ITUPublications
Recommandations

Union internationale des télécommunications
Secteur des radiocommunications

Recommandation UIT-R BS.2088-2

(11/2025)

Série BS: Service de radiodiffusion sonore

Format de fichier détaillé pour l'échange international de programmes audio avec métadonnées

Avant-propos

Le rôle du Secteur des radiocommunications est d'assurer l'utilisation rationnelle, équitable, efficace et économique du spectre radioélectrique par tous les services de radiocommunication, y compris les services par satellite, et de procéder à des études pour toutes les gammes de fréquences, à partir desquelles les Recommandations seront élaborées et adoptées.

Les fonctions réglementaires et politiques du Secteur des radiocommunications sont remplies par les Conférences mondiales et régionales des radiocommunications et par les Assemblées des radiocommunications assistées par les commissions d'études.

Politique en matière de droits de propriété intellectuelle (IPR)

L'UIT attire l'attention sur la possibilité que l'application ou la mise en œuvre de la présente Recommandation puisse donner lieu à l'utilisation d'un droit de propriété intellectuelle. L'UIT ne prend pas position en ce qui concerne l'existence, la validité ou l'applicabilité des droits de propriété intellectuelle, qu'ils soient revendiqués par un membre de l'UIT ou par une tierce partie étrangère à la procédure d'élaboration des Recommandations.

À la date d'approbation de la présente Recommandation, l'UIT avait été avisée de l'existence d'une propriété intellectuelle, protégée par des brevets, dont l'acquisition pourrait être requise pour mettre en œuvre la présente Recommandation. Toutefois, comme il ne s'agit peut-être pas de renseignements les plus récents, il est vivement recommandé aux développeurs de consulter les renseignements pertinents de l'UIT-R concernant les brevets à l'adresse https://www.itu.int/en/ITU-R/study-groups/Pages/itu-r-patent-information.aspx.

Séries des Recommandations UIT-R

(Également disponible en ligne: https://www.itu.int/publ/R-REC/en)

Série	Titre
BO	Diffusion par satellite
BR	Enregistrement pour la production, l'archivage et la diffusion; films pour la télévision
BS	**Service de radiodiffusion sonore**
BT	Service de radiodiffusion télévisuelle
F	Service fixe
M	Services mobile, de radiorepérage et d'amateur y compris les services par satellite associés
P	Propagation des ondes radioélectriques
RA	Radio astronomie
RS	Systèmes de télédétection
S	Service fixe par satellite
SA	Applications spatiales et météorologie
SF	Partage des fréquences et coordination entre les systèmes du service fixe par satellite et du service fixe
SM	Gestion du spectre
SNG	Reportage d'actualités par satellite
TF	Émissions de fréquences étalon et de signaux horaires
V	Vocabulaire et sujets associés

Note: *Cette Recommandation UIT-R a été approuvée en anglais aux termes de la procédure détaillée dans la Résolution UIT-R 1.*

Publication électronique
Genève, 2026

RECOMMANDATION UIT-R BS.2088-2*

Format de fichier détaillé pour l'échange international de programmes audio avec métadonnées

(2015-2019-2025)

Domaine d'application

Cette Recommandation spécifie le format de fichier audio *Broadcast Wave 64 Bit* (BW64), comprenant les nouveaux fragments <ds64>, <axml>, <bxml>, <sxml> et <chna>, qui permettent d'acheminer des fichiers multicanaux de grande taille et les métadonnées, notamment celles relatives au modèle de définition audio (ADM) spécifié dans la Recommandation UIT-R BS.2076.

Mots clés

Fichier, format de fichier, métadonnées, WAVE, BW64, échange, programme audio, WAV, BWF, RIFF, RF64, fichier WAVE, audio en immersion, modèle de définition audio (ADM), modèle ADM série (S-ADM)

L'Assemblée des radiocommunications de l'UIT,

considérant

a) que les supports d'enregistrement fondés sur l'informatique, y compris les disques et les bandes de données, ont pénétré dans tous les domaines de la production audio pour la radiodiffusion, à savoir l'édition non linéaire, la restitution à l'antenne et l'archivage;

b) que l'informatique offre de notables avantages en termes de souplesse d'exploitation, de flux de production et d'automatisation des stations et qu'elle est donc intéressante pour la modernisation des studios existants et pour la conception de nouvelles installations de studio;

c) que l'adoption d'un unique format de fichier pour l'échange de signaux simplifierait beaucoup l'interfonctionnement d'équipements individuels et de studios distants tout en facilitant l'intégration souhaitable des opérations d'édition, de restitution à l'antenne et d'archivage;

d) qu'un ensemble minimal d'informations relatives à la diffusion doit être inclus dans le fichier afin de décrire les métadonnées liées au signal audio;

e) que, pour assurer la compatibilité entre applications de complexités diverses, il faut adopter un ensemble minimal de fonctions communes à toutes les applications capables de traiter le format de fichier recommandé;

f) que la Recommandation UIT R BS.646 définit le format audionumérique utilisé en production audio pour la diffusion radiophonique et télévisuelle;

g) que la compatibilité avec les formats de fichier actuellement disponibles sur le marché permettrait de réduire les efforts que les entreprises doivent déployer pour mettre en œuvre ce format dans les équipements;

h) que, pour le champ «coding history» et pour d'autres métadonnées associées, un format normalisé simplifierait l'utilisation de l'information après l'échange des programmes;

---

* La Commission d'études 6 des radiocommunications a apporté des modifications de forme à la présente Recommandation en 2026 conformément à la Résolution UIT-R 1.

i) que la qualité d'un signal audio dépend du traitement appliqué à ce signal, notamment en cas de codage/décodage non linéaire lors de réductions du débit binaire;

j) que dans les systèmes audio évolués, il est nécessaire que les métadonnées associées au signal audio soient acheminées dans le fichier;

k) que les systèmes audio évolués utilisent diverses configurations multicanaux comprenant des éléments audio basés sur un canal, un objet ou une scène, comme spécifié dans la Recommandation UIT-R BS.2051;

l) que les métadonnées relatives au signal audio utilisées dans les systèmes sonores évolués sont définies dans la Recommandation UIT-R BS.2076, les définitions communes faisant l'objet de la Recommandation UIT-R BS.2094 et la représentation série des métadonnées (S-ADM) faisant l'objet de la Recommandation UIT-R BS.2125;

m) que le format décrit dans la Recommandation UIT-R BS.1352 a des limites en ce qui concerne la taille des fichiers et sa capacité d'acheminer des métadonnées supplémentaires;

n) que les fichiers audio multicanaux pourraient potentiellement avoir une taille supérieure à 4 gigaoctets,

recommande

1 que, pour l'échange de programmes audio, les paramètres, la fréquence d'échantillonnage (partie 1), la profondeur binaire (parties 4 et 5) et la préaccentuation (partie 6) du signal audio soient déterminés conformément aux parties applicables de la Recommandation UIT-R BS.646;

2 que le format de fichier spécifié dans l'Annexe 1 soit utilisé pour l'échange de programmes dans les cas d'utilisation suivants:

- dans les environnements fondés sur des fichiers WAVE, où les applications de diffusion fondées sur des fichiers WAVE souhaitent effectuer une mise à jour pour gérer du contenu en immersion, tout en maintenant la compatibilité avec les versions ultérieures;
- dans les flux de travail reposant sur des fichiers, où il y aura une bibliothèque mixte comprenant à la fois d'anciens contenus fondés sur des fichiers WAVE et du contenu en immersion;
- dans les flux de travail reposant sur des fichiers, où un enrobage unique des données et des métadonnées dans un seul paquet est préféré.

NOTE – L'Annexe 4 indique les modifications apportées aux spécifications de l'Annexe 1 par rapport à la version précédente de la présente Recommandation, à titre d'information uniquement.

Annexe 1
(normative)

Spécification du format de fichier BW64

1 Introduction

Le format BW64 repose sur le format de fichier audio WAVE (décrits dans l'Annexe 2), qui est un type de fichier spécifié dans le format RIFF (Resource Interchange File Format). Les fichiers WAVE contiennent spécifiquement des données audio. Le module de construction fondamental du format de

fichier RIFF, appelé fragment, contient un groupe d'éléments informationnels étroitement associés. Il est composé d'un identificateur de fragment, d'une valeur d'entier représentant la longueur en octets et les informations acheminées. Un fichier RIFF se compose d'un ensemble de fragments. Ce format BW64 utilise les principaux éléments du format décrits dans la norme EBU Tech 3306.

Le format de fichier BWF, décrit dans la Recommandation UIT-R BS.1352, est associé à un certain nombre de limites, en particulier:

- Taille de fichier de 4 gigaoctets au maximum.
- Absence de prise en charge des éléments audio multicanaux évolués en raison de métadonnées relatives au signal audio limitées.
- Prise en charge inadaptée des métadonnées techniques.

Le format BW64 décrit dans la présente Recommandation vise à supprimer ces limites et à maintenir autant que faire se peut la compatibilité avec le format décrit dans la Recommandation UIT-R BS.1352, avec lequel il partage de nombreux éléments principaux.

La demande concernant le transfert de métadonnées est en hausse, en particulier le transfert des métadonnées relatives au modèle de définition audio (ADM) conformément à la Recommandation UIT-R BS.2076. La présente Recommandation comprend une définition des fragments <axml>, <bxml> et <sxml> utilisés pour l'enregistrement et le transfert de métadonnées sous la forme d'un élément XML respectivement au format UTF-8, au format compressé et au format série.

Le fragment <chna> décrit dans la présente Recommandation vise avant tout à établir le lien depuis chaque piste d'un fichier BW64 vers les identifiants dans les métadonnées ADM définis dans la Recommandation UIT-R BS.2076.

Outre sa vocation première d'établir un lien entre chaque piste du fichier et ses métadonnées ADM associées, le fragment <chna> permet également d'accéder plus rapidement aux identifiants ADM sans avoir à accéder aux métadonnées XML (si les identifiants ont l'une des valeurs prédéfinies pour les configurations ADM types). Étant donné que le fragment <chna> peut avoir une taille fixe et est placé avant les fragments <data>, <axml>, <bxml> et <sxml>, il est plus facile d'avoir accès à son contenu, de le générer ou de le modifier à la volée.

Dans le présent document, les types de données sont utilisés conformément à l'Annexe 3.

2 Description du format BW64

2.1 Contenu d'un fichier au format BW64

Un fichier au format BW64 doit commencer par l'en-tête «WAVE» et comporter au moins les fragments suivants:

```
<WAVE-form> ->
    BW64('WAVE'
        <ds64-ck>       // Fragment ds64 pour l'adressage 64 bits
        <fmt-ck>        // Format du signal audio: MIC/non-MIC
        <chna-ck>       // Fragment chna pour la consultation ADM
        <axml-ck>       // Fragment axml pour l'acheminement des données XML
        <bxml-ck>       // Fragment bxml pour l'acheminement des métadonnées XML
                        // compressées
        <sxml-ck>       // Fragment sxml pour l'acheminement des métadonnées XML
                        // associées à un sous-fragment ou des données audio
```

```
        <wave-data>)     // Données audio
```

NOTE 1 – Des fragments additionnels peuvent être présents dans le fichier et certains d'entre eux peuvent sortir du cadre de la présente Recommandation. Les applications ne sont pas tenues d'interpréter ou d'utiliser ces fragments, de sorte que l'intégrité des données contenues dans des fragments inconnus ne peut être garantie. Toutefois, les applications compatibles transmettent de manière transparente ces fragments inconnus. D'autres fragments, y compris les fragments <bext> et <ubxt> du format BWF, peuvent figurer dans les fichiers BW64 définis dans la présente Recommandation. Cependant, les applications conformes à la Recommandation UIT-R BS.2088 (BW64) peuvent lire des métadonnées XML pouvant aussi contenir des métadonnées converties à partir d'anciens fragments, y compris les fragments <bext> et <ubxt> du format BWF.

NOTE 2 – Il pourrait être admis de placer le fragment <axml>, <bxml> ou <sxml> après le fragment <data>, étant donné que les métadonnées XML auront probablement une longueur inconnue et il pourrait être plus pratique de connaître la position de départ des échantillons audio dans le fichier.

2.2 Fragments déjà définis dans le cadre de la norme RIFF/WAVE

La norme RIFF/WAVE utilise un certain nombre de fragments qui sont déjà définis. Il s'agit des suivants:

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

Ces fragments sont décrits aux § 2.6.1 à 2.6.3.

Le format RIFF/WAVE est un sous-ensemble du format décrit dans la Recommandation UIT-R BS.1352. La Recommandation UIT-R BS.1352 contient les fragments additionnels suivants:

- <bext>
- <ubxt>

Ces fragments peuvent figurer dans le format BW64. Toutefois, le contenu de ces fragments devrait être converti en des métadonnées XML contenues dans un fragment associé au format XML (voir les paragraphes 10 et 11). Les applications devraient utiliser des métadonnées XML. On trouvera dans le Rapport UIT-R BS.2388 des informations concernant l'utilisation du format BW64.

2.3 Nouveaux fragments et nouvelles structures dans le format BW64

Les nouveaux fragments introduits pour le format BW64 sont les suivants:

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

Ces fragments sont décrits aux paragraphes 3 à 8.

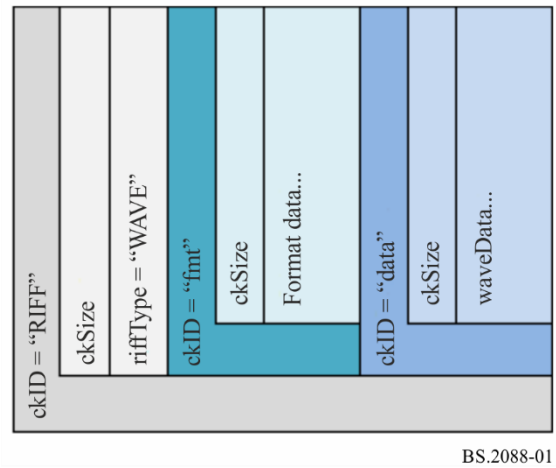
2.4 Utilisation du fragment <ds64> pour permettre l'utilisation de fichiers de plus de 4 gigaoctets

Cette limite de 4 gigaoctets est due au fait que les formats RIFF/WAVE et BWF n'autorisent qu'un adressage 32 bits. Avec 32 bits, 4294967296 octets (= 4 gigaoctets) au maximum peuvent être adressés. Pour résoudre ce problème, un adressage 64 bits est nécessaire. La Figure 1 montre la

structure d'un fichier RIFF/WAVE conventionnel de base, où les champs ckSize sont des nombres de 32 bits représentant la taille du fragment auquel ils sont associés.

FIGURE 1

Structure d'un fichier RIFF/WAVE de base



Si l'on se contente de porter la taille de chaque champ d'un fichier BWF, à 64 bits, le fichier ainsi obtenu n'est pas compatible avec le format RIFF/WAVE type – remarque évidente mais importante.

La solution adoptée consiste à définir un nouveau format RIFF fondé sur des champs de 64 bits, appelé BW64, identique au format RIFF/WAVE original, avec les modifications suivantes:

- L'identifiant 'BW64' est utilisé à la place de 'RIFF' dans les quatre premiers octets du fichier.
- Un fragment <ds64> (data size 64) est obligatoirement ajouté et doit être le premier fragment après «BW64 chunk».

Le fragment <ds64> a deux valeurs entières à 64 bits obligatoires, qui remplacent les deux champs de 32 bits du format RIFF/WAVE:

- bw64Size (remplace le champ taille du fragment <RIFF>)
- dataSize (remplace le champ taille du fragment <data>)

Pour les deux champs de 32 bits du format RIFF/WAVE, les règles suivantes s'appliquent:

Si la valeur à 32 bits contenue dans le champ n'est pas 0xFFFFFFFF, alors cette valeur à 32 bits est utilisée.

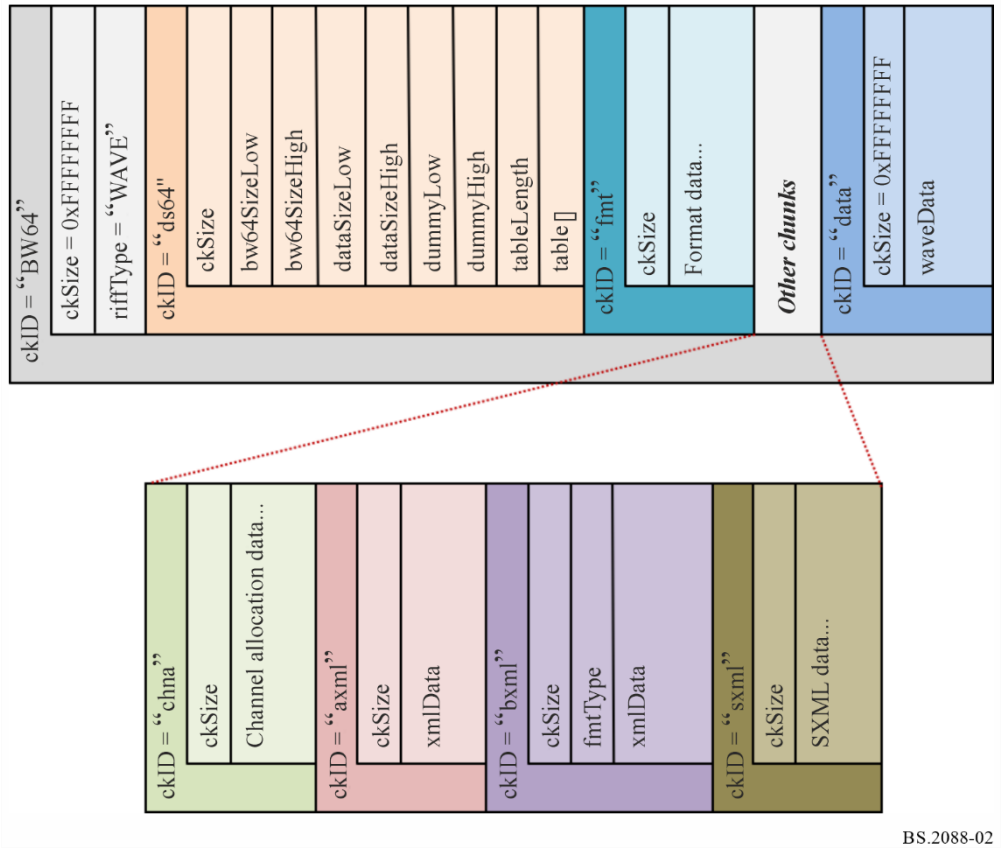
Si la valeur à 32 bits contenue dans le champ est 0xFFFFFFFF, la valeur à 64 bits du fragment 'ds64' est utilisée à la place.

- Un ensemble de structures (voir l'Annexe 1) avec des fragments additionnels à 64 bits est possible, à titre facultatif.

La Figure 2 montre la structure complète du format BW64, où les valeurs du champ ckSize pour les fragments <BW64> et <data> sont mis à 0xFFFFFFFF, afin de leur permettre d'utiliser les valeurs à 64 bits du fragment <ds64>.

FIGURE 2

Structure d'un fichier BW64



NOTE – La taille des données des fragments peut être variable. Le début de chaque fragment est aligné au niveau des mots par rapport au début du fichier BW64 afin de maintenir la compatibilité avec le format BWF spécifié dans la Recommandation UIT-R BS.1352. Si la taille du fragment correspond à un nombre impair d'octets, un octet de remplissage de valeur nulle est inséré après le fragment. La valeur de ckSize n'inclut toutefois pas l'octet de remplissage.

2.5 Assurer la compatibilité entre le format RIFF/WAVE et le format BW64

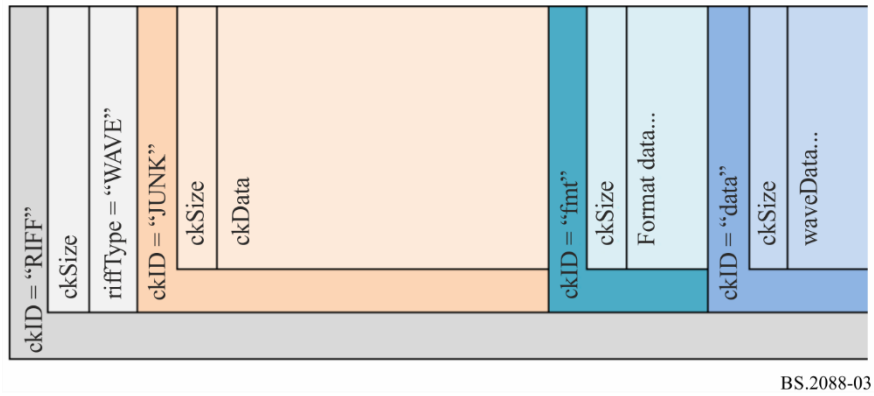
Malgré des fréquences d'échantillonnage plus élevées et la présence d'éléments audio multicanaux, certains fichiers audio de production feront forcément moins de 4 gigaoctets et devraient par conséquent rester au format RIFF/WAVE version courte (décrit dans l'Annexe 2). Le problème est qu'une application d'enregistrement ne peut pas savoir à l'avance si l'élément audio enregistré qu'elle est en train de compiler fera plus de 4 gigaoctets ou non une fois l'enregistrement terminé (c'est-à-dire savoir si elle doit ou non utiliser le format BW64).

La solution est de permettre à l'application d'enregistrement de passer du format RIFF/WAVE au format BW64 à la volée lorsque la taille limite de 4 gigaoctets est atteinte, tout en poursuivant l'enregistrement.

Pour ce faire, on réserve un espace supplémentaire dans le format RIFF/WAVE en insérant un fragment <JUNK> de la même taille qu'un fragment <ds64>. Cet espace réservé n'a pas de signification pour la version courte du format WAVE, mais il deviendra le fragment <ds64> s'il est nécessaire de passer au format BW64. La Figure 3 montre ce fragment <JUNK> fictif qui est placé avant le fragment <fmt>.

FIGURE 3

Structure d'un fichier avec fragment JUNK



Lorsqu'elle commencera un enregistrement, une application compatible BW64 créera un fichier RIFF/WAVE dont le premier fragment sera le fragment <JUNK>. Tout en effectuant l'enregistrement, elle vérifiera la taille du fichier RIFF et des données. Si le fichier et les données dépassent 4 gigaoctets, l'application:

- remplacera le fragment ckID <JUNK> par le fragment <ds64> (le fragment <JUNK> devient donc le fragment <ds64>);
- insérera la taille du fichier RIFF et la taille du fragment 'data' dans le fragment <ds64>;
- mettra la taille du fichier RIFF et la taille du fragment 'data' dans les champs de 32 bits à 0xFFFFFFFF;
- remplacera le champ ID 'RIFF' par le champ 'BW64' dans les quatre premiers octets du fichier;
- poursuivra l'enregistrement.

2.6 Structures et fragments existants dans le format RIFF/WAVE

Les fragments qui existent dans le format RIFF/WAVE sont les suivants:

```
struct RiffChunk            // déclare la structure RiffChunk
{
    CHAR   ckID[4];         // 'RIFF'
    DWORD  ckSize;          // 4 octets pour la taille du fichier RIFF/WAVE classique
    CHAR   riffType[4];     // 'WAVE'
};

struct FormatChunk          // déclare la structure FormatChunk
{
    CHAR   ckID[4];         // 'fmt'
    DWORD  ckSize;          // 4 octets pour la taille du fragment 'fmt '
    WORD   formatTag;       // WAVE_FORMAT_PCM = 0x0001, etc.
    WORD   channelCount;    // 1 = mono, 2 = stéréo, etc.
    DWORD  sampleRate;      // 32000, 44100, 48000, etc.
    DWORD  bytesPerSecond;  // important uniquement pour les formats compressés
    WORD   blockAlignment;  // taille de conteneur (en octets) d'un ensemble
                            // d'échantillons
```

```
    WORD   bitsPerSample;   // bits valides par échantillon 16, 20 ou 24
    WORD   cbSize;          // devrait être exclu lorsque le champ extraData n'est
                            // pas utilisé mais, lorsqu'il est présent,
                            // doit être mis à zéro
    CHAR   extraData[22];   // données supplémentaires du champ
                            // WAVE_FORMAT_EXTENSIBLE au besoin,
                            // ne doit pas être utilisé car le champ cbSize sera
                            // mis à zéro ou absent.
};

struct DataChunk            // déclare la structure DataChunk
{
    CHAR   ckID[4];         // 'data'
    DWORD  ckSize;          // 4 octets pour la taille du fragment 'data'
    CHAR   waveData[];      // échantillons audio
};
```

Les intervalles vides entre crochets indiquent qu'un nombre variable d'éléments peut être utilisé (y compris aucun).

2.6.1 Éléments du fragment <RIFF>

Le fragment <RIFF> est le fragment de premier niveau du fichier.

Champ	Description
ckID	Suite de 4 caractères {'R', 'I', 'F', 'F'} utilisée pour identifier le fragment.
ckSize	4 octets pour la taille du fichier.
riffType	Suite de 4 caractères {'W', 'A', 'V', 'E'} indiquant que le fichier est un fichier audio de type WAVE.

2.6.2 Éléments du fragment <fmt>

Le fragment <fmt> contient les informations sur les formats des échantillons audio enregistrés dans le fragment <data>.

Champ	Description
ckID	Suite de 4 caractères {'f', 'm','t', ' '} utilisée pour identifier le fragment.
ckSize	4 octets pour la taille du fragment.
formatTag	2 octets représentant le format des échantillons audio. La valeur 0x0001 signifie qu'il s'agit d'un format MIC, et 0x0000 désigne les formats inconnus.
channelCount	2 octets indiquant le nombre de pistes audio contenues dans le fichier.
sampleRate	4 octets indiquant la fréquence d'échantillonnage des éléments audio en Hz.
bytesPerSecond	Nombre moyen d'octets de données audio à transférer par seconde. Le logiciel de reproduction peut estimer la capacité tampon nécessaire en utilisant cette valeur.
blockAlignment	Alignement de blocs (en octets) des données audio. Le logiciel de reproduction a besoin de traiter de multiples octets de données **blockAlignment** à la fois, de sorte que la valeur de **blockAlignment** peut être utilisée pour l'alignement du tampon.

Champ	Description
bitsPerSample	Nombre de bits par échantillon et par canal. Chaque canal est censé avoir la même résolution de codage des échantillons. Si ce champ n'est pas nécessaire, il doit être mis à zéro.
cbSize	Taille en octets de la structure extraData.
extraData	Données supplémentaires utilisées pour enregistrer l'information WAVE_FORMAT_EXTENSIBLE. Ne doit pas être utilisé dans le format BW64.

Le fragment FormatChunk est déjà le fragment de format spécialisé pour les données audio MIC.

La suite extraData du champ FormatChunk est utilisée lorsque le champ formatTag est mis à 0xFFFE (WAVE_FORMAT_EXTENSIBLE). Étant donné que les éléments audio multicanaux devraient être décrits par des métadonnées ADM, l'utilisation du champ formatTag devrait être évitée. Toutefois, les mises en œuvre devraient être capables de lire un fichier contenu ce champ et de le traiter correctement.

Afin de s'assurer que le champ FormatChunk ne contredit pas les informations contenues dans les fragments <chna>, <axml>, <bxml> et <sxml>, il est recommandé de mettre le champ formatTag à 0x0001 pour les éléments audio MIC, et à 0x0000 (formatTag = inconnu) pour tous les autres éléments audio ayant un autre format.

2.6.3 Éléments du fragment <data>

Le fragment <data> permet d'enregistrer les échantillons audio.

Champ	Description
ckID	Suite de 4 caractères {‘d’, ‘a’, ‘t’, ‘a’} utilisée pour identifier le fragment.
ckSize	4 octets pour la taille du fragment.
waveData	Lieu d'enregistrement des échantillons audio. Les échantillons sont enregistrés en commençant par le bit de plus faible poids. De multiples pistes sont stockées par entrelacement, échantillon par échantillon. Par exemple, pour un élément audio à 2 pistes de 16 bits:

Octet	Échantillon	Piste
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 Fragment de premier niveau BW64

3.1 Définition

Le fragment de premier niveau <BW64> remplace le fragment <RIFF> qui est utilisé dans les fichiers de 32 bits. La présence de ce fragment signifie qu'il doit y avoir le fragment <ds64> pour lire les fichiers de 64 bits. Le fragment <BW64> est décrit ci-après:

```
struct BW64Chunk           // déclare la structure BW64Chunk
{
    CHAR ckID[4];        // 'BW64'
    DWORD ckSize;        // 0xFFFFFFFF signifie ne pas utiliser ces données,
                         // utiliser bw64SizeHigh et bw64SizeLow dans le
                         // fragment 'ds64' à la place
    CHAR BW64Type[4];   // 'WAVE'
};
```

3.2 Éléments du fragment <BW64>

Champ	Description
ckID	Suite de 4 caractères {'b', 'w', '6', '4'} utilisée pour identifier le fragment.
ckSize	4 octets qui doivent être mis à 0xFFFFFFFF pour indiquer que cette valeur de taille n'est pas utilisée et que le fragment <ds64> doit être utilisé pour déterminer les tailles.
BW64Type	Suite de 4 caractères {'W', 'A', 'V', 'E'} indiquant que le fichier est un fichier audio de type WAVE.

4 Fragments DS64 et JUNK

4.1 Définitions

Le fragment <ds64> achemine une valeur de 64 bits indiquant la taille du fichier, le fragment <data> et une suite de valeurs de 64 bits indiquant la taille d'autres fragments pouvant être définis. La structure du fragment <ds64> est présentée ci-après, suivie de la structure de la table **ChunkSize64** qui achemine la taille des fragments pouvant être définis (autres que le fragment <data>). Les intervalles vides entre crochets indiquent qu'un nombre variable d'éléments peut être utilisé (y compris aucun).

```
struct DataSize64Chunk     // déclare la structure DataSize64Chunk
{
    CHAR ckID[4];          //'ds64', identifiant de fragment FOURCC
    DWORD ckSize;          // 4 octets pour la taille du fragment <ds64>
    DWORD bw64SizeLow;     // 4 octets de faible poids pour la taille du fragment <BW64>
    DWORD bw64SizeHigh;    // 4 octets de poids fort pour la taille du fragment <BW64>
    DWORD dataSizeLow;     // 4 octets de faible poids pour la taille du fragment <data>
    DWORD dataSizeHigh;    // 4 octets de poids fort pour la taille du fragment <data>
    DWORD dummyLow;        // valeur fictive pour la compatibilité
    DWORD dummyHigh;       // valeur fictive pour la compatibilité
    DWORD tableLength;     // nombre d'entrées valides dans la suite "table"
    ChunkSize64 table[];   // suite de tailles de fragment pour les fragments de plus
                           // de 4 gigaoctets
};
struct ChunkSize64         // déclare la structure ChunkSize64
{
    CHAR ckID[4];         // identifiant du fragment nécessitant un adressage à 64 bits;
                          // par exemple. 'axml' est utilisé lorsque le fragment <axml>
```

```
                        // fait plus de 4 gigaoctets
    DWORD ckSizeLow;    // 4 octets de poids faible pour la taille du fragment
    DWORD ckSizeHigh;   // 4 octets de poids fort pour la taille du fragment
};
```

Le fragment <JUNK> réserve la place du fragment <ds64> qui est utilisé dans le cas où un fichier audio de 32 bits en cours de production doit être converti ultérieurement à la volée en fichier de 64 bits. La taille du fragment <JUNK> doit correspondre à la taille du fragment <ds64> qui pourrait le remplacer. La structure du fragment est présentée ci-après:

```
struct JunkChunk          // déclare la structure JunkChunk
{
    CHAR   ckID[4];      // 'JUNK'
    DWORD  ckSize;       // 4 octets pour la taille du fragment 'JUNK'. Doit être au
                         // moins 28 si le fragment sert à réserver la place pour un
                         // fragment 'ds64'.
    CHAR   ckData[];    // octets fictifs
};
```

4.2 Éléments du fragment <ds64>

Champ	Description
ckID	Suite de 4 caractères {‘d’, ‘s’, ‘6’, ‘4’} utilisée pour identifier le fragment.
ckSize	4 octets pour la taille du fragment <ds64>.
bw64SizeLow	4 octets de faible poids pour la taille du fragment <BW64>. La taille des données à 64 bits est exprimée sous la forme 0xHHHHLLLL si <bw64SizeLow> et <bw64SizeHigh> sont 0xLLLL et 0xHHHH, respectivement. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
bw64SizeHigh	4 octets de poids fort pour la taille du fragment <BW64>. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
dataSizeLow	4 octets de faible poids pour la taille du fragment <data>. La taille des données à 64 bits est exprimée sous la forme 0xHHHHLLLL si <dataSizeLow> et <dataSizeHigh> sont 0xLLLL et 0xHHHH, respectivement. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
dataSizeHigh	4 octets de poids fort pour la taille du fragment <data>. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
dummyLow	Valeur fictive sur 4 octets qui doit être ignorée lorsqu'elle est lue et mise à zéro en mode écriture. Elle permet de garantir la compatibilité avec la spécification EBU Tech 3306 FR64, qui utilise cette valeur pour acheminer l'information sur la taille du fragment <fact> qui n'existe pas dans le format BW64.
dummyHigh	Valeur fictive sur 4 octets qui doit être ignorée lorsqu'elle est lue et mise à zéro en mode écriture. A la même fonction que le champ <dummyLow>.
tableLength	Nombre d'entrées valides dans la suite «ChunkSize64 table».
ChunkSize64 table	Suite de tailles de fragment pour les fragments de plus de 4 gigaoctets.

La table **ChunkSize64** est spécifiée ci-après. Une suite de structures **ChunkSize64** est utilisée pour enregistrer la longueur de tous les fragments autres que le fragment <data> dans la partie opérationnelle du fragment <ds64>. Actuellement, le seul type de fragment autre que le fragment <data> qui soit susceptible de dépasser 4 gigaoctets est le fragment <axml> (possible dans les fichiers audio fondés sur des objets de très grande taille).

Champ	Description
ckID	Suite de 4 caractères utilisée pour faire le lien vers le champ <ckID> du fragment nécessitant un adressage 64 bits. Par exemple, la suite de 4 caractères {‘a’, ‘x’, ‘m’, ‘l’} est utilisée pour le fragment <axml>.
ckSizeLow	4 octets de faible poids pour le fragment renvoyant à <ckID>. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
ckSizeHigh	4 octets de poids fort pour le fragment renvoyant à <ckID>. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.

4.3 Éléments du fragment <JUNK>

Champ	Description
ckID	Suite de 4 caractères {‘J’, ‘U’, ‘N’, ‘K’} utilisée pour identifier le fragment.
ckSize	4 octets pour la taille du fragment <JUNK>. Doit être égal ou supérieur à 28 pour réserver la place du fragment <ds64>.
ckData	Données fictives devant être ignorées.

5 Fragment AXML

5.1 Définition

Le fragment <axml> peut contenir tout type de données compatibles avec la version 1.0 ou une version ultérieure du format XML, qui est un format d'échange de données répandu [1]. Il est à noter que le fragment <axml> peut contenir des fragments XML provenant de plusieurs schémas. Il peut apparaître dans un ordre quelconque avec les autres fragments RIFF à l'intérieur du même fichier.

Le fragment <axml> est composé d'un en-tête suivi de données compatibles avec le format XML. La longueur totale du fragment n'est pas fixe.

On trouvera au paragraphe 11 un exemple d'utilisation possible du fragment <axml> dans le format BW64 pour acheminer des métadonnées pour la diffusion, y compris les paramètres contenus dans les anciens fragments <bext> et <ubxt>.

```
struct axml_chunk
{
    CHAR    ckID[4];        // {'a','x','m','l'}
    DWORD   ckSize;         // taille du fragment <axml> en octets
    CHAR    xmlData[];     // données textuelles en XML
};
```

Étant donné que les données XML peuvent prendre plus de 4 gigaoctets, il faudra peut-être utiliser le fragment <ds64> pour que le fragment <axml> puisse contenir un champ de 64 bits. On trouvera ci-après un pseudo-code pour montrer comment procéder en utilisant la table dans le fragment <ds64>.

```
DataSize64Chunk.tableLength = 1;   // nombre d'entrées valides dans “table”
DataSize64Chunk.table[0] = {
    ChunkSize64.ckID = {`a`, `x`, `m`, `l`};   // ID du fragment <axml>
     ckSizeLow = xxxx     // 4 octets de faible poids pour la taille du fragment
     ckSizeHigh = xxxx    // 4 octets de poids fort pour la taille du fragment
}
```

5.2 Éléments du fragment <axml>

ckID	Suite de 4 caractères {‘a’, ‘x’, ‘m’, ‘l’} utilisée pour identifier le fragment.
ckSize	Taille de la section du fragment contenant les données en octets (ne comprend pas les 8 octets utilisés par les champs ckID et ckSize.)
xmlData	Informations textuelles en XML.

Les données ont une structure XML hiérarchique et sont enregistrées dans des chaînes textuelles au format XML, version 1.0 ou ultérieure.

Si l'appareil de réception ne peut pas interpréter le contenu du fragment <axml> conformément aux spécifications indiquées dans l'élément XML, la totalité du fragment est ignorée.

6 Fragment BXML

6.1 Définition

Le fragment <bxml> peut contenir les données XML compressées, à la place du fragment <axml>.

Le fragment <bxml> est composé d'un en-tête suivi des données XML compressées à l'aide de la méthode de compression spécifiée dans **fmtType**. La longueur totale du fragment n'est pas fixe.

```
struct bxml_chunk
{
    CHAR    ckID[4];        // {'b','x','m','l'}
    DWORD   ckSize;         // taille du fragment <bxml> en octets
    WORD    fmtType;        // type de méthode de compression, 0x0001=”gzip”, etc.
    CHAR   xmlData[];     // données textuelles en XML compressées à l'aide de la méthode
                          // de compression
};
```

Étant donné que les données XML compressées peuvent prendre plus de 4 gigaoctets, il faudra peut-être utiliser le fragment <ds64> pour que le fragment <bxml> puisse contenir un champ de 64 bits. On trouvera ci-après un pseudo-code pour montrer comment procéder en utilisant la table dans le fragment <ds64>.

```
DataSize64Chunk.tableLength = 1;   // nombre d'entrées valides dans “table”
DataSize64Chunk.table[0] = {
    ChunkSize64.ckID = {`b`, `x`, `m`, `l`};   // ID du fragment <bxml>
     ckSizeLow = xxxx     // 4 octets de faible poids pour la taille du fragment
     ckSizeHigh = xxxx    // 4 octets de poids fort pour la taille du fragment
}
```

6.2 Éléments du fragment <bxml>

ckID	Suite de 4 caractères {‘b’, ‘x’, ‘m’, ‘l’} utilisée pour identifier le fragment.
ckSize	Taille de la section du fragment contenant les données en octets (ne comprend pas les 8 octets utilisés par les champs ckID et ckSize.)
fmtType	Valeur de 2 octets qui représente la méthode de compression du texte XML. La valeur 0x0001 correspond à la méthode de compression gzip (IETF RFC 1952). On utilise la valeur 0x0000 pour le texte XML non compressé.
xmlData	Code XML compressé à l'aide de la méthode de compression indiquée par fmtType.

7 Fragment SXML

7.1 Définition

Le fragment <sxml> peut contenir tout type de données XML compressées ou non compressées compatibles avec la version 1.0 ou une version ultérieure du format XML en association avec des segments de données audio. Il peut apparaître dans un ordre quelconque avec les autres fragments RIFF à l'intérieur du même fichier.

Le fragment <sxml> est composé d'un en-tête suivi de sous-fragments **(SubXMLChunk)** avec des données XML compressées ou non compressées comme spécifié par **fmtType**. Chaque **SubXMLChunk** correspond à un nombre unique d'échantillons contigus avec les **SubXMLChunk**s adjacents. Le fragment <sxml> est complété par une table facultative de points d'alignement, qui permet un accès par horodate à un certain **SubXMLChunk**. La longueur totale du fragment <sxml> n'est pas fixe.

Le fragment <sxml> peut être utilisé pour le transport de métadonnées variant dans le temps, par exemple pour une représentation série du modèle ADM spécifié dans la Recommandation UIT-R BS.2125.

```
struct sxml_chunk
{
    CHAR    ckID[4];              // {'s','x','m','l'}
    DWORD   ckSize;               // taille du fragment <sxml> en octets
    WORD    fmtType;              // type de méthode de compression, 0x0001=”gzip”, etc.
    DWORD  subXMLCkTbSizeLow;     //  4  octets  de  faible  poids  pour  la  taille  de
                                  // nSubXMLChunks + SubXMLChunk table[]
    DWORD  subXMLCkTbSizeHigh;    //  4  octets  de  poids  fort  pour  la  taille  de
                                  // nSubXMLChunks + SubXMLChunk table[]
    DWORD   nSubXMLChunks;        // nombre de sous-fragments avec des données XML
    SubXMLChunk table[];          // suite de sous-fragments avec des données XML
    DWORD   nAlignmentPoints;     // nombre de points d'alignement
    AlignmentPoint table[];       // suite de points d'alignement
};

struct SubXMLChunk
{
    DWORD   subXMLChunkSize;      // taille de SubXMLChunk en octets
    DWORD   nSamplesSubDataChunk;// nombre d'échantillons audio associés à SubXMLChunk
    CHAR    xmlData[];           // données XML compressées ou non compressées
};

struct AlignmentPoint
{
    DWORD   subXMLChunkByteOffsetLow;   // 4 octets de faible poids pour le décalage
                                        // de SubXMLChunk en octets
    DWORD   subXMLChunkByteOffsetHigh;  // 4 octets de poids fort pour le décalage
                                        // de SubXMLChunk en octets
    DWORD   nSamplesAlignPointLow;      // 4 octets de faible poids pour le nombre
                                   // d'échantillons correspondant au point d'alignement
```

```
    DWORD    nSamplesAlignPointHigh;    // 4 octets de poids fort pour le nombre
                                        // d'échantillons correspondant au point d'alignement
};
```

Étant donné que les données XML compressées ou non compressées peuvent prendre plus de 4 gigaoctets, il faudra peut-être utiliser le fragment <ds64> pour que le fragment <sxml> puisse contenir un champ de 64 bits. On trouvera ci-après un pseudo-code pour montrer comment procéder en utilisant la table dans le fragment <ds64>.

```
DataSize64Chunk.tableLength = 1;   // nombre d'entrées valides dans "table"
DataSize64Chunk.table[0] = {
    ChunkSize64.ckID = {`s`, `x`, `m`, `l`};   // ID du fragment <sxml>
     ckSizeLow = xxxx     // 4 octets de faible poids pour la taille du fragment
     ckSizeHigh = xxxx    // 4 octets de poids fort pour la taille du fragment
}
```

7.2 Éléments du fragment <sxml>

Champ	Description
ckID	Suite de 4 caractères {'s', 'x', 'm', 'l'} utilisée pour identifier le fragment.
ckSize	Taille de la section du fragment contenant les données en octets (ne comprend pas les 8 octets utilisés par les champs ckID et ckSize.)
fmtType	Valeur de 2 octets qui représente la méthode de compression du texte XML. La valeur 0x0001 correspond à la méthode de compression gzip (IETF RFC 1952). On utilise la valeur 0x0000 pour les données XML non compressées.
subXMLCkTbSizeLow	4 octets de faible poids pour la taille de SubXMLChunk table[], y compris les 4 octets du champ nSubXMLChunks. La taille des données à 64 bits est exprimée sous la forme 0xHHHHLLLL si <subXMLCkTbSizeLow> et <subXMLChTbSizeHigh> sont 0xLLLL et 0xHHHH, respectivement. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
subXMLCkTbSizeHigh	4 octets de poids fort pour la taille de SubXMLChunk table[], y compris les 4 octets du champ nSubXMLChunks.
nSubXMLChunks	Nombre d'entrées valides dans la suite «SubXMLChunk table».
SubXMLChunk table	Suite de sous-fragments avec des données XML.
nAlignmentPoints	Nombre d'entrées valides dans la suite «AlignmentPoint table».
AlignmentPoint table	Suite de points d'alignement.

La table **SubXMLChunk** est spécifiée comme suit.

Champ	Description
subXMLChunkSize	Taille de la section du sous-fragment SubXMLChunk contenant les données en octets, non compris les 8 octets utilisés par subXMLChunkSize et nSamplesSubDataChunk.
nSamplesSubDataChunk	Nombre d'échantillons audio par canal associés au sous-fragment SubXMLChunk.
xmlData	Données XML ou données XML compressées à l'aide de la méthode de compression indiquée par fmtType.

La table **AlignmentPoint** est spécifiée comme suit.

Champ	Description
subXMLChunkByteOffsetLow	4 octets de faible poids pour le décalage en octets du début d'un sous-fragment SubXMLChunk par rapport au point d'alignement, décalage exprimé en octets par rapport au début du fragment <sxml>, non compris les 8 octets utilisés par ckID et ckSize. Le décalage à 64 bits est exprimé sous la forme 0xHHHHLLLL si <subXMLChunkByteOffsetLow> et <subXMLChunkByteOffsetHigh> sont 0xLLLL et 0xHHHH, respectivement. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
subXMLChunkByteOffsetHigh	4 octets de poids fort pour le décalage en octets du début d'un sous-fragment SubXMLChunk par rapport au point d'alignement. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
nSamplesAlignPointLow	4 octets de faible poids pour le nombre d'échantillons audio par canal correspondant à l'horodate du point d'alignement par rapport au début du fragment <data>. Le nombre à 64 bits est exprimé sous la forme 0xHHHHLLLL si <nSamplesAlignPointLow> et <nSamplesAlignPointHigh> sont 0xLLLL et 0xHHHH, respectivement. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.
nSamplesAlignPointHigh	4 octets de poids fort pour le nombre d'échantillons correspondant à l'horodate. On indique la quantité sans signe de 32 bits en commençant par les bits de plus faible poids.

8 Fragment CHNA

8.1 Définition

Le fragment <chna> est un fragment défini spécifiquement pour être utilisé avec le modèle ADM spécifié dans la Recommandation UIT-R BS.2076. Le fragment <chna> est composé d'un en-tête suivi du nombre de pistes et du nombre d'identifiants UID de piste utilisés. Il est suivi d'une suite de structures ID contenant chacune les identifiants correspondant aux identifiants des éléments ADM.

La taille du fragment dépend du nombre d'identifiants UID de piste à définir. Le nombre de structures ID doit être égal ou supérieur au nombre d'identifiants UID de piste utilisés. En autorisant un nombre de structures ID supérieur au nombre d'identifiants UID, il est possible de faciliter la mise à jour et l'ajout de nouveaux identifiants dans le fragment, sans avoir à en modifier la taille. Par exemple, il se peut qu'on ne sache pas au départ combien d'identifiants UID seront générés; par conséquent, le nombre de structures ID dans le fragment est mis à 64 (ce nombre étant considéré comme suffisant pour l'opération à accomplir). Le logiciel génère ensuite 55 identifiants UID (par exemple), qui remplissent les 55 premières structures ID et les 9 structures ID restantes sont donc mises à zéro.

L'identifiant ADM à l'intérieur du fragment peut renvoyer aux métadonnées ADM acheminées dans le fragment <axml>, <bxml> ou <sxml> ou dans un fichier externe de définitions communes. Si les 4 derniers chiffres hexadécimaux de l'identifiant ont une valeur d'au plus 0x0FFF, ils sont alors définis selon les définitions communes contenues dans la Recommandation UIT-R BS.2094 – *Définitions communes pour le Modèle de définition audio* (par exemple, définition des canaux 'avant gauche' et 'avant droit'). Tous les identifiants avec des valeurs d'au moins 0x1 000 seront considérés comme ayant une définition personnalisée et seront donc contenus dans le fragment <axml>, <bxml> ou <sxml> à l'intérieur du fichier.

La structure audioID contient un index vers la piste utilisée dans le fragment <data> (qui contient les échantillons audio), qui commence par la valeur 1 pour la première piste. Elle contient un identifiant UID pour la piste, qui sera contenu dans les métadonnées ADM. Les éléments audio d'une piste peuvent varier tout au long d'un fichier; dans ce cas, il y aura un identifiant UID différent pour chaque définition. Par conséquent, il est possible d'avoir plusieurs identifiants UID pour chaque piste. Les deux autres valeurs dans la structure renvoient aux identifiants des éléments audioTrackFormat et audioPackFormat du modèle ADM. Le modèle ADM permet d'omettre audioTrackFormat et audioStreamFormat si le type de format de l'enveloppe des caractéristiques audio est MIC linéaire, auquel cas il est fait référence à audioChannelFormat au lieu de audioTrackFormat.

```
struct chna_chunk
{
    CHAR    ckID[4];        // {'c','h','n','a'}
    DWORD   ckSize;         // taille du fragment <chna>
    WORD    numTracks;      // nombre de pistes utilisées
    WORD    numUIDs;        // nombre d'identifiants UID de piste utilisés
    audioID ID[N];          // identifiants pour chaque piste (où N >= numUIDs)
};

struct audioID
{
    WORD    trackIndex;     // index de piste dans le fichier
    CHAR    UID[12];        // valeur audioTrackUID
    CHAR    trackRef[14];   // référence audioTrackFormatID ou audioChannelFormatID
    CHAR    packRef[11];    // référence audioPackFormatID
    CHAR    pad;            // octet de remplissage pour garantir un nombre pair
                            // d'octets
}
```

8.2 Éléments du fragment <chna>

ckID Suite de 4 caractères {‘c’,‘h’,‘n’,‘a’}[1] utilisée pour identifier le fragment.

ckSize Taille de la section du fragment contenant les données en octets (ne tient pas compte des 8 octets utilisés par ckID et ckSize.)

numTracks Nombre de pistes utilisées dans le fichier. Même si une piste contient plus d'un jeu d'identifiants, cela reste une seule piste.

[1] **Remarque**: La définition DWORD ckID = «chna» ne serait pas unique. Différentes architectures produisent des ordres de caractères différents. Par conséquent, nous définissons char ckID[4] = {‘c’,‘h’,‘n’,‘a’} à la place.

numUIDs Nombre d'identifiants UID utilisés dans le fichier. Étant donné qu'il est possible de donner de multiples identifiants UID pour une seule piste (couvrant différentes périodes), il peut s'agir d'une valeur supérieure à celle du champ **numTracks**. Cette valeur doit correspondre au nombre d'identifiants défini dans le champ **ID**.

ID Structure contenant le jeu d'identifiants audio de référence pour la piste. Contient N identificateurs où N >= numUIDs. Lorsque numUIDs est inférieur à N, le contenu des identifiants de piste non utilisés est mis à zéro. Lors de la lecture du fragment, la valeur de N peut être déduite du champ ckSize, puisque ckSize = 4 + (N * 40), donc N = (ckSize – 4) / 40.

trackIndex Index de piste dans le fichier, commençant à 1. Correspond directement à l'ordre des pistes entrelacées dans le fragment <data>.

UID Valeur audioTrackUID de la piste. La suite de caractères a le format ATU_xxxxxxxx où x est un chiffre hexadécimal.

trackRef Référence audioTrackFormatID de la piste. La suite de caractères a le format AT_xxxxxxxx_xx, où x est un chiffre hexadécimal. Le format AC_xxxxxxxx_00 (le suffixe «00» sert de remplissage pour que la chaîne corresponde au format de la chaîne audioTrackFormatID, et n'a pas de signification particulière), où x est un chiffre hexadécimal, est également utilisé lorsque les deux éléments audioTrackFormat et audioStreamFormat pour l'enveloppe des caractéristiques audio correspondant à MIC linéaire sont omis et qu'il est directement fait référence à audioChannelFormat dans le code XML ADM.

packRef Référence audioPackFormatID de la piste. La suite de caractère a le format AP_xxxxxxxx où x est un chiffre hexadécimal. Lorsque le champ audioPackFormatID n'est pas requis (lorsque le champ audioStreamFormat renvoie à un champ audioPackFormat plutôt qu'à un champ audioChannelFormat), ce champ doit être rempli avec des valeurs nulles.

pad Octet unique pour garantir que la structure audioID a un nombre pair d'octets.

Lorsqu'un **identifiant** n'est pas utilisé, le champ **trackIndex** doit avoir la valeur zéro et les autres champs doivent être des chaînes nulles de la même longueur que les chaînes ID habituelles utilisées. Par conséquent, la chaîne nulle du champ packRef serait composée de 11 caractères néant (valeur zéro ASCII) et le champ trackRef serait composé de 14 caractères néant.

8.3 Exemples

On trouvera ci-après quelques exemples illustrant le fonctionnement du fragment <chna>. Dans chaque exemple, le pseudo-code utilise une notation de type chaîne pour les identifiants (par exemple, «AT_00010001_01»), où, dans la pratique, il convient d'utiliser une suite de caractères pour éviter l'inclusion de caractères nuls de terminaison (on procèderait donc ainsi {‘A’,’T’,’_’,’0’,’0’,’0’,’1’,’0’,’0’,’0’,’1’,’_’,’0’,’1’}).

8.3.1 Fichier stéréo simple

La plupart des fichiers audio existants sont encore des fichiers stéréo à deux canaux, la première piste contenant le canal gauche et la seconde le canal droit. Dans le modèle ADM, le canal gauche est défini avec l'identifiant AT_00010001_01, et le canal droit avec l'identifiant AT_00010002_01. Le paquet stéréo est défini avec l'identifiant AP_00010002.

Le pseudo-code est le suivant:

```
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`; };
```

Le nombre de structures ID est de 2, il n'y a donc pas de structure ID non utilisée dans cet exemple.

Lorsque, dans le modèle ADM, les deux éléments audioTrackFomat et audioStreamFormat sont omis et qu'il est fait référence à audioChannelFormat, on utilise le code suivant.

```
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 Exemple avec un objet simple

Il se peut que les objets audio couvrent uniquement une partie de la durée du fichier audio. Pour économiser de l'espace, les objets qui ne se chevauchent pas peuvent partager la même piste. On aurait dans ce cas de multiples identifiants UID dans la même piste. Dans cet exemple, on utilise en outre plus grand nombre de structures ID (32 en l'espèce) que d'identifiants numUID pour monter comment les structures ID non utilisées sont mises à zéro.

```
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 première piste contient 3 identifiants UID, donc 3 objets différents (avec les identifiants de piste AT_00031001_01, AT_00031003_01 et AT_00031004_01) à différents emplacements indiquant le temps dans le fichier. La seconde piste contient un identifiant UID, donc un objet. Cet objet aura le même identifiant de paquet (AP_00031001) que le premier objet de la piste 1, ce qui indique que le premier objet contient deux canaux acheminés à la fois dans la piste 1 et dans la piste 2. Les métadonnées ADM acheminées dans le fragment <axml>, <bxml> ou <sxml> seraient utilisées pour préciser l'attribution des canaux et des pistes.

8.3.3 Exemple avec de multiples contenus

Le fichier BW64 pourrait contenir de multiples objets dans un seul fichier, par exemple un mélange 5.1 principal sur les 6 premières pistes, avec un mélange stéréo en langue étrangère sur les deux pistes suivantes. La Recommandation UIT-R BS.1738 présente plusieurs configurations; cet exemple montre comment le scénario de production 5 décrit dans cette Recommandation peut être pris en charge avec le fragment <chna>. Ce scénario prévoit 8 pistes: les 6 premières contiennent un mélange complet 5.1 et les 2 dernières pistes contiennent un mélange stéréo de son international. Le fragment <chna> correspondant est le suivant:

```
ckID = {'c','h','n','a'};
ckSize = 84;
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[1]={ trackIndex=3; UID="ATU_00000003"; trackRef="AT_00010003_01"; packRef="AP_00010003"; pad='\0`; };
ID[1]={ trackIndex=4; UID="ATU_00000004"; trackRef="AT_00010004_01"; packRef="AP_00010003"; pad='\0`; };
ID[1]={ trackIndex=5; UID="ATU_00000005"; trackRef="AT_00010005_01"; packRef="AP_00010003"; pad='\0`; };
ID[1]={ trackIndex=6; UID="ATU_00000006"; trackRef="AT_00010006_01"; packRef="AP_00010003"; pad='\0`; };
ID[1]={ trackIndex=7; UID="ATU_00000007"; trackRef="AT_00010001_01"; packRef="AP_00010002"; pad='\0`; };
ID[1]={ trackIndex=8; UID="ATU_00000008"; trackRef="AT_00010002_01"; packRef="AP_00010002"; pad='\0`; };
```

Les métadonnées ADM dans le fragment <axml>, <bxml> ou <sxml> contiendront des informations sur la manière dont les deux mélanges sont découpés.

9 Règles applicables aux fragments XML

Il existe trois fragments différents qui peuvent acheminer des métadonnées XML: <axml>, <bxml> et <sxml>. Si l'objectif premier de ces fragments est d'acheminer des métadonnées XML ADM (comme spécifié dans la Recommandation UIT-R BS.2076) ou des métadonnées S-ADM (comme spécifié dans la Recommandation UIT-R BS.2125), il leur est également possible d'acheminer d'autres métadonnées XML, telles que les métadonnées pour la diffusion décrites au paragraphe 11. Étant donné que plusieurs fragments peuvent acheminer des métadonnées XML, il existe un risque que les métadonnées d'un fragment soient contraires à celles d'un autre fragment. Les règles suivantes doivent donc être appliquées:

1 Il ne doit pas y avoir plus d'une instance d'un fragment XML donné.

2 Si des métadonnées ADM sont acheminées:

   a) elles doivent figurer soit dans le fragment <axml> soit dans le fragment <bxml>, mais pas dans les deux;

   b) le fragment <chna> contenant des références croisées pour les métadonnées ADM doit être présent.

3 Si des métadonnées S-ADM sont acheminées, elles doivent figurer uniquement dans le fragment <sxml>.

4 Si des métadonnées ADM et des métadonnées S-ADM sont acheminées, elles doivent être indépendantes les unes des autres (pas de références croisées entre elles).

5 Si d'autres métadonnées sont acheminées (autres que ADM ou que S-ADM):

   a) elles peuvent être acheminées avec les métadonnées ADM et S-ADM dans le même fragment si on le souhaite;

b) le contenu de ces «autres métadonnées» ne doit pas être redondant avec ce qui est décrit dans les métadonnées ADM ou S-ADM existantes;

c) si les «autres métadonnées» renvoient à des métadonnées ADM ou S-ADM, les métadonnées ADM ou S-ADM auxquelles il est fait référence doivent être présentes dans le fichier.

10 Compatibilité avec la Recommandation UIT-R BS.1352

Dans la mesure où le format BWF (Recommandation UIT-R BS.1352) est la version courte du format de fichier RIFF/WAVE (décrit dans l'Annexe 2) avec des fragments supplémentaires, le plus important étant les fragments <bext> et <ubxt>, il est nécessaire de comprendre comment assurer la compatibilité entre le format BWF et le format BW64.

Fragments BWF Rec. UIT-R BS.1352-4	Fragments BW64 Rec. UIT-R BS.2088-2	Marche à suivre
<fmt>	<fmt>	Utiliser de manière conventionnelle
<data>	<data>	Utiliser de manière conventionnelle
<fact>	<fact>	Utiliser de manière conventionnelle
–	<ds64>	Voir les § 2.4 et 4
–	<JUNK>	Voir les § 2.4 et 4
–	<chna>	Voir le paragraphe 8 pour l'attribution des canaux. Note: La Recommandation UIT-R BS.2088-0 ne prend pas en charge la référence à audioChannelFormat.
–	<axml>, <bxml> ou <sxml>	Voir les paragraphes 5 à 7. Utiliser pour les métadonnées pour la diffusion qui figureraient dans le fragment <bext> ou <ubxt>.
<bext> ou <ubxt>	(<bext> ou <ubxt>)	En cas de lecture du fragment <bext> ou <ubxt>, il devrait être converti au fragment <axml>, <bxml> ou <sxml> correspondant pour acheminer les métadonnées ADM et toutes autres métadonnées XML relatives à la diffusion. Voir le paragraphe 11 pour en savoir plus.

11 Générer des métadonnées pour la diffusion XML

Selon la Recommandation UIT-R BS.1352, les métadonnées pour la diffusion sont acheminées dans les fragments <bext> et <ubxt>. Ces fragments contiennent uniquement les champs d'une longueur fixe qui sont spécifiés, ce qui rend impossible la prise en charge de toutes autres métadonnées relatives à la diffusion. Les fragments <axml>, <bxml> et <sxml> dans le format BW64 peuvent acheminer n'importe quel type de métadonnées XML, et peuvent donc servir à acheminer les métadonnées pour la diffusion, y compris les paramètres figurant dans les fragments <bext> et <ubxt>.

Pour acheminer les paramètres <bext> ou <ubxt> dans les fragments <axml>, <bxml> et <sxml>, il convient d'utiliser la structure XML ci-après, dans laquelle les commentaires avec le préfixe 'BEXT' (le préfixe 'UBXT' est utilisé quand des paramètres <ubxt> sont acheminés) indiquent les paramètres contenus dans le fragment <bext>.

```
<?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>
```

Le code XML ci-dessus est fondé sur les schémas de métadonnées EBUCore [2] et AESCore [3], qui sont compatibles avec la Recommandation UIT-R BS.2076.

Lorsqu'on lit un fichier BWF UIT-R BS.1352 dans l'intention de le convertir en fichier BW64, les fragments <bext> ou <ubxt> devraient être convertis au langage XML décrit ici en vue de les insérer dans le fragment <axml>, <bxml> ou <sxml>.

12 Extension des fichiers BW64

L'extension «.wav» est l'extension définie pour les fichiers conformes au format BW64. Les anciens logiciels peuvent ainsi lire les fragments du fichier qu'ils comprennent (essentiellement les fragments <fmt> et <data>), afin qu'au minimum les échantillons audio soient accessibles.

Il n'est pas recommandé d'utiliser une autre extension de fichier lors de la création d'un fichier BW64, mais on peut s'attendre à ce qu'une extension «.bw64» soit utilisée à tort. Par conséquent, les logiciels capables de lire les fichiers BW64 doivent pouvoir lire cette autre extension de fichier.

13 Bibliographie

[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

Annexe 2
(pour information)

Format RIFF de fichier audio (.WAV)

Les informations contenues dans cette Annexe sont extraites des documents spécifiant le format de fichier RIFF. Elles ne figurent ici qu'à titre d'information, faute de sources extérieures fiables auxquelles se référer.

1 Format des fichiers audio de type WAVE

Le format WAVE est défini comme suit. Les logiciels devraient ignorer tous les fragments inconnus qui seront rencontrés, comme avec toutes les formes de type RIFF. Toutefois, le fragment <fmt-ck> apparaît toujours avant le fragment <wave-data>, et ces deux fragments sont obligatoires dans un fichier WAVE.

```
<WAVE-form> ->
    RIFF('WAVE'
        <fmt-ck>    // Fragment de format
        [<fact-ck>]   // Fragment factuel
        [<other-ck>]  // Autres fragments facultatifs
        <wave-data>)  // Données audio
```

Les fragments WAVE sont décrits dans les paragraphes suivants:

1.1 Fragment de format WAVE

Le fragment de format WAVE <fmt-ck> spécifie le format des données audio <wave-data>. Il est défini comme suit:

```
<fmt-ck> ->fmt(<common-fields>
               <format-specific-fields>)
<common-fields> ->
    Struct {
        WORD  wFormatTag;           // Catégorie de format
        WORD  nChannels;            // Nombre de canaux
        DWORD nSamplesPerSec;       // Fréquence d'échantillonnage
        DWORD nAvgBytesPerSec;      // Estimation du tampon
        WORD  nBlockAlign;          // Longueur de bloc de données
    }
```

Les champs contenus dans la portion <common-fields> du fragment sont les suivants:

Champ	Description
wFormatTag	Nombre indiquant la catégorie de format WAVE du fichier. Le contenu de la portion <format-specific-fields> du fragment <fmt-ck> ainsi que l'interprétation des données audio, dépendent de cette valeur.
nchannels	Nombre de canaux représentés dans les données audio: 1 pour la monophonie et 2 pour la stéréophonie.
nSamplesPerSec	Fréquence d'échantillonnage (en échantillons par seconde) à laquelle chaque canal doit être reproduit.
nAvgBytesPerSec	Nombre moyen d'octets de données audio à transférer par seconde. Le logiciel de reproduction peut estimer la capacité tampon nécessaire en utilisant cette valeur.
nBlockAlign	Alignement de blocs (en octets) des données audio. Le logiciel de reproduction a besoin de traiter de multiples octets de données <nBlockAlign> à la fois, de sorte que la valeur de ce champ peut être utilisée pour l'alignement du tampon.

Le champ <format-specific-fields> se compose de zéro, un ou plusieurs octets de paramètres. Les paramètres insérés dépendent de la catégorie de format WAVE. On trouvera de plus amples informations dans les paragraphes qui suivent. Le logiciel de reproduction doit être écrit de façon à permettre (et à ignorer) tous paramètres <format-specific-fields> inconnus apparaissant à la fin du champ.

1.2 Catégories de format WAVE

La catégorie de format d'un fichier WAVE est spécifiée par la valeur du champ <wFormatTag> du fragment <fmt>. La représentation des données dans le champ <wave-data> et le contenu du champ <format-specific-fields> du fragment <fmt> dépendent de la catégorie de format.

Les catégories de format WAVE non propriétaire ouvert actuellement défini, sont notamment les suivantes:

wFormatTag	Valeur	Catégorie de format
WAVE_FORMAT_UNKNOWN	0x0000	Inconnu
WAVE_FORMAT_PCM	0x0001	Format MIC
WAVE_FORMAT_IEEE_FLOAT	0x0003	Flottant IEEE
WAVE_FORMAT_EXTENSIBLE	0xFFFE	Format Wave extensible – Déterminé par le sous-format

NOTE – Seuls les formats WAVE_FORMAT_PCM et WAVE_FORMAT_UNKNOWN sont actuellement utilisés pour les fichiers BW64. Le paragraphe 2 ci-après donne des détails sur le format WAVE MIC. Le paragraphe 3 donne des informations générales sur d'autres formats WAVE. D'autres formats WAVE pourront être définis ultérieurement.

Auparavant, on utilisait la catégorie WAVE_FORMAT_EXTENSIBLE pour les fichiers multicanaux, mais cette utilisation devrait être évitée dans le futur.

1.3 Fragment factuel

Le fragment factuel <fact-ck> contient des informations variables selon les fichiers, concernant le contenu des fichiers WAVE non MIC. Ce fragment n'est donc pas utilisé dans cette version du format BW64. Ce fragment est défini comme suit:

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

Le champ <dwSampleLength> représente la longueur des données dans les échantillons. Le champ <nSamplesPerSec> de l'en-tête du format WAVE est utilisé en association avec le champ <dwSampleLength> pour déterminer la longueur des données en secondes.

Le fragment factuel est requis pour tous les nouveaux formats WAVE non MIC. Il n'est pas requis pour les fichiers WAVE_FORMAT_PCM types.

Le fragment factuel sera élargi pour inclure toute autre information requise par de futurs formats WAVE. Les champs ajoutés seront placés après le champ <dwSampleLength>. Les applications pourront utiliser le champ de dimension de fragment pour déterminer quels champs sont présents.

1.4 Autres fragments facultatifs

Un certain nombre d'autres fragments sont spécifiés pour usage dans le format WAVE. Les détails de ces fragments sont donnés dans la spécification du format WAVE et dans ses mises à jour ultérieures.

NOTE – Le format WAVE peut prendre en charge d'autres fragments facultatifs qui peuvent être inclus dans des fichiers WAVE pour transporter des informations spécifiques. Ces fragments sont considérés comme étant privés et seront ignorés par les applications qui ne peuvent pas les interpréter.

2 Format MIC

Si le champ <wFormatTag> du fragment <fmt-ck> est mis à la valeur WAVE_FORMAT_PCM, les données audio se composent d'échantillons représentés en format MIC. Pour les données audio MIC, le champ <format-specific-fields> est défini comme suit:

```
<PCM-format-specific> ->
    struct {
        WORD  nBitsPerSample;    // Longueur d'échantillon
    }
```

Le champ <nBitsPerSample> spécifie le nombre de bits de données utilisés pour représenter chaque échantillon dans chaque canal. S'il y a plusieurs canaux, la longueur d'échantillon est la même pour chaque canal.

Le champ <nBlockAlign> doit être égal à la formule suivante, après avoir été arrondi au plus proche entier:

nChannels × BytesPerSample

La valeur de BytesPerSample doit être calculée en arrondissant nBitsPerSample au plus proche octet entier supérieur. Lorsque le mot de l'échantillon audio est inférieur à un nombre entier d'octets, les bits de plus fort poids de l'échantillon audio sont placés dans les bits de plus fort poids du mot de données et les bits de données non utilisés adjacents au bit de poids le plus faible doivent être mis à zéro.

Pour les données MIC, le champ <nAvgBytesPerSec> du fragment <fmt> doit être égal à la formule suivante.

nSamplesPerSec × nBblockAlign

NOTE 1 – Conformément à la spécification initiale du format WAVE, il est possible, par exemple, de regrouper des échantillons à 20 bits provenant de 2 canaux en 5 octets qui se partageront un seul octet pour les bits de plus faible poids des deux canaux. La présente Recommandation spécifie un nombre entier d'octets par échantillon audio, afin de réduire toute ambiguïté dans les mises en œuvre et d'obtenir une compatibilité maximale dans l'échange de programmes.

2.1 Mise en paquet des données pour fichiers WAVE MIC

Dans un fichier WAVE monophonique, les échantillons sont enregistrés consécutivement. Pour les fichiers WAVE stéréophoniques, le canal 0 représente le canal de gauche et le canal 1 le canal de droite. Dans les fichiers WAVE multicanaux, les échantillons sont entrelacés.

Les schémas suivants montrent la mise en paquets des données pour des fichiers WAVE mono et stéréo à 8 bits:

Mise en paquet des données pour MIC mono 8 bits

Échantillon 1	Échantillon 2	Échantillon 3	Échantillon 4
Canal 0	Canal 0	Canal 0	Canal 0

Mise en paquet des données pour MIC stéréo 8 bits

Échantillon 1		Échantillon 2	
Canal 0 (gauche)	Canal 1 (droite)	Canal 0 (gauche)	Canal 1 (droite)

Les schémas suivants montrent la mise en paquet des données pour fichiers WAVE mono et stéréo à 16 bits:

Mise en paquet des données pour MIC mono à 16 bits

Échantillon 1		Échantillon 2	
Canal 0 octet de poids faible	Canal 0 octet de poids fort	Canal 0 octet de poids faible	Canal 0 octet de poids fort

Mise en paquet des données pour MIC stéréo à 16 bits

Échantillon 1			
Canal 0 (gauche)	Canal 0 (gauche)	Canal 1 (droite)	Canal 1 (droite)
octet de poids faible	octet de poids fort	octet de poids faible	octet de poids fort

2.2 Format des données dans les échantillons

Chaque échantillon est contenu dans un nombre entier d'octets i dont la longueur est le plus petit nombre d'octets nécessaires. L'octet de plus faible poids est enregistré le premier. Les bits qui représentent l'amplitude d'échantillonnage sont enregistrés aux positions binaires de plus fort poids du nombre i, les autres bits étant mis à zéro.

Par exemple, si la longueur d'échantillon (indiquée par le champ <nBitsPerSample>) est de 12 bits, chaque échantillon est codé sur un entier de deux octets. Les quatre bits de plus faible poids du premier octet (de plus faible poids) sont mis à zéro. Le format des données et les valeurs extrêmes de diverses longueurs d'échantillons audio MIC se présentent comme suit:

Longueur d'échantillon	Format des données	Valeur maximale	Valeur minimale
Un à huit bits	Entier non signé	255 (0xFF)	0
Au moins neuf bits	Entier signé i	Plus grande valeur positive de i	Plus grande valeur négative de i

Par exemple, les valeurs maximale, minimale et médiane des données audio MIC codées sur 8 bits et sur 16 bits sont les suivantes:

Format	Valeur maximale	Valeur minimale	Valeur médiane
MIC 8 bits	255 (0xFF)	0	128 (0x80)
MIC 16 bits	32 767 (0x7FFF)	–32 768 (–0x8000)	0

2.3 Exemples de fichiers WAVE MIC

Exemple de fichier WAVE MIC avec une fréquence d'échantillonnage de 11,025 kHz, mono, 8 bits par échantillon:

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

Exemple de fichier WAVE MIC avec une fréquence d'échantillonnage de 22,05 kHz, stéréo, 8 bits par échantillon:

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

2.4 Enregistrement des données WAVE

Le champ <**wave-data**> contient les données WAVE. Il est défini comme suit:

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

2.5 Fragment factuel

Le fragment factuel <fact-ck> contient des informations importantes sur le contenu du fichier WAVE. Ce fragment est défini comme suit:

```
<fact-ck> ->   fact(<dwFileSize:DWORD>)   // Nombre d'échantillons
```

Ce fragment n'est pas requis pour les fichiers MIC.

Le fragment factuel sera élargi pour inclure toute autre information requise par de futurs formats WAVE. Les champs ajoutés seront placés après le champ <dwFileSize>. Les applications pourront utiliser le champ de dimension de fragment pour déterminer quels champs sont présents.

2.6 Autres fragments facultatifs

Un certain nombre d'autres fragments sont spécifiés pour usage dans le format WAVE. Les détails de ces fragments sont donnés dans la spécification du format WAVE et dans ses mises à jour ultérieures.

NOTE 1 – Le format WAVE peut prendre en charge d'autres fragments facultatifs qui peuvent être inclus dans des fichiers WAVE pour transporter des informations spécifiques. Ces fragments sont considérés comme étant privés et seront ignorés par les applications qui ne peuvent pas les interpréter.

3 Extension du format WAVE

La structure étendue du format WAVE, ajoutée au fragment <fmt-ck>, est utilisée pour définir toutes les données audio de format autre que MIC. Elle est décrite comme suit. La structure étendue du format WAVE général est utilisée pour tous les formats autres que MIC.

```
typedef struct waveformat_extended_tag {
    WORD  wFormatTag;      // type de format
    WORD  nChannels;       // nombre de canaux (c'est-à-dire mono, stéréo...)
    DWORD nSamplesPerSec;  // fréquence d'échantillonnage
    DWORD nAvgBytesPerSec; // estimation du tampon
    WORD  nBlockAlign;     // longueur de bloc de données
    WORD  wBitsPerSample;  // nombre de bits par échantillon de données mono
    WORD  cbSize;          // comptage d'octets de la forme étendue
} WAVEFORMATEX;
```

Champ	Description
wFormatTag	Type de fichier WAVE.
nChannels	Nombre de canaux représentés dans les données audio: 1 pour la monophonie et 2 pour la stéréophonie.
nSamplesPerSec	Fréquence d'échantillonnage du fichier WAVE. La valeur de ce champ doit être 48 000 ou 44 100, etc. Cette fréquence est également utilisée par le champ de longueur d'échantillon contenu dans le fragment factuel afin de déterminer la durée des données.
nAvgBytesPerSec	Débit moyen. Le logiciel de reproduction peut estimer la capacité tampon nécessaire en utilisant la valeur <nAvgBytesPerSec>.
nBlockAlign	Alignement de bloc (en octets) des données dans le fragment <data-ck>. Le logiciel de reproduction a besoin de traiter de multiples octets de données <nBlockAlign> à la fois, de sorte que la valeur de ce champ peut être utilisée pour l'alignement du tampon.

Champ	Description
wBitsPerSample	Nombre de bits par échantillon et par canal. Chaque canal est censé avoir la même résolution de codage des échantillons. Si ce champ n'est pas nécessaire, il doit être mis à zéro.
cbSize	Longueur en octets des informations supplémentaires contenues dans l'en-tête de format WAVE, à l'exclusion de la longueur de la structure d'extension WAVEFORMATEX.

NOTE – Les champs qui suivent le champ <cbSize> contiennent des informations spécifiques nécessaires pour le format défini dans le champ <wFormatTag>.

Annexe 3
(normative)

Définitions des types de données de base

On trouvera ci-après les étiquettes atomiques, qui sont des étiquettes renvoyant à des types de données de base. Le type de données équivalent en langage C est également indiqué.

Étiquette	Signification	Type en langage C
<CHAR>	Entier signé de 8 bits	Caractère avec signe
<BYTE>	Entier non signé de 8 bits	Caractère sans signe
<INT>	Entier signé de 16 bits commençant par le bit de faible poids	Entier signé
<WORD>	Quantité non signée de 16 bits commençant par le bit de faible poids	Entier non signé
<LONG>	Entier signé de 32 bits commençant par le bit de faible poids	Longueur avec signe
<DWORD>	Quantité non signée de 32 bits commençant par le bit de faible poids	Longueur sans signe
<FLOAT>	Nombre de 32 bits à virgule flottante IEEE	Flottant
<DOUBLE>	Nombre de 64 bits à virgule flottante IEEE	Double
<STR>	Chaîne (séquence de caractères)	
<ZSTR>	Chaîne se terminant par ZERO	
<BSTR>	Chaîne avec un préfixe d'un octet (8 bits)	
<WSTR>	Chaîne avec un préfixe d'un mot (16 bits)	
<BZSTR>	Chaîne se terminant par ZERO avec un préfixe d'un octet	

Annexe 4
(informative)

Modifications apportées aux spécifications de l'Annexe 1

1 Modifications apportées entre les Recommandations UIT-R BS.2088-0 et BS.2088-1

Dans la révision 1 de la présente Recommandation, les modifications suivantes sont apportées aux spécifications de l'Annexe 1:

– Adjonction du fragment BXML au paragraphe 6.

– Adjonction du fragment SXML au paragraphe 7.

– Adjonction d'une nouvelle fonction en cas d'omission des éléments audioTrackFormat et audioStreamFormat au paragraphe 8.

2 Modifications apportées entre les Recommandations UIT-R BS.2088-1 et BS.2088-2

La révision 2 de la présente Recommandation vise à apporter des précisions sur le traitement des fragments utilisés dans d'autres formats de fichier WAVE décrits dans l'Annexe 1:

– Clarification du traitement des fragments aux paragraphes 2.1, 2.2 et 10.

– Ajout d'une méthode de génération de code XML à partir du fragment <ubxt> au paraphe 11.