ITU出版物
建议书

国际电信联盟
无线电通信部门

ITU-R BS. 2088-2 建议书

(11/2025)

BS系列：广播业务（声音）

带有元数据的音频节目素材国际交换的长文件格式

前言

无线电通信部门的作用是确保所有无线电通信业务，包括卫星业务，合理、公平、有效和经济地使用无线电频谱，并开展没有频率范围限制的研究，在此基础上通过建议书。

无线电通信部门制定规章制度和政策的职能由世界和区域无线电通信大会以及无线电通信全会完成，并得到各研究组的支持。

知识产权

国际电联提请注意：本建议书的应用或实施可能涉及使用已申报的知识产权。国际电联对无论是其成员还是建议书制定程序之外的其他机构提出的有关已申报的知识产权的证据、有效性或适用性不表示意见。

至本建议书批准之日止，国际电联未收到实施本建议书可能需要的受专利保护的知识产权的通知。但需要提醒实施者注意的是，这可能并非最新信息，因此大力提倡他们通过下列网站查询适当的 ITU-R 专利信息：https://www.itu.int/en/ITU-R/study-groups/Pages/itu-r-patent-information.aspx。

ITU-R 建议书系列

（可同时在以下网址获得：https://www.itu.int/publ/R-REC/zh）

系列	标题
BO	卫星传输
BR	用于制作、存档和播放的记录；用于电视的胶片
BS	**广播业务（声音）**
BT	广播业务（电视）
F	固定业务
M	移动、无线电测定、业余无线电以及相关卫星业务
P	无线电波传播
RA	射电天文
RS	遥感系统
S	卫星固定业务
SA	空间应用和气象
SF	卫星固定和固定业务系统之间频率共用和协调
SM	频谱管理
SNG	卫星新闻采集
TF	时间信号和标准频率发射
V	词汇和相关课题

注：本ITU-R建议书英文版已按ITU-R第1号决议规定的程序批准。

电子出版物
2026年，日内瓦

ITU-R BS.2088-2 建议书*

带有元数据的音频节目素材
国际交换的长文件格式

（2015-2019-2025年）

范围

本建议书包括64位广播波（BW64）音频文件格式的规范，包括新块<ds64>、<axml>、<bxml>、<sxml>和<chna>，其能使文件携带大的多声道文件以及元数据，包括ITU-R BS.2076建议书中定义的音频定义模型（ADM）。

关键词

文件、文件格式、元数据、波形、BW64、交换、音频节目、WAV、BWF、RIFF、RF64、波形文件、沉浸式音频、音频定义模型（ADM）、串行ADM（S-ADM）

国际电联无线电通信全会，

考虑到

a) 基于信息技术的存储介质，包括数据光盘和磁带，已经渗透到采用无线广播的音频产品的所有领域，即非线性编辑、空中播放和存档；

b) 本技术在运营灵活性、生产流程和站台自动化方面表现出强大的优势，并且因此在现有演播室的升级以及新的演播室设施的设计方面广受欢迎；

c) 信号交换采用单一的文件格式，极大简化了个别装备和远端演播室之间的互操作能力，它还便于编辑、空中播放和存档的综合操作；

d) 记录与音频信号相关的元数据的文件中必须包括广播相关信息的最小集；

e) 为保证复杂程度不同的应用程序之间的兼容性，必须对功能的最小集协商一致，它对所有能够处理被推荐文件格式的应用是共同的；

f) ITU-R BS.646建议书定义用于无线电和电视广播的音频产品中使用的数字音频格式；

g) 与现有可用商业文件格式的兼容能减少在设备中应用这种格式时所需的行业努力；

h) 历史信息的编码的标准格式将简化节目交换之后信息的使用；

i) 音频信号的品质受信号处理过程的影响，特别是在比特率降低过程中受非线性编码和解码的使用的影响；

j) 高级音频系统要求在文件中携带与音频相关联的元数据；

k) 高级音频系统使用多种多声道配置，包括声道、对象和场景音频等，这些在ITU-R BS.2051建议书中都有列举；

---

* 2026年3月，无线电通信第6研究组根据ITU-R第1号决议，对该建议书进行了编辑性修订。

l) ITU-R BS.2076建议书中指定了用于高级声音系统的音频相关元数据，并在ITU-R BS.2094建议书中指定了其通用定义，并在ITU-R BS.2125建议书中指定了元数据的序列化表示（S-ADM）；

m) ITU-R BS.1352建议书在文件大小及其携带附加元数据的能力方面有限制；

n) 多声道音频很可能比4 GB的文件更大，

建议

1 对于音频节目交换，应按照ITU-R BS.646建议书的相关部分设置音频信号参数、抽样频率（第1部分）、位深度（第4和5部分）以及预加重（第6部分）；

2 附件1中详述的文件格式应当用于以下用例中的音频节目互换：

- 在基于波形文件的环境下，基于波形文件的广播应用程序想要升级以处理拟真内容，并同时维持向前兼容性；
- 在基于文件的工作流中，将存在旧系统的基于波形文件的内容以及拟真内容的混合库；
- 在基于文件的工作流中，一个单包数据附加元数据封装是首选的。

注 – 附件4说明了本建议书的上一版对附件1中的规范所做的更改，仅供参考。

附件1
（规范性）

BW64文件格式的规范

1 引言

BW64格式是资源交换文件格式（RIFF）中详述的一种文件，以波形音频文件格式（在附件2中有所描述）为基础。WAVE文件明确包括音频数据。RIFF文件格式的基本结构块，即简称为块，包括一组密切相关的信息元。它包括块标识符，一个代表字节的长度和承载的信息的整数值。一个RIFF文件由块的组合构成。如EBU技术3306中所描述，本BW64格式使用格式的核心元素。

对于BWF文件格式、ITU-R BS.1352建议书有很多局限性，最明显的就是：

- 最大文件大小小于4 GB。
- 由于音频相关元数据有限，不支持高级多声道音频。
- 对技术元数据的支持不足。

本建议书中所描述的BW64格式，旨在克服这些局限性，并通过ITU-R BS.1352建议书中的格式（有很多共享的核心元素）尽可能多地维持兼容性。

根据ITU-R BS.2076建议书，对元数据的传输，尤其是音频定义模型（ADM）元数据的传输的需求不断增长。本建议书包括<axml>、<bxml>和<sxml>块的定义，分别以压缩和序列化格式，用于存储和传输UTF-8编写的XML格式元数据。

本建议书中描述的<chna>块的主要用途是，将参考从BW64文件中的每个轨道提供至ITU-R BS.2076建议书中定义的ADM元数据中的ID。

除了完成将文件中的每个轨道与其关联ADM元数据链接这个主要目的之外，<chna>块还允许更快地访问ADM ID，而无须访问XML元数据（如果ID在标准ADM配置预定义的值范围内）。由于<chna>块的大小可以固定，并且位于<data>、<axml>、<bxml>和<sxml>块之前，因此可以更容易地即时访问、生成或修改其内容。

根据附件3使用这篇文档中的数据类型。

2 BW64格式描述

2.1 BW64格式文件的内容

BW64格式文件的开始须采用强制“WAVE”标头以及至少包括如下块：

```
<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
```

注 1 – 另外的块也可以出现在文件中。有些可能超出了本建议书讨论的范围。这些应用可能或可能不说明或使用这些块，因此不能保证该未知块中包括的数据的完整性。但是，符合要求的应用必须在未知块上透明地传送。其他块，包括BWF生成的<bext>块和<ubxt>块可能会出现在本建议书规定的BW64文件中。不过，符合ITU-R BS.2088（BW64）建议书的应用可以读取XML元数据，这些元数据也可能包含从传统块转换而来的元数据，包括BWF <bext> 和<ubxt>块。

注 2 – 由于XML元数据可能是文件中的音频取样的未知长度和已知起始位置，将<axml>、<bxml>或<sxml>块置于<data>块之后是允许的且可能更实用。

2.2 定义为RIFF/WAVE标准部分的现有的块

RIFF/WAVE标准使用已经定义的一些块。其中有：

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

这些块在第2.6.1至2.6.3节中进行了描述。

RIFF/WAVE是ITU-R BS.1352建议书格式的一个子集。ITU-R BS.1352建议书包含这些附加块：

- <bext>
- <ubxt>

这些块可以包含在BW64格式中。但是，这些块中包含的内容应转换为包含在XML相关块中的XML元数据（见第10和11节）。应用程序应使用XML元数据。有关BW64格式使用的信息见ITU-R BS.2388号报告。

2.3 BW64格式中的新块和结构

为BW64引入的新块如下：

- <BW64>
- <ds64>
- <JUNK>
- <axml>、<bxml>或<sxml>
- <chna>

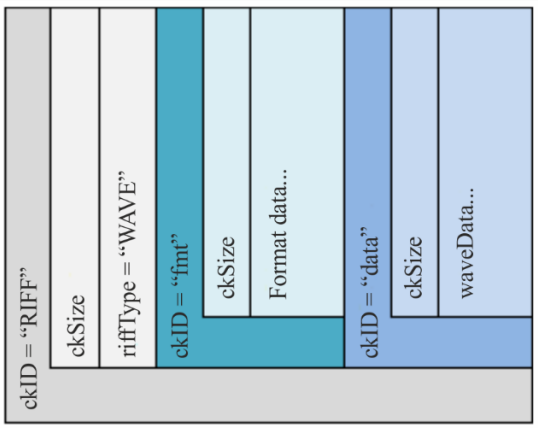
这些块在第3节至第8节有所描述。

2.4 使用<ds64>块让大于4 GB的文件得以使用

4 GB屏障的原因是RIFF/WAVE与BWF中的32位寻址。在32位的情况下，最大4294967296字节 = 4 GB能够被寻址。为解决该问题，需要64位寻址。图1呈现了一个基本的、常用的RIFF/WAVE文件的结构，其中ckSize字段是32位数字，表示其块的大小。

图1

基本RIFF/WAVE文件结构



仅是将BWF中的每个字段对象的大小改为64位所产生的文件与标准RIFF/WAVE格式（一个明显的但重要的观察所得）不兼容。

所采用的方法就是，定义一个基于64位的新RIFF，称作BW64，等同于原版RIFF/WAVE格式，但以下的几种变化除外：

- 文件的前四个字节里用的ID是“BW64”而不是“RIFF”。
- 添加了一个强制命令<ds64>（数据大小64）块，该强制命令块需要作为继“BW64块”之后的第一个。

“ds64”块有两个强制64位整数值，代替RIFF/WAVE格式中的两个32位域对象：

- bw64尺寸（代替<RIFF>块的尺寸域）

- 数据尺寸（代替<data>块的尺寸域）

以下规则应用于RIFF/WAVE格式的所有的两个32位域：

如果位域中的32位值不是0xFFFFFFFF，则使用这个32位值。

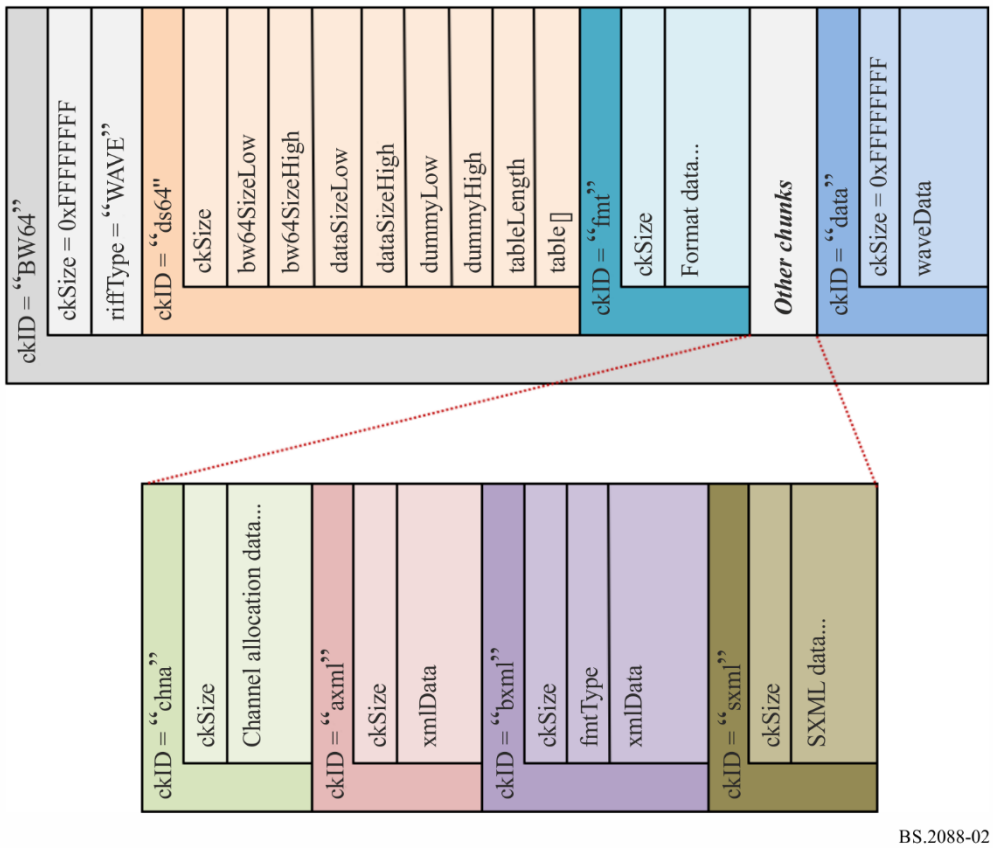
如果位域中的32位值是0xFFFFFFFF，则取而代之的是使用‘ds64’块中的64位值。

- 一个可选择的结构阵列（见附件1）（带有附加64位区块）是可能的。

BW64文件格式的完整结构如图2所示，其中将<BW64>和<data>块的ckSize值设置为0xFFFFFFFF，以允许它们使用<ds64>块中的64位大小值。

图2

BW64文件结构



注 – 块的数据大小可以是可变的。每个块的开始都相对于BW64文件的开始进行字对齐，以保持与ITU-R BS.1352建议书中指定的BWF的兼容性。如果块大小为奇数字节，则将在块后写入值为零的填充字节。但是，ckSize值不包括填充字节。

2.5 在RIFF/WAVE和BW64之间实现相容性

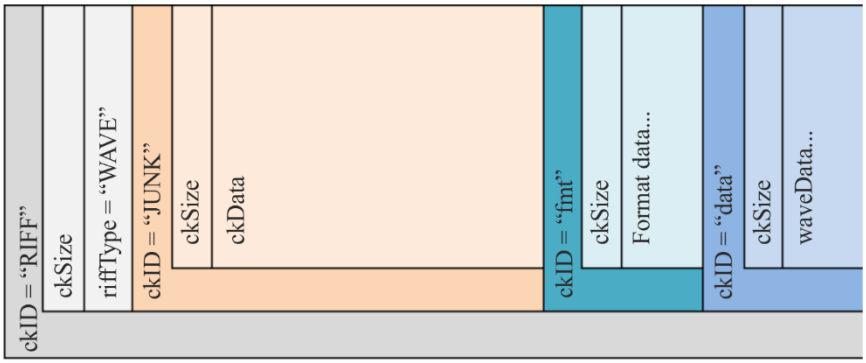
尽管有更高的抽样频率和多声道音频，一些生产的音频文件将不可避免地小于4 GB，且这些音频文件须置于短格式RIFF/WAVE格式中（如附件2所描述）。问题是，一个记录应用不能提前知道其正在编译的所记录下的音频录制是否会在录制的最后超过4 GB（即记录应用是否需要用到BW64）。

解决方法是，使记录应用能够在线以4 GB大小为限度从RIFF/WAVE切换到BW64，且记录仍在进行。

这是通过在RIFF/WAVE中预留额外空间来实现的，即插入一个同<ds64>块尺寸相同的<JUNK>块。这个预留的空间对于短格式WAVE没有意义，但如果有必要转换成BW64，它会变成<ds64>块。图3解释了置于<fmt>块之前的<JUNK>占位符块。

图3

带JUNK块的文件结构



在记录的开头，一个BW64-aware应用会创造出一个标准的RIFF/WAVE（带有一个“JUNK”块）作为第一个块。在记录的时候，BW64-aware应用会检查RIFF和数据大小。如果它们超过了4 GB，应用就会：

- 用<ds64>块代替ckID <JUNK>。（这会将先前的<JUNK>块转换成一个<ds64>块）。
- 在<ds64>块中插入RIFF尺寸、“数据”块尺寸。
- 在32位域将RIFF尺寸、“数据”块尺寸设置成0xFFFFFFFF
- 在文件的前四个字节中，用BW64代替ID“RIFF”。
- 继续记录。

2.6 RIFF/WAVE格式中现存的块和结构

RIFF/WAVE格式中现存的块和结构如下：

```
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
};
```

空数组括号表示有很多元素可使用（包括零）。

2.6.1 <RIFF>块的元素

<RIFF>块是文件的顶层。

字段	描述
ckID	这是4字符数组{‘R’, ‘I’, ‘F’, ‘F’}，用于块识别。
ckSize	文件大小的4字节值。
riffType	这是4字符数组{‘W’, ‘A’, ‘V’, ‘E’}，表示文件是个WAVE类型音频文件。

2.6.2 <fmt>块的元素

<fmt>块包含关于储存在<data>块中的音频样本格式的信息。

字段	描述
ckID	这是4字符数组{‘f’, ‘m’,‘t’, ‘ ’}，用于块识别。
ckSize	文件大小的4字节值。
formatTag	这是个2字节值，代表音频样本格式。0x0001的值的意义为，格式是PCM，0x0000为未知格式。
channelCount	这是2字节值，表示文件中的音轨数量。
sampleRate	这是个4字节值，代表Hz中音频的抽样率。
bytesPerSecond	每秒平均字节数，在此须发送波形数据。播放软件可以使用此值估算缓冲区大小。

blockAlignment	波形数据的块校准（字节）。播放软件每次需要处理多个数据的**blockAlignment**字节，因此，**blockAlignment**用于缓冲校准。
bitsPerSample	这是每声道每样本的位数。假定每个声道具有相同的抽样分辨率。如果不需要此字段，则应将其设置为零。
cbSize	extraData结构的字节大小。
extraData	额外数据用于储存WAVE_FORMAT_EXTENSIBLE信息，不在BW64中使用。

FormatChunk已经是一个专门的格式块，用于PCM音频数据。

当formatTag被设置为0XFFFE（WAVE_FORMAT_EXTENSIBLE）时，使用FormatChunk中的extraData数组。多声道音频应当使用ADM元数据，避免formatTag的使用。然而，它的实施是可能的，读取含有该formatTag的文件，并谨慎实施。

为确保FormatChunk不与<chna>、<axml>、<bxml>和<sxml>块的任何信息产生冲突，建议为PCM音频设置0x0001 formatTag，为所有其他非PCM音频设置0x0000（formatTag = 未知）。

2.6.3 <data>块的元素

<data>块用于储存音频样本。

字段	描述
ckID	这是4字符数组{‘d’, ‘a’, ‘t’, ‘a’}，用于块识别。
ckSize	块大小的4字节值。
waveData	音频样本储存位置。样本是按照小尾数字节序存储的。多音轨音频是在逐样本基础上交叉储存的。例如，对于16位双音轨音频来说：

字节	样本	音轨
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 BW64顶层块

3.1 定义

在32位大小的文件中，使用的是<BW64>顶层块而不是<RIFF>块。通过读取这个块，一个 <ds64>块须作为读取64位大小文件而存在。<BW64>块表示如下：

```
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 <BW64>块的元素

字段	描述
ckID	这是4字符数组{'b', 'w', '6', '4'}，用于块识别。
ckSize	必须将4字节值设置为0xFFFFFFFF，以指示未使用此大小值，并且须使用<ds64>块来确定大小。
BW64Type	这是4字符数组{'W', 'A', 'V', 'E'}，表示文件是个WAVE类型音频文件。

4 DS64和JUNK块

4.1 定义

<ds64>块为文件大小、<data>块和一组其他的可定义的块的64位尺寸值承载64位尺寸值。<ds64>块的结构表示如下，并附有**ChunkSize64**表格（带有可定义的块（除<data>）的尺寸）结构。空数组括号表示有很多元素可使用（包括零）。

```
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
};
```

<JUNK>块是<ds64>块的一个占位符（如果一个32比特大小的音频文件正在生成，也许随后需要在线转化为64比特大小的文件，需要用到<ds64>块）。<JUNK>的大小须与将要代替它的潜在的<ds64>块的大小匹配。块的结构如下：

```
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 <ds64>块的元素

字段	描述
ckID	这是4字符数组{'d', 's', '6', '4'}，用于块识别。
ckSize	<ds64>块的4字节尺寸。
bw64SizeLow	这是<BW64>块的低4字节尺寸。如果<bw64SizeLow>和<bw64SizeHigh>分别是0xLLLL和0xHHHH，64位数据尺寸则表示为0xHHHHLLLL。32位无符数字是小字节序格式。
bw64SizeHigh	这是<BW64>代码块的高4字节尺寸。32位无符数字是小字节序格式。
dataSizeLow	这是<数据>代码块的低4字节尺寸。如果<dataSizeLow>和<dataSizeHigh>分别是0xLLLL和0xHHHH，64位数据尺寸则表示为0xHHHHLLLL。32位无符数字是小字节序格式。
dataSizeHigh	这是<数据>代码块的高4字节大小。32位无符数字是小字节序格式。
dummyLow	这是一个4字节的伪值，读取时应忽略，写入时应设置为零。它的存在是为了确保与EBU Tech 3306 RF64规范的兼容性，EBU Tech 3306 RF64规范使用此值携带关于BW64格式中不存在的<fact>块的大小信息。
dummyHigh	这是一个4字节的伪值，读取时应忽略，写入时应设置为零。它的目的与<dummyLow>相同。
tableLength	这是数组“ChunkSize64表格”中的有效输入。
ChunkSize64 table	这是超过4 GB的块的块尺寸数组。

以下详述了**ChunkSize64**表格。**ChunkSize64**结构的数组用来储存<ds64>块的可选部分中的除了<data>以外的任何块的长度。目前，<axml>块（可能在超大的基于对象的音频文件中）可能是唯一的大小超过4 GB的除了<data>以外的块类型。

字段	描述
ckID	这个4字符数组是用于参考块（需要64位寻址）的<ckID>。例如，4字符数组{‘a’, ‘x’, ‘m’, ‘l’}用于<axml>块。
ckSizeLow	这是关于<ckID>的块的低4字节尺寸。32位无符数字是小字节序格式。
ckSizeHigh	这是关于<ckID>的块的高4字节尺寸。32位无符数字是小字节序格式。

4.3 <JUNK>块的元素

字段	描述
ckID	这是4字符数组{‘J’, ‘U’, ‘N’, ‘K’}，用于块识别。
ckSize	<JUNK>块的4字节大小。 <ds64>块的占位符须至少应为28。
ckData	无效数据，可忽略。

5 AXML块

5.1 定义

<axml>块也许包含一些符合XML 1.0格式或更新的格式的数据，是数据交换[1]的一个普遍格式。注意，<axml>块也许包含来自不止一个Schema的XML碎片。它可能在同一个文件中与另一个RIFF块以任何顺序出现。

<axml>块由一个标头（标头下有符合XML格式的数据）组成。块的总长度不固定。

在第11节可以看到一个案例，即如何使用BW64中的<axml>块承载广播元数据（包括之前的<bext>和 <ubxt>块中的参数）。

```
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
};
```

由于XML可能大于4 GB，它可能有必要用<ds64>块来允许一个64位大小字段用于<axml>块。以下内容是一些伪代码，说明如何用<ds64>块中的表格数组来达到这个目的。

```
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 <axml>块的元素

ckID 这是4字符数组{‘a’, ‘x’, ‘m’, ‘l’}，用于块识别。

ckSize 这是字节块中的数据部分的大小。（其不包括ckID和ckSize使用的8字节。）

xmlData 该字段包含XML中的文本信息。

XML数据结构是分级的，且数据是依照XML 1.0格式或更新的格式以文本字符串的形式储存的。

如果接收设备无法按照XML中规定的规范解释<axml>块的内容，则将忽略整个块。

6 BXML块

6.1 定义

<bxml>块可能包含压缩的XML数据，而不是<axml>块。

<bxml>块包含一个标头，后跟由**fmtType**中指定的压缩方法压缩的XML数据。块的总长度不是固定的。

```
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
};
```

由于压缩的XML数据可能大于4 GB，它可能有必要用<ds64>块来允许一个64位大小字段用于<bxml>块。以下内容是一些伪代码，说明如何用<ds64>块中的表格数组来达到这个目的。

```
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 <bxml>块的元素

ckID 这是4字符数组{‘b’, ‘x’, ‘m’, ‘l’}，用于块识别。

ckSize 这是字节块中的数据部分的大小。（不包括ckID和ckSize使用的8字节。）

fmtType 这是一个2字节的值，表示XML文本的压缩方法。0x0001的值表示压缩方法是gzip（IETF RFC 1952）。0x0000用于未压缩的XML文本。

xmlData 该字段包含由fmtType指示的压缩方法压缩的XML代码。

7 SXML块

7.1 定义

<sxml>块可以包含符合XML 1.0格式或与音频数据段相关联的压缩或未压缩XML的任何数据。它可能以任何顺序与同一文件中的其他RIFF块一起出现。

<sxml>块包含一个标头，后跟带有**fmtType**指定的压缩或未压缩XML数据的子块（**SubXMLChunk**）。每个**SubXMLChunk**对应于与相邻**SubXMLChunks**相邻的唯一数量的音频样本。<sxml>块由一个可选的对齐点表完成，该表允许基于时间戳的方式访问选定的**SubXMLChunk**。<sxml>块的总长度不固定。

<sxml>块可用于传输时变元数据，例如，ITU-R BS.2125建议书中指定的ADM的串行表示。

```
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
};
```

由于压缩或未压缩的XML数据可能占用4 GB以上，因此可能需要使用<ds64>块来允许<sxml>块的64位大小字段。以下内容是一些伪代码，说明如何用<ds64>块中的表格数组来达到这个目的。

```
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 <sxml>块的元素

字段	描述
ckID	这是4字符数组{‘s’, ‘x’, ‘m’, ‘l’}，用于块识别。
ckSize	这是字节块中的数据部分的大小。其不包括ckID和ckSize使用的8字节。）
fmtType	这是一个2字节的值，代表XML文本的压缩方法。值0x0001表示压缩方法为gzip（IETF RFC 1952）。0x0000用于未压缩的XML数据。
subXMLCkTbSizeLow	这是SubXMLChunk table[]的低4字节大小，包括nSubXMLChunks字段的4个字节。如果<subXMLCkTbSizeLow>和<subXMLChTbSizeHigh>分别为0xLLLL和0xHHHH，则64位数据大小表示为0xHHHHLLLL。32位无符数字是小字节序格式。
subXMLCkTbSizeHigh	这是SubXMLChunk table[]的高4字节大小，包括nSubXMLChunks字段的4个字节。
nSubXMLChunks	这是数组“SubXMLChunk table”中有效条目的数量。
SubXMLChunk table	这是包含XML数据的子块数组。
nAlignmentPoints	这是数组“AlignmentPoint table”中有效条目的数量。
AlignmentPoint table	这是对齐点的数组。

SubXMLChunk table规定如下。

字段	描述
subXMLChunkSize	这是SubXMLChunk的数据部分的大小，以字节为单位，不包括subXMLChunkSize和nSamplesSubDataChunk使用的8个字节。
nSamplesSubDataChunk	这是与SubXMLChunk关联的每个声道的音频样本数。
xmlData	该字段包含XML数据或通过fmtType指示的压缩方法压缩的XML数据。

AlignmentPoint table指定如下。

字段	描述
subXMLChunkByteOffsetLow	这是SubXMLChunk的起始字节偏移量，其对齐点从<sxml>块的开头以字节表示，不包括ckID和ckSize使用的8个字节。这是SubXMLChunk的起始字节偏移量的低4字节。如果<subXMLChunkByteOffsetLow>和<subXMLChunkByteOffsetHigh>分别为0xLLLL和0xHHHH，则64位数据大小将表示为0xHHHHLLLL。32位无符数字是小字节序格式。
subXMLChunkByteOffsetHigh	这是具有对齐点的SubXMLChunk的起始字节偏移量的高4字节。32位无符数字是小字节序格式。
nSamplesAlignPointLow	这是从<data>块开始以每个声道的音频样本表示的对齐点的时间戳。这是时间戳抽样计数的低4字节。如果<nSamplesAlignPointLow>和<nSamplesAlignPointHigh>分别为0xLLLL和0xHHHH，则64位计数表示为0xHHHHLLLL。32位无符数字是小字节序格式。
nSamplesAlignPointHigh	这是时间戳抽样计数的高4字节。32位无符数字是小字节序格式。

8 CHNA块

8.1 定义

如ITU-R BS.2076建议书中定义的，<chna>块是一个为ADM的使用而明确定义的块。<chna>块由一个标头（标头下有音轨的数量和使用的音轨UID的数量）组成。之后是一个ID结构的数组，每个ID结构包含与ADM元素ID相对应的ID。

块的大小取决于要被定义的音轨UID。ID结构的数量须等于或大于使用的音轨UID的数量。通过允许ID结构的数量必须超过UID的数量，可以便于为块更新ID和添加新的ID而无须改变块的大小。例如，也许起初不知道会生成多少个UID，所以，如果块中的ID结构数量设置为64（这是由制订人考虑的，使设置的数量一定多于他们完成任务所需要的量）；随后软件生成55个UID（初始UID的例数），填上了起初的55个ID结构，因此剩下的9个ID结构可设置为零值。

块中的ADM ID可以引用<axml>，<bxml>或<sxml>块中或外部通用定义文件中携带的ADM元数据。如果ID的最后四个十六进制数字的值为0x0FFF或更低，则它们在ITU-R BS.2094建议书 – 音频定义模型的通用定义（例如，“FrontLeft”和“FrontRight”的声道定义）中定义为通用定义。任何值为0x1 000及以上的ID都被定义为自定义，因此将包含在文件中的<axml>、<bxml>或<sxml>块中。

audioID结构包含一个<data>块中使用的轨道的索引（包含音频样本），第一个轨道的值从1开始。它包含轨道的UID，ADM元数据将包含该UID。一个音轨的音频元素在一个文件的过程中可能是不同的；在这种情况下，每个定义都有一个不同的UID。因此，每个轨道可能有多个UID。结构中的其他两个值是对ADM的audioTrackFormat和audioPackFormat元素的

ID的引用。如果音频本质的格式类型是线性PCM，则ADM允许省略audioTrackFormat和audioStreamFormat。然后，引用audioChannelFormat而不是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 <chna>块的元素

ckID 这是4字符数组{‘c’,‘h’,‘n’,‘a’}[1]，用于块识别。

ckSize 块的数据段的长度（不包括ckID和ckSize使用的8字节）。

numTracks 文件中使用的音轨的数量。即使一个轨道含有多于一组ID，仍然仅是一个音轨。

numUIDs 文件中使用的UID的数量。因为可以给单声道多个UID（覆盖不同的时间段），这个值可以比**numTracks**大。这个值须与**ID**中定义的ID的数量相匹配。

ID 结构包含一组音频参考ID用于轨道。该数组含有N个ID，其中N>=numUID。当numUID小于N时，未使用的音轨ID的内容被设置为零。当读取块时，N的值可以来自ckSize，因为ckSize = 4 + （N * 40），所以N = (ckSize – 4) / 40。

trackIndex 文件中的轨道的检索，从1开始。这直接与<data>块中交叉的块的顺序相对应。

UID 音轨的音轨UID值。字符数组有格式ATU xxxxxxxx，其中x是个十六进制数字。

trackRef 轨道的TheaudioTrackFormatID参考。字符数组有格式AT_xxxxxxxx_xx，其中，x是个十六进制数字。格式AC_xxxxxxxx_00（后缀“00”将填充字符串以匹配audioTrackFormatID字符串的格式，并且不带任何含义），其中，x是十六进制数字，当省略线性PCM的音频本质的audioTrackFormat和

[1] **备注：**定义DWORD ckID = “chna”不是唯一的。不同的结构会产生不同的字符顺序。因此，我们定义char ckID[4] = {‘c‘,‘h‘,‘n‘,‘a‘}。

audioStreamFormat，并且在ADM XML代码中直接引用audioChannelFormat时，也使用。

packRef 轨道的audioPackFormatID参考。字符数组的格式为AP_xxxxxxxx，其中，x是十六进制数字。当不需要audioPackFormatID时（当audioStreamFormat指的是audioPackFormat而不是audioChannelFormat时），此字段须填充空值。

pad 一个单字节，确保音频ID结构有字节的一个偶数。

当一个**ID**未被使用时，**trackIndex**的值须为零，且其他字段须有空字符串（空字符串的长度与平常使用的ID字符串相同）。所以，packRef的空字符串将由11个空字符（ASCII值零）组成，且trackRef将由14个空字符组成。

8.3 资料性案例

为了帮助说明<chna>块的操作，此处描述了一些简单的示例。每个示例中的伪代码都为ID使用类似字符串的符号（例如“AT_00010001_01”），在实践中，须使用字符数组来避免在结尾处包含空终结字符（因此实际上可以这样进行：
{‘A’,’T’,’_’,’0’,’0’,’0’,’1’,’0’,’0’,’0’,’1’,’_’,’0’,’1’}）。

8.3.1 简单立体声文件

大多数的现有音频文件仍是双声道立体声文件，第一个轨道包含左声道，第二个轨道包含右声道。ADM有一个左声道定义，带有的ID是AT_00010001_01，右声道定义带有的ID是AT_00010002_01。立体包装定义带有的ID是AP_00010002。

伪代码表示如下：

```
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`; };
```

ID结构的数量是2，所以在该例子中没有未使用的ID结构。

当ADM省略了audioTrackFomat和audioStreamFormat以及对audioChannelFormat的引用时，将使用以下代码。

```
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 简单的基于对象的例子

音频对象可能仅覆盖音频文件中的一个时间分段。为节省空间，非重叠的对象可能分享同一个轨道。这就是同一个轨道中会出现多个UID的情况。这个例子使用的ID结构（这个例子中有32个）比UID数量多，以显示如何将未使用的ID结构设置成零。

```
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`; };
```

第一个轨道包含3个UID，因此在文件内的不同时间位置将包含3个不同的对象（轨道ID为AT_00031001_01，AT_00031003_01和AT_00031004_01）。第二轨道包含一个UID，因此包含一个对象。该对象与轨道1中的第一个对象具有相同的包ID（AP_00031001）。这表明第一个对象包含在轨道1和轨道2中携带的两个信道。在<axml>、<bxml>或<sxml>块中承载的ADM元数据将用于阐明信道和轨道的分配。

8.3.3 多内容案例

BW64文件可以在单个文件中包含多个内容，例如在前6个轨道上有主混音5.1，在接下来的2个轨道上有一个外语立体声混音。ITU-R BS.1738建议书含有几种配置，而且案例会呈现建议书中的生产情况5是如何在<chna>块中被处理的。这种情况包含8个轨道，前6个轨道包含一个5.1完整混音，后2个轨道包含一个立体声国际混音。<chna>结果如下：

```
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`; };
```

<axml>、<bxml>或<sxml>块中的ADM元数据将包含信息，即两个混音是如何分开的。

9 XML块的规则

有三种不同的块可以承载XML元数据：<axml>、<bxml>和<sxml>。虽然这些块的主要目的是携带ADM XML元数据（如ITU-R BS.2076建议书中所规定）或S-ADM元数据（如ITU-R BS.2125建议书中所规定），但它们也可以携带其他XML元数据，如第11节中所述的广播元数据。由于多个块能够承载XML元数据，因此一个块中的元数据可能与另一个块中的元数据冲突。因此，应适用下列规则：

1 任何特定XML块的实例不得超过一个。

2 如果正在携带ADM元数据：

a) 它须仅出现在<axml>或<bxml>块中，而不同时出现在两者中；

b) 必须存在一个交叉引用ADM元数据的<chna>块。

3 如果正在携带S-ADM元数据，则它须出现在<sxml>块中。

4 如果同时携带了ADM元数据和S-ADM元数据，则它们须彼此独立（即，彼此不交叉引用）。

5 如果携带其他元数据（即非ADM或非S-ADM）：

a) 如果需要，它可以与ADM和S-ADM元数据一起放在同一块中；

b) 此“其他元数据”的内容不应代表现有ADM或S-ADM元数据中已经描述的任何内容；

c) 如果“其他元数据”交叉引用ADM或S-ADM元数据，则文件中须存在引用的ADM或S-ADM元数据。

10 与ITU-R BS.1352建议书的相容性

由于BWF格式（ITU-R BS.1352建议书）是带有额外块的短格式RIFF/WAVE文件格式（如附件2中所描述的），尤其是<bext>和<ubxt>块，需要理解BWF和BW64之间的相容性。

BWF块 ITU-R BS.1352-4 建议书	BW64块 ITU-R BS.2088-2 建议书	如何操作
<fmt>	<fmt>	常规使用
<data>	<data>	常规使用
<fact>	<fact>	常规使用
–	<ds64>	见2.4和4节
–	<JUNK>	见2.4和4节
–	<chna>	有关声道分配，请参见第8节。 注意：ITU-R BS.2088-0建议书不支持引用audioChannelFormat
–	<axml>, <bxml> 或 <sxml>	请参阅第5至7节。用于<bext>或<ubxt>块中存在的广播元数据
<bext> 或 <ubxt>	(<bext> 或<ubxt>)	如果读取一个<bext>或<ubxt>块，由应转变到对应的<axml>、<bxml>和<sxml>块数据以承载ADM和任何其他与广播相关的XML元数据。更多详细信息见第11节

11 生成XML广播元数据

ITU-R BS.1352建议书承载<bext>和<ubxt>块中的广播元数据。这些块有固定的字段长度且这些块限于指定的字段，因此阻止任何其他的与广播相关的元数据被携带。BW64中的<axml>、<bxml>和<sxml>块可以携带任何XML元数据，因此可以用于携带广播元数据，包括<bext>和<ubxt>块中的参数。

应当用以下的XML结构来携带<axml>、<bxml>或<sxml>块中的<bext>/<ubxt>参数，XML结构中，有前置代码“BEXT”（携带参数时使用'UBXT <ubxt> '）的注释表示<bext>块参数。

```
<?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: Originator/UBXT: uOriginator-->
        </name>
      </contactDetails>
      <organisationDetails>
        <organisationName>
          <!--BEXT: OriginatorReference/UBXT: uOriginatorReference-->
        </organisationName>
      </organisationDetails>
    </creator>

    <description typeDefinition="bextDescription" or "ubxtDescription">
      <dc:description>
        <!--BEXT: Description/UBXT: uDescription-->
      </dc:description>
    </description>
    <date>
      <!--BEXT: OriginationDate/UBXT: OriginationDate and
          BEXT: OriginationTime/UBXT: OriginationTime below-->
      <created startDate="2000-10-10" startTime="12:00:00"/>
    </date>

    <format>
      <audioFormatExtended>
        <!--BEXT: TimeReference/UBXT: TimeReference 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: CodingHistory/UBXT: uCodingHistory-->
      </technicalAttributeString>
    </format>

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

上文的XML是基于EBU核心[2]和AES核心[3]元数据Schema数据库，Schema数据库与ITU-R BS.2076建议书兼容。

如果要把一个ITU-R BS.1352 BWF文件转换成一个BW64文件而对ITU-R BS.1352 BWF文件进行读取，应将<bext>或<ubxt>块转换成此处描述的XML，以包括在<axml>、<bxml>或<sxml>块内。

12 BW64格式文件的文件扩展名

遵照BW64格式的文件的扩展名定义为“.wav”。这使遗产软件能够读取它理解的文件中的块（首先是<fmt>和<data>），所以至少可以进入音频样本。

虽然不建议在生成BW64文件时使用任何其他文件扩展名，但是可以预料到“.bw64”扩展名可能会被不恰当地使用。因此，读取BW64文件的软件须允许该替代文件扩展名。

13 参考资料

[1] Extensible Markup Language (XML) 1.0 W3C Recommendation 26-November-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

附件2
（资料性）

RIFF WAVE（.WAV）文件格式

本附件中的信息源于微软RIFF文件格式的规范文件。它只用于资料性信息。因缺乏可靠的外部资源作参考，所以信息被收录到附件中。

1 波形音频文件格式（WAVE）

WAVE格式定义如下。软件须忽略遇到的任何未知块，就像所有RIFF表单一样。但是，<fmt ck>总是在<wave data>之前发生，并且这两个块在WAVE文件中都是必需的。

```
<WAVE-form> ->
    RIFF('WAVE'
        <fmt-ck>    // Format chunk
        [<fact-ck>]   // Fact chunk
        [<other-ck>]  // Other optional chunks
        <wave-data>)  // Sound data
```

WAVE块在下列各节中描述：

1.1 WAVE格式块

WAVE 格式块<fmt-ck>规定<wave-data>的格式。<fmt-ck>定义如下：

```
<fmt-ck> ->fmt(<common-fields>
               <format-specific-fields>)
<common-fields> ->
    Struct {
        WORD  wFormatTag;            // Format category
        WORD  nChannels;             // Number of channels
        DWORD nSamplesPerSec;        // Sampling rate
        DWORD nAvgBytesPerSec;       // For buffer estimation
        WORD  nBlockAlign;           // Data block size
    }
```

块的<common-fields>部分中的字段描述如下：

字段	描述
wFormatTag	指示文件的WAVE格式类别的一个号码。<fmt-ck>的<format-specific-fields>部分的内容和波形数据的解释取决于该值。
nchannels	在波形数据中代表信道数，如1代表单声道或2代表立体声。
nSamplesPerSec	抽样率（在每秒抽样中），在此须对每个通路复制。

nAvgBytesPerSec	每秒平均字节数，在此须发送波形数据。播放软件可以使用此值估算缓冲区大小。
nBlockAlign	波形数据的块校准（字节）。播放软件每次需要处理多个数据的<nBlockAlign>字节，因此<nBlockAlign>用于缓冲校准。

<format-specific-fields>由0或多个参数的字节组成。哪个参数出现取决于WAVE格式类别，详细描述参见以下节。播放软件应允许（并忽略）在此字段结束时出现的任何未知<format-specific-fields>参数。

1.2 WAVE格式种类

WAVE文件的格式类别由<fmt>块的<wFormatTag>字段的内容规定。<wave-data>中的数据表示以及<fmt>块<format-specific-fields>的内容取决于格式类别。

当前定义的开放非专有WAVE格式种类如下：

wFormatTag	值	格式类别
WAVE_FORMAT_UNKNOWN	0x0000	未知
WAVE_FORMAT_PCM	0x0001	PCM格式
WAVE_FORMAT_IEEE_FLOAT	0x0003	IEEE浮点
WAVE_FORMAT_EXTENSIBLE	0xFFFE	波形格式扩展 – 由子格式决定

注 – 目前BW64只使用WAVE_FORMAT_PCM和WAVE_FORMAT_UNKNOWN格式。PCM WAVE 格式的详细描述在下述的第2节中。其他WAVE格式的通用信息在第3节中描述。其他WAVE格式可能在今后规定。

以前，WAVE_FORMAT_EXTENSIBLE会被用于多声道文件，但在未来，应该避免这种使用。

1.3 实际块

<fact-ck>存储关于非PCM WAVE文件内容的重要信息。因此，该块没有用于BW64格式的这个版本。此块定义如下：

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

<dwSampleLength>代表样本中数据的长度。来自波形格式标头的<nSamplesPerSec>字段与<dwSampleLength>字段结合使用，以按秒数决定数据长度。

实际块被要求用于所有的新的非PCM WAVE格式。此块不是标准WAVE_FORMAT_PCM文件所要求的。

实际块将被扩展以包括所有未来WAVE格式所要求的其他信息。增加的字段将出现在<dwSampleLength>字段之后。具体应用使用块大小字段以确定所出现的字段。

1.4 其他任选块

规定其他块使用WAVE格式。这些块的详细说明在WAVE格式规范中给出并随时发布更新。

注 – WAVE格式可支持WAVE文件中包括的承载特定信息的其他任选块。这些被认为是专用块并被不能对其进行识别的应用所忽略。

2 PCM格式

如果<fmt-ck>的<wFormatTag>字段设置为WAVE_FORMAT_PCM，波形数据由PCM格式表示的抽样组成。对于PCM波形数据，<format-specific-fields>定义如下：

```
<PCM-format-specific> ->
    struct {
        WORD  nBitsPerSample;     // Sample size
    }
```

<nBitsPerSample>字段规定：用于表示每通路的每个抽样所用的数据比特数。如果有多个信道，每个信道的抽样大小是相同的。

<nBlockAlign>字段须等于下列公式，四舍五入到下一个整数。

$$nChannels \times BytesPerSample$$

BytesPerSample的值的计算须采用将nBitsPerSample四舍五入到下一个整数字节。当音频抽样字小于字节的整数时，音频抽样的最高有效位设置于数据字的最高有效位上，最低有效位相邻的未使用的数据须被设置为0。

对于PCM数据，<fmt>块的<nAvgBytesPerSec>字段应与下述公式相同。

$$nSamplesPerSec \times nBblockAlign$$

注1 – 最初的WAVE规范允许2个声道的最低有效位的单个字节，例如2个声道的20比特抽样打包到5个共享字节。本建议书规定每个音频抽样的整数字节，以减少具体实施中的模糊性并实现最大限度的互换兼容性。

2.1 PCM WAVE文件的数据打包

在单声道的WAVE文件中，抽样是连续存储的。对于立体声WAVE文件，声道0表示左声道，声道1表示右声道。多声道WAVE文件中，抽样是隔行交织的。

下表表示8比特单声道和立体声WAVE文件的数据打包：

8比特单声道PCM的数据打包

样本1	样本2	样本3	样本4
声道0	声道0	声道0	声道0

8比特立体声PCM的数据打包

样本1		样本2	
声道0 （左）	声道1 （右）	声道0 （左）	声道1 （右）

下表表示16比特单声道和立体声WAVE文件的数据打包：

16比特单声道PCM的数据打包

样本1		样本2	
声道0 低阶字节	声道0 高阶字节	声道0 低阶字节	声道0 高阶字节

16比特立体声PCM的数据打包

样本1			
声道0（左）	声道0（左）	声道1（右）	声道1（右）
低阶字节	高阶字节	低阶字节	高阶字节

2.2 样本的数据格式

每个样本包含在整数i中。i的大小是包含特定样本大小所需要的最小的字节数。最低有效字节最先存储。表示样本幅度的比特存储在i的最高有效位。其余比特设置为0。

例如，如果样本大小（<nBitsPerSample>中记录）是12比特，则在2 字节整数中存储每个样本。第1个（最低有效）字节的最低有效4比特置为0。各种大小的PCM波形样本的数据格式和最大和最小值如下：

样本大小	数据格式	最大值	最小值
1到8比特	无符号整数	255（0xFF）	0
9或更多bits	有符号整数i	i的最大正值	i的最大负值

例如，8比特和16比特PCM波形数据的最大、最小和中间值如下：

格式	最大值	最小值	中间值
8比特PCM	255 (0xFF)	0	128 (0x80)
16比特PCM	32767 (0x7FFF)	−32768 (−0x8000)	0

2.3 PCM WAVE文件的范例

PCM WAVE文件的**范例**，11.025 kHz抽样率、单声道、每样本8比特：

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

PCM WAVE文件的**范例**，22.05 kHz抽样率、立体声、每样本8比特：

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

2.4 WAVE数据的存储

<**wave-data**>包括波形数据定义如下：

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

2.5 实际块

<fact-ck>实际块存储关于WAVE文件内容的重要信息。此块定义如下：

```
<fact-ck> ->    fact(<dwFileSize:DWORD>)      // Number of samples
```

此块不是PCM文件所要求的。

实际块将被扩展以包括所有未来WAVE格式所要求的其他信息。增加的字段将出现在<dwFileSize>字段之后。具体应用使用块大小字段以确定所出现的字段。

2.6 其他任选块

为了使用，大量的其他块以WAVE格式定义。这些块的详情在WAVW格式的规范中和任何之后的更新问题中给出。

注 1 – WAVE格式可支持WAVE文件中包括的承载特定信息的其他任选块。这些被认为是专用块并被不能对其进行识别的应用所忽略。

3 WAVE格式扩展

扩展的波形格式结构加入<fmt-ck>，用于定义所有非PCM格式波数据，并描述如下。一般扩展的波形格式结构用于所有非PCM格式。

```
typedef struct waveformat_extended_tag {
    WORD  wFormatTag;      // format type
    WORD  nChannels;       // number of channels (i.e. mono, stereo...)
    DWORD nSamplesPerSec;  // sample rate
    DWORD nAvgBytesPerSec; // for buffer estimation
    WORD  nBlockAlign;     // block size of data
    WORD  wBitsPerSample;  // number of bits per sample of mono data
    WORD  cbSize;          // the count in bytes of the extra size
} WAVEFORMATEX;
```

字段	描述
wFormatTag	定义WAVE文件类型。
nChannels	波形中的声道数，1代表单声道，2代表立体声。
nSamplesPerSec	波形文件抽样率的频率。须为48000或 44100等。它也在实际块中用于抽样大小入口以确定数据的持续时间。
nAvgBytesPerSec	平均数据率。播放软件可以采用<nAvgBytesPerSec>值估计缓冲大小。

nBlockAlign	<data-ck>中数据的块校准（字节）。播放软件需要同时处理多个 数据的<nBlockAlign>字节，因此<nBlockAlign>值可用于缓冲校准。
wBitsPerSample	这是每声道每样本的比特数。假定每声道具有相同的抽样分辨率。如果不需要此字段，则应设置为0。
cbSize	WAVE格式标头中额外信息的字节大小，不包括WAVEFORMATEX结构的大小。

注 – <cbSize>字段之后的字段，包括<wFormatTag>中定义WAVE格式所需要的特定信息。

附件3
（规范性）

原始数据类型定义

以下是关于原始数据类型的原子标签。等效的C数据类型也在此列举出来。

标签	含义	C类型
<CHAR>	8位有符号整数	有符号字符型
<BYTE>	8位无符号整数	无符号字符型
<INT>	小字节序格式的16位有符号整数	有符号整数
<WORD>	小字节序格式的16位无符号数字	无符号整数
<LONG>	小字节序格式的32位有符号整数	有符号长整型
<DWORD>	小字节序格式的32位无符号数字	无符号长整型
<FLOAT>	32位IEEE浮点数	浮点类型Float
<DOUBLE>	64位IEEE浮点数	浮点类型Double
<STR>	字符串（一个字符序列）	
<ZSTR>	NULL结尾的字符串	
<BSTR>	带有字节（8位）大小前缀的字符串	
<WSTR>	带有字（16位）大小前缀的字符串	
<BZSTR>	带有字节大小前缀的NULL结尾的字符串	

附件4
（资料性）

附件1中规范的变更

1 ITU-R BS.2088-0建议书到BS.2088-1建议书的修改

本建议书的修订版1对附件1中的规范进行了以下更改：

– 在第6节中添加了BXML块。

– 在第7节中添加了SXML块。

– 在第8节中增加了一个新功能，用于省略audioTrackFormat和audioStreamFormat。

2 ITU-R BS.2088-1建议书到BS.2088-2建议书的修改

本建议书修订2澄清了附件1中其他wave文件格式所用块的处理：

– 澄清第2.1、2.2和10段中块的处理。

– 第11节还包括从块增加的XML生成方法<ubxt>。