización de ficheros de tamaño superior a 4 Gbytes

La razón de la limitación a 4 Gbytes es el direccionamiento de 32 bits en los ficheros RIFF/WAVE y BWF. Con 32 bits se pueden direccionar un máximo de 4294967296 bytes = 4 Gbytes. Para resolver este problema se precisa un direccionamiento de 64 bits. En la Fig. 1 se muestra la estructura de un fichero básico convencional RIFF/WAVE, en el que los campos ckSize son números de 32 bits que representan los tamaños de sus segmentos.

FIGURA 1

Estructura del fichero RIFF/WAVE básico



Al cambiar únicamente el tamaño de todos los campos en un fichero BWF para sean de 64 bits se genera un fichero que no es compatible con el formato normalizado RIFF/WAVE – una observación obvia pero importante.

El planteamiento adoptado consiste en definir un nuevo RIFF basado en 64 bits denominado BW64 que es idéntico al formato RIFF/WAVE original, salvo por los cambios siguientes:

- se utiliza el ID 'BW64' en lugar del 'RIFF' en los primeros cuatro bytes de la fila;
- se añade un segmento obligatorio <ds64> (tamaño de datos 64) que tiene que ser el primer segmento después de «BW64 chunk».

El segmento 'ds64' tiene dos valores enteros obligatorios de 64 bits, que sustituyen a los dos campos de 32 bits del formato RIFF/WAVE:

- bw64Size (sustituye al campo tamaño del segmento <RIFF>);
- dataSize (sustituye al campo correspondiente al tamaño del segmento <data>).

Para todos los campos dobles de 32 bits del formato RIFF/WAVE aplican las siguientes normas:

Si el valor de 32 bits en el campo no es 0xFFFFFFFF, entonces se utiliza este valor de 32 bits.

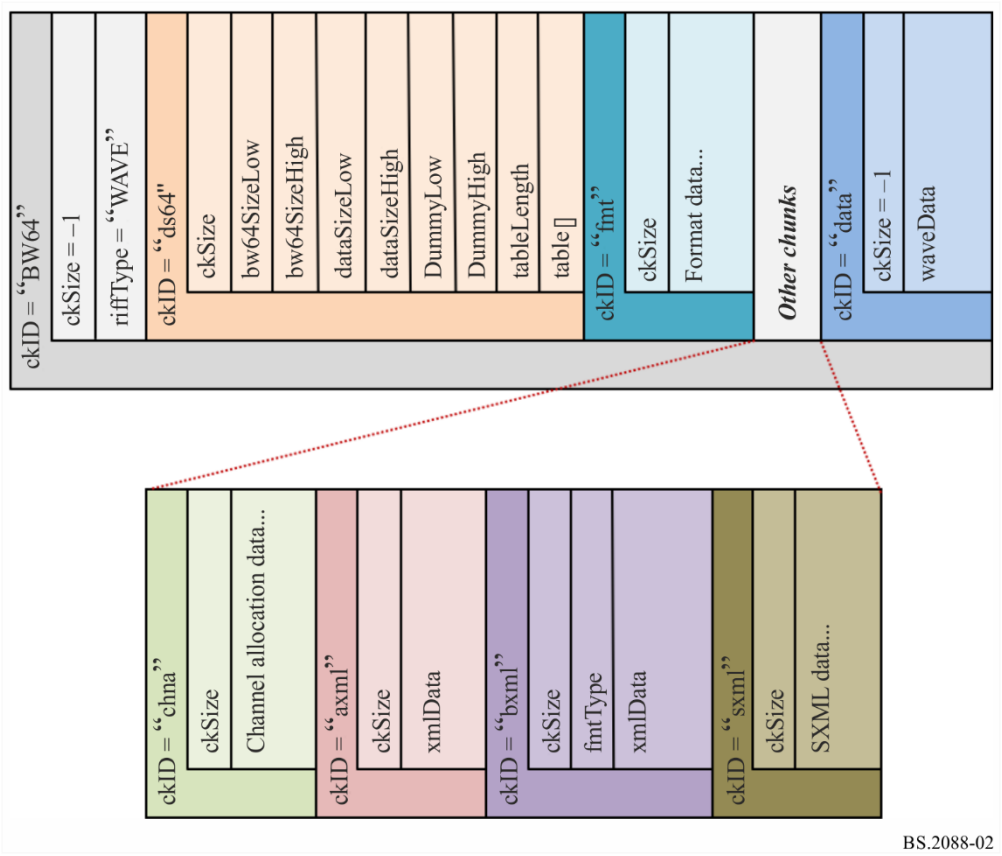
Si el valor de 32 bits en el campo es 0xFFFFFFFF, se utiliza en su lugar el valor de 64 bits en el segmento 'ds64'.

- Es posible una disposición facultativa de estructuras (véase el Anexo 1) con tamaños adicionales de los segmentos de 64 bits.

En la Fig. 2 se muestra la estructura completa de un formato de fichero BW64, en el que los valores ckSize para los segmentos <BW64> y <data> se fijan a 0xFFFFFFFF, para permitir que utilicen valores con tamaños de 64 bits a partir del segmento <ds64>.

FIGURA 2

Estructura de fichero BW64



NOTA – El tamaño de los segmentos en datos puede ser variable. El inicio de cada segmento está alienado por palabras al inicio del fichero BW64 para mantener la compatibilidad con BWF, especificado en la Recomendación UIT-R BS.1352. Si el tamaño del segmento es un número impar de bytes, se escribirá después del segmento un byte de relleno de valor cero. El valor de ckSize, sin embargo, no incluye el byte de relleno.

2.5 Compatibilidad entre RIFF/WAVE y BW64

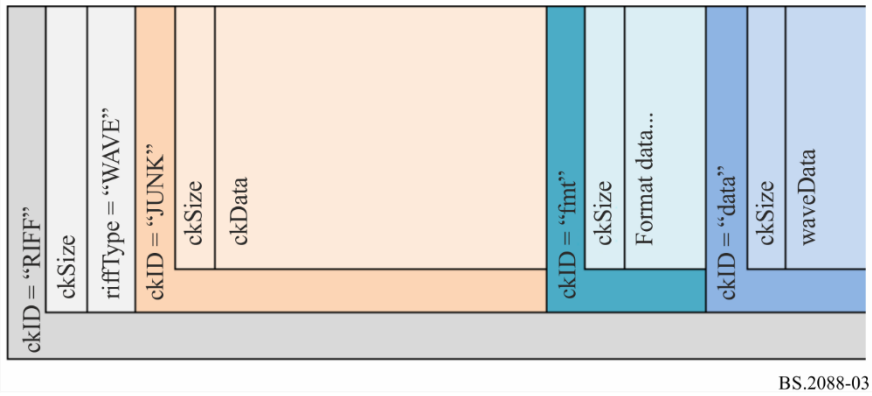
Aunque se utilicen frecuencias de muestreo más elevadas y audio multicanal, algunos archivos de audio de producción serán inevitablemente inferiores a 4 Gbytes y deberían por lo tanto mantenerse en el formato RIFF/WAVE corto (descrito en el Anexo 2). El problema surge porque una aplicación de grabación no puede saber con antelación si el audio grabado que está compilando superará los 4 Gbytes o no al final de la grabación (es decir, si necesita utilizar BW64 o no).

La solución consiste en permitir que la aplicación de grabación conmute de RIFF/WAVE a BW64 sobre la marcha en el límite de tamaño de 4 Gbytes mientras se está realizando la grabación.

Esto se consigue reservando espacio adicional en RIFF/WAVE, insertando un segmento <JUNK> que tiene el mismo tamaño que un segmento <ds64>. Este espacio reservado no significa nada para la forma corta de WAVE, pero se transformará en el segmento <ds64>, si es necesaria la transición a BW64. El diagrama de la Fig. 3 muestra el segmento marcador de posición <JUNK> antes del segmento <fmt>.

FIGURA 3

Estructura de fichero con un segmento JUNK



Al inicio de la grabación, una aplicación consciente de BW64 creará un formato RIFF/WAVE normalizado con un segmento 'JUNK' como primer segmento. Durante la grabación, comprobará los tamaños del RIFF y de los datos. Si supera los 4 Gbytes, la aplicación:

- sustituirá el ckID <JUNK> por el segmento <ds64>. (Esto transforma el segmento <JUNK> en un segmento <ds64>);
- introducirá el tamaño del RIFF y el tamaño del segmento 'data' en el segmento <ds64>;
- fijará el tamaño del RIFF y el tamaño del segmento 'data' en los campos de 32 bits a 0xFFFFFFFF;
- sustituirá el ID 'RIFF' por 'BW64' en los primeros cuatro bytes del fichero;
- continuará con la grabación.

2.6 Segmentos y estructuras existentes en el formato RIFF/WAVE

Los segmentos que existen en el formato RIF/WAVE se muestran a continuación:

```
struct RiffChunk            // declare RiffChunk structure
{
    CHAR   ckID[4];         // 'RIFF'
    DWORD  ckSize;          // 4 byte size of the traditional RIFF/WAVE file
    CHAR   riffType[4];     // 'WAVE'
};

struct FormatChunk          // declare FormatChunk structure
{
    CHAR   ckID[4];         // 'fmt '
    DWORD  ckSize;          // 4 byte size of the 'fmt ' chunk
    WORD   formatTag;       // WAVE_FORMAT_PCM = 0x0001, etc.
    WORD   channelCount;    // 1 = mono, 2 = stereo, etc.
    DWORD  sampleRate;      // 32000, 44100, 48000, etc.
    DWORD  bytesPerSecond;  // only important for compressed formats
    WORD   blockAlignment;  // container size (in bytes) of one set of samples
    WORD   bitsPerSample;   // valid bits per sample 16, 20 or 24
    WORD   cbSize;          // should be excluded as extraData is not used, but if
```

```
                            // present shall be set to zero
    CHAR   extraData[22];   // extra data of WAVE_FORMAT_EXTENSIBLE when necessary,
                            // shall not be used as cbSize will be zero or not present.
};

struct DataChunk            // declare DataChunk structure
{
    CHAR   ckID[4];         // 'data'
    DWORD  ckSize;          // 4 byte size of the 'data' chunk
    CHAR   waveData[ ];     // audio samples
};
```

Los corchetes vacíos indican que se puede utilizar un número variable de elementos (incluido cero).

2.6.1 Elementos del segmento <RIFF>

El segmento <RIFF> es el fichero de mayor nivel.

Campo	Descripción
ckID	Conjunto de 4 caracteres {'R', 'I', 'F', 'F'} utilizados para la identificación del segmento.
ckSize	Valor de 4 bytes del tamaño del fichero.
riffType	Conjunto de 4 caracteres {'W', 'A', 'V', 'E'} que indica que el fichero es un fichero de audio de tipo WAVE.

2.6.2 Elementos del segmento <fmt >

El segmento <fmt > contiene información sobre los formatos de las muestras de audio almacenadas en el segmento <data>.

Campo	Descripción
ckID	Conjunto de 4 caracteres {'f', 'm', 't', ' '} utilizado para la identificación del segmento.
ckSize	Valor de 4 bytes del tamaño del segmento.
formatTag	Valor de 2 bytes que representa el formato de las muestras de audio. El valor 0x0001 significa que el formato es MIC, 0x0000 es para formatos desconocidos.
channelCount	Valor de 2 bytes que indica el número de pistas de audio en el fichero.
sampleRate	Valor de 4 bytes que indica la velocidad de muestreo de audio en Hz.
bytesPerSecond	Número medio de bytes por segundo en que se deberán transferir los datos de forma de onda. El soporte lógico de reproducción puede estimar el tamaño de la memoria intermedia utilizando este valor.

blockAlignment	Alineación de bloques (en bytes) de los datos de forma de onda. El soporte lógico de reproducción necesita procesar un múltiplo de bytes **blockAlignment** de los datos de una vez, de forma que el valor de **blockAlignment** se pueda utilizar para el alineamiento de la memoria intermedia.
bitsPerSample	Número de bits por muestra y por canal. Cada canal se supone que tiene la misma resolución de muestreo. Si este campo no es necesario, se deberá poner a cero.
cbSize	Tamaño en bytes de la estructura extraData.
extraData	Datos adicionales utilizados para almacenar la información de WAVE_FORMAT_EXTENSIBLE. No se debe utilizar en BW64.

FormatChunk ya es un formato de segmento especializado para datos de audio PCM.

El conjunto extraData en FormatChunk se utiliza cuando formatTag se pone a 0xFFFE (WAVE_FORMAT_EXTENSIBLE). Puesto que el audio multicanal se debe describir utilizando metadatos ADM, debe evitarse el uso de este formatTag. Sin embargo, debe ser posible que las implementaciones puedan leer un fichero que contenga este formatTag y lo puedan manejar de forma adecuada.

Para garantizar que FormatChunk no contradice la información de los segmentos <chna>, <axml>, <bxml> y <sxml>, se recomienda fijar formatTag a 0x0001 para audio MIC y a 0x0000 (formatTag = desconocido) para todas las señales de audio que no sean MIC.

2.6.3 Elementos del segmento <data>

El segmento <data> sirve para almacenar las muestras de audio.

Campo	**Descripción**
ckID	Conjunto de 4 caracteres {'d', 'a', 't', 'a'} utilizado para la identificación del segmento.
ckSize	Valor de 4 bytes del tamaño del segmento.
waveData	Aquí se almacenan las muestras de audio. Las muestras se almacenan en el orden little-endian (empezando por el byte menos significativo). Se almacenan múltiples pistas intercalándolas muestra a muestra. Por ejemplo, para audio de dos pistas de 16 bits:

Byte	**Muestra**	**Pista**
0	0 – LSB	1
1	0 – MSB	1
2	0 – LSB	2
3	0 – MSB	2
4	1 – LSB	1
5	1 – MSB	1
6	1 – LSB	2
7	1 – MSB	2

3 Segmento BW64 de máximo nivel

3.1 Definición

El segmento <BW64> de máximo nivel se utiliza en lugar del segmento <RIFF> usado en los ficheros de 32 bits. Leer este segmento implica que tiene que haber un segmento <ds64> para leer los tamaños de 64 bits. El segmento <BW64> se muestra a continuación:

```
struct BW64Chunk           // declare BW64Chunk structure
{
    CHAR ckID[4];          // 'BW64'
    DWORD ckSize;          // 0xFFFFFFFF means don't use this data, use
                           // bw64SizeHigh and bw64SizeLow in 'ds64' chunk instead
    CHAR BW64Type[4];      // 'WAVE'
};
```

3.2 Elementos del segmento <BW64>

Campo	Descripción
ckID	Conjunto de 4 caracteres {'b', 'w', '6', '4'} utilizado para la identificación del segmento.
ckSize	Valor de 4 bytes que deberá fijarse a 0xFFFFFFFF para indicar que este valor de tamaño no se utiliza y que deberá utilizarse en su lugar el segmento <ds64> para determinar los tamaños.
BW64Type	Conjunto de 4 caracteres {'W', 'A', 'V', 'E'} que indica que el fichero es un fichero de audio de tipo WAVE.

4 Segmentos DS64 y JUNK

4.1 Definiciones

El segmento <ds64> transporta valores de tamaño de 64 bits para el tamaño del fichero, el segmento <data> y un conjunto de valores de tamaño de 64 bits de otros posibles segmentos. La estructura del segmento <ds64> se muestra a continuación, seguida por la estructura para **ChunkSize64** que transporta los tamaños de otros posibles segmentos (distintos de <data>). Los corchetes vacíos indican que se puede utilizar un número variable de elementos (incluido cero).

```
struct DataSize64Chunk     // declare DataSize64Chunk structure
{
    CHAR ckID[4];          // 'ds64', FOURCC chunk identifier
    DWORD ckSize;          // 4 byte size of the <ds64> chunk
    DWORD bw64SizeLow;     // low 4 byte size of <BW64> chunk
    DWORD bw64SizeHigh;    // high 4 byte size of <BW64> chunk
    DWORD dataSizeLow;     // low 4 byte size of <data> chunk
    DWORD dataSizeHigh;    // high 4 byte size of <data> chunk
    DWORD dummyLow;        // dummy value for cross compatibility
```

```
    DWORD dummyHigh;        // dummy value for cross compatibility
    DWORD tableLength;      // number of valid entries in array "table"
    ChunkSize64 table[ ];   // array of chunk sizes for chunks exceeding 4 Gbytes
};

struct ChunkSize64          // declare ChunkSize64 structure
{
    CHAR ckID[4];           // chunk ID of chunk which needs 64bit addressing;
                            // e.g. 'axml' is used when <axml> chunk exceeds 4 Gbytes
    DWORD ckSizeLow;        // low 4 byte chunk size
    DWORD ckSizeHigh;       // high 4 byte chunk size
};
```

El segmento <JUNK> es un marcador de posición para el segmento <ds64> que se utiliza si se está generando un fichero de audio de 32 bits de tamaño que pueda ser necesario convertir sobre la marcha en un fichero de 64 bits de tamaño. El tamaño de <JUNK> debe ajustarse al tamaño del posible segmento <ds64> que lo sustituirá. La estructura del segmento se muestra a continuación:

```
struct JunkChunk             // declare JunkChunk structure
{
    CHAR   ckID[4];          // 'JUNK'
    DWORD  ckSize;           // 4 byte size of the 'JUNK' chunk. This must be at
                             // least 28 if the chunk is intended as a place-holder
                             // for a 'ds64' chunk.
    CHAR   ckData[];         // dummy bytes
};
```

4.2 Elementos del segmento <ds64>

Campo	Descripción
ckID	Conjunto de 4 caracteres {'d', 's', '6', '4'} utilizado para la identificación del segmento.
ckSize	Tamaño de 4 bytes del segmento <ds64>.
bw64SizeLow	Tamaño reducido de 4 bytes del segmento <BW64>. El tamaño de los datos de 64 bits se expresa como 0xHHHHLLLL si <bw64SizeLow> y <bw64SizeHigh> son 0xLLLL y 0xHHHH, respectivamente. La cantidad no asignada de 32 bits está en formato little-endian.
bw64SizeHigh	Tamaño elevado de 4 bytes del segmento <BW64>. La cantidad no asignada de 32 bits está en formato little-endian.
dataSizeLow	Tamaño reducido de 4 bytes del segmento <data>. El tamaño de datos de 64 bits se expresa como 0xHHHHLLLL si <dataSizeLow> y <dataSizeHigh> son 0xLLLL y 0xHHHH, respectivamente. La cantidad no asignada de 32 bits está en formato little-endian.

dataSizeHigh	Tamaño elevado de 4 bytes del segmento <data>. La cantidad no asignada de 32 bits está en formato little-endian.
dummyLow	Valor ficticio de 4 bytes que deberá ignorarse cuando se lee y deberá fijarse a cero cuando se escribe. Existe para garantizar la compatibilidad con la especificación Tech 3306 RF64 de la UER, que utiliza este valor para transmitir información de tamaño sobre el segmento <fact> que no existe en el formato BW64.
dummyHigh	Valor ficticio de 4 bytes que deberá ignorarse cuando se lee y se deberá fijar a cero cuando se escribe. Su objeto es el mismo que el de <dummyLow>.
tableLength	Número de valores válidos en el conjunto «ChunkSize64 table»
ChunkSize64 table	Conjunto de tamaños de segmento para segmentos que superan los 4 Gbytes.

El cuadro **ChunkSize64** se especifica de la forma siguiente. Se utiliza un conjunto de estructuras **ChunkSize64** para almacenar la longitud de cualquier segmento distinto de <data> en la parte opcional del segmento <ds64>. Actualmente, el único tipo de segmento distinto de <data> que es probable que supere el tamaño de 4 Gbytes sería el segmento <axml> (posible en ficheros de audio basados en objetos extremadamente grandes).

Campo	**Descripción**
ckID	Conjunto de 4 caracteres utilizado para referirse al <ckID> del segmento que necesita direccionamiento de 64 bits. Por ejemplo, el conjunto de 4 caracteres {'a', 'x', 'm', 'l'} se utiliza para el segmento <axml>.
ckSizeLow	Tamaño reducido de 4 bytes del segmento que se refiere a <ckID>. La cantidad no asignada de 32 bits está en formato little-endian.
ckSizeHigh	Tamaño elevado de 4 bytes del segmento que se refiere a <ckID>. La cantidad no asignada de 32 bits está en formato little-endian.

4.3 Elementos del segmento <JUNK>

Campo	**Descripción**
ckID	Conjunto de 4 caracteres {'J', 'U', 'N', 'K'} utilizado para la identificación del segmento.
ckSize	Tamaño de 4 bytes del segmento <JUNK>. Debe ser por lo menos 28 para ser un marcador de posición del segmento <ds64>.
ckData	Datos ficticios que se deben ignorar.

5 Segmento AXML

5.1 Definición

El segmento <axml> puede incluir cualquier dato conforme con el formato XML 1.0 o posterior, un formato muy extendido para el intercambio de datos [1]. Cabe destacar que el segmento <axml> puede incluir fragmentos XML de más de un esquema. Se puede producir en cualquier orden con otros segmentos RIFF dentro del mismo fichero.

El segmento <axml> está constituido por un encabezamiento seguido por datos acordes con el formato XML. La longitud global del segmento no es fija.

En el § 11se muestra un ejemplo de cómo se puede utilizar el segmento <axml> en BW64 para transportar metadatos de radiodifusión, incluidos los parámetros de los anteriores segmentos <bext> y <ubxt>.

```
struct axml_chunk
{
    CHAR    ckID[4];       // {'a','x','m','l'}
    DWORD   ckSize;        // size of the <axml> chunk in bytes
    CHAR    xmlData[];    // text data in XML
};
```

Puesto que XML puede ocupar más de 4 Gbytes podría ser necesario utilizar el segmento <ds64> para permitir un campo de tamaño de 64 bits para el segmento <axml>. A continuación figura un seudocódigo para mostrar cómo se puede conseguir, utilizando el conjunto «table» en el segmento <ds64>.

```
DataSize64Chunk.tableLength = 1;   // number of valid entries in array "table"
DataSize64Chunk.table[0] = {
    ChunkSize64.ckID = {`a`, `x`, `m`, `l`};   // chunk ID for the <axml> chunk
     ckSizeLow = xxxx     // low 4 byte chunk size
     ckSizeHigh = xxxx    // high 4 byte chunk size
}
```

5.2 Elementos del segmento <axml>

Campo	Descripción
ckID	Conjunto de 4 caracteres {'a', 'x', 'm', 'l'} utilizado para la identificación del segmento.
ckSize	Tamaño de la porción de datos del segmento en bytes. (No incluye los 8 bytes utilizados por ckID y ckSize.)
xmlData	Este campo contiene la información de texto en XML.

La estructura de datos XML es jerárquica y los datos se almacenan en cadenas de texto de conformidad con el formato XML 1.0 o superior.

Si el dispositivo receptor no puede interpretar el contenido del segmento <axml> de conformidad con la especificación determinada en el XML, se debe ignorar la totalidad del segmento.

6 Segmento BXML

6.1 Definición

El segmento <bxml> puede contener datos XML comprimidos en lugar del segmento <axml>.

El segmento <bxml> consta de un encabezamiento seguido por los datos XML comprimidos utilizando el método de compresión especificado en **fmtType**. La longitud total del segmento no es fija.

```
struct bxml_chunk
{
    CHAR    ckID[4];       // {'b','x','m','l'}
    DWORD   ckSize;        // size of the <bxml> chunk in bytes
    WORD    fmtType;       // type of compression method, 0x0001=”gzip”, etc.
    CHAR    xmlData[];    // XML text data compressed by the compression method
};
```

Puesto que XML puede ocupar más de 4 Gbytes podría ser necesario utilizar el segmento <ds64> para permitir un campo de tamaño de 64 bits para el segmento <bxml>. A continuación figura un seudocódigo para mostrar cómo se puede conseguir, utilizando el conjunto «table» en el segmento <ds64>.

```
DataSize64Chunk.tableLength = 1;   // number of valid entries in array “table”
DataSize64Chunk.table[0] = {
    ChunkSize64.ckID = {`b`, `x`, `m`, `l`};   // chunk ID for the <bxml> chunk
     ckSizeLow = xxxx     // low-4-byte chunk size
     ckSizeHigh = xxxx    // high-4-byte chunk size
}
```

6.2 Elementos del segmento <bxml>

ckID	Conjunto de 4 caracteres {'b', 'x', 'm', 'l'} utilizado para la identificación del segmento.
ckSize	Tamaño de la porción de datos del segmento en bytes. (No incluye los 8 bytes utilizados por ckID y ckSize.)
fmtType	Valor de 2 bytes que representa el método de compresión del texto XML. El valor 0x0001 significa que el método de compresión es gzip (IETF RFC 1952). 0x0000 se utiliza para el texto XML sin comprimir.
xmlData	Este campo contiene el código XML comprimido con el método de compresión indicado en fmtType.

7 Segmento SXML

7.1 Definición

El segmento <sxml> puede contener cualquier dato XML comprimido o sin comprimir compatible con el formato XML 1.0 o posterior asociado con segmentos de datos de audio. Puede aparecer en cualquier orden con otros segmentos RIFF en el mismo fichero.

El segmento <sxml> consta de un encabezamiento seguido por subsegmentos **(SubXMLChunk)** con datos XML comprimidos o sin comprimir, según se especifique en **fmtType**. Cada **SubXMLChunk** corresponde a un único número de muestras de audio contiguas al **SubXMLChunks** adyacente. El

segmento <sxml> se completa con un conjunto «table» de puntos de alineación optativo, que permite acceder por sellos de tiempo al **SubXMLChunk** seleccionado. La longitud total del segmento <sxml> no es fija.

El segmento <sxml> puede utilizarse para el transporte de metadatos variables en el tiempo, por ejemplo, una representación en serie del ADM especificado en la Recomendación UIT-R BS.2125.

```
struct sxml_chunk
{
    CHAR    ckID[4];              // {'s','x','m','l'}
    DWORD   ckSize;               // size of the <sxml> chunk in bytes
    WORD    fmtType;              // type of compression method, 0x0001=”gzip”, etc.
    DWORD   subXMLCkTbSizeLow;    //  low  4  byte  of  size  of  nSubXMLChunks  +
                                  // SubXMLChunk table[]
    DWORD   subXMLCkTbSizeHigh;   //  high  4  byte  of  size  of  nSubXMLChunks  +
                                  // SubXMLChunk table[]
    DWORD   nSubXMLChunks;        // number of sub-chunks with XML data
    SubXMLChunk table[];          // array of sub-chunks with XML data
    DWORD   nAlignmentPoints;     // number of alignment points
    AlignmentPoint table[];       // array of alignment points
};

struct SubXMLChunk
{
    DWORD   subXMLChunkSize;      // size of SubXMLChunk in bytes
    DWORD   nSamplesSubDataChunk;// number of audio samples associated with SubXMLChunk
    CHAR    xmlData[];           // compressed or uncompressed XML data
};

struct AlignmentPoint
{
    DWORD  subXMLChunkByteOffsetLow;  // low 4 byte of SubXMLChunk byte offset
    DWORD  subXMLChunkByteOffsetHigh; // high 4 byte of SubXMLChunk byte offset
    DWORD  nSamplesAlignPointLow;     // low 4 byte of alignment point sample count
    DWORD  nSamplesAlignPointHigh;    // high 4 byte of alignment point sample count
};
```

Puesto que XML comprimido o sin comprimir puede ocupar más de 4 Gbytes podría ser necesario utilizar el segmento <ds64> para permitir un campo de tamaño de 64 bits para el segmento <sxml>. A continuación figura un seudocódigo para mostrar cómo se puede conseguir, utilizando el conjunto «table» en el segmento <ds64>.

```
DataSize64Chunk.tableLength = 1;   // number of valid entries in array “table”
DataSize64Chunk.table[0] = {
    ChunkSize64.ckID = {`s`, `x`, `m`, `l`};   // chunk ID for the <sxml> chunk
     ckSizeLow = xxxx     // low-4-byte chunk size
     ckSizeHigh = xxxx    // high-4-byte chunk size
}
```

7.2 Elementos del segmento <sxml>

Campo	Descripción
ckID	Conjunto de 4 caracteres {'s', 'x', 'm', 'l'} utilizado para la identificación del segmento.
ckSize	Tamaño de la porción de datos del segmento en bytes. No incluye los 8 bytes utilizados por ckID y ckSize.
fmtType	Valor de 2 bytes que representa el método de compresión del texto XML. El valor 0x0001 indica que el método de compresión es gzip (IETF RFC 1952). 0x0000 se utiliza para indicar datos XML sin comprimir.
subXMLCkTbSizeLow	Tamaño reducido de 4 bytes del SubXMLChunk table[], incluidos los 4 bytes del campo nSubXMLChunks. El tamaño de datos de 64 bits se expresa como 0xHHHHLLLL si <subXMLCkTbSizeLow> y <subXMLChTbSizeHigh> son 0xLLLL y 0xHHHH, respectivamente. La cantidad sin signo de 32 bits está en formato little-endian.
subXMLCkTbSizeHigh	Tamaño elevado de 4 bytes del SubXMLChunk table[], incluidos los 4 bytes del campo nSubXMLChunks.
nSubXMLChunks	Número de entradas válidas en la matriz «SubXMLChunk table».
SubXMLChunk table	Matriz de subsegmentos con datos XML.
nAlignmentPoints	Número de entradas válidas en la matriz «AlignmentPoint table».
AlignmentPoint table	Matriz de puntos de alineación.

La matriz **SubXMLChunk** se especifica de la siguiente manera:

Campo	Descripción
subXMLChunkSize	Tamaño de la sección de datos del segmento SubXMLChunk en bytes, excluidos los 8 bytes utilizados por los segmentos subXMLChunkSize y nSamplesSubDataChunk.
nSamplesSubDataChunk	Número de muestras de audio por canal asociadas con el segmento SubXMLChunk.
xmlData	Este campo contiene los datos XML o datos XML comprimidos utilizando el método de compresión indicado en fmtType.

La matriz **AlignmentPoint** se especifica de la siguiente manera:

Campo	Descripción
subXMLChunkByteOffsetLow	Desplazamiento de byte de inicio de un segmento SubXMLChunk con el punto de alineación expresado en bytes con respecto al inicio del segmento <sxml>, excluidos los 8 bytes utilizados por ckID y ckSize. Tamaño reducido de 4 bytes del desplazamiento de byte de inicio de

	SubXMLChunk. El tamaño de datos de 64 bits se expresa como 0xHHHHLLLL si <subXMLChunkByteOffsetLow> y <subXMLChunkByteOffsetHigh> son 0xLLLL y 0xHHHH, respectivamente. La cantidad de 32 bits sin signo está en formato little-endian.
subXMLChunkByteOffsetHigh	Tamaño elevado de 4 bytes del desplazamiento de byte de inicio de SubXMLChunk con punto de alineación. La cantidad de 32 bits sin signo está en formato little-endian.
nSamplesAlignPointLow	Sello de tiempo del punto de alineación expresado en muestras de audio por canal desde el principio del segmento <data>. Tamaño reducido de 4 bytes de la cuenta de muestras de sello de tiempo. La cuenta de 64 bits se expresa como 0xHHHHLLLL si <nSamplesAlignPointLow> y <nSamplesAlignPointHigh> son 0xLLLL y 0xHHHH, respectivamente. La cantidad de 32 bits sin signo está en formato little-endian.
nSamplesAlignPointHigh	Tamaño elevado de 4 bytes de la cuenta de muestras de sello de tiempo. La cantidad de 32 bits sin signo está en formato little-endian.

8 Segmento CHNA

8.1 Definición

El segmento <chna> es un segmento específicamente definido para su uso con ADM como se especifica en la Recomendación UIT-R BS.2076. El segmento <chna> comprende un encabezamiento seguido por el número de pistas y por el número de UID de pista utilizados. A continuación figura un conjunto de estructuras de ID que contiene cada una de ellas los ID correspondientes a los ID de elementos ADM.

El tamaño del segmento depende el número de UID de pista por definir. El número de estructuras ID debe ser igual o superior al número de UID de pista utilizado. Permitir que el número de estructuras ID supere al número de UID puede facilitar la actualización y la adición de nuevos ID al segmento sin tener que cambiar el tamaño del segmento. Por ejemplo, puede que no esté claro cuántos ID se generarán al principio, de forma que, si el número de estructuras ID en el segmento se fija a 64 (considerado por el diseñador como más que suficiente para esta tarea), el soporte lógico genera entonces 55 UID (número estimado de UID iniciales) que rellenan las primeras 55 estructuras ID y las restantes 9 estructuras ID se fijan a valores cero.

Los ID del ADM dentro del segmento pueden hacer referencia a metadatos ADM transportados en los segmentos <axml>, <bxml> y <sxml> o en un fichero de definición común externo. Si los cuatro últimos dígitos hexadecimales de los ID tienen un valor igual o inferior a 0x0FFF entonces se definen como definiciones comunes de la Recomendación UIT-R BS.2094-0 – *Definiciones comunes para el modelo de definición de audio* (por ejemplo, definiciones de canal para 'FrontLeft' y 'FrontRight'). Cualesquiera ID con valores de 0x1 000 y superiores se definen como definiciones comunes, de forma que estén incluidos en los segmentos <axml>, <bxml> y <sxml> dentro del fichero.

La estructura audioID incluye un índice para la pista utilizada en el segmento <data> (que incluye las muestras de audio), empezando con el valor 1 para la primera pista. Contiene un UID para la pista, que estará incluido en los metadatos ADM. Los elementos de audio de una pista pueden ser diferentes a lo largo de un fichero; en este caso, existirá un UID diferente para cada definición. Por lo tanto es

posible tener múltiples UID para cada pista. Los otros dos valores en la estructura son referencias a los ID de los elementos audioTrackFormat y audioPackFormat del ADM. ADM permite la omisión de audioTrackFormat y audioStreamFormat si el tipo de formato de la esencia de audio es PCM lineal. En ese caso, se hace referencia a audioChannelFormat en lugar de audioTrackFormat.

```
struct chna_chunk
{
    CHAR    ckID[4];        // {'c','h','n','a'}
    DWORD   ckSize;         // size of the <chna> chunk
    WORD    numTracks;      // number of tracks used
    WORD    numUIDs;        // number of track UIDs used
    audioID ID[N];          // IDs for each track  (where N >= numUIDs)
};

struct audioID
{
    WORD    trackIndex;     // index of track in file
    CHAR    UID[12];        // audioTrackUID value
    CHAR    trackRef[14];   // audioTrackFormatID or audioChannelFormatID reference
    CHAR    packRef[11];    // audioPackFormatID reference
    CHAR    pad;            // padding byte to ensure even number of bytes
}
```

8.2 Elementos del segmento <chna>

ckID Conjunto de 4 caracteres {'c','h','n','a'}[1] para la identificación del segmento.

ckSize Tamaño de la sección de datos del segmento en bytes. (No incluye los 8 bytes utilizados por ckID y ckSize.)

numTracks Número de pistas utilizado en el fichero. Aunque una pista incluya más de un conjunto de ID sigue siendo sólo una pista.

numUIDs Número de UID utilizados en el fichero. Puesto que es posible dar a una única pista múltiples UID (que cubran diferentes periodos de tiempo), podría ser un valor superior a **numTracks**. Este valor debería corresponder al número de ID definidos en **ID**.

ID Estructura que incluye el conjunto de ID de referencia de audio para la pista. Este conjunto contiene N ID, siendo N >= numUIDs. Cuando numUIDs es inferior a N el contenido de los ID de pista no utilizados se fija a cero. Cuando se lee el segmento, el valor de N se puede obtener a partir de ckSize, puesto que ckSize = 4 + (N * 40), de forma que N = (ckSize – 4) / 40.

trackIndex Índice de la pista en el fichero, comenzando por 1. Corresponde directamente al orden de las pistas entrelazadas en el segmento <data>.

UID Valor de audioTrackUID de la pista. El conjunto de caracteres tiene el formato ATU_xxxxxxxx donde x es un dígito hexadecimal.

---

[1] **Nota:** La definición de DWORD ckID = «chna» no sería única. Diferentes arquitecturas generan diferentes órdenes de caracteres. Por lo tanto, definimos el carácter ckID[4] = {'c','h','n','a'} en su lugar.

trackRef Referencia de audioTrackFormatID de la pista. El conjunto de caracteres tiene el formato AT_xxxxxxxx_xx donde x es un dígito hexadecimal. También se utiliza el formato AC_xxxxxxxx_00 (el sufijo «00» rellena la cadena para ajustarse al formato de la cadena audioTrackFormatID y no tiene significado), siendo x un dígito hexadecimal, cuando se omiten tanto audioTrackFormat y audioStreamFormat para la esencia de audio de la PCM lineal y audioChannelFormat está directamente referenciado en el código XML ADM.

packRef Referencia de audioPackFormatID de la pista. El conjunto de caracteres tiene el formato AP_xxxxxxxx donde x es un dígito hexadecimal. Cuando no se precisa audioPackFormatID (cuando audioStreamFormat se refiere a un audioPackFormat en lugar de a un audioChannelFormat) este campo debe rellenarse con valores nulos.

pad Un único byte para asegurar que la estructura audioID tiene un número par de bytes.

Cuando un **ID** no se está utilizando se debe otorgar al **trackIndex** un valor cero y a los restantes campos cadenas nulas con la misma longitud que la cadena normal de ID utilizada. De esta forma la cadena de nulos para packRef estará constituida por 11 caracteres nulos (valor cero en ASCII) y trackRef estará constituido por 14 caracteres nulos.

8.3 Ejemplos de carácter informativo

Para contribuir a ilustrar el funcionamiento del segmento <chna> se describen algunos ejemplos simples. El seudocódigo en cada ejemplo utiliza la notación en cadena para los ID (por ejemplo, «AT_00010001_01»), donde en la práctica haya que utilizar un conjunto de caracteres para evitar la inclusión de un carácter de terminación nulo al final (realmente se haría de esta forma: {'A','T','_','0','0','0','1','0','0','0','1','_','0','1'}).

8.3.1 Fichero estereofónico simple

La mayoría de los ficheros de audio existentes siguen siendo ficheros estereofónicos de dos canales, donde la primera pista contiene el canal izquierdo y la segunda pista el canal derecho. El ADM tiene una definición de canal izquierdo con un ID de AT_00010001_01 y de canal de derecho con un ID de AT_00010002_01. La definición del paquete estereofónico tiene un ID de AP_00010002.

El seudocódigo se muestra a continuación:

```
ckID = {‘c’,’h’,’n’,’a’};
ckSize = 84;
numTracks = 2;
numUIDs = 2;
ID[0]={ trackIndex=1; UID=“ATU_00000001”; trackRef=“AT_00010001_01”; packRef=“AP_00010002”; pad=‘\0`; };
ID[1]={ trackIndex=2; UID=“ATU_00000002”; trackRef=“AT_00010002_01”; packRef=“AP_00010002”; pad=‘\0`; };
```

El número de estructuras de ID es 2, por lo que no hay estructuras de ID sin utilizar en este ejemplo.

Cuando ADM omite tanto audioTrackFomat como audioStreamFormat y hace referencia a audioChannelFormat, se utiliza el siguiente código.

```
ckID = {‘c’,’h’,’n’,’a’};
ckSize = 84;
numTracks = 2;
```

```
numUIDs = 2;
ID[0]={ trackIndex=1; UID="ATU_00000001"; trackRef="AC_00010001_00"; packRef="AP_00010002"; pad='\0`; };
ID[1]={ trackIndex=2; UID="ATU_00000002"; trackRef="AC_00010002_00"; packRef="AP_00010002"; pad='\0`; };
```

8.3.2 Ejemplo basado en objeto simple

Los objetos de audio pueden cubrir únicamente una porción del tiempo en el fichero de audio. Para ahorrar espacio, pueden compartir la misma pista objetos que no se solapen. En este caso se pueden producir múltiples UID en la misma pista. Este ejemplo también utiliza más estructuras de ID (32 en este caso) que numUID para mostrar cómo las estructuras de ID se fijan a cero.

```
ckID = {'c','h','n','a'};
ckSize = 1284;
numTracks = 2;
numUIDs = 4;
ID[0]={ trackIndex=1; UID="ATU_00000001"; trackRef="AT_00031001_01"; packRef="AP_00031001"; pad='\0`; };
ID[1]={ trackIndex=1; UID="ATU_00000002"; trackRef="AT_00031003_01"; packRef="AP_00031002"; pad='\0`; };
ID[2]={ trackIndex=1; UID="ATU_00000003"; trackRef="AT_00031004_01"; packRef="AP_00031003"; pad='\0`; };
ID[3]={ trackIndex=2; UID="ATU_00000004"; trackRef="AT_00031002_01"; packRef="AP_00031001"; pad='\0`; };
ID[4]={ trackIndex=0; UID=['\0']*12;      trackRef=['\0']*14;        packRef=['\0']*11;     pad='\0`; };
   :
ID[31]={ trackIndex=0; UID=['\0']*12;      trackRef=['\0']*14;        packRef=['\0']*11;     pad='\0`; };
```

La primera pista contiene 3 UID, de forma que incluirá 3 objetos diferentes (con los ID de pista AT_00031001_01, AT_00031003_01 y AT_00031004_01) en diferentes ubicaciones temporales en el fichero. La segunda pista incluye un UID, de forma que sólo contiene un objeto. Este objeto contiene el mismo ID de paquete (AP_00031001) que el primer objeto en la pista 1. Esto sugiere que el primer objeto contiene dos canales transportados tanto en la pista 1 como en la pista 2. Los metadatos ADM transportados en los segmentos <axml>, <bxml> y <sxml> se utilizarían para aclarar la ubicación de los canales y de las pistas.

8.3.3 Ejemplo de contenido múltiple

El fichero BW64 podría incluir múltiples contenidos en un único fichero, tales como una mezcla 5.1 principal en las primeras 6 pistas, con una mezcla estereofónica de idiomas extranjeros en las siguientes 2 pistas. La Recomendación UIT-R BS.1738 incluye diversas configuraciones y el ejemplo mostrará cómo se pueden tratar 5 casos de producción a partir de la Recomendación dentro del segmento <chna>. Este caso incluye 8 pistas, conteniendo las 6 primeras una mezcla completa 5.1 y las 2 pistas siguientes una mezcla internacional estereofónica. El <chna> resultante se muestra a continuación:

```
ckID = {'c','h','n','a'};
ckSize = 324;
numTracks = 8;
numUIDs = 8;
ID[0]={ trackIndex=1; UID="ATU_00000001"; trackRef="AT_00010001_01"; packRef="AP_00010003"; pad='\0`; };
ID[1]={ trackIndex=2; UID="ATU_00000002"; trackRef="AT_00010002_01"; packRef="AP_00010003"; pad='\0`; };
ID[2]={ trackIndex=3; UID="ATU_00000003"; trackRef="AT_00010003_01"; packRef="AP_00010003"; pad='\0`; };
ID[3]={ trackIndex=4; UID="ATU_00000004"; trackRef="AT_00010004_01"; packRef="AP_00010003"; pad='\0`; };
ID[4]={ trackIndex=5; UID="ATU_00000005"; trackRef="AT_00010005_01"; packRef="AP_00010003"; pad='\0`; };
ID[5]={ trackIndex=6; UID="ATU_00000006"; trackRef="AT_00010006_01"; packRef="AP_00010003"; pad='\0`; };
```

```
ID[6]={ trackIndex=7; UID="ATU_00000007"; trackRef="AT_00010001_01"; packRef="AP_00010002"; pad='\0`; };
ID[7]={ trackIndex=8; UID="ATU_00000008"; trackRef="AT_00010002_01"; packRef="AP_00010002"; pad='\0`; };
```

Los metadatos ADM en los segmentos <axml>, <bxml> y <sxml> incluirán información de cómo se dividen las dos mezclas.

9 Reglas para los segmentos XML

Hay tres segmentos diferentes que pueden transportar metadatos XML: <axml>, <bxml> y <sxml>. Si bien el objetivo primario de estos segmentos es transportar metadatos XML ADM (como se especifica en la Recomendación UIT-R BS.2076) o metadatos S-ADM (como se especifica en la Recomendación UIT-R BS.2125), también pueden transportar otro tipo de metadatos XML, como los metadatos de radiodifusión descritos en el § 11. Al haber múltiples segmentos capaces de transportar metadatos XML, se corre el riesgo de que los metadatos de un segmento contradigan los metadatos de otro segmento. Por consiguiente, serán de aplicación las siguientes reglas:

1 No habrá más de un ejemplar de cualquier segmento XML concreto.

2 Si se transportan metadatos ADM:

   a) sólo aparecerán en el segmento <axml> o el segmento <bxml>, no en ambos;

   b) habrá un segmento <chna> presente en las referencias cruzadas de metadatos ADM.

3 Si se transportan metadatos S-ADM, sólo aparecerán en el segmento <sxml>.

4 Si se transportan tanto metadatos ADM como metadatos S-ADM, serán independientes unos de otros (es decir, no habrá referencias cruzadas entre ellos).

5 Si se transportan otros metadatos (es decir, ni ADM ni S-ADM):

   a) podrán transportarse junto con los metadatos ADM y S-ADM en el mismo segmento;

   b) el contenido de los «otros metadatos» no representará nada ya descrito en los metadatos ADM o S-ADM existentes;

   c) de haber referencias cruzadas entre los «otros metadatos» y los metadatos ADM o S-ADM, los metadatos ADM o S-ADM estarán presentes en el fichero.

10 Compatibilidad con la Recomendación UIT-R BS.1352

Puesto que el formato BWF (Recomendación UIT-R BS.1352) es el formato de fichero corto de RIFF/WAVE (descrito en el Anexo 2) con segmentos adicionales, en particular el segmento <bext>, es necesario comprender la compatibilidad entre BWF y BW64.

Segmentos BWF Rec. UIT-R BS.1352-3	Segmentos BW64 Rec. UIT-R BS.2088-0	Segmentos BW64 Rec. UIT-R BS.2088-1	Tratamiento
<fmt>	<fmt>	<fmt>	Uso convencional
<data>	<data>	<data>	Uso convencional
<fact>	<fact>	<fact>	Uso convencional [, aunque probablemente es redundante, por lo que se podría omitir]
–	<ds64>	<ds64>	Véase el § 2.4 y el § 4
–	<JUNK>	<JUNK>	Véase el § 2.4 y el § 4

Segmentos BWF Rec. UIT-R BS.1352-3	Segmentos BW64 Rec. UIT-R BS.2088-0	Segmentos BW64 Rec. UIT-R BS.2088-1	Tratamiento
–	<chna>	<chna>	Véase el § 8 para la atribución de canal. Nota: la Recomendación UIT-R BS.2088-0 no soporta la referencia audioChannelFormat
–	<axml>	<axml>, <bxml> o <sxml>	Véanse los § 5 a 7 para la atribución de canales. Uso para los metadatos de radiodifusión que pudiera haber en el segmento <bext>
<bext>	–	–	Cuando se lee un segmento <bext>, hay que convertir los datos de los segmentos <axml>, <bxml> y <sxml> correspondientes para transportar ADM y cualesquiera otros metadatos XML relacionados con la radiodifusión. Véase el § 10 para más detalles

11 Generación de metadatos de radiodifusión XML

La Recomendación UIT-R BS.1352 transporta metadatos de radiodifusión en los segmentos <bext> y <ubxt>. Estos segmentos tienen campos de longitud fija y están limitados a los campos especificados, evitando así cualquier otra radiodifusión relacionada con metadatos que se puedan transportar. Los segmentos <axml>, <bxml> y <sxml> en BW64 puede transportar cualesquiera metadatos XML y se puede utilizar por lo tanto para transportar metadatos de radiodifusión, entre ellos los parámetros en los segmentos <bext> y <ubxt>.

Para transportar los parámetros <bext>/<ubxt> en los segmentos <axml>, <bxml> y <sxml> se debe utilizar la estructura XML siguiente, en la que los comentarios con el prefijo 'BEXT' indican los parámetros del segmento <bext>/<ubxt>.

```
<?xml version="1.0" encoding="UTF-8"?>
<ebuCoreMain                    xmlns="urn:ebu:metadata-schema:ebuCore_2015"
xmlns:dc="http://purl.org/dc/elements/1.1/"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <coreMetadata>
    <creator>
      <contactDetails>
        <name>
          <!--BEXT: bextOriginator -->
        </name>
      </contactDetails>
      <organisationDetails>
        <organisationName>
          <!--BEXT: bextOriginatorReference -->
        </organisationName>
      </organisationDetails>
    </creator>

    <description typeDefinition="bextDescription">
      <dc:description>
        <!--BEXT: bextDescription -->
      </dc:description>
    </description>
    <date>
      <!--BEXT: bextOriginationDate and bextOriginationTime below-->
      <created startDate="2000-10-10" startTime="12:00:00"/>
    </date>
```

```
    <format>
      <audioFormatExtended>
        <!--BEXT: bextTimeReference below-->
        <audioProgramme audioProgrammeID="..." start="00:00:00:00">
          <!--Other audioProgramme metadata here -->
        </audioProgramme>
        <!--Other ITU-R BS.2076 ADM metadata here -->
      </audioFormatExtended>
      <technicalAttributeString typeDefinition="CodingHistory">
        <!--BEXT: bextCodingHistory -->
      </technicalAttributeString>
    </format>

    <identifier formatLabel="UMID"
formatLink="http://www.ebu.ch/metadata/cs/ebu_IdentifierTypeCodeCS.xml#1.1">
      <dc:identifier>
        <!--BEXT: bextUMID-->
      </dc:identifier>
    </identifier>
  </coreMetadata>
</ebuCoreMain>
```

XML se basa en los esquemas de metadatos EBUCore [2] y AESCore [3] que son compatibles con la Recomendación UIT-R BS.2076.

Cuando se lea un fichero BWF de la Recomendación UIT-R BS.1352 con la intención de convertirlo en un fichero BW64, se deben convertir los segmentos <bext>/<ubxt> en el XML descrito aquí para su inclusión en los segmentos <axml>, <bxml> y <sxml>.

12 Ampliación de fichero del fichero de formato BW64

La ampliación de fichero de los ficheros acordes al formato BW64 se define como «.wav». Esto permite al soporte lógico anterior leer los segmentos en el fichero que entiende (en primer lugar <fmt> y <data>), de forma que se pueda acceder por lo menos a las muestras de audio.

Aunque no se recomienda el uso de cualesquiera ampliaciones de ficheros alternativas cuando se generan ficheros BW64, es probable que se utilicen de forma inapropiada ampliaciones «.bw64». Por lo tanto, el soporte lógico que lee ficheros BW64 debe tolerar estas ampliaciones de ficheros alternativas.

13 Bibliografía

[1] Extensible Markup Language (XML) 1.0 W3C Recommendation 26 de noviembre de 2008 http://www.w3.org/TR/2008/REC-xml-20081126.

[2] EBU Tech 3293, «EBU Core Metadata Set v.1.6».

[3] AES 60-2011, «AES standard for audio metadata – Core audio metadata».

[4] IETF: RFC 1952, «GZIP file format specification version 4.3,» Internet Engineering Task Force, Reston, VA, May, 1996. http://tools.ietf.org/html/rfc1952.

Anexo 2
(informativo)

Formato de fichero RIFF WAVE (.WAV)

La información de este Anexo se ha tomado de los documentos de especificación del formato de fichero RFF. Se incluye únicamente con carácter informativo. Se incluye debido a la falta de fuentes de información externas fiables de referencia.

1 Formato de fichero audio de forma de onda (WAVE)

El formato de WAVE se define como sigue. El *software* debe ignorar cualesquiera segmentos desconocidos encontrados, con todas las formas RIFF. Sin embargo, <fmt-ck> aparece siempre antes que <wave-data> (datos de onda), y estos dos segmentos son obligatorios en un fichero WAVE.

```
<WAVE-form> ->
    RIFF('WAVE'
        <fmt-ck>       // Segmento de formato
        [<fact-ck>]    // Segmento ampliado
        [<other-ck>]   // Otros segmentos facultativos
        <wave-data>)   // Datos radiofónicos
```

Los segmentos WAVE se describen en las secciones siguientes:

1.1 Segmento de formato WAVE

El segmento de formato WAVE <fmt-ck> especifica el formato de <wave data>. <fmt-ck> se define como sigue:

```
<fmt-ck> ->fmt(<common-fields>
               <format-specific-fields>)
<common-fields> ->
    Struct {
        WORD  wFormatTag;           // Categoría de formato
        WORD  nChannels;            // Número de canales
        DWORD nSamplesPerSec;       // Velocidad de muestreo
        DWORD nAvgBytesPerSec;      // Para estimación de la memoria intermedia
        WORD  nBlockAlign;          // Tamaño de bloque de datos
    }
```

Los campos en la porción <common fields> (campos comunes) del segmento son los siguientes:

Campo	Descripción
wFormatTag	Número que indica la categoría del formato WAVE del fichero. El contenido de la porción <format-specific-fields> (campos específicos de formato) del segmento 'fmt' y la interpretación de los datos de la forma de onda, dependen de este valor.

nchannels	Número de canales representado en los datos de forma de onda, a saber, 1 para monofonía o 2 para estereofonía.
nSamplesPerSec	Velocidad de muestreo (en muestras por segundo) a la cual se debe reproducir cada canal.
nAvgBytesPerSec	Número medio de octetos por segundo en que se deben transferir los datos de forma de onda. El soporte lógico de reproducción puede estimar el tamaño de la memoria intermedia utilizando este valor.
nBlockAlign	Alineación de bloques (en bytes) de los datos de forma de onda. El soporte lógico de reproducción necesita procesar a la vez un múltiplo de bytes de datos <nBlockAlign> (número de alineación de bloque), con el fin de que el valor de <nBlockAlign> se pueda utilizar para la alineación de la memoria intermedia.

El campo (<format-specific-fields>) consta de cero, uno o más bytes de parámetros. Los parámetros que aparecen dependen de la categoría del formato WAVE: para más detalles, véanse las secciones siguientes. El soporte lógico de reproducción debe permitir (y pasar por alto) cualesquiera parámetros <format-specific-fields> desconocidos que aparezcan al final de este campo.

1.2 Categorías de formato WAVE

La categoría de formato de un fichero WAVE es especificada por el valor del campo <wFormatTag> (rótulo de formato) del segmento 'fmt'. La representación de datos en <wave data>, y el contenido de <format-specific-fields> del segmento 'fmt', dependen de la categoría del formato.

Entre las categorías de formato WAVE abiertas que no son privadas actualmente definidas cabe citar las siguientes:

wFormatTag	Valor	Categoría de formato
WAVE_FORMAT_UNKNOWN	0x0000	Desconocido
WAVE_FORMAT_PCM	0x0001	Formato MIC
WAVE_FORMAT_IEEE_FLOAT	0x0003	IEEE flotante
WAVE_FORMAT_EXTENSIBLE	0xFFFE	Formato de onda extensible – determinado por SubFormat

NOTA – Actualmente sólo se utilizan los formatos WAVE_FORMAT_PCM y WAVE_FORMAT_ UNKNOWN con BW64. En el § 2 siguiente se dan detalles sobre el formato PCM WAVE. En el § 3 figura información general sobre otros formatos WAVE. En el futuro se pueden definir otros formatos WAVE.

En el pasado se habrían utilizado formatos WAVE_FORMAT_EXTENSIBLE para ficheros multicanal pero eso debe evitarse en el futuro.

1.3 Segmento ampliado

El segmento <fact-ck> almacena información importante sobre el contenido de ficheros WAVE que no son MIC. Por tanto, este segmento no se utiliza en la presente versión del formato BW64. Este segmento se define como sigue:

```
<fact-ck> -> fact( <dwSampleLength:DWORD> )
```

<dwSampleLength> representa la longitud de los datos en las muestras. El campo <nSamplesPerSec> del encabezamiento del formato de onda se utiliza junto con al campo <dwSampleLength> para determinar la longitud de los datos en segundos.

El segmento ampliado se requiere para todos los formatos WAVE nuevos que no son MIC. Este segmento no se precisa para los ficheros normalizados WAVE_FORMAT_PCM.

El segmento ampliado se extenderá para incluir cualquier otra información requerida por futuros formatos WAVE. Los campos añadidos aparecerán después del campo <dwSampleLength>. Las aplicaciones pueden utilizar el campo de tamaño del segmento para determinar los campos que están presentes.

1.4 Otros segmentos facultativos

Se especifican algunos otros segmentos para su utilización en el formato WAVE. Los detalles de estos segmentos figuran en la especificación del formato WAVE y en cualesquiera actualizaciones publicadas ulteriormente.

NOTA – El formato WAVE puede admitir otros segmentos facultativos que pueden ser incluidos en ficheros WAVE para transportar información específica. Se considera que éstos son segmentos privados y serán omitidos por aplicaciones que no pueden interpretarlos.

2 Formato MIC

Si el campo <wFormatTag> del <fmt-ck> se pone a WAVE_FORMAT_PCM, los datos de forma de onda consisten en muestras representadas en formato MIC. Para los datos de forma de onda MIC, <format-specific-fields> se define como sigue:

```
<PCM-format-specific> ->
    struct {
        WORD  nBitsPerSample;    // Tamaño de muestra
    }
```

El campo <nBitsPerSample> (número de bits por muestra) especifica el número de bits de datos utilizado para representar cada muestra de cada canal. Si hay múltiples canales, el tamaño de la muestra es igual para cada canal.

El campo <nBlockAlign> debe ser igual a la siguiente fórmula, redondeada hasta el siguiente número entero:

$$\text{nchannels} \times \text{BytesPerSample}$$

El valor de BytesPerSample debe calcularse redondeando nBitsPerSample hasta el siguiente número entero. Cuando la palabra de muestra audio es menor que un número entero de bytes, los bits más significativos de la muestra audio se colocan en los bits más significativos de la palabra de datos, y los bits de datos no utilizados adyacentes al bit menos significativo se pondrán a cero.

Para datos MIC, el campo <nAvgBytesPerSec> (número medio de bytes por segundo) del segmento 'fmt' debe ser igual a la siguiente fórmula:

$$\text{nSamplesPerSec} \times \text{nBblockAlign}$$

NOTA 1 – La especificación WAVE original permite, por ejemplo, el empaquetamiento en 5 bytes de muestras de 20 bits tomadas de dos canales, compartiéndose un solo byte para los bits menos significativos de ambos canales. Esta Recomendación especifica un número completo de bytes por muestra audio con el fin de reducir ambigüedades en las implementaciones y alcanzar la máxima compatibilidad en los intercambios.

2.1 Empaquetado de datos para ficheros PCM WAVE

En un fichero WAVE monocanal, las muestras se almacenan consecutivamente. Para ficheros WAVE estereofónicos, el canal 0 representa el canal izquierdo y el canal 1 representa el canal derecho. En ficheros WAVE multicanal, las muestras están entrelazadas.

En los siguientes diagramas se observa el empaquetado de datos para ficheros WAVE monofónicos y estereofónicos de 8 bits:

Empaquetado de datos para MIC monofónica de 8 bits

Muestra 1	Muestra 2	Muestra 3	Muestra 4
Canal 0	Canal 0	Canal 0	Canal 0

Empaquetado de datos para MIC estereofónica de 8 bits

Muestra 1		Muestra 2	
Canal 0 (izquierda)	Canal 1 (derecha)	Canal 0 (izquierda)	Canal 1 (derecha)

En los siguientes diagramas se observa el empaquetado de datos para ficheros WAVE monofónicos y estereofónicos de 16 bits:

Empaquetado de datos para MIC monofónica de 16 bits

Muestra 1		Muestra 2	
Canal 0 octeto de orden inferior	Canal 0 octeto de orden superior	Canal 0 octeto de orden inferior	Canal 0 octeto de orden superior

Empaquetado de datos para MIC estereofónica de 16 bits

Muestra 1			
Canal 0 (izquierda)	Canal 0 (izquierda)	Canal 1 (derecha)	Canal 1 (derecha)
octeto de orden inferior	octeto de orden superior	octeto de orden inferior	octeto de orden superior

2.2 Formato de datos de las muestras

Cada muestra está contenida en un entero i. El tamaño de i es el número más pequeño de octetos requeridos para contener el tamaño de muestra especificado. El octeto menos significativo se almacena primero. Los bits que representan la amplitud de la muestra se almacenan en los bits más significativos de i, y los bits restantes se ponen a cero.

Por ejemplo, si el tamaño de la muestra (grabada en <nBitsPerSample> (número de bits por muestra)) es 12 bits, cada muestra se almacena en un entero de 2 octetos. Los cuatro bits menos significativos del primer octeto (menos significativo) se ponen a cero. El formato de datos y los valores máximos y mínimos de las muestras de forma de onda MIC de varios tamaños son como sigue:

Tamaño de muestra	Formato de datos	Valor máximo	Valor mínimo
Uno a ocho bits	Entero sin signo	255 (0xFF)	0
Nueve o más bits	Entero con signo i	Valor positivo máximo de i	Valor más negativo de i

Por ejemplo, los valores máximo, mínimo y medio para datos de forma de onda MIC de 8 bits y de 16 bits son como sigue:

Formato	Valor máximo	Valor mínimo	Valor medio
8-bit MIC	255 (0xFF)	0	128 (0x80)
16-bit MIC	32 767(0x7FFF)	–32 768(–0x8000)	0

2.3 Muestra de ficheros PCM WAVE

Ejemplo de un fichero PCM WAVE con velocidad de muestreo de 11,025 kHz, monofónico, 8 bits por muestra:

```
RIFF('WAVE' fmt(1, 1, 11025, 11025, 1, 8)
          data(<wave-data>) )
```

Ejemplo de un fichero PCM WAVE con velocidad de muestreo de 22,05 kHz, estereofónico, 8 bits por muestra:

```
RIFF('WAVE' fmt(1, 2, 22050, 44100, 2, 8)
          data(<wave-data>) )
```

2.4 Almacenamiento de datos WAVE

<**wave data**> contiene los datos de forma de onda y se define como sigue:

```
<wave-data> -> { <data-ck> }
    <data-ck> -> data( <wave-data> )
```

2.5 Segmento ampliado

El segmento ampliado <fact-ck> almacena información importante sobre el contenido del fichero WAVE. Este segmento se define como sigue:

```
<fact-ck> ->  fact(<dwFileSize:DWORD>)    // Número de muestras
```

Este segmento no se requiere para ficheros MIC.

El segmento ampliado se extenderá para incluir cualquier otra información requerida por futuros formatos WAVE. Los campos añadidos aparecerán después del campo <dwFileSize> (tamaño de fichero). Las aplicaciones pueden utilizar el campo de tamaño del segmento para determinar los campos que están presentes.

2.6 Otros segmentos facultativos

Se especifican algunos otros segmentos para utilización en el formato WAVE. Los detalles de estos segmentos figuran en la especificación del formato WAVE y cualesquiera actualizaciones publicadas ulteriormente.

NOTA 1 – El formato WAVE puede admitir otros segmentos facultativos que pueden ser incluidos en ficheros WAVE para transportar información específica. Se considera que éstos son segmentos privados y serán omitidos por aplicaciones que no pueden interpretarlos.

3 Extensión del formato WAVE

La estructura de formato de onda ampliada añadida a <fmt-ck> se utiliza para definir todos los datos de onda de formato que no son MIC, y se describe como sigue. La estructura de formato de forma de onda ampliada general se utiliza para todos los formatos que no son MIC.

```
typedef struct waveformat_extended_tag {
    WORD  wFormatTag;      // tipo de formato
    WORD  nChannels;       // número de canales (monofónicos, estereofónicos)
    DWORD nSamplesPerSec;  // velocidad de muestreo
    DWORD nAvgBytesPerSec; // para estimación de la memoria intermedia
    WORD  nBlockAlign;     // tamaño de bloque de datos
    WORD  wBitsPerSample;  // número de bits por muestra de datos monofónicos
    WORD  cbSize;          // cómputo en octetos del tamaño suplementario
} WAVEFORMATEX;
```

Campo	Descripción
wFormatTag	Define el tipo de fichero WAVE.
nChannels	Número de canales en la onda, 1 para monofonía, 2 para estereofonía.
nSamplesPerSec	Frecuencia de la velocidad de muestreo del fichero. Debe ser 48 000 ó 44 100, etc. Esta velocidad es utilizada también por la entrada de tamaño de muestra en el segmento ampliado para determinar la duración de los datos.
nAvgBytesPerSec	Velocidad media de datos. El soporte lógico de reproducción puede estimar el tamaño de la memoria intermedia utilizando el valor <nAvgBytesPerSec> (número medio de octetos por segundo).
nBlockAlign	La alineación de bloques (en octetos) de los datos de <data-ck> (segmento de datos). El soporte lógico de reproducción tiene que procesar a la vez un múltiplo de octetos de datos (<nBlockAlign>), de modo que el valor de <nBlockAlign> se pueda utilizar para la alineación de la memoria intermedia.
wBitsPerSample	Este es el número de bits por muestra por canal. Se supone que cada canal tenga la misma resolución de muestreo. Si este campo no es necesario, se debe poner a cero.
cbSize	El tamaño en octetos de la información suplementaria en el encabezamiento del formato WAVE no incluye el tamaño de la estructura WAVEFORMATEX.

NOTA – Los campos que siguen al campo <cbSize> (tamaño) contienen información específica necesaria para el formato WAVE definido en el campo <wFormatTag> (rótulo de formato). Cualesquiera formatos WAVE que puedan ser utilizados en el BWF se especificarán en Suplementos a la presente Recomendación.

Anexo 3
(normativo)

Definiciones de tipos de datos primitivos

Las siguientes son etiquetas atómicas que se refieren a tipos de datos primitivos. También se enumera el tipo de datos C equivalente.

Etiqueta	Significado	Tipo C
<CHAR>	Número entero de 8 bits con signo	Carácter con signo
<BYTE>	Número entero de 8 bits sin signo	Carácter sin signo
<INT>	Número entero de 16 bits con signo en formato little-endian	Entero con signo
<WORD>	Cantidad de 16 bits sin signo en formato little-endian	Entero sin signo
<LONG>	Número entero de 32 bits con signo en formato little-endian	Longitud con signo
<DWORD>	Cantidad de 32 bits con signo en formato little-endian	Longitud sin signo
<FLOAT>	Número con coma flotante IEEE de 32 bits	Flotante
<DOUBLE>	Número con coma flotante IEEE de 64 bits	Doble
<STR>	Cadena (secuencia de caracteres)	
<ZSTR>	Cadena terminada en NULL	
<BSTR>	Cadena con prefijo de tamaño en bytes (8 bits)	
<WSTR>	Cadena con prefijo de tamaño de palabra (16 bits)	
<BZSTR>	Cadena terminada en NULL con prefijo de tamaño en bytes	

Anexo 4
(informativo)

Modificaciones en la especificación del Anexo 1

1 Recomendación UIT-R BS.2088-1

En la Revisión 1 de esta Recomendación se introducen los siguientes cambios en la especificación del Anexo 1:

– Adición del segmento BXML en el § 6.

– Adición del segmento SXML en el § 7.

– Adición de una nueva función de omisión de audioTrackFormat y audioStreamFormat en el § 8.