# XMF Specification

**Incorporating all Recommended Practices**

*Document version 1.2*
*November 1, 2007*

*Specification for XMF Meta File Format*
*Type O & Type 1 XMF Files (SMF + DLS)*
*Description of SMF Meta Event for XMF Patch Type Prefix*
*XMF Meta File Format Updates 1.01*
*XMF Compression Definition for "zlib"*
*XMF Meta File Format Version 2.0*
*Type 2 XMF Files (Mobile XMF)*
*Audio Clips for Mobile XMF*
*ID3 Metadata for XMF Files*

**Published by:**
The MIDI Manufacturers Association
Los Angeles, CA

This document is a combination of the eXtensible Music File (XMF) “Recommended Practices” adopted by AMEI/MMA from 2001 through 2007:

**RP-030** Specification for XMF Meta File Format 1.0
**RP-031** Type 0 & Type 1 XMF Files (DLS + SMF)
**RP-032** SMF Meta Event for XMF Patch Type Prefix
**RP-039** XMF Meta File Format Updates 1.01
**RP-040** XMF Compressions Definition for “zlib”
**RP-042a** Type 2 XMF File (Mobile XMF)
**RP-043** Specification for XMF Meta File Format 2.0
**RP-045** Audio Clips for Mobile XMF
**RP-047** ID3 Metadata for XMF Files

Printed 2007

MMA
POB 3173
La Habra CA 90632-3173

# Specification for XMF Meta File Format

**Version 1.00b**

**Revised 10/10/2001**

**Published by:**
The MIDI Manufacturers Association
Los Angeles, CA

## PREFACE

XMF stands for **eXtensible Music Format**.

XMF is a low-overhead meta-file format for bundling collections of data resources (i.e. file images) in one or more formats into a single file. XMF brings stability, structure, ease of handling, transportability, optional data compression, optional data security, and a consistent meta-data framework to resource collections.

Separate Recommended Practice documents define the rules for various Types of XMF files, each intended to address a particular set of purposes. For example, RP 031 defines Type 0 and Type 1 XMF Files, which allow for the bundling of Standard MIDI File (SMF) and Downloadable Sounds (DLS) file images. Additional XMF File Types can be added via the Recommended Practice process at any time, without requiring any changes in the underlying XMF meta-file format.

Printed 2001

MMA
PO Box 3173
La Habra CA 90632-3173

# Table of Contents

# 0. Definition of Terms

**extended ASCII** - A text character encoding standard based on the American Standard Code for Information Interchange, as used in other MMA specifications.

**DTD** - Document Type Definition. In XML, a document in XML format which defines a set of XML elements for use in other XML documents.

**GUID** - Globally Unique Identifier, a 16-byte integer uniquely identifying an item. See Chapter 10, "DEC/HP Network Computing Architecture Remote Procedure Call RunTime Extensions Specification Version OSF TX1.0.11", Steven Miller, July 23 1992. (Same definition used in DLS <DLSID> chunk.)

**parser** - Program or device for reading a data format.

**player** - Program or device for playing XMF files.

**RDF** - Resource Description Framework, an XML-based metadata format standard defined by the World Wide Web Consortium (w3c). See http://www.w3c.org/RDF.

**resource** - Block of formatted data corresponding to a file. For example: SMF, DLS, WAV, text, JPG, or manufacturer's non-standard format.

**RMID** - RIFF MIDI file, originally defined in Microsoft(R) Windows(R) Multimedia Programmers Reference, Microsoft Press, 1991, p. 8-31 (now out of print).

**Unicode** - A text character encoding standard that includes all major world scripts in a simple and consistent manner. See http://www.unicode.org.

**URI** - Uniform Resource Indicator, the formal term for a family of names/addresses that refer to resources, in the form of short text strings. 'URL' is an informal term, and is a special case of URI that includes more popular URI schemes such as http:, ftp:, mailto:, etc. See http://www.w3c.org/Addressing.

**VLQ** - Variable Length Quantity, a binary number format with a variable number of bytes. See section 4.1.

**XString** - eXtensible String, a text string format with a leading length integer in VLQ form and no terminating character. See section 4.2.

# 1. Introduction

XMF stands for **eXtensible Music Format**.

XMF is a low-overhead meta-file format for bundling collections of data resources (i.e. file images) in one or more formats into a single file. XMF brings stability, structure, ease of handling, transportability, optional data compression, optional data security, and a consistent meta-data framework to resource collections.

Separate MMA Recommended Practice documents define the rules for various Types of XMF files, each intended to address a particular set of purposes. For example, a separate Recommended Practice document defines Type 0 and Type 1 XMF Files, which allow for the bundling of Standard MIDI File (SMF) and Downloadable Sounds (DLS) file images. Additional XMF File Types can be added via the MMA Recommended Practice process at any time, without requiring any changes in the underlying XMF meta-file format.

**Bundling –**XMF is primarily intended to bundle existing standard music and sound file formats – such as SMF, DLS, and WAV – and not to replace any of them. It is assumed that XMF implementors will have access to existing playback APIs or devices able to render the formats contained in an XMF collection, and will pass the contained resources off to them for rendering, in accordance with Recommended Practice guidelines. An XMF collection may be either a flat list of resources, or hierarchically structured to any depth.

**XMF File Types –** Like XML, the XMF meta-file format is a generalized structure that can be used for many purposes, but is not a directly usable format. Only specific XMF File Types, as defined in separate MMA Recommended Practice documents, can be rendered. In XML terms, an XMF File Type specification would be analogous to a DTD, defining how the container will be used for one purpose or application area.

**Extensibility –** XMF is eXtensible by the content creator and application developer, in four ways:

- Custom resource types can be stored in any XMF file
- Custom meta-data fields are supported, and can be displayed to the end user
- Custom data compression algorithms can be used
- Custom security algorithms can be used

However, the use of custom resource types, compression algorithms, or security algorithms will in most cases diminish file portability. XMF readers are required to ignore all unrecognized resource types, **UnpackerID**s, and **MetaDataItems**, and each XMF File Type Recommended Practice specifies player capabilities and behaviors, so there is no danger of misinterpretation.

**Hierarchically Structured Collections –** XMF uses a hierarchical containment paradigm, expressed in a new low-overhead tree data structure (not RIFF) with simple and consistent parsing rules.

**Scaling Data Structure –** Minimized file size is essential if XMF is to be usable on all playback platforms and networks, including small mobile devices. To that end, the container data structure is scalable. Lightweight and breakaway data structures minimize overhead, and smaller resource collections need fewer bytes to wrap. To conserve space and transmission bandwidth, chunk boundaries are not constrained to word, double-word, or other processor-specific boundaries. Every data item is either of known size or is length-delimited, so it's never necessary to parse through the internals of any item just to move ahead to the next item. There is no upper limit on file size, resource size, or number of resources.

**A Tree of Nodes –** The container hierarchy is expressed as a tree of nodes, starting at the top of the file with a single root node. In the simplest case the **RootNode** holds one single resource – for example, one SMF file. More often the **RootNode** will be a folder containing further nodes – resource nodes and/or further folder nodes, nested to arbitrary depth. In some situations literally hierarchical resource layout will be desirable, and in other situations the hierarchical tree is best used as a small index holding pointers to large resource blocks appearing elsewhere. Both options are available.

**External Resource References –** To facilitate both the sharing of resources among multiple XMF collections and the dynamic publishing of resources on the Internet, any node in the tree can optionally reference a resource in another file, or at an Internet http: address (URI), rather than a data block inside the XMF file.

**Meta-Data –** Each node in the tree (both files and folders) may optionally have any number of independent meta-data items. Each meta-data item can be either a Standard item or a Custom (i.e. application-specific or end-user-defined) item, and the item's contents can be of any length or data type including binary data. Any meta-data item may include multiple content variations, keyed to the playback device's preferred language and country. Meta-data is implemented as a field in the **NodeHeader**; when no meta-data is needed, this field collapses to exactly 1 byte per resource or folder.

**Unpackers –** Each node in the tree (resource or folder) may optionally be individually secured, data-compressed, or otherwise processed with any number of encoders. Each required decoding operation is indicated by a Unpacker selector and the size of the decoded result (for use in memory/disk allocation). Each node can have an independent list of **UnpackerIDs** indicating the sequence of decoding operations required to recover 'clear' playable data (for example: decryption, followed by data decompression, followed by watermark check). These lists appear directly as a field in the **NodeHeader**; when no decoding is needed, the list collapses to exactly 1 byte per resource or folder.

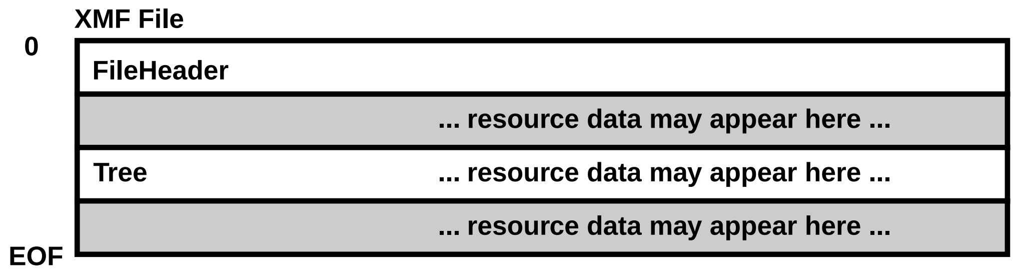
# 2. XMF File Structure

An XMF file begins with exactly one **FileHeader**, and includes exactly one **Tree**.

The **FileHeader** contains the minimum information needed to get a **Tree** parser started (see **2.1. FileHeader Structure**).

The **Tree** describes the resources (SMF file images, DLS file images, WAV file images, etc.) in the collection, expresses the collection's structure, and indicates where to find the resources. Resources may appear either inside the XMF file, or in other files (see **2.2. Tree Structure**).

Resources that appear inside the XMF file may be located either in the space between the **FileHeader** and the **Tree**, within the **Tree**, or in the space between the **Tree** and the end of the file.


XMF File
0
FileHeader
... resource data may appear here ...
Tree
... resource data may appear here ...
... resource data may appear here ...
EOF

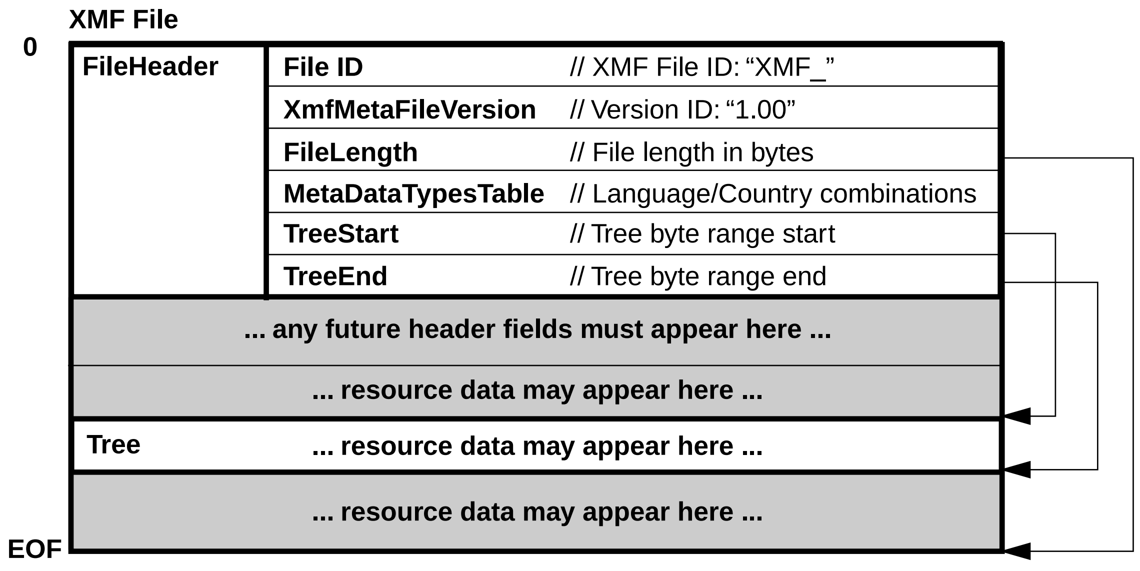
## 2.1. FileHeader Structure

Fields in the **FileHeader** indicate the meta-file spec version, the file's total length, the location of the **Tree**'s start and end, and contain global meta-data format tables.

- The **FileID** field identifies the file as an XMF file. **FileID** is always the 4-byte ASCII sequence: **`XMF_`**
- **XmfMetaFileVersion** is the version of the **XMF Meta-File Format Specification** (this document) to which the file conforms, expressed as a 4-byte ASCII sequence with a decimal point in the second position. This is version 1.00 of the specification, so the byte sequence should be: **`1.00`**
- **FileLength** is the total length of the XMF file, in bytes, in **VLQ** format (see section 4.1). Note that the value contained in this field must reflect the length of this field.

  **Note for XMF file creators:** The value for this field will depend in part on the size of the VLQ used here. Because a change in value may force a change in VLQ size, determining the value and VLQ size may require iteration.
- **MetaDataTypesTable** contains shared information on the languages and countries for which **NodeMetaData** contents are provided, as detailed in the **3. XMF Meta-Data** section of this document.
- **TreeStart** is the offset from the start of the XMF file to the first byte of the **Tree**, in bytes, in **VLQ** format (see section 4.1).
- **TreeEnd** is the offset from the start of the XMF file to the last byte of the **Tree**, in bytes, in **VLQ** format (see section 4.1).

  **Note:** The **FileLength**, **TreeStart**, and **TreeEnd** fields should **not** be used to create space between the **FileHeader** and **Tree**, or between the **Tree** and the end of the file, for storing hidden data. XMF parsers will not find that data, so XMF tools will strip that data when editing an XMF file, and XMF players will not be able to access that data.



## 2.2. Tree Structure

The **Tree** describes the resources in the collection (SMF file images, DLS file images, WAV file images, etc.), expresses the collection's hierarchical structure, and indicates where to find the resources. Resources may be stored directly within the tree, elsewhere in the same XMF file, or in external files, as detailed below under **Reference Types**.

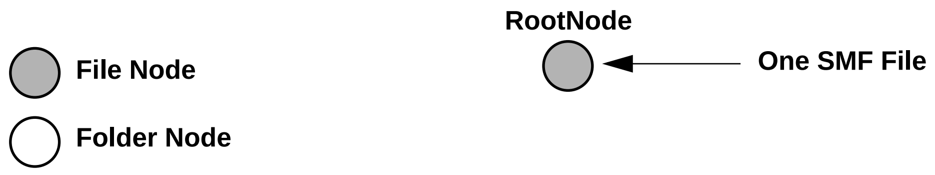
The **Tree** is built using two kinds of **Node**:

- **FolderNodes** – each of which can contain one or more additional nodes, including further **FolderNodes**, and
- **FileNodes** – each of which describes exactly one resource (SMF, DLS, WAV, etc.) and indicates where to find it.

The **Tree** always begins with one logical root, a node called **RootNode**.

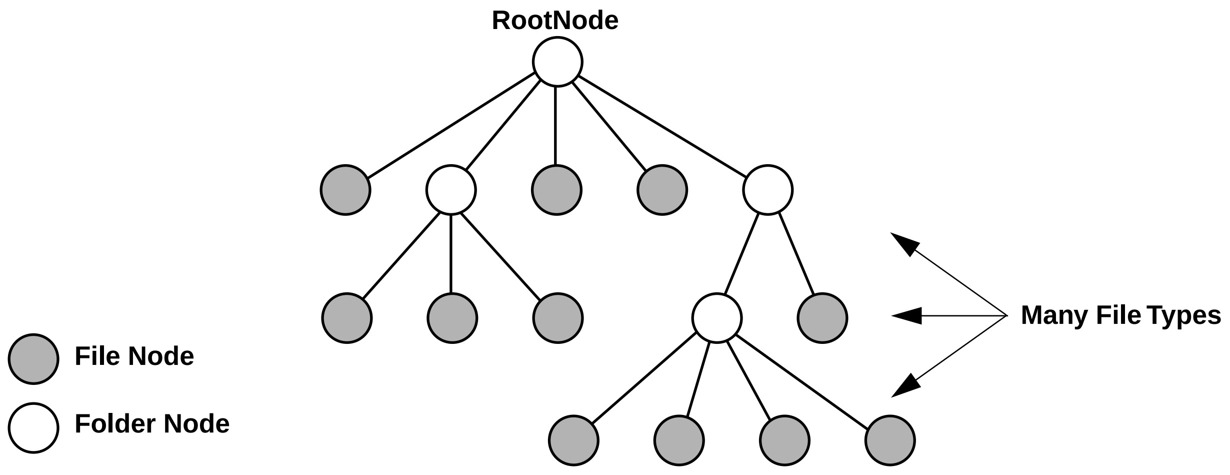
In simple XMF files, the **RootNode** will be a **FileNode** holding exactly one resource.

**Simplest Tree Hierarchy:
Single Node**



More often it'll be a **FolderNode** holding several further **FileNode**s, and perhaps additional **FolderNode**s (which may be hierarchically nested to any depth).

**Typical Tree Hierarchy:
Many Nodes**

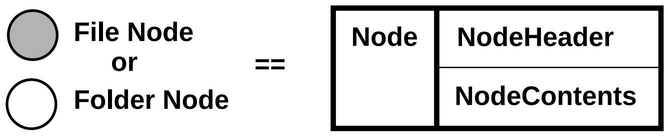


### 2.2.1. Node Structure

Every **Node** in the **Tree** consists of two parts:

- **NodeHeader** and
- **NodeContents**.

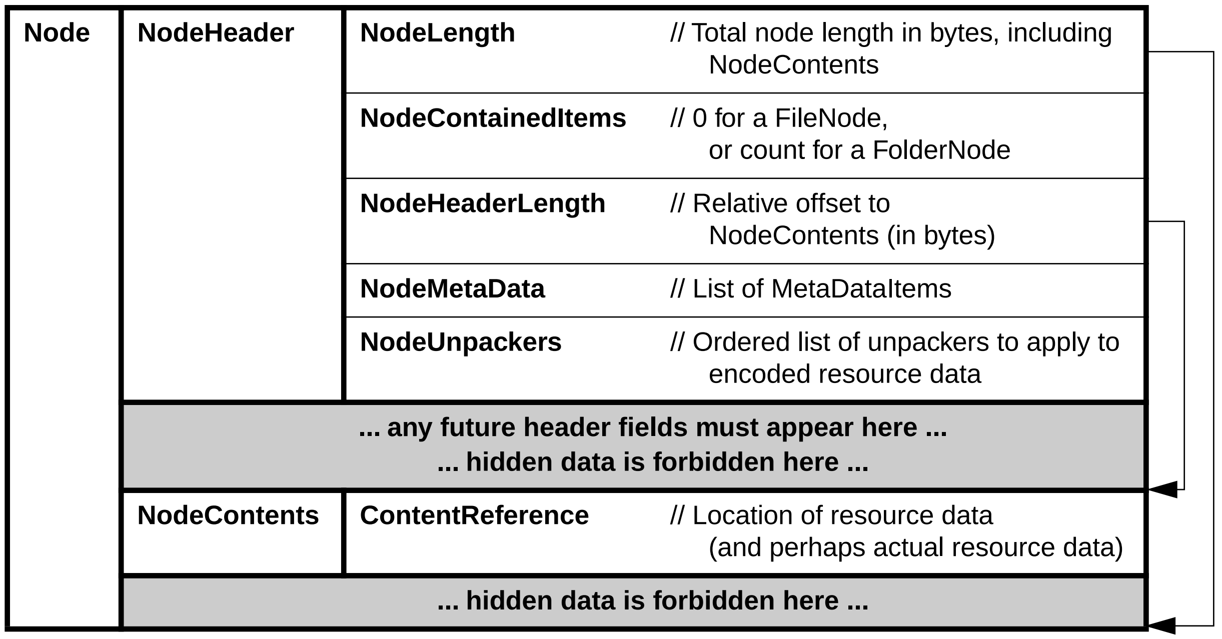
This is true for both **FileNode**s and **FolderNode**s, including the **RootNode**.



### 2.2.1.1. NodeHeader Structure

The **NodeHeader** fields implement the **Tree** structure, and hold information about the resource data blocks that the **Node** represents.

> **Note:** Unlike many resource file formats, in most cases XMF does not require **Nodes** to have names or resource ID numbers. Where needed or desired, this can be achieved via standard meta-data fields.



| | | | |
|---|---|---|---|
| **Node** | **NodeHeader** | **NodeLength** | // Total node length in bytes, including NodeContents |
| | | **NodeContainedItems** | // 0 for a FileNode, or count for a FolderNode |
| | | **NodeHeaderLength** | // Relative offset to NodeContents (in bytes) |
| | | **NodeMetaData** | // List of MetaDataItems |
| | | **NodeUnpackers** | // Ordered list of unpackers to apply to encoded resource data |
| | **... any future header fields must appear here ... ... hidden data is forbidden here ...** | | |
| | **NodeContents** | **ContentReference** | // Location of resource data (and perhaps actual resource data) |
| | **... hidden data is forbidden here ...** | | |

- **NodeLength** allows the parser to move ahead to the next **Node** at the same level of the **Tree**, including skipping over any **Node** it doesn't recognize or doesn't care about. **VLQ** format (see section 4.1). Note that the value contained in this field must reflect the length of this field.

  **Note for XMF file creators:** The value for this field will depend in part on the size of the VLQ used here. Because a change in value may force a change in VLQ size, determining the value and VLQ size may require iteration.

  **Note:** This field should not be used to create space to store hidden data in the gap between the **NodeHeader** and the **NodeContents**. XMF parsers will not find that data, so XMF tools will strip that data when editing an XMF file, and XMF players will not be able to access that data.

- **NodeContainedItems** indicates whether the **Node** is a **FileNode** (resource) or **FolderNode** (container for further **Nodes**). For **FileNode**s, **NodeContainedItems** is 0. For **FolderNodes**, **NodeContainedItems** is the number of directly contained **Nodes** (non-recursive count, i.e. not counting any **Nodes** contained in child **FolderNodes**). **VLQ** format (see section 4.1). Since 0 is used to indicate a **FileNode**, empty **FolderNodes** are not allowed in XMF.

- **NodeHeaderLength** is the relative offset from the start of the **NodeHeader** to the start of the **NodeContents**. This is a hedge to preserve backwards compatibility in future, as it will allow older parsers to skip over any new **NodeHeader** fields that may be added to later versions of the XMF meta-file format specification. **VLQ** format (see section 4.1).

  **Note:** As a result, all future versions of XMF will have to support the whole initial field set (**NodeLength**, **NodeContainedItems**, **NodeHeaderLength**, **NodeMetaData**, and **NodeUnpackers**), and any additions will have to appear after **NodeUnpackers**.

  **Note:** This field should **not** be used to create space to store hidden data in the gap between the **NodeHeader** and the **NodeContents**. XMF parsers will not find that data, so XMF tools will strip that data when editing an XMF file, and XMF players will not be able to access that data.

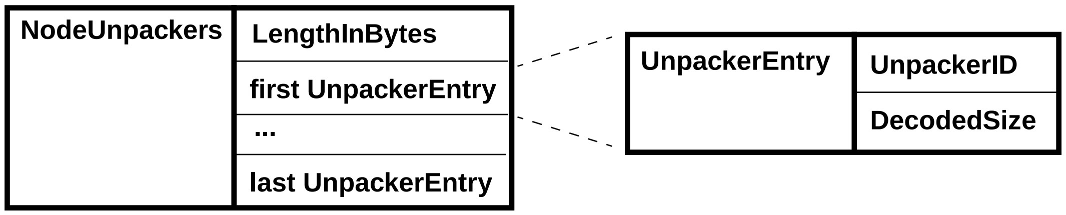
- **NodeMetaData** contains zero or more **MetaDataItems** pertaining to the resource data referenced in the **NodeContents**, as detailed in section **3. XMF Meta-Data**.
- **NodeUnpackers** is an ordered, length-delimited list specifying a sequence of zero or more decoding operations that must be applied to the resource data referenced in the **NodeContents** in order to recover clear, usable (playable) data. Unpackers will typically be used for data compression and intellectual property protection. See section 2.2.1.1.1. for details.

#### 2.2.1.1.1. NodeUnpackers Structure

**NodeUnpackers** is an ordered, length-delimited list specifying a sequence of zero or more decoding operations that must be applied to the resource data referenced in the **NodeContents** in order to recover clear, usable (i.e. playable) data. Unpackers will typically be used for data compression and intellectual property protection. **LengthInBytes** is a **VLQ.**

Each decoding operation is expressed as one **UnpackerEntry**, consisting of an **UnpackerID** (see section 5.1) and the size in bytes of the resulting decoded data block. This tells the XMF parser what size buffer to allocate for the decoding operation's output buffer. For format, interpretation, and registration of **UnpackerID**s, see section 5.1. **DecodedSize** is a **VLQ** (see section 4.1).

If **DecodedSize** is 0, determination of the decode buffer size is left to the player. This is intended to support differences in player capabilities, including variable bitrate codecs and variable levels of quality.

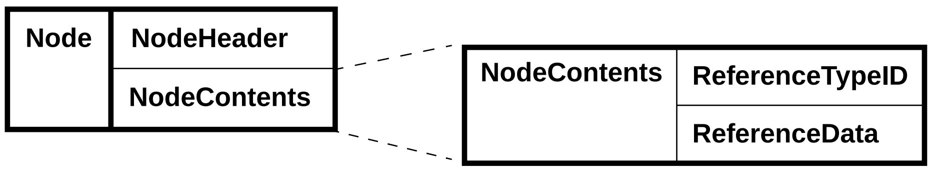


Data compression which is native to the resource file format (such as IMA 4:1 in WAV files) should not be reflected in **NodeUnpackers**, as that processing is handled outside of the XMF parser's scope, in the type-specific playback APIs and devices.

> **Note: FolderNodes** appearing in the **Tree** must not use any unpackers that would render the folder's contained **Nodes** illegible. This rule arises because otherwise we'd need to invent a way to describe how the **Tree** continues on the inside of that 'black box'. If you wish to 'black box' encode a folder, however, it is possible to store it outside the **Tree**, as a detached **Node** (see section 2.2.1.2.1), and point the **FolderNode**'s **ContentReference** at the detached **Node**.

### 2.2.1.2. NodeContents Structure

The **NodeContents** part of a **Node** consists is a structure indicating where to find the described resource. The resource may appear directly within the **NodeContents** structure, elsewhere within the current file, as an external file, or as a resource in another XMF file's collection, as described below.



Every **NodeContents** structure consists of two parts:

- A **ReferenceTypeID** in **VLQ** form indicating an access mechanism. Any **Node** may use any **Reference-TypeID**; this is true for both **FileNode**s and **FolderNode**.
- A **ReferenceData** field with the data necessary to support that **ReferenceTypeID**. The format of the **Refer-enceData** depends on the value of the **ReferenceTypeID**, as section 2.2.1.2.1. shows.

### 2.2.1.2.1. Reference Types

Each **ReferenceTypeID** represents one unique reference type, or way of finding the resource data that the **Node** describes. **ReferenceTypeID**s may only be defined and assigned by the MMA.

Note that the same collection of resources can be stored in many different ways, depending on which reference types are used for each **Node** – even though the **Tree**'s logical structure remains the same for all such variations – so decisions about **ReferenceType** usage will dramatically affect the overall layout of an XMF file.

See also: **Appendix: Notes on the XMF Meta-File Format**

| ReferenceTypeID | | ReferenceData | |
|---|---|---|---|
| **ID** | **Description** | **Description** | **Format** |
| **1** | **In-Line Resource** | The resource data block | (the actual encoded resource data block) |
| **2** | **In-File Resource** | Offset in bytes from the start of the XMF file to the resource data block | **VLQ** |
| **3** | **In-File Node** | Offset in bytes from the start of the XMF file to a Node whose **ContentReference** leads to the resource data block | **VLQ** |
| **4** | **External Resource File** | URI* of an external resource file** (SMF, WAV, DLS, etc.), e.g.: **file://myOtherFile.wav** | **XString** |
| **5** | **External XMF Resource** | URI* of an external XMF file**, including specific resource name, e.g.: **file://myOtherXmfFile.xmf#myResource** | **XString** |

* The URI stored in the ReferenceData must be coded according to RFC 2396 (US-ASCII with restricted character set). To access characters which cannot be represented in this coding, use the %HH convention to represent the byte values; coding standards for non-European characters in URIs are defined at http://www.w3.org/TR/charmod/.

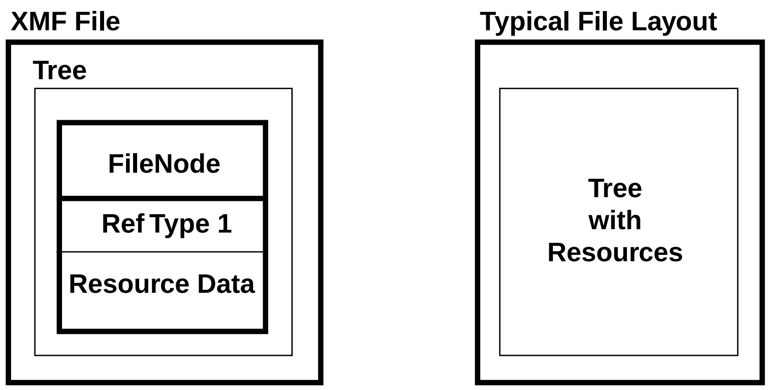
** If the XMF player is displaying a copyright notice for the XMF file, and detects that the external file has a different copyright notice, then the player must display both copyright notices. If the XMF player detects that the external file contains digital rights management (DRM) information, then the XMF player must not use the external file in any way that would violate the DRM usage restrictions.

Note: This RP does not specify the use of any particular DRM technology. Future RPs may define DRM systems for XMF, perhaps using the Unpackers mechanism. Developers interested in preserving interoperability are advised to check the status of this work before introducing their own DRM technologies into XMF.

Each **ReferenceTypeID** is detailed below.

### ReferenceTypeID 1: In-Line Resource

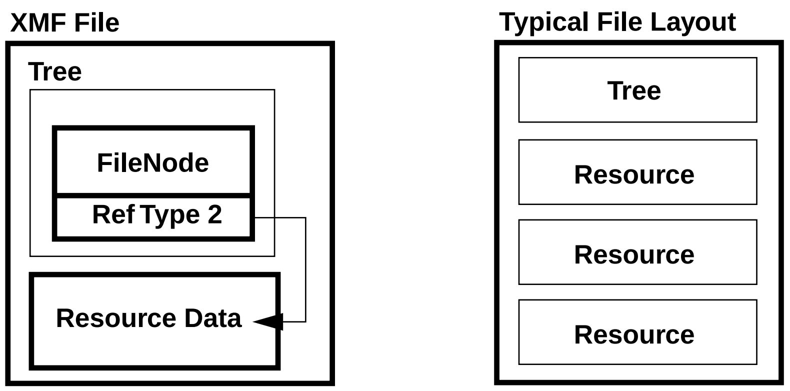
**ReferenceTypeID 1** means the resource data block appears in-line, immediately after the **ReferenceTypeID**, and encoded per the **NodeUnpackers**. In other words, the resource data is stored directly in the **Tree**. Type 1 may be appropriate for small and/or simple files, but is not recommended for large or complicated files as it works against the intended use of the **Tree** as an access accelerator. **FolderNode**s should generally use **ReferenceTypeID** 1, as should XMF files containing only one resource.



**Typical Use Cases:** Simple XMF files, or when an author wishes to pack (encrypt, data-compress, etc.) an entire folder of resources in a single operation.

### ReferenceTypeID 2: In-File Resource

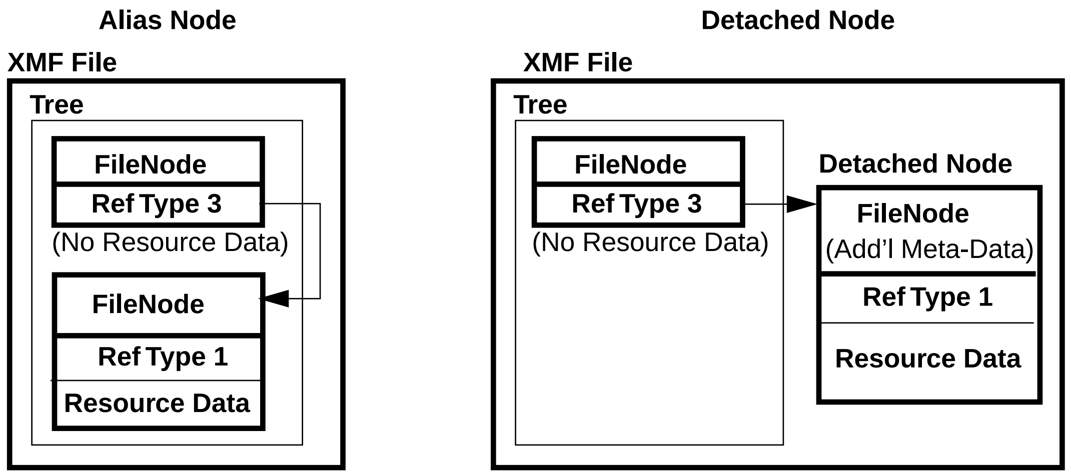
**ReferenceTypeID 2** points to a resource data block appearing elsewhere in the same XMF file, and encoded per **NodeUnpackers**. Typically the target block will be outside the **Tree**; however this can also be used for an alias to a resource appearing in another **Node** in the **Tree**. **ReferenceTypeID** 2 is recommended when the number of resources or the complexity of the **Tree** is high, as it keeps the bulky resources out of the **Tree**, making it easier to load the entire **Tree** in one seek operation.

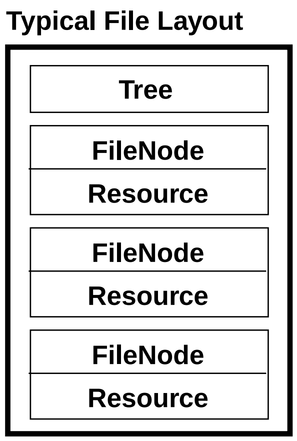


**Typical Use Cases:** Large resource collections in which in-line resource placement would make Tree navigation cumbersome. **ReferenceTypeID** 2 may also be simpler for XMF generator programs to assemble.

## ReferenceTypeID 3: In-File Node

**ReferenceTypeID 3** points to a further **Node** within the same file. If the target **Node** is also in the main **Tree**, this becomes an alias to another resource. More typically, the target **Node** will be detached from the **Tree**, in order to store additional meta-data in the detached node, without clogging the main **Tree** – again, with the aim of optimizing the **Tree**.





**Typical Use Cases:** Resources needing large amounts of meta-data, or situations in which content developers wish to use a single resource for multiple purposes (such as stand-ins, temporary placeholders, or multiple artists sharing one resource – such as DLS – in their work).

> **Note:** In **Node**s using **ReferenceTypeID** 3, the **NodeUnpacker**s in the referenced **Node** override any **NodeUnpacker**s in the referring node. Any **NodeUnpacker**s indicated in the referring node should be ignored.
>
> **Note:** The target node will not necessarily hold the target resource directly, as it has a **ContentReference** of its own. However, to avoid excessively long or circular seek chains, the target resource data must be found within 4 reference indirections, otherwise the XMF parser must fail and return a 'Too many reference indirec-tions' error.

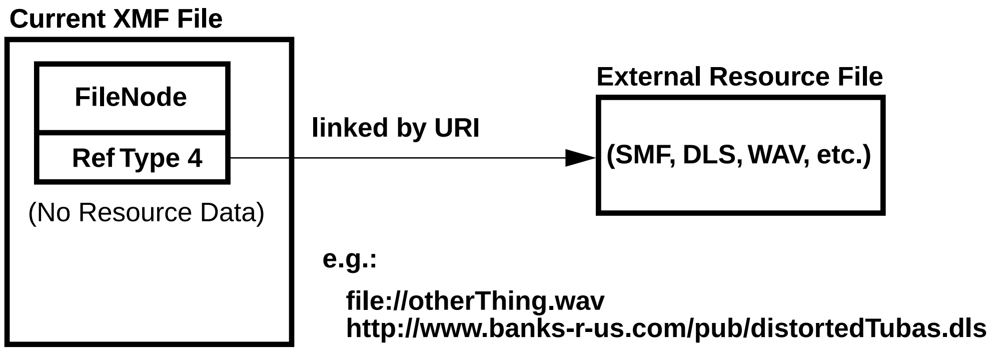
## ReferenceTypeID 4: External Resource File

**ReferenceTypeID 4** points to external whole resource files (such as SMF, WAV, or DLS files), using the W3C's Uniform Resource Identifier (URI) format. (URI is the formal superset of URL.) URI Schemes **file:** and **http:** should be supported if the playback platform is capable of them; additional schemes may be supported in a future version of this specification. If the external file is encoded with XMF encoders, **NodeUnpackers** should reflect that fact. **ReferenceTypeID** 4 can allow for very small XMF files, can allow external files to be shared by multiple XMF files, can allow rapidly changing resources to be decoupled from the rest of the XMF collection, and can layer uniform meta-data onto resources that don't support it directly.

The format for URIs is defined and described at the following URLs:

**http://www.ietf.org/rfc/rfc2396.txt**

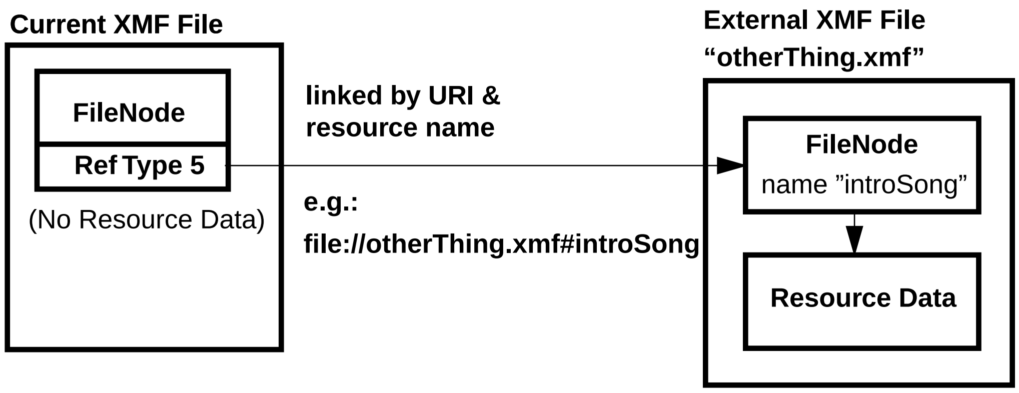
**http://www.w3.org/Addressing/**



**Typical Use Cases:** This reference type has all the same resource sharing and aliasing benefits as **ReferenceTypeID** 3, with the additional flexibility of allowing the target resource to live outside of the XMF file. This creates many new possibilities because the target file can be updated after the XMF file is distributed. For example, newscasts: a single local XMF file on a computer could link to an audio file on a server that is updated hourly; every time the XMF file is opened, the latest news plays. For interactive content development groups, **ReferenceTypeID** 4's file-level indirection may make management of large media asset collection more flexible; assets subject to frequent change can be kept as external files for easier updating without disturbing the main XMF asset collection.

## ReferenceTypeID 5: External XMF Resource

**ReferenceTypeID 5** points to resources stored in (or referenced by) external XMF files, also via URI (same schemes), but with the resource name indicated using the URI convention (append the filename with a hash [**#**] followed by the desired Node Name). For a type 5 reference to succeed, the target resource must include a Node Name field in its meta-data, and that name must match the referring name. **ReferenceTypeID** 5 has all the advantages of **ReferenceTypeID** 4, but allows the target resource to reside within an XMF file – and as long as the names agree, the link will survive all changes to the target file's structure and other contents.



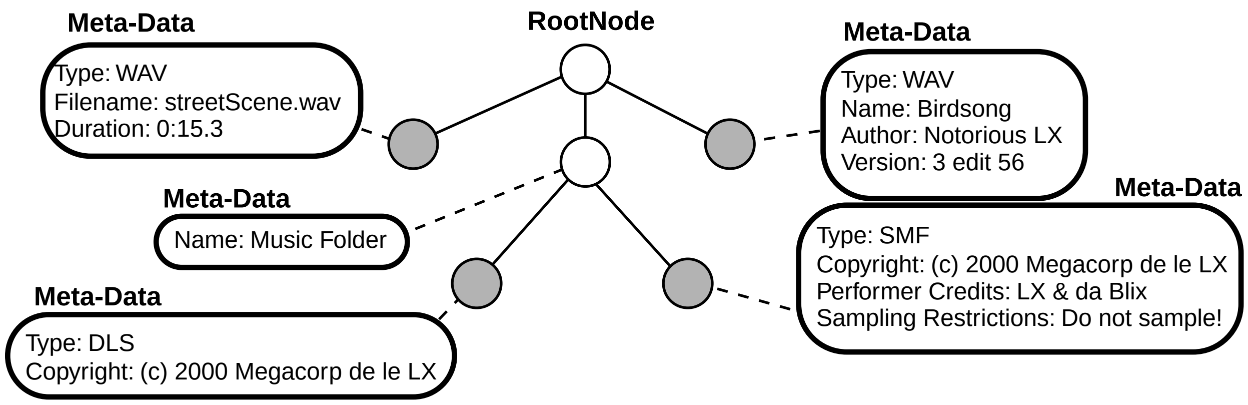
**Typical Use Cases:** This reference type has all the same resource sharing and aliasing benefits as type 4, with the additional flexibility of allowing the target resource to live inside another XMF file.

> **Note:** In **Node**s using **ReferenceTypeID** 5, the **NodeUnpacker**s in the referenced **Node** override any **NodeUnpacker**s in the referring node. Any **NodeUnpacker**s indicated in the referring node should be ignored.
>
> **Note:** The target node will not necessarily hold the target resource directly, as it has a **ContentReference** of its own. However, to avoid excessively long or circular seek chains, the target resource data must be found within 4 reference indirections, otherwise the XMF parser must fail and return a 'Too many XMF indirections' error.

# 3. XMF Meta-Data

Meta-data is ancillary informational or logistical data related to a resource or folder in an XMF file, essentially forming a flexible 'property sheet' for the described item. As an abstract example, a copyright notice might be one **MetaDataItem**, and a performer credit would be a second **MetaDataItem**.

**Typical Meta-Data Use**



## 3.1. Meta-Data Semantics

XMF can accommodate any number of **MetaDataItem**s for any resource or folder in the collection. Meta-data contents may be either text or binary data, depending on the requirements of the particular field.

The requirements of meta-data standardization, efficiency, and extensibility pull in different directions. To accommodate all three, and still optimize for the most frequently-occurring fields (such as copyright notices and title), XMF supports two categories of meta-data fields: a pre-defined set of Standard Fields for common use, and an open space for arbitrary Custom Fields that end users or applications can freely create and use for non-standard purposes (i.e. application-specific internal data, special user notes, etc.).

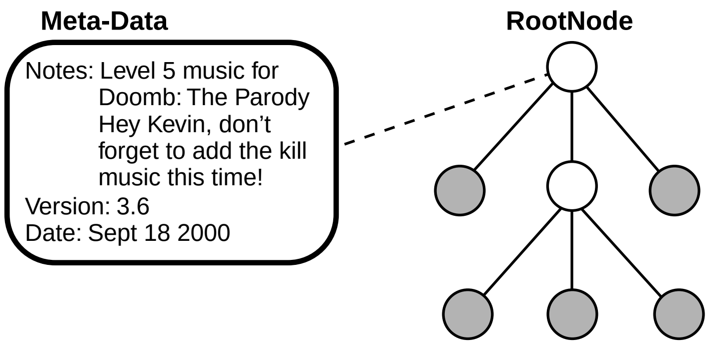
**Support for Internationally Distributed Content –** For each **MetaDataItem** present in a given **Node**, the content creator or distributor can optionally supply multiple alternate versions of the field contents, with each version keyed to one or more Language/Country combinations. This allows an XMF player that knows its geographic location and its user's natural language preference to select and display the appropriate meta-data content version. For example, album liner notes can be provided in English, Japanese Katakana, French, German, Italian, and Spanish, and only the appropriate version will be presented to the user. Text can be encoded in Extended ASCII, Unicode, or compressed Unicode, and can optionally be hidden from the file's end user. Binary meta-data can also be keyed to language and country. However, only the one best content version for each **MetaDataItem** should be presented to the user at any given time.

> **Note:** Future XMF Recommended Practices may specify an XML semantic equivalent to this extensible meta-data mechanism, perhaps using the W3C's Resource Description Format (RDF) – another system that permits multiple (and customized) meta-data etymologies to be used together.

### 3.1.1. Meta-Data Scope Rules

The scope of a **MetaDataItem** depends on where in the Tree it appears.

- Meta-data attached to the **RootNode** is considered to pertain to the XMF file as a whole.


Meta-Data
RootNode
Notes: Level 5 music for Doomb: The Parody Hey Kevin, don't forget to add the kill music this time!
Version: 3.6
Date: Sept 18 2000

- Meta-data attached to a **FolderNode** is in most cases considered to pertain to all the **FileNode**s and **FolderNode**s it contains, to any nesting depth. If multiple **MetaDataItems** of the same **FieldID** apply to a given **FileNode**, only the most local of those **MetaDataItems** should be used.
- Meta-data attached to a **FileNode** is considered to pertain to that resource only.

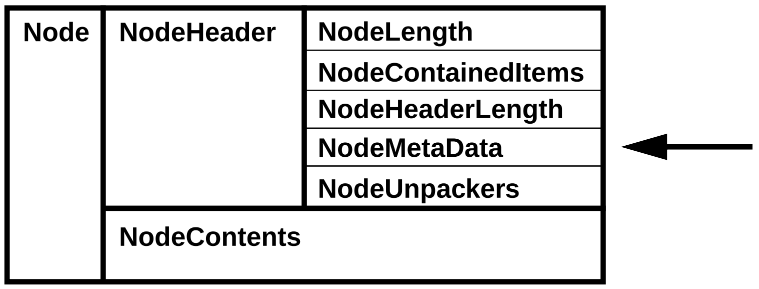
## 3.2. Meta-Data Structures

The meta-data system has two parts:

- Meta-Data information for specific **Nodes** appears in the **NodeMetaData** field of the **NodeHeader** structure, as a list of **MetaDataItem**s, and
- Shared information supporting the multi-country/multi-language meta-data feature appears in the **Meta-DataTypesTable** in the **FileHeader** structure.

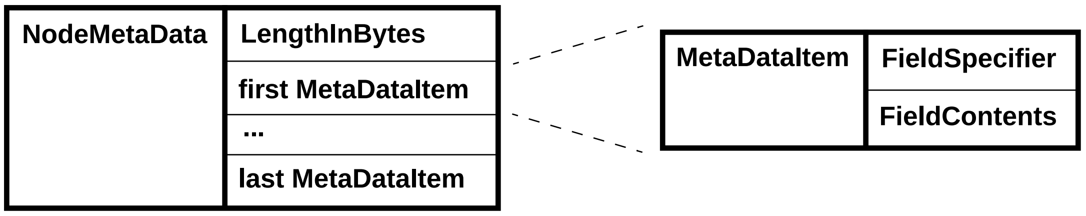
## 3.2.1. NodeMetaData Structure

Recall that meta-data for a **FileNode** or **FolderNode** is stored in the **NodeMetaData** field of the **NodeHeader** structure.



The **NodeMetaData** structure is a length-delimited list of **MetaDataItems**. It may contain any number of **Meta-DataItem**s. **LengthInBytes** is a **VLQ** (see section 4.1).

### 3.2.1.1. MetaDataItem Structures



Each **MetaDataItem** consists of two parts, which are further described in the following sections:

- One **FieldSpecifier** structure, and
- One **FieldContents** structure.

#### 3.2.1.1.1. FieldSpecifier Structure

in XMF, there are two kinds of meta-data fields:

- Standard fields, for common uses, and
- Custom fields, for special purposes.

Each kind uses a different **FieldSpecifier** format:

- If the first byte of the **FieldSpecifier** is 0, then this **MetaDataItem** contains a Standard meta-data field, and the following **VLQ** holds the **FieldID**. In other words, all Standard **FieldSpecifiers** start with a zero byte, followed by the field number in **VLQ** form. See section 5.2. for the defined Standard **FieldID**s.
- If the first byte of the **FieldSpecifier** is not 0, then this **MetaDataItem** contains a Custom meta data field, and the **FieldSpecifier** is the field's name, in **XString** form.

**FieldSpecifier:**

either **Standard Field Specifier**

**VLQ( 0 ), VLQ( FieldID )**

Example: **`0x00 0x04`**
means the standard
Filename On Disk meta-data field

or **Custom Field Specifier**

**XString( "Your Custom Field Name Goes Here as Text" )**

Example: **`0x16 'This is my Custom Name'`**
means a Custom meta-data item named
'This is my Custom Name'

**Custom Field Specifiers** are formatted as text strings in **XString** form (see section 4.2), and so can use any field specifier at all – the only limitation is that the length cannot be zero. We expect that in most cases users will want custom meta-data to be human readable, and in these cases the desired name in plain text would typically be used directly for the Custom Field Specifier.

> **Example:** If a publisher wanted a Custom field named "Canto Catalog Filename", then the string "Canto Catalog Filename" would appear literally in the XMF file as the Custom Field Specifier, in **XString** form.

**Standard FieldID**s are integers in **VLQ** form. Only the MMA may assign Standard **FieldID**s. This document defines several Standard **FieldID**s (see section 5.2). It is expected that new Standard Fields will be defined in future MMA Recommended Practices. If a parser does not recognize a Standard **FieldID**, it should not attempt to process the **FieldContents** for that **MetaDataItem**.

### 3.2.1.1.2. FieldContents Structure

There are two kinds of meta-data **FieldContents**:

- **Universal Contents**, where just one version of content is present for the **MetaDataItem**. The contents may be either Extended ASCII text or binary data, but not Unicode.
- **International Contents**, where multiple alternate versions of the content are provided, each one keyed to a language and one or more countries. Each alternate may contain either Extended ASCII text, Unicode text, compressed Unicode text, or binary data, and may be either visible to the user or hidden.

**FieldContents** for a Standard meta-data **FieldID** may be Universal, International, or either, as per that **FieldID**'s definition (see section 5.2).

**FieldContents** for a Custom meta-data item may be either Universal or International, on a field-by-field basis.

For Universal contents, the **FieldContents** structure is a **VLQ** of zero (**`0x00`**), then the **LengthInBytes** of the following data in **VLQ** form, a **StringFormatTypeID** in **VLQ** form, and finally the data block. See section 3.2.2.1 for the defined values for a **StringFormatTypeID**.

> Note that the **LengthInBytes** is the number of data bytes plus the size of the **StringFormatTypeID**. As a result, **FieldID**s which require no data, such as flags, can simply use a **LengthInBytes** of 0, and omit the **StringFormatTypeID**. In other words, the **FieldContents** for an empty Universal field is just **`0x00 0x00`**.

For International contents, the **FieldContents** structure is a list of **ContentVersion** structures, prefixed with both a **NumberOfVersions** count and a **LengthInBytes** (both in **VLQ** format). Each **ContentVersion** structure consists of two parts: a **MetaDataType** specifier in **VLQ** form, indicating a string format/language/country combination (see below); and the length-delimited **VersionData**. A zero length indicates no **VersionData**.

**FieldContents:**

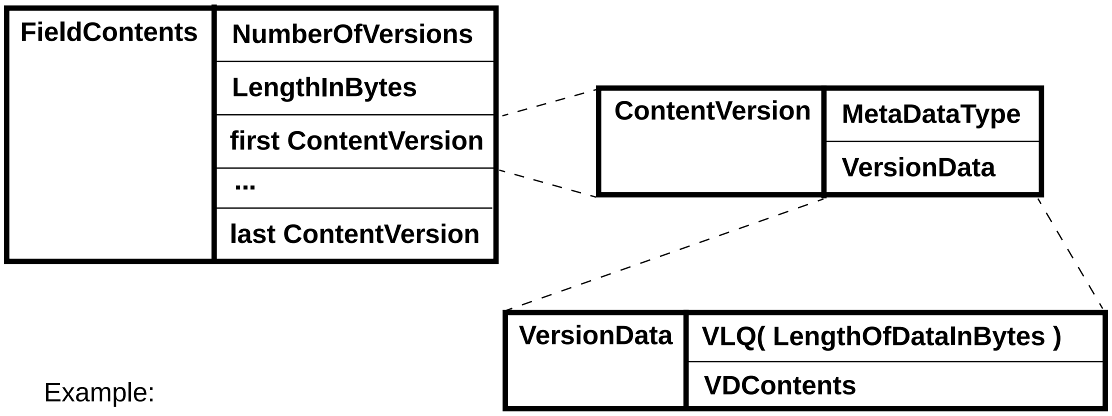
either **Universal Contents** without Country/Language version selection

**VLQ( 0 ), VLQ( LengthInBytes ), VLQ( StringFormatTypeID ), UniversalData**

Example:

```
// FieldContents ------------------------------------

 0x00         // Initial zero indicates UniversalContents
 0x81 0x01    // 129 bytes follow
 0x07         // Binary data, hidden
 <128 bytes of binary data>
```

or **International Contents** with Country/Language version selection



Example:

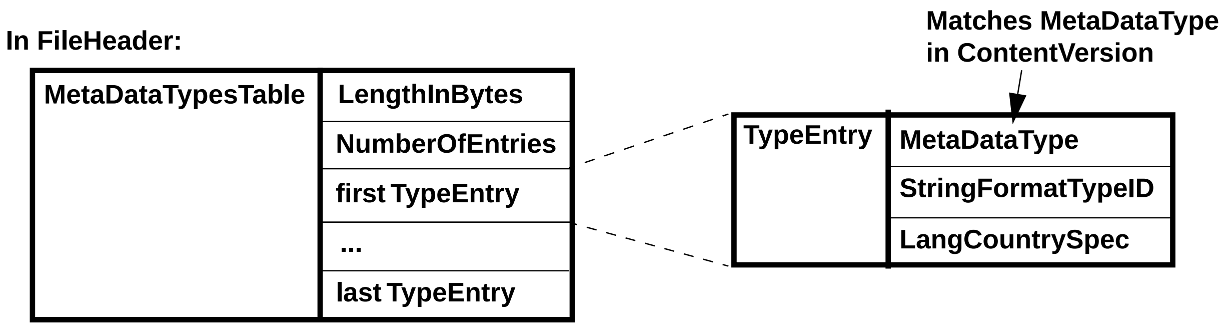
```
// FieldContents ---------------------------------------

0x03    // NumberOfVersions: 3 versions
        // -- Initial non-zero indicates InternationalContents
0x30     // 48 bytes follow


// First ContentVersion -----------------
0x03                     // For MetaDataType 3,
0x0C ‘Hello, world’      // VersionData is this XString

// Second ContentVersion -----------------
0x01                      // For MetaDataType 1,
0x12 ‘Bonjour, la France’ // VersionData is this XString

// Third ContentVersion ------------------
0x02                     // For MetaDataType 2,
0x0F ‘Bonjour, Quebec’   // VersionData is this XString
```

## 3.2.2. MetaDataTypesTable Format

To avoid repeating a Language/Country and **StringFormatTypeID** for every **ContentVersion** present in the XMF file, all unique language/country/format combinations are stored in the **FileHeader** in a list called the **MetaDataTypesTable**. The **MetaDataTypesTable** is delimited both by length (to allow parsers to skip over the table) and by count (for easier detection of out-of-range **MetaDataType** values).

To obtain a **ContentVersion**'s intended **LangCountrySpec** and **StringFormatTypeID**, a parser would search the **MetaDataTypesTable** for a **TypeEntry** for the **MetaDataType** used in the **ContentVersion**. **MetaDataType** values must start at 1 and go up to **NumberOfEntries**, but are not required to appear in the **MetaDataTypes** table in any particular order.

**LengthInBytes**, **NumberOfEntries**, and **StringFormatTypeID** are all **VLQ**s. **LangCountrySpec** is in **XString** form.



Example:

```
0x16    // LengthInBytes is 22
0x03    // NumberOfEntries

// First TypeEntry
0x01            // MetaDataType 1
0x00            // StringFormatID 0 is visible ASCII
0x05 ‘fr-fr’    // LangCountrySpec says French / France

// Second TypeEntry
0x03            // MetaDataType 3 (order is not required)
0x00            // StringFormatID 0 is visible ASCII
0x02 ‘en’       // LangCountrySpec says English / all countries

// Third TypeEntry
0x02            // MetaDataType 2
0x00            // StringFormatID 0 is visible ASCII
0x05 ‘fr-ca’    // LangCountrySpec says French / Canada
```

### 3.2.2.1. StringFormatTypeID Definitions

**StringFormatTypeID**s are integers in **VLQ** form, and may only be defined and assigned by the MMA. This document defines eight **StringFormatTypeID**s:

**0 –** Extended ASCII, visible to the user
**1 –** Extended ASCII, hidden from the user
**2 –** Unicode, visible to the user
**3 –** Unicode, hidden from the user
**4 –** Compressed Unicode, visible to the user
**5 –** Compressed Unicode, hidden from the user
**6 –** Binary data, visible to the user
**7 –** Binary data, hidden from the user

**Note:** In this coding, the low bit of the **StringFormatTypeID** integer acts as a visible/hidden flag. Any **StringFormatTypeID**s assigned in future should continue this pattern.

**Unicode Compression** is as defined in the Unicode Consortium document **Unicode Technical Report #6**. This compression scheme involves both a moving 7-bit window across the 16-bit Unicode character space, and a fixed 7-bit window into Roman character space. This allows full access to international characters at string sizes roughly comparable to ASCII.

Please refer to:

**http://www.unicode.org/unicode/reports/tr6/**

**Unicode Format –** Unicode contents for **StringFormatTypeID**s 2 and 3 must conform to version 2.0 of the Unicode standard, in UTF-16 coding. It is not required to start the Unicode contents with a Byte Order Mark (0xFEFF), because the byte order of the entire XMF file may be determined from the first two bytes of the **FileHeader** structure (section 2.1). When importing text content into Unicode from other text encodings, developers should be careful to observe correct transcoding, based on the ISO/IEC 10646 character set.

### 3.2.2.2. LangCountrySpec Format and Selection Behavior

**LangCountrySpec** syntax and registry are as defined in the Internet Engineering Task Force's HTTP standard (RFC 2068 at section 3.10), as derived from RFC 1766, ISO 639 (language codes) and ISO 3166 (country codes).

For example, the following are valid **LangCountrySpec**s:

English for all countries: **`en`**

English for the US: **`en-us`**

English for Canada: **`en-ca`**

French for Canada: **`fr-ca`**

English for the US and Canada only: **`en-us,ca`**

An XMF playback program or device should make an effort to determine its location and its user's natural language at the time the XMF file is first opened, and select the appropriate **ContentVersion** for every meta-data item it accesses in the XMF file. When multiple **ContentVersion**s are available for a **MetaDataItem**, only the one best match should be displayed, never multiple **ContentVersion**s at once for the same meta-data item. The system's location and country must not be changed during parsing of a given XMF file, to ensure synchronization of all such selections for that file.

Please refer to:

**http://ietf.org/rfc/rfc1766.txt**
**http://ietf.org/rfc/rfc2068.txt**
**http://www.oasis-open.org/cover/iso639a.html**
**http://www.din.de/gremien/nas/nabd/iso3166ma/codlstp1/en_listp1.html**

# 4. Atomic Structures

The **VLQ** and **XString** structures are the basis of most data fields in XMF files.

## 4.1. VLQ Format

To save space and avoid all maximum size limitations, most XMF integers are represented via the Variable Length Quantity number format (per “Standard MIDI Files 1.0”, p. 2). Basically, a **VLQ** is an integer that’s however many bytes long it needs to be to hold the given value. The last byte is signalled by a clear high bit, and all other bytes have the high bit set. This leaves 7 bits of usable data per byte, so turning a **VLQ** into a binary integer means an iterative process of shifting an accumulator left 7 bits, masking the next byte’s high bit off, and adding. One-byte **VLQ**s are considered ‘last byte,’ and must have a high bit of zero.

For example:

```
0      =     0x00  =     0x00
64     =     0x40  =     0x40
127    =     0x7f  =     0x7f
128    =     0x80  =     0x81 0x00
8192   =     0x2000=     0xC0 0x00
16383 =     0x3fff=     0xff 0x7f
2097151=    0x1fffff=   0xff 0xff 0x7f
```

|  |  | hi bit |  |
|---|---|---|---|
| VLQ | first Byte | 1 | 7 data bits |
|  | ··· | 1 | 7 data bits |
|  | last Byte | 0 | 7 data bits |

## 4.2. XString Format

The **XString** format used in several places is a length-delimited ASCII text string, where the length is encoded in a leading **VLQ** rather than a fixed-size integer. This is similar to the Pascal String of olde, but allows the string to have any length, and allows a parser to skip over any **XString** it isn’t interested in, without having to scan for a terminating character.

|  |  |
|---|---|
| XString | VLQ( LengthOfTextInBytes ) |
|  | Text Data |

# 5. Managed ID Formats and Assignment

This section describes the format, management, and assignment of the number spaces for **UnpackerID**s, Standard Meta-Data **FieldID**s, and **ResourceFormatID**s.

## 5.1. UnpackerID Format and Assignment

In a **NodeUnpackers** list in a **NodeHeader**, an **UnpackerID** selects a decoding algorithm to apply to the encoded resource data referenced in the **NodeContents**. Every **UnpackerID** is prefixed by an **UnpackerTypeID** that picks one of four number spaces, each of which uses a different format and mechanism to prevent assignment collisions, as detailed in the following four subsections.

### 5.1.1. Standard UnpackerIDs

**Standard UnpackerID**s are prefixed with **UnpackerTypeID 0** (**`0x00`**).

Standard **UnpackerID**s are reserved for unpacker types that all XMF parsers are required to implement, and may only be defined by the MMA. There will probably be very few Standard **UnpackerID**s, so in practice they'll all fit into one byte for the foreseeable future.

A **Standard UnpackerID** consist of one integer in **VLQ** form.

Standard **UnpackerID**s may only be defined by the MMA.

| UnpackerID | Unpacker Name | Algorithm Reference |
|---|---|---|
| **0** | No Unpacker | (No operation) |

**Example:**

```
0x00 // Standard UnpackerID follows
0x00 // No Unpacker
```

**Note:** Developers who wish to support XMF to the fullest degree are advised to check the current status of the XMF Files unpackers work.

### 5.1.2. MMA Manufacturer UnpackerIDs

**MMA Manufacturer UnpackerID**s are prefixed with **UnpackerTypeID 1** (**`0x01`**).

MMA Manufacturer **UnpackerID**s are reserved for manufacturer-specific unpacker types. Each MMA Manufacturer is free to create **UnpackerID**s based on their own MMA Manufacturer ID, subject to the following format restrictions. Manufacturers are free to either publish their **UnpackerID**s or keep them proprietary.

MMA Manufacturer **UnpackerID**s contain two parts: first the company's assigned MMA Manufacturer ID in ordinary 1-byte or 3-byte form, then a **VLQ** containing the company's internal unpacker ID.

Note that all MMA Manufacturer IDs are either one byte long or three bytes long, as indicated by the first byte (**`0x00`** means a three-byte ID).

**Examples:**

```
// Example 1: 3-byte Mfr ID
0x01 // MMA Mfr UnpackerID follows
0x00 0x7c 0x7f // Electric Kazoo MMA Mfr. ID (3-byte since hibyte = 0)
0x0C // Electric Kazoo's Unpacker type 12

// Example 2: 1-byte Mfr ID
0x01 // MMA Mfr UnpackerID follows
0x7c // KazooTronics MMA Mfr. ID (1-byte since hibyte not zero)
0x81 0x01 // KazooTronics's Unpacker type 129
```

### 5.1.3. Registered UnpackerIDs

**Registered UnpackerID**s are prefixed with **UnpackerTypeID 2** (**`0x02`**).

Registered **UnpackerID**s are reserved for custom or proprietary unpacker types, but are not linked to MMA Manufacturer IDs, and may only be defined and assigned by the MMA. This allows non-MMA manufacturers access to eXtensibility without incurring the higher byte count of Non-Registered **UnpackerID**s.

A registered **UnpackerID** consists of one integer in **VLQ** form.

To obtain a Registered **UnpackerID**, or for the MMA's registration policy, see **http://www.midi.org**. For the current list of Registered **UnpackerID**s, see **http://www.midi.org**.

**Example:**

```
0x02 // Registered UnpackerID follows
0x47 // Registered Unpacker type 71
     // (must be listed on MMA website to be valid)
```

## 5.1.4. Non-Registered UnpackerIDs

**Non-Registered UnpackerID**s are prefixed with **UnpackerTypeID 3** (**`0x03`**).

Non-Registered **UnpackerID**s are reserved to allow non-MMA manufacturers to use arbitrary private compression and data protection mechanisms without MMA registration. To avoid collisions, a Non-Registered **UnpackerID**s must be generated as a Globally Unique Identifier (GUID).

A Non-Registered **UnpackerID** consists of one GUID in **VLQ** form.

While GUIDs are defined to be 16 bytes long, typically length will increase to 19 bytes when converted to **VLQ** form. However, in the conversion to **VLQ** any contiguous runs of high-order zero bits will be truncated at 7-bit boundaries, perhaps producing a **VLQ** shorter than 19 bytes.

**Example:**

```
0x03 // Non-Registered UnpackerID follows
<16-byte GUID in VLQ form appears here>
```

## 5.2. Meta-Data Standard FieldID Format and Assignment

Standard **FieldID**s for XMF Meta-Data are integers in **VLQ** format. This document defines seven Standard **FieldID**s for logistical and informational purposes. Only the MMA may assign Standard **FieldID**s. Note that up to 127 Standard **FieldID**s can be defined without exceeding a 1-byte **VLQ**.

## 5.2.1. Standard FieldID Assignments

**Important:** Future MMA Recommended Practices for XMF are expected to define further Standard **FieldIDs**, addressing informational meta-data (i.e. authoring and commerce information such as copyright notices and credits) and rights management, with guidelines for cross-coding to and from major relevant standards.

The following table contains the Standard FieldID assignments.

**Note:** In defining new Standard meta-data fields, every **FieldID** must be defined as requiring either Universal or International **FieldContents**, or allowing the use of either.

| FieldID | Field Name and Notes | Valid for Node Types | Contents Format |
|---|---|---|---|
| **0** | **XMF File Type**<br>E.g. XMF Type 0, or XMF Type 1, etc., including RevisionID.<br>Values to follow in later RPs. | File or Folder,<br>but only allowed in the **RootNode** | Universal, Binary (hidden or visible)<br>See section 5.2.2 for **FieldContents** format. |
| **1** | **Node Name**<br>Name must be unique in the XMF file. | File or Folder | Universal, Extended ASCII (hidden or visible) |
| **2** | **Node ID Number**<br>Optional, for parser convenience. | File or Folder | Universal Binary (hidden or visible)<br>**VLQ** |
| **3** | **Resource Format**<br>E.g. SMF, DLS, WAV, etc.<br>Values to follow in later RPs. | File | Universal, Binary (hidden or visible)<br>See section 5.3 for **FieldContents** format. |
| **4** | **Filename on Disk**<br>Excluding filename extension. | File | Universal, Extended ASCII (hidden or visible) |
| **5** | **Filename Extension on Disk**<br>For OS's other than Mac OS. | File | Universal, Extended ASCII (hidden or visible)<br>Including dot, e.g. "`.dls`". |
| **6** | **Mac OS File Type and Creator**<br>Formatted as 8 ASCII characters, e.g. "`TYPECR8R`" | File | Universal, Extended ASCII (hidden or visible) |
| **7** | **MIME Type**<br>(e.g. "`audio/wav`") | File | Universal, Extended ASCII (hidden or visible) |
| **8** | **Title**<br>Cross-coding guidelines expected to follow in a later RP. | File or Folder | Universal, Extended ASCII (hidden or visible),<br>or International |
| **9** | **Copyright Notice**<br>Cross-coding guidelines expected to follow in a later RP. | File or Folder | Universal, Extended ASCII (hidden or visible)<br>or International |
| **10** | **Comment**<br>Cross-coding guidelines expected to follow in a later RP. | File or Folder | Universal, Extended ASCII (hidden or visible)<br>or International |

## 5.2.2. Field Contents Interpretation Notes

### FieldID 0: XMF File Type

The **XMF File Type** meta-data field identifies the Type to which the XMF file conforms, for example XMF Type 0 or XMF Type 1, and the specification Revision level within that Type. This field is only valid in the **RootNode**; if the reading parser encounters it elsewhere, it must not process it as a file type indicator.

Note that this is distinct from the **XmfMetaFileVersion** appearing in the **FileHeader**, which indicates the version of the **XMF Meta-File Format Specification** being used.

The **FieldContents** of an XMF File Type **MetaDataItem** is a **FileTypeID** followed by a **RevisionID**, both in **VLQ** form. **FileTypeID**s are integers corresponding directly to XMF File Types (i.e. 0x07 means XMF Type 7), and may only be assigned by the MMA, typically through separate Recommended Practice documents (see for example **Type 0 and Type 1 XMF Files**). **RevisionID**s are integers corresponding directly to revision numbers of the given XMF File Type (i.e. 0x00 means initial release, 0x03 means third revision), and may only be assigned by the MMA.

## 5.3. ResourceFormatID Format and Assignment

Every **FileNode** must include in its **NodeMetaData** one **MetaDataItem** indicating the resource's data format (e.g. SMF, DLS, WAV, etc.), using the **ResourceFormat** Standard meta-data field (Standard meta-data **FieldID** 3). Every **ResourceFormatID** is prefixed by a **FormatTypeID** that picks one of four number spaces, each of which uses a different format and mechanism to prevent assignment collisions, as detailed in the following four subsections. The **FieldContents** for a **ResourceFormat MetaDataItem** depends on the **FormatTypeID**. This closely parallels the **UnpackerID** scheme.

### 5.3.1. Standard ResourceFormatIDs

**Standard ResourceFormatID**s are prefixed with **FormatTypeID 0** (`0x00`), followed by the **ResourceFormatID** in VLQ form.

Standard **ResourceFormatID**s are reserved for standard data formats such as SMF, DLS, WAV, etc. that all XMF parsers for a given XMF File Type are required to implement. There will probably be very few Standard **ResourceFormatID**s, so in practice they'll all fit into one byte for the foreseeable future.

| **ResourceFormatID** | **Format Name** | **Defined in RP for XMF File Type** |
|---|---|---|
| **0** | Standard MIDI File (SMF), Type 0 | XMF Type 0 and 1 |
| **1** | SMF, Type 1 | XMF Type 1 |
| **2** | Downloadable Sounds (DLS), Level 1 | XMF Type 0 and 1 |
| **3** | DLS, Level 2 | XMF Type 0 and 1 |
| **4** | DLS, Level 2.1 | XMF Type 0 and 1 |

**Note:** Standard **ResourceFormatID**s may only be defined by the MMA, and will typically be added via the Recommended Practice documents that define new XMF File Types. Developers who wish to support XMF to the fullest degree are advised to check the current status of the XMF Files work.

**Example:**

```
// MetaDataItem -----------------------------------

// FieldSpecifier:
0x00 // VLQ( 0 ): Standard meta-data FieldID follows
0x03 // VLQ( 3 ): FieldID 3: This is a Resource Format MetaDataItem

// FieldContents:
0x00 // VLQ( 0 ): Universal contents follows
0x02 // VLQ( 3 ): LengthInBytes 3: 3 bytes follow
0x06 // VLQ( 6 ): Visible Binary data follows
// UniversalData starts here
0x00 // VLQ( 0 ): FormatTypeID 0: Standard ResourceFormatID follows
0x01 // VLQ( 1 ): ResourceFormatID 1: Resource is in SMF Type 1 format
```

## 5.3.2. MMA Manufacturer ResourceFormatIDs

**MMA Manufacturer ResourceFormatID**s are prefixed with **FormatTypeID 1** (**`0x01`**).

MMA Manufacturer **ResourceFormatID**s are reserved for manufacturer-specific resource types. Each MMA Manufacturer is free to create **ResourceFormatID**s based on their own MMA Manufacturer ID, subject to the following format restrictions. Manufacturers are free to either publish their **ResourceFormatID**s or keep them proprietary.

A MMA Manufacturer **ResourceFormatID** contains two parts: first the company's assigned MMA Manufacturer ID in ordinary 1-byte or 3-byte form, then a **VLQ** containing the company's internal format ID.

Note that all MMA Manufacturer IDs are either one byte long or three bytes long, as indicated by the first byte (**`0x00`** means a three-byte ID).

**Example:**

```
// MetaDataItem -------------------------------------

// FieldSpecifier:
0x00 // VLQ( 0 ): Standard meta-data FieldID follows
0x03 // VLQ( 3 ): FieldID 3: This is a Resource Format MetaDataItem

// FieldContents:
0x00 // VLQ( 0 ): Universal contents follows
0x06 // VLQ( 6 ): LengthInBytes 6: 6 bytes follow
0x06 // VLQ( 6 ): Visible Binary data follow
// UniversalData starts here
0x01 // VLQ( 1 ): FormatTypeID 1: MMA Mfr ResourceFormatID follows
0x00 0x7c 0x7f // Electric Kazoo MMA Mfr. ID
0x0A // VLQ( 10 ): Resource is in Electric Kazoo’s format type 10
```

## 5.3.3. Registered ResourceFormatIDs

**Registered ResourceFormatID**s are prefixed with **FormatTypeID 2** (**`0x02`**).

Registered **ResourceFormatID**s are reserved for custom or proprietary resource formats, but are not linked to MMA Manufacturer IDs, and may only be defined and assigned by the MMA. This allows non-MMA manufacturers access to eXtensibility without incurring the higher byte count of Non-Registered **ResourceFormatID**s.

A Registered **ResourceFormatID** consists of one integer in **VLQ** form.

To obtain a Registered **ResourceFormatID**s, or for the MMA's registration policy, see **http://www.midi.org**. For the current list of Registered **ResourceFormatID**s, see **http://www.midi.org**.

**Example:**

```
// MetaDataItem -------------------------------------

// FieldSpecifier:
0x00 // VLQ( 0 ): Standard meta-data FieldID follows
0x03 // VLQ( 3 ): FieldID 3: This is a Resource Format MetaDataItem

// FieldContents:
0x00 // VLQ( 0 ): Universal contents follows
0x04 // VLQ( 4 ): LengthInBytes 4: 4 bytes follow
0x06 // VLQ( 6 ): Visible Binary data follow
// UniversalData starts here
0x02  // VLQ( 2 ): FormatTypeID 2: Registered ResourceFormatID follows
0x81 0x46 // VLQ( 197 ): Resource is in Registered format number 197
          // (must be listed on MMA website to be valid)
```

## 5.3.4. Non-Registered ResourceFormatIDs

**Non-Registered ResourceFormatID**s are prefixed with **FormatTypeID 3** (**`0x03`**).

Non-Registered **ResourceFormatID**s are reserved to allow non-MMA manufacturers to use arbitrary private resource formats without MMA registration. To avoid collisions, a Non-Registered **ResourceFormatID**s must be generated as a Globally Unique Identifier (GUID).

A Non-Registered **ResourceFormatID** consists of one GUID in **VLQ** form.

While GUIDs are defined to be 16 bytes long, typically length will increase to 19 bytes when converted to **VLQ** form. However, in the conversion to **VLQ** any contiguous runs of high-order zero bits will be truncated at 7-bit boundaries, perhaps producing a **VLQ** shorter than 19 bytes.

**Example:**

```
// MetaDataItem -------------------------------------

// FieldSpecifier:
0x00 // VLQ( 0 ): Standard meta-data FieldID follows
0x03 // VLQ( 3 ): FieldID 3: This is a Resource Format MetaDataItem

// FieldContents:
0x00 // VLQ( 0 ): Universal contents follows
0x15 // VLQ( 21 ): LengthInBytes 21: 21 bytes follow
0x06 // VLQ( 6 ): Visible Binary data follow
// UniversalData starts here
0x03  // VLQ( 3 ): FormatTypeID 3: Registered ResourceFormatID follows
<19 bytes appear here> // GUID in VLQ form
```

# Appendix: Notes on the XMF Meta-File Format

**Topics:**

## Purpose of XMF

In recent years the need for a standard, universal file-bundling format for music and sound data has become clear. This is especially true in interactive multimedia and Internet entertainment delivery where the audio producer's output is often not just a single file, but rather a collection of related files for each title, scene, or setting. In many cases more than one file format is needed – for example, games usually use both SMF and digital audio files.

Further, some media file formats tend by their nature to create dependencies between files. Notably, when the DLS file format is used for content-specific samples (such as sung lyrics), there is an implied link between the DLS file and the SMF file that uses it.

Attempting to use unbundled collections of files poses a number of difficult problems for the content producer, the content user, and the tool developer:

- Hard to keep the collection together
- Hard to move the collection from one computer to another
- Hard to move the collection to different operating systems (file types, .zip vs. sit, etc.)
- Hard to ensure version synch among files in the collection
- Hard to keep other files out of the collection
- Each file type has its own meta-data conventions (SMF, DLS, WAV, etc.)
- No standard way to add meta-data notations to the entire collection
- Hard to express how the collection should be used – what does 'play' or 'load' mean for the collection?
- No consistent cross-platform data-compression standard for a collection, so files tend to be large
- No consistent cross-platform information security (intellectual property protection) standard for a collection, so business models are limited
- Relationship between the music/sound content creator and the programmer/user is complicated by all the above ambiguities

XMF addresses all of these problems, bringing important benefits to all interested parties:

- **For users of music and sound media** XMF makes working with music and sound material easier by reducing the collection to a single file. Expectations of playback behavior for each area of application are made clear through a series of separate Recommended Practices. If desired, the differences between media types can be made invisible – it doesn't matter whether the XMF file contains an SMF file or a WAV file, it loads and plays just the same in either case.

- **For producers of music and sound media** XMF provides increased confidence in the collection's integrity by guarding against misuse in a variety of ways, as well as the increased convenience of delivering just a single file and being able to attach meta-data to the collection as a whole. The addition of data security features makes new forms of business possible – for example, think of telephone networks handling per-use micro-payments for cell phone ring tones, or DLS banks that need to be unlocked with a purchased key.
- **For developers of playback software and applications** XMF adds (in separate Recommended Practices) unambiguous interpretation rules to collections of music and sound materials, builds on top of the existing media-specific playback APIs (SMF, DLS, WAV, etc.), and is a cross-platform standard. Any needed custom resource types (for example: softsynth patch banks, configuration setups, etc.) can stored in XMF files, and will be ignored by other playback software. In future, new resource types could be adopted by the MMA for 'playlists' of SMF phrases or samples, enabling new XMF applications ranging from cross-platform remixers to a 'universal session file' format.
- **For developers of authoring software** XMF provides a gateway to cross-platform and interactive playback environments, including those beyond computers.

## Advice for XMF Readers

If your program is asked to play an XMF file, then you'll have some XMF parser code sitting on top of playback APIs that support SMF, DLS, WAV, and/or other resource types. XMF's scope does not extend into the internals of those resource types or their playback API's – only resource access, and the setup of the virtual playback studio.

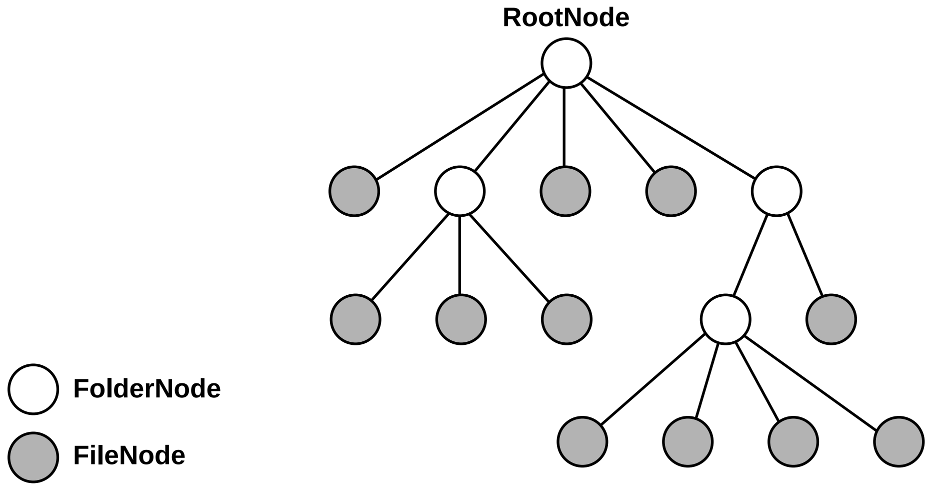
Your XMF reader has to know two things:

**How do I know what resources are in, and/or referred to by, the XMF file?**
**How do I know what I'm supposed to do with those resources?**

## How do I know what resources are in, and/or referred to by, the XMF file?

XMF files are logically structured as hierarchical **Tree**s, very much like a computer file system with 'files' (resources) located in 'folders' that may be nested to any depth.

**Example Tree Hierarchy**



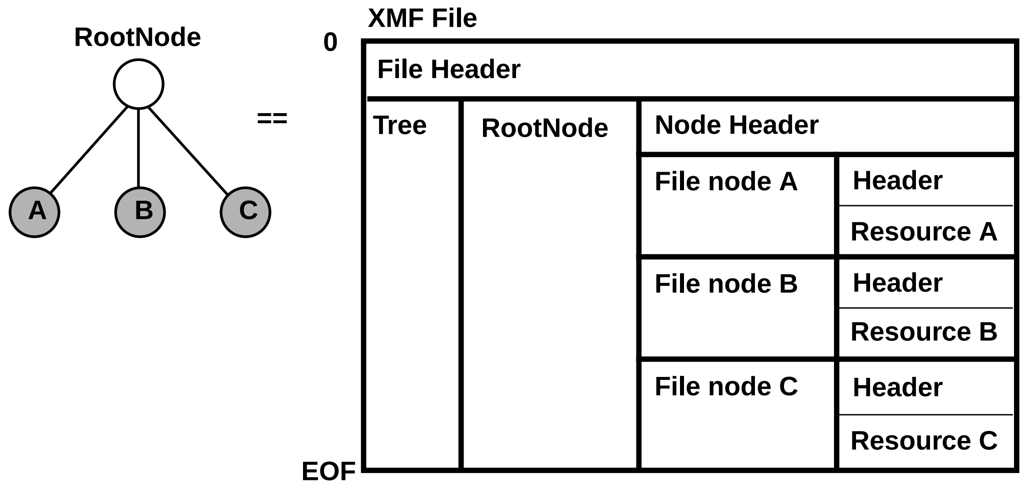
As with file systems, the structure and depth of the **Tree** is arbitrary and under the control of the end user. Therefore, your XMF reader code must be able to handle any arbitrary **Tree** a user may invent, and must actually scan the **Tree** in order to determine what resources are present (or referred to) in a given XMF file.

## Resource Storage and File Layout

You should be aware that the resources described by a given logical **Tree** may be stored in many places, depending on the resource Reference Type used in each **FileNode**, and this can produce many different storage profiles. Resources may be stored directly within the **Tree**, elsewhere in the same XMF file, or in external resource files (SMF, WAV, DLS, etc.) or external XMF files.

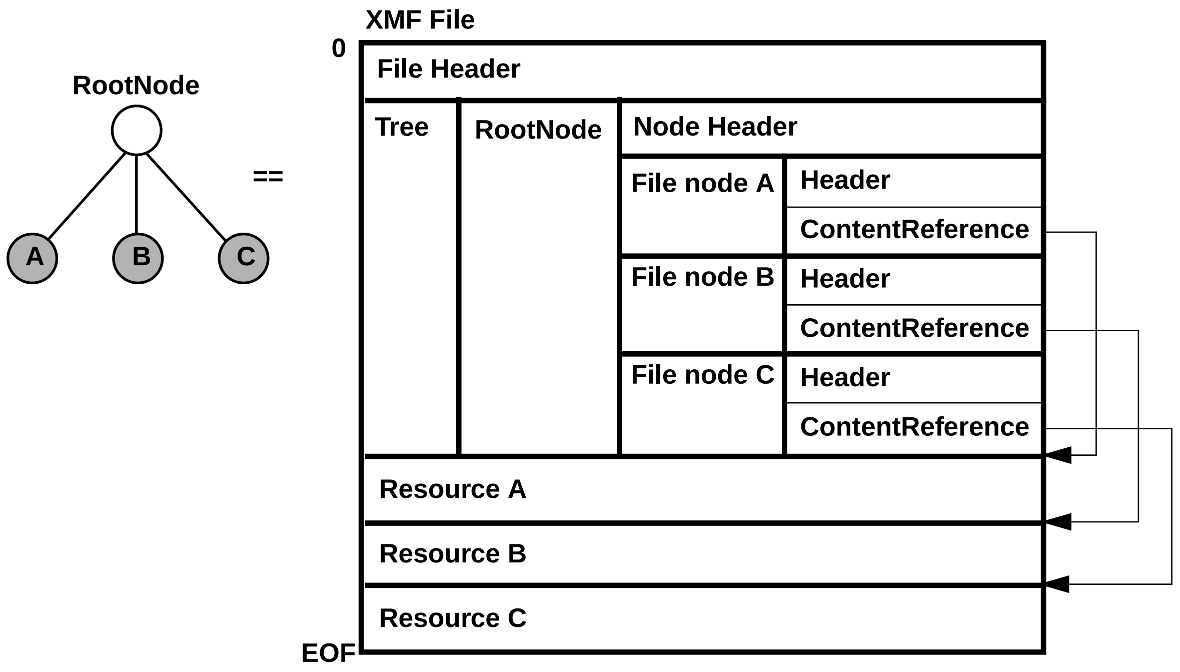
Storing the resources directly in the **Tree** produces hierarchical containment:

### Hierarchical Storage Example



The resources can also be stored elsewhere in the file, producing a flat storage profile and a much smaller **Tree**:

**Flat Storage Example**

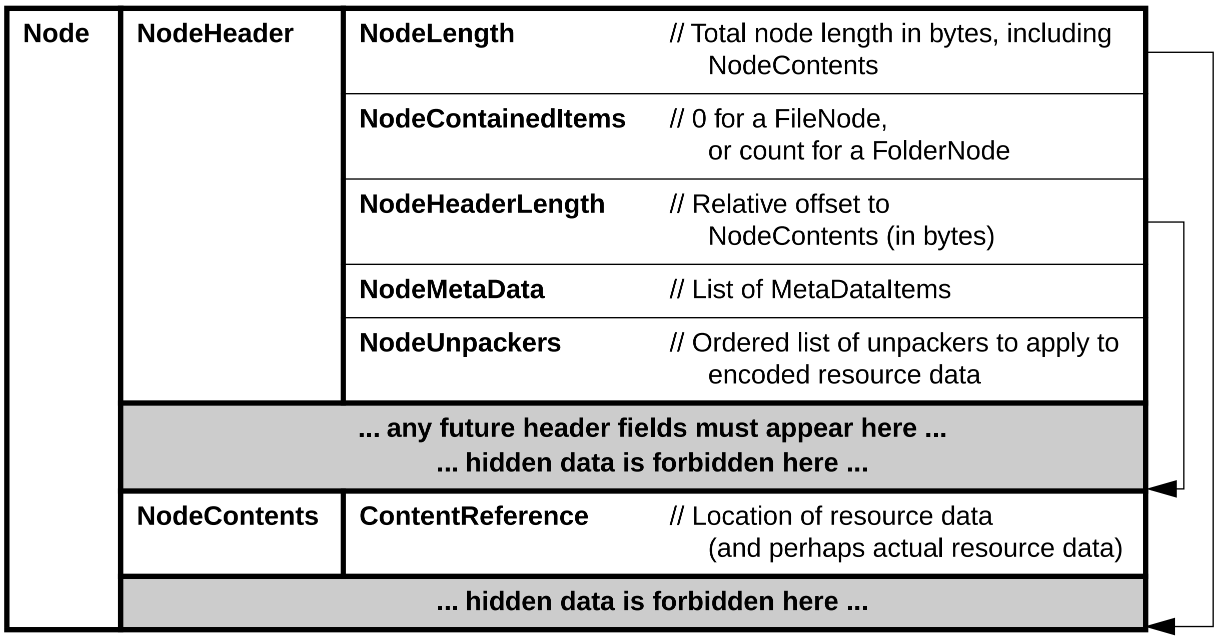


Many other storage profiles are also available, including XMF files that are only collections of references to external files (see **2.2.1.2.1. Reference Types** in the **XMF Meta-File Format Specification**).

## Parsing the Tree

Each file or folder in the **Tree** is represented as a **Node**, and all **Node**s use the same data structure. You start your parse at the **Tree**'s **RootNode**, whose location is indicated in the **FileHeader**'s **TreeStart** field.

The structure for all **Node**s is:



| | | | |
|---|---|---|---|
| **Node** | **NodeHeader** | **NodeLength** | // Total node length in bytes, including NodeContents |
| | | **NodeContainedItems** | // 0 for a FileNode, or count for a FolderNode |
| | | **NodeHeaderLength** | // Relative offset to NodeContents (in bytes) |
| | | **NodeMetaData** | // List of MetaDataItems |
| | | **NodeUnpackers** | // Ordered list of unpackers to apply to encoded resource data |
| | **... any future header fields must appear here ... ... hidden data is forbidden here ...** | | |
| | **NodeContents** | **ContentReference** | // Location of resource data (and perhaps actual resource data) |
| | **... hidden data is forbidden here ...** | | |

- **NodeLength** is a **VLQ** described below under **Navigating Nodes**.

  **Note:** This field should **not** be used to store hidden data in the gap after the end of the **NodeContents**. XMF parsers will not find that data, so XMF tools will strip that data when editing an XMF file, and XMF players will not be able to access that data.

- **NodeContainedItems** is a **VLQ** that indicates whether the **Node** is a **FileNode** (resource) or **FolderNode** (container for further **Nodes**), and for **FolderNodes** indicates the number of directly contained **Nodes**. If a **FolderNode**, then the **NodeContents** describes catenated **Nodes** for the contained files and/or folders. Since 0 is used to indicate a **FileNode**, empty **FolderNodes** are not allowed in XMF.
- **NodeHeaderLength** is a **VLQ** that takes you to the **NodeContents** part. This is redundant for XMF version 1.00, but preserves backwards compatibility if MMA adds more fields in future.

  **Note:** This field should **not** be used to store hidden data in the gap between the **NodeHeader** and the **NodeContents**. XMF parsers will not find that data, so XMF tools will strip that data when editing an XMF file, and XMF players will not be able to access that data.

- **NodeMetaData** contains meta-data for the resource. (See **3. XMF Meta-Data** section of the **XMF Meta-File Format Specification** for format).
- **NodeUnpackers** is a list of the decoding operations you have to apply to the stored resource data block to get back clear data that you can play (see **2.2.1.1. NodeHeader Structure** section of the **XMF Meta-File Format Specification**). Unpackers are usually data decompressors, or decrypters or other info-security operations.
- **NodeContents** is a structure indicating where the resource data lives.

Note that you won't have to actually go get the resource data for every **Node** – just the ones you know you need.

When you do need a resource, a **NodeContents** can use any of five different Reference Types, each describing a different way to get from the **Node** to the resource data it describes:

- **1:** Immediate, in-line resource data
- **2:** Pointer to raw resource data elsewhere in the file
- **3:** Pointer to a resource **Node** elsewhere in file (with its own **NodeContents** structure)
- **4:** Reference to an external whole file (i.e. SMF, DLS, WAV)
- **5:** Reference to a named resource in an external XMF file.

See section **2.2.1.2. NodeContents Structure** for parsing details.

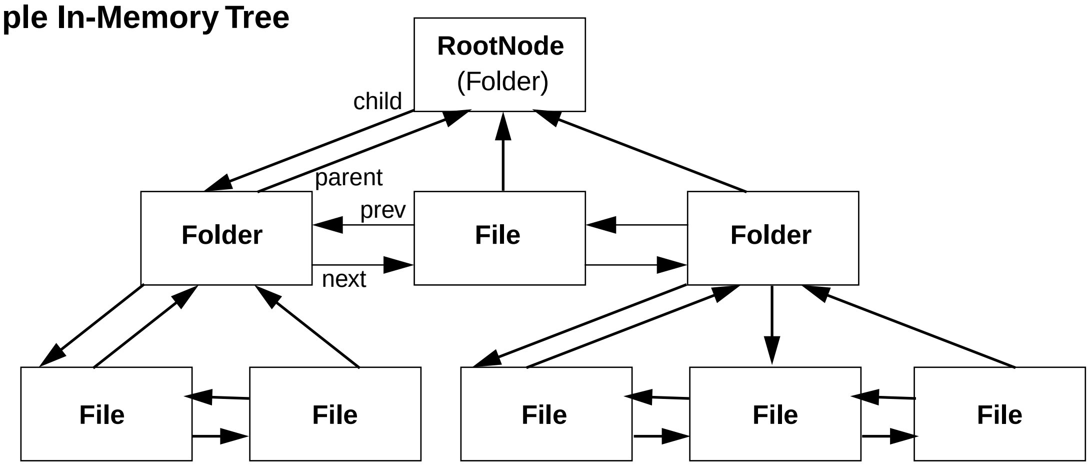
## Navigating Nodes

Navigation among **Nodes** is handled via the **NodeLength** field. It indicates the **Node** structure's length in bytes (including **NodeContents**), so you can use it to jump ahead to the next **Node** at the same level of the **Tree**.

For a **FolderNode**, **NodeLength** also tells you when you've read the last contained item – just compare your read pointer against the parent folder's expected ending offset (or decrement the **NodeContainedItems**).

Note that **NodeLength** only facilitates forward motion through the tree. If you anticipate the need to move around the **Tree** more flexibly, then the first time you parse the Tree you may want to build an in-memory tree of your own as you go – with **`next`**, **`prev`**, **`parent`**, and **`child`** pointers to accelerate subsequent random & backwards access.

**Example In-Memory Tree**



## How do I know what I'm supposed to do with those resources?

The XMF meta-file format per se has no required player behaviors. However, each XMF File Type may specify a different set of rules for how players, converters, and other tools and applications should handle the resources in an XMF file. These rules are defined in a separate Recommended Practice document defining each XMF File Type. A **MetaDataItem** in the **RootNode** indicates the XMF File Type of the file you're parsing, and the Type should also be indicated via filename extension, MIME type, and/or Mac OS File Type (as appropriate for the context). Each player, tool, or application must keep track of which XMF File Types it knows how to handle, and should exercise caution and discretion when opening an unrecognized XMF File Type.

## Resource Relationships

While an XMF file may be a simple bag of independently useful resources, some XMF use contexts will involve relationships between resources – for example, a game scene may need several SMFs played in a certain order, or an SMF may need a specific DLS bank to play correctly. The XMF meta-file structure as presently conceived does not directly express such relationships (other than via folder groupings), instead relying on meta-data, new resource types (included as part of the same XMF collection), and Recommended Practice documents defining particular XMF file Types, to express them.

Resource relationships fall into two major categories, further explained below:

**Resource Groupings**
**Resource Dependencies**

## Resource Groupings

Music and sound authors can use the folder hierarchy to arrange resources into arbitrary groups – by project, by media type, or any other desired criteria. Custom meta-data could also be used to create groupings via tagging, as could naming conventions using the standard meta-data **Node Name** field, or the **Filename on Disk** field. Although any XMF user could set up their own rules for what to do with those groupings, XMF as presently conceived does not include any universal rules governing them; that kind of interpretation is left to negotiations between the people and tools who create XMF files and the people and tools that use XMF files. Some specific standard behaviors may be defined in the XMF Recommended Practice documents that define specific XMF file Types – for example, **Autostart** and **Preload** are defined in the Recommended Practice document for XMF Type 0 and Type 1.

This leaves the door open for any number of application-specific grouping and interpretation conventions – i.e. a game publisher might invent one way of doing things, and a phone network might invent another that better fits its requirements, but the commonly readable resources from either source's XMF files could always be interchanged, and a flexible XMF authoring tool could create both kinds of files.

## Resource Dependencies

XMF as presently conceived does not include any generalized mechanism for directly representing resource dependencies. Where necessary, specific kinds of dependence can be represented by inventing new resource types and including such resources in XMF files. For example, some SMFs will rely on patch banks for custom synth types, and this dependency is expected to be represented in a new **BankMap** resource type. We expect to be able to handle any other important resource dependencies that come to light in future in a similar way.

## What do I do if I can't do what's expected?

Parsers should anticipate the following error conditions, and report them to the user if appropriate in the context of the intended user experience:

– Too many XMF indirections
– Too many reference indirections
– Can't access required resource
– Can't **Preload** requested resource (XMF Type 0 and 1)
– Can't **Autostart** requested resource (XMF Type 0 and 1)
– Can't parse XMF file
– External `file:` access not supported
– External `http:` access not supported
– Unrecognized Standard meta-data **FieldID**
– Unrecognized Standard **UnpackerID**
– Unrecognized Registered **UnpackerID**
– No unpacker for requested **UnpackerID**
– Unrecognized Standard **ResourceFormatID**
– Unrecognized Registered **ResourceFormatID**
– No handler for requested **ResourceFormatID**

## Advice for XMF Writers

This section contains suggestions for creating XMF files.

## How to Build an XMF File

The use of **VLQ**s and the hierarchical nature of the **Tree** means that XMF files are most easily assembled in hierarchical order, starting from the most deeply nested point and working up the tree to the **RootNode**.

The suggested procedure is:

**1.** Collect all resources (SMF, WAV, DLS, etc.) you want to bundle into the XMF file, as files and/or data blocks in memory.

**2.** Collect the locations (URI's, paths) of all external resources you want to reference from the XMF file.

**3.** Collect all the meta-data you want to apply to each bundled and referenced resource.

**4.** Apply all desired encoders to each resource, recording the unencoded size and **UnpackerID** for each operation, and the order of operations.

**5.** Plan your bundled resource storage.

You have a great deal of discretion in terms of how to put the resources together. The simplest approach is to store all resources directly in the **Tree**. However if you want to optimize the **Tree** you may want to store the resources elsewhere in the file, use aliases, and/or use detached **Nodes** to store each item's meta-data and unpackers outside of the **Tree**. See **Resource Storage and File Layout**.

**6.** Create the **Node** structure (**NodeHeader** and **ContentReference** structures) for every bundled and referenced resource. This is where meta-data and **UnpackerID**s are associated with the resources.

**7.** Create the **Tree** structure by catenating all the **Nodes** and stitching them together with their **NodeLength** fields. You should start at the deepest point in the hierarchy and work up the **Tree** to the **RootNode**, for three reasons:

A) Outer chunk lengths depend on inner chunk lengths;

B) The size of each **VLQ** field depends on the number it must hold; and

C) A **FolderNode**'s contained **Nodes** appear in-line.

**8.** If you're storing resources outside the **Tree** (Reference Type 2), or using detached **Nodes** (Reference Type 3), catenate those blocks before or after the **Tree**.

**9.** Create the **FileHeader** structure and catenate it to the start of the file.

This completes the XMF file.

## Suggested Practices

- If your XMF file contains more than one resource, keep the **Tree** lean by storing resources outside of the **Tree** (via ReferenceType 2), and place the **Tree** immediately after the **FileHeader**. In most cases this allows the **Tree** to be retrieved in a single seek or **http:** fetch, and it can then be used as an accelerator for accessing individual resources.

- If you use detached **Node**s (ReferenceType 3), don't use the same meta-data **FieldSpecifier**s in both the detached **Node** and **Node** in the **Tree**. Although XMF readers are required to use both (to avoid data loss), such duplication could be confusing to end users.
- If you use detached **Node**s (ReferenceType 3), don't duplicate the **NodeUnpackers** list in both the detached **Node** and the **Node** in the **Tree**; or, if you must, be very sure the two lists agree. Although XMF readers are required to use the detached version, such duplication could be confusing to system designers, and could cause unnecessary unpacker searches.

# AMEI/MMA
# Recommended Practice RP-031
## Type 0 and Type 1 XMF Files (SMF & DLS)

**[Abstract]**

This document describes the preferred simple method for bundling a collection of any number of SMF and DLS file images into a single XMF file. Player behavior is also specified. This document defines new XMF Standard ResourceFormatIDs, and Standard Meta-Data FieldIDs, amending the separate MMA document Specification for XMF Meta File Format (RP-030). Familiarity with the XMF meta-file format is assumed.

XMF Type 0 and Type 1 are identical, except that the SMF file images in an XMF Type 0 file are guaranteed to be streamable. Every XMF Type 0 and 1 File must contain at least one SMF file; inclusion of DLS files is optional. Reliable playback performance is achieved by requiring DLS and General MIDI capabilities in all players. There are no player API requirements. Any one SMF file image within the XMF file may be optionally designated to 'Autostart' when the XMF file is loaded for playback. SMF and DLS file images may optionally be marked for Preloading. Certain restrictions apply to the contents of the bundled SMF and DLS file images, and to the supported XMF meta-file features.

**Note:** Future Recommended Practices are expected to address the use of additional instrument formats and further resource types in XMF files. Developers are advised to check the status of this ongoing work before developing their own extensions to XMF Type 0 and 1.

## 1. Introduction

MIDI sequences stored in SMF files and performed on a combination of custom wavetable and General MIDI instruments represent a powerful, flexible, potentially interactive, and sometimes space-efficient musical medium. This medium is suitable for a broad range of musical genres, and suitable for unattended playback in end-user terminal devices such as desktop and laptop computers, web browsers, interactive televisions, and more powerful mobile devices.

This class of MIDI application implies a link between the musical event (notes) content and the custom wavetable instruments required to present it. Because the standards defining these two media types – the Standard MIDI File (SMF) and the Downloadable Sounds file (DLS) – conceive each as a separate file, the need for a data-driven bundling and linking solution has arisen.

The MMA has previously addressed this need once, in TSB Item #157, Recommended Use of the RMID File Format. However, certain expansion limitations in the approach endorsed in that Recommended Practice have led to the present proposal. The XMF meta-file fomat used in Type 0 and Type 1 XMF Files is different from the RIFF meta-file format use in the Extended RMID format, and this difference produces several functional differences between the two Recommended Practices. As a result, some developers may find the Type 0 and Type 1 XMF file formats a more general and flexible bundling solution for SMF and DLS data.

Differences between XMF Types 0/1 and Extended RMID include:

- XMF files with equivalent contents will tend to be slightly smaller than Extended RMID.
- Whereas Extended RMID supports one SMF file image and one DLS file image, XMF Type 0 and Type 1 files may optionally contain multiple SMF file images and/or multiple DLS files, arranged by the content creator into arbitrary folder hierarchies. This makes the XMF formats suitable for both archive files and presentation files. If the Extended RMID approach were to be extended to support arbitrary hierarchies or large numbers of contained files, the size difference between the two Recommended Practices would be significantly greater.
- The XMF file may incorporate by reference external SMF and DLS files, as well as SMF file images and DLS file images contained in other XMF files, via file: or http: access. This facilitates data sharing and dynamically served, netborne files (although some players may not support these URI schemes, and will have to decline to play XMF files that require them).

- XMF meta-data also allows multiple alternate contents for each field, keyed to the end user's country and language, and optionally in Unicode encoding to support all major international character sets; Extended RMID's meta-data system uses ASCII only. We plan to define guidelines for import/export interoperability with major standard meta-data systems, whereas no such guidelines have been proposed for the system used in Extended RMID.
- XMF files may contain custom meta-data fields defined by the content creator, publisher, or distributor. While this would be possible to achieve in Extended RMID, no specific mechanism has been proposed to date.
- The playback system's required capabilities and behavior are specified for XMF Type 0 and Type 1 files, including General MIDI support and Autostart and Preload functions. In Extended RMID, no playback system requirements are specified.
- XMF Type 0 and Type 1 files have a 4 GB maximum size, and a 4 GB maximum resource size limit, whereas Extended RMID files (like all RIFF-based formats) are limited to 2 GB.

# 2. File Format

XMF Type 0 and XMF Type 1 files are based on the XMF Meta-File Format, with the following further requirements:

- There are certain restrictions on what ResourceTypes are allowed to appear in the file,
- Certain newly defined meta-data fields are required, and
- Certain newly defined optional meta-data fields are available for use

To support these requirements, the Specification for XMF Meta File Format (RP-030) is hereby extended with new ResourceFormatIDs and new meta-data Standard FieldIDs.

## 2.1. About the XMF Meta File Format

XMF Type 0 and XMF Type 1 use the XMF meta-file format, which is defined in detail in the separate MMA document Specification for XMF Meta File Format (RP-030) and its amendments.

XMF is intended to support resource bundling in a low-overhead manner suitable for use in a broad range of playback platforms including very simple mobile devices (cell phones, PDAs, etc.). XMF allows multiple data blocks to be bundled into a single file, in a folder hierarchy similar to a computer file system. Each data block or folder may have any amount of meta-data associated with it, and each data block may be optionally data-compressed and optionally encrypted. Data resources in other files may be included by reference.

## 2.2. Definition of Type 0 and Type 1 XMF Files

XMF Type 0 and Type 1 are identical, except that the SMF file images in an XMF Type 0 file are guaranteed to be streamable. Differences between the two are described at sections 2.2.1. and 2.2.2., and common requirements are described at section 2.2.3.

Every XMF Type 0 and 1 File must contain at least one SMF file; inclusion of DLS files is optional.

### 2.2.1. Type 0 XMF File

XMF Type 0 is intended for SMF + DLS collections whose SMF parts can be played by streaming. Note that it is believed that for reliable playback, any needed DLS file images must be completely loaded before SMF playback begins.

A Type 0 XMF File is defined as an XMF file with the following characteristics:

- All SMF file images present in the XMF collection must be playable by streaming.
  It is believed that only SMF Type 0 data that does not include embedded looping and branching messages can be streamed; SMF Type 1 file images, and SMF Type 0 file images containing embedded looping and branching messages, cannot be streamed.

- All FileNodes contain either SMF Type 0 file images, or DLS file images, or data blocks in custom formats. No provision is made for digital audio file images (WAV, AIFF, MP3, etc.) appearing in an XMF Type 0 file, other than those appearing as components of DLS file images. It is expected that future Recommended Practices will define new XMF File Types with guidelines and behaviors for digital audio files.
- The RootNode contains one MetaDataItem (using the XMF File Type FieldID) identifying the file as a Type 0 XMF File. This is Revision 0 of Type 0, so UniversalData is 0x00 0x00.

Further characteristics are described below at heading 2.2.3.

### 2.2.2. Type 1 XMF File

XMF Type 1 is intended for SMF + DLS collections, with no expectation that the SMF parts can be played by streaming. Therefore, all needed SMF and DLS file images must be completely loaded before SMF playback begins.

A Type 1 XMF File is defined as an XMF file with the following characteristics:

- All FileNodes contain either SMF Type 1 or Type 0 file images, or DLS file images, or data blocks in custom formats.
- No provision is made for digital audio file images (WAV, AIFF, MP3, etc.) appearing in an XMF Type 1 file, other than those appearing as components of DLS file images. It is expected that future Recommended Practices will define new XMF File Types with guidelines and behaviors for digital audio files.
- The RootNode contains one MetaDataItem (using the XMF File Type FieldID) identifying the file as a Type 1 XMF File. This is Revision 0 of Type 1, so UniversalData is 0x01 0x00.

Further characteristics are described below at heading 2.2.3.

### 2.2.3. Common Characteristics for Type 0 and Type 1

Further, both Type 0 and Type 1 XMF Files must have the following characteristics:

- Conformance to the separate Specification for XMF Meta File Format (RP-030) document.
- The contained file images may be freely arranged into files and folders however desired, to any nesting depth, as described in the Specification for XMF Meta File Format.
- Subject to the limitations described in this document, each FileNode or FolderNode in the XMF file may optionally include in its NodeMetaData any number of MetaDataItem, as defined in the Specification for XMF Meta File Format.
- As per the Specification for XMF Meta File Format, each FileNode containing resource data must indicate the resource format by including in its NodeMetaData one ResourceFormatID MetaDataItem (Standard Meta-Data field number 2), using an appropriate ResourceFormatID.
- Note: Use of FileNodes containing data in custom formats may reduce content portability, as not all players will be able to recognize and use that data.
- As per the Specification for XMF Meta File Format, a FileNode may refer to resource data in a separate file via the file: or http: URI schemes (see ReferenceTypeIDs 4 and 5). However, for Type 0 and Type 1 XMF files only the following data file types may be referenced in this way:
  - Separate SMF files
  - SMF file images stored in other XMF files
  - Separate DLS files
  - DLS file images stored in other XMF files
  - Separate custom data files
  - Custom data file images stored in other XMF files

  No provision is made for digital audio file images (WAV, AIFF, MP3, etc.) referenced from an XMF Type 0 or Type 1 file, other than those appearing as components of referenced DLS file images. It is expected that future Recommended Practices will define new XMF file Types with guidelines and behaviors for digital audio files.

**Note:** Use of the file: or http: URI schemes may reduce content portability, as some players may not have file systems or support web access. A player encountering an XMF file which depends on the use of a URI scheme that the player does not support must decline to play that file. If appropriate in the intended user experience, the player should report this error condition to the user (e.g. 'External http: access not supported' or 'External file: access not supported").

## 2.3. XMF Standard ResourceFormatID Definitions for SMF and DLS

Every FileNode in an XMF Type 0 or Type 1 file that contains an SMF or DLS file image must include in its NodeMetaData one MetaDataItem identifying its data's resource format. The FieldSpecifier for this MetaDataItem must be Standard FieldID 2 (Resource Format), and the FieldContents must be the appropriate Standard ResourceFormatID from the following list. No other Standard ResourceFormatIDs may appear in the XMF file (see previous notes on digital audio files).

Standard ResourceFormatIDs may only be assigned by the MMA.

| ResourceFormat | Interpretation |
|---|---|
| 0 | Standard MIDI File (SMF), Type 0 |
| 1 | SMF, Type 1 |
| 2 | Downloadable Sounds (DLS), Level 1 |
| 3 | DLS, Level 2 |
| 4 | DLS, Level 2.1 |

**Note:** This Recommended Practice hereby adds the preceding Standard ResourceFormatIDs to the Specification for XMF Meta File Format (RP-030), amending that document.

## 2.4. XMF Meta-Data Standard Fields for Type 0 and Type 1 Files

XMF Type 0 and Type 1 files introduce two new XMF Standard Meta-Data FieldIDs, as detailed in this section.

| FieldID | Field Name and Notes | Valid for Node Types | Contents Format |
|---|---|---|---|
| 11 | Autostart<br>Node Name of the FileNode containing the SMF image to autostart when this XMF file loads | File or Folder, but only allowed in the RootNode | Universal<br>Extended ASCII<br>(hidden or visible) |
| 12 | Preload | File | Universal,<br>No contents – just<br>0x00 0x00 |

**Note:** This Recommended Practice hereby adds the preceding Meta-Data Standard FieldIDs to the Specification for XMF Meta File Format (RP-030), amending that document.

### 2.4.1. Meta-Data FieldID 11: Autostart

The Autostart meta-data field is optional, and is intended to allow the content creator to define what 'play' means for an XMF file that contains multiple SMFs, by designating exactly one of the SMFs as the 'Autostart' target.

This field may appear in the RootNode of a Type 0 or Type 1 XMF File. This field is only valid in the RootNode; if the reading parser encounters it elsewhere, it must not process it as an autostart indicator.

The FieldContents of an Autostart MetaDataItem is a Universal Extended ASCII string containing the Node Name of one SMF file image to be automatically started when the player loads the XMF file. The desired SMF FileNode must contain one MetaDataItem with the matching Node Name (Standard FieldID 1).

Parsers should ignore any Autostart item that does not target an SMF file image.

### 2.4.2. Meta-Data FieldID 12: Preload

The Preload meta-data field is optional, and may be used to pre-load specific SMF and DLS file images into the player's playback sequencer and/or DLS synth device, in order to make them immediately available for use upon a subsequent event from a user or program. For DLS this means making the instruments available in the synthesizer. For SMF this means preparing the sequence for playback, but not actually starting playback.

This field may appear in any FolderNode without restriction, and in any FileNode that describes a DLS or SMF file image. If a Preload field appears in a FolderNode, the player should preload all SMF and DLS files in that folder and in any of its descendents. If a Preload field appears in a FileNode that describes a custom data format then behavior is left to the player's discretion, in order to preserve flexibility for custom extensions.

The FieldContents of a Preload MetaDataItem is empty, as no parameter is needed. That is, UniversalContents is two VLQs, both with value 0: **0x00 0x00**.

## 2.5. VLQ Maximum Values

In XMF Type 0 and Type 1, VLQs shall not exceed the following maximum values. These limits are intended to guide parser implementors as to what size integer to use when reading each VLQ.

| VLQ | Max Value | Min Bits Unsigned Integer |
|---|---|---|
| FileHeader: | | |
| FileLength | 4 GB (4,294,967,295) | 32 |
| TreeStart | 4 GB (4,294,967,295) | 32 |
| TreeEnd | 4 GB (4,294,967,295) | 32 |
| NodeHeader: | | |
| NodeLength | 4 GB (4,294,967,295) | 32 |
| NodeContainedItems | 65,535 | 16 |
| NodeHeaderLength | 4 GB (4,294,967,295) | 32 |
| NodeMetaData: | | |
| LengthInBytes | 4 GB (4,294,967,295) | 32 |
| Standard FieldID | 65,535 | 16 |
| Custom Field Specifier (XString length) | 65,535 | 16 |
| FieldContents: | | |
| length for UniversalContents | 4 GB (4,294,967,295) | 32 |
| NumberOfStrings for | 65,535 | 16 |
| MetaDataType for InternationalContents | 65,535 | 16 |
| VersionData length for | 4 GB (4,294,967,295) | 32 |
| NodeUnpackers: | | |
| ListSizeInBytes | 65,535 | 16 |
| DecodedSize | 4 GB (4,294,967,295) | 32 |
| NodeContents: | | |
| ReferenceTypeID | 5 | 3 |
| MetaDataTypesTable: | | |
| LengthInBytes | 65,535 | 16 |
| NumberOfEntries | 65,535 | 16 |
| MetaDataType in TypeEntry | 65,535 | 16 |
| StringFormatTypeID | 7 | 3 |

## 3. SMF and DLS Content Requirements

The XMF Type 0 and Type 1 instrument access scheme requires slight limitations on the contents of the included SMF and DLS files, as detailed below. XMF file creation tools should enforce these rules where possible.

The General MIDI system is used to support common instruments, and the DLS system is used to support custom instruments and audio premixes including recorded performances. SMF content may be authored for GM1 or GM2, however the presence of a GM2 synth in the XMF player is not guaranteed. DLS content may be authored for DLS Level 1 or DLS Level 2, relying on DLS' fallback mechanism.

### 3.1. SMF Content

SMF content is unrestricted, except that:

- Every Track in an SMF will use either General MIDI instruments supplied by the playback system, or custom instruments in DLS form, and supplied via the XMF file. Each Track may independently choose either option. Within GM, both GM1 and GM2 can be used. Therefore, every SMF Track must start by specifying that choice using an XMF Patch Type Prefix Meta event (**0xFF 0x60 <len> <param>**). This SMF Meta event is defined in the separate document SMF Meta Event for XMF Patch Type Prefix (RP-032).
    - The General MIDI 1 instrument set (and GM1 system behavior) is chosen by default, so no initial message is required to select GM1. The XMF Patch Type Prefix Meta event selecting GM1 (parameter 0x01) at the start of a Track is permitted, but redundant: **0xFF 0x60 0x01 0x01**.
    - If a Track has been written to take advantage of the General MIDI 2 instrument set and/or controller responses, the Track should begin with an XMF Patch Type Prefix Meta event selecting GM2 (parameter 0x02): **0xFF 0x60 0x01 0x02**.
    - If a Track has been written for the custom DLS instruments supplied via the XMF file, the Track should begin with an XMF Patch Type Prefix Meta event selecting DLS (parameter 0x03): **0xFF 0x60 0x01 0x03**.
- No SMF Track may be reassigned to a different instrument set (GM1, GM2, or DLS) at any time. Therefore, the SysEx messages GM1 System On, GM2 System On, Turn DLS On, Turn DLS Off and GM System Off should not appear in any SMF Track, and must be ignored by players. XMF Patch Type Prefix Meta events will only be processed if they appear as the first message in an SMF Track; if they appear anywhere else in an SMF Track, they must be ignored.

### 3.2. DLS Content

Any XMF file that requires any DLS instruments must supply those instruments via the XMF file.

The content of DLS file images in an XMF Type 0 or Type 1 file is unrestricted, except that to avoid presenting the player with ambiguities, no Bank/Program assignments in any DLS file image in (or referred to from) the XMF file may overlap or conflict with those of any other DLS file image in (or referred to from) the same XMF file.

Due to the custom of numbering programs from zero in DLS files, content creators using multiple DLS file images in a given XMF file should exercise care to avoid overlapping program numbers.

## 4. Playback System Specification

To maximize the reliability of XMF Type 0 and XMF Type 1 playback across different platforms and players, all players are required to support DLS and General MIDI. The player's expected behavior for a simple 'load-and-play' operation is defined.

### 4.1. Required MIDI Playback Capabilities

This sections details the MIDI Playback capabilities required for XMF Type 0 and XMF Type 1 files.

The General MIDI system is used to support common instruments, and the DLS system is used to support custom instruments and audio premixes including recorded performances. SMF content may be authored for GM 1 or GM 2. DLS content may be authored for DLS Level 1 or DLS Level 2, relying on DLS' fallback mechanism.

The implementation of any required part may be in software, hardware, or any combination thereof. There is no requirement for an Application Programming Interface (API) for any of the parts.

General MIDI player requirements are described in the Recommended Practice document General MIDI System Level 1 and related documents. DLS player requirements are described in the Recommended Practice documents DLS Level 1.0, DLS Level 2.0, and DLS Level 2.1.

#### 4.1.1. Required MIDI Capabilities for XMF Type 0 Files

To play a Type 0 XMF File, the playback system must include an SMF playback sequencer and supporting technology able to play an SMF Type 0 file image by streaming from a local file system via the URI file: protocol, or from a server via the URI http: protocol.

Streaming playback is intended as an optional feature to support quicker player start-up, not a required behavior for playing Type 1 XMF Files. The player may wait until an XMF Type 0 file is completely received before beginning playback, if desired.

Further characteristics are described below at heading 4.1.3.

#### 4.1.2. Required MIDI Capabilities for XMF Type 1 Files

To play a Type 1 XMF File, the playback system must include an SMF playback sequencer able to play a locally queued SMF Type 0 or SMF Type 1 file image.

Streaming playback is not required.

Further characteristics are described below at heading 4.1.3.

#### 4.1.3. Common MIDI Requirements for XMF Type 1 and Type 0

Further, to play a Type 1 or Type 0 XMF File, the playback system must have the following minimum MIDI instrument rendering capabilities:

- MIDI synth device(s) conforming to the General MIDI specifications. GM 1 is required; GM 2 is encouraged but optional. All GM Instrument data must be supplied or managed by the player, so that the XMF content can rely on the GM instruments without having to supply them directly in the XMF file. A DLS device may be used for GM rendering, at the player implementor's discretion.
- When SMF content is authored for GM2 but only a GM1 synth is available, it is recommended, but optional, for the XMF player to filter the MIDI stream appropriately, in order to prevent generation of obviously bad sounds.
- MIDI synth device(s) conforming to the DLS specifications, for custom DLS instruments supplied via the XMF file. DLS Level 1 is required; DLS Level 2 and 2.1 are encouraged but optional.
- MIDI connection between the SMF playback sequencer tracks and the synth(s).

### 4.2. Required Behavior

The only required behavior for an XMF Type 0 or XMF Type 1 file player is the procedure for loading and starting the file, which includes the Autostart and Preload features.

The player is not required to provide any automatic handling of, or specific behavior with regard to, any SMF and DLS file images other than those marked for Preload and Autostart. It is expected that content and software developers who wish to include multiple SMF and multiple DLS file images will have their own purposes in mind – such as archiving or interactive variation selections – and will use their own API-level commands to retrieve that content from their XMF files.

### Loading an XMF Type 0 or Type 1 File and Starting Playback

Upon loading a Type 0 or Type 1 XMF file for playback, the playback system must take the following actions, in the indicated order.

1. Before loading the DLS device with any DLS file images from an XMF file, all previously loaded DLS instruments must be unloaded. This prevents incorrect instruments from being used inadvertently.
2. Load the DLS synth with all DLS files marked for Preload. (See Standard Meta-Data field number 12). For non-streaming SMFs, players may intelligently load only those instruments that the SMF calls for.
3. Load the SMF playback sequencer with all SMF file images marked for Preload or Autostart. (See section 2.4). Only one Autostart SMF is allowed.
4. Make MIDI connections between SMF tracks and synth devices, according to the initial synth assignment XMF Patch Type Prefix Meta event in each track of the SMF. (See section 3.1.)
5. If an SMF file image was marked for Autostart in the XMF file (see section 2.4), have the playback sequencer start it. If no Autostart SMF was indicated in the XMF file the player should not start any SMF playback, but rather stand ready for the user or program to issue a Play command (via user interface or API) at any time.
   For Type 0 XMF Files, the player may at its own election stream the SMF file image from its disk file or server, rather than delaying the start of playback until the entire SMF file image has been received.
6. The initial XMF Patch Type Prefix Meta events used to assign each SMF Track to a synth device should be processed only once at routing setup time, and not processed again during playback.

## 5. External File Type Identifiers

The preferred external identifiers for XMF Type 0 and Type 1 files are as follows:

| | XMF Type 0 | XMF Type 1 |
|---|---|---|
| Filename Extension | **`.xmf0`** | **`.xmf1`** |
| MIME Type | **`audio/xmf0`** | **`audio/xmf1`** |
| Mac OS File Type | **`Xmf0`** | **`Xmf1`** |

The use of filename extensions is recommended for all XMF Type 0 and Type 1 files irrespective of platform, in order to preserve external file type infomation should the file be transferred to a Windows computer.

On Mac OS computers the Mac OS File Types should be used.

For Internet-hosted XMF files the MIME Types should be used.

#### Examples

```
myStreamableXmfFile.xmf0
Unstreamable.xmf1
```

MIDI Manufacturers Association
Los Angeles CA
www.midi.org

# AMEI/MMA
# Recommended Practice RP-032
## SMF Meta-Event for XMF Patch Type Prefix

**[Abstract]**

XMF Type 0 and Type 1 files contain Standard MIDI Files (SMF). Each SMF Track in such XMF files may be designated to use either standard General MIDI 1 or General MIDI 2 instruments supplied by the player, or custom DLS instruments supplied via the XMF file. This document defines a new SMF Meta-Event to be used for this purpose.

**[XMF Patch Type Prefix Meta-Event]**

The XMF Patch Type Prefix Meta-Event is defined as follows:

```
FF 60 <len> <param>
```

In a Type 0 or Type 1 XMF File, this meta-event specifies how to interpret subsequent Program Change and Bank Select messages appearing in the same SMF Track: as General MIDI 1, General MIDI 2, or DLS. In the absence of an initial XMF Patch Type Prefix Meta-Event, General MIDI 1 (instrument set and system behavior) is chosen by default.

In a Type 0 or Type 1 XMF File, no SMF Track may be reassigned to a different instrument set (GM1, GM2, or DLS) at any time. Therefore, this meta-event should only be processed if it appears as the first message in an SMF Track; if it appears anywhere else in an SMF Track, it must be ignored.

| | |
|---|---|
| `<param> = 0x01` | General MIDI 1. GM1 is chosen by default, so starting an SMF Track with this meta-event selecting GM1 is redundant, but still permitted. Instruments will be automatically supplied and managed by the player, not supplied in the XMF file. Syntax: `[FF 60 01 01]` |
| `<param> = 0x02` | General MIDI 2. The SMF Track has been written to take advantage of the General MIDI 2 instrument set and/or controller responses. Instruments will be automatically supplied and managed by the player, not supplied in the XMF file. If GM2 is not available, GM1 will be used. Syntax: `[FF 60 01 02]` |
| `<param> = 0x03` | DLS. The SMF Track has been written for the custom DLS instruments supplied via the XMF file. Instruments will be supplied via the XMF file, not supplied by the player. Syntax: `[FF 60 01 03]` |

**[Relationship to System Exclusive Messages]**

In a Type 0 XMF File or a Type 1 XMF File, no SMF Track may be reassigned to a different instrument set (GM1, GM2, or DLS) at any time after the initial XMF Patch Type Prefix Meta-Event. Therefore, the SysEx messages GM1 System On, GM2 System On, Turn DLS On, Turn DLS Off and GM System Off should never appear in the same SMF Track as an XMF Patch Type Prefix Meta-Event, and must be ignored by players if they do appear.

MIDI Manufacturers Association
Los Angeles CA
www.midi.org

# MMA Technical Standards Board/ AMEI MIDI Committee

Recommend Practice (RP-039)

**XMF Meta File Format Updates 1.01**

**Abstract:**

These 9 Updates to **XMF Meta File Format** (RP-030) and **Type 0 and Type 1 XMF Files** (RP-031) are for purposes of editorial clarification and technical enhancement, and increment the version number of each specification.

**Background:**

These changes are motivated by implementation experiences since the initial XMF Specifications were approved; background for each change is discussed separately in the proposal.

## Contents

# 1. Updates to "Specification for XMF Meta File Format" (RP-030)

There are a total of 5 changes for the Specification for XMF Meta File Format, 2 technical and 3 editorial.

## 1.1 Add New ReferenceTypeID 6: XMF File URI and Node ID Number

### 1.1.1 Discussion

Currently the only way for an XMF file node to point to a file image stored in another XMF file is with ReferenceTypeID 5, which uses the URI of the XMF file and the node name (which must match the contents of a Node Name metadata field). In some applications it would be more efficient to identify the target node by Node ID Number metadata field, rather than by name. This new ReferenceTypeID provides a way to do this that works for both external XMF files and nodes in the same XMF file. ReferenceTypeID 6 implements this in a manner similar to ResourceTypeID 5, but with a Node ID Number instead of a Node Name. In other words, ReferenceTypeID 6 uses two parameters: URI of the XMF file in XString form, and the target node ID number in VLQ form.

### 1.1.2 Spec Text Changes

The text needs to be changed in two places:

- In section 2.2.1.2.1, insert new table entry for ReferenceTypeID 6:

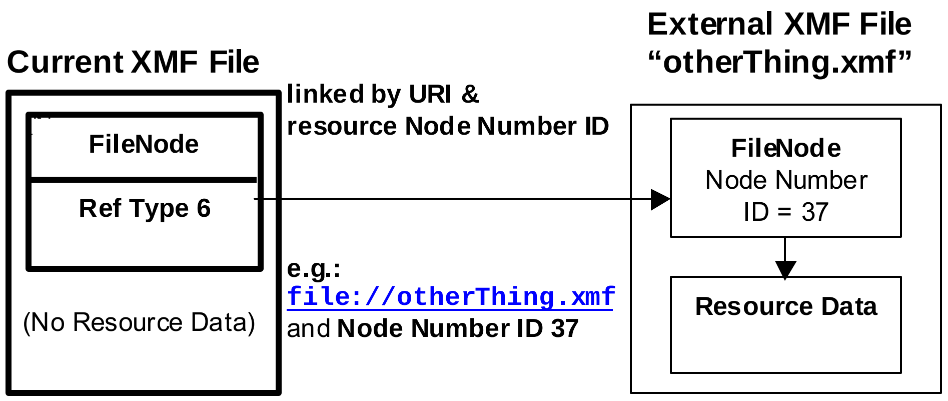
| ID | Description | Description | Format |
|---|---|---|---|
| 6 | **XMF File URI and Node ID Number** | URI* of an external XMF file**, plus a VLQ integer specifying the target resource by Node ID Number, e.g. **`file://myOtherFile.xmf`** and Node ID Number **37**<br>**Or**<br>Empty URI (meaning look in the same XMF file), plus a VLQ integer specifying the target resource by node ID, e.g. **""** and Node ID Number **37** | URI: **XString**<br>Node ID number: **VLQ** |

- In section **2.2.1.2.1**, insert new description of ReferenceTypeID 6 after description of ReferenceTypeID 5:

**<begin new text>**

**ReferenceTypeID 6: XMF File URI and Node ID**

**ReferenceTypeID 6** points to resources stored in (or referenced by) the same XMF file or external XMF files, identifying the file via URI (same schemes), but identifying the resource by Node Number ID, in **VLQ** form. For references to nodes in the same file, the URI will be empty (**XString** will have length **VLQ** of zero and no text characters). Use of the hash [#] convention in the URI is not permitted as that could conflict with the Node ID Number. For a type 6 reference to succeed, the target resource must include a Node ID Number field in its meta-data, whose contents must match the referring Node Number ID. **ReferenceTypeID** 6 has all the advantages of **ReferenceTypeID** 5, but in some applications may be more efficient – and as long as the numbers agree, the link will survive all changes to the target file's structure and other contents.



**Typical Use Cases:** This reference type has all the same resource sharing and aliasing benefits as ReferenceTypeID 5, but with better efficiency in some applications.

> **Note:** In **Nodes** using **ReferenceTypeID** 6, the **NodeUnpackers** in the referenced Node override any **NodeUnpackers** in the referring node. Any **NodeUnpackers** indicated in the referring node should be ignored.
>
> **Note:** The target node will not necessarily hold the target resource directly, as it has a **ContentReference** of its own. However, to avoid excessively long or circular seek chains, the target resource data must be found within 4 reference indirections, otherwise the XMF parser must fail and return a 'Too many XMF indirections' error.

**<end new text>**

#### 1.1.3 Effect of Changes

Existing parsers conforming to version 1.00 of the meta file format spec will not support the new feature. Existing XMF files conforming to version 1.00 of the meta file format spec will be completely readable by parsers conforming to version 1.01.

## 1.2 Note Limitations of ReferenceTypeID 2

#### 1.2.1 Discussion

For the same reasons as change 1.1, the following editorial note should be added, pointing out the issue with ReferenceTypeID 2:

#### 1.2.2 Spec Text Changes

In section **2.2.1.2.1**, insert the following text at start of description of ReferenceTypeID 2:

> "**Note: ReferenceTypeID** 2 should only be used with resource formats that include length delimiters. This is because **ReferenceTypeID** 2 does not provide any indication of the length of the target data. As a result, with non-length delimited resource formats there is no way to tell where the target resource ends. For non-inline storage of resource formats that are not length delimited, it is better to store the resource in a separate **Node** (perhaps detached from the Tree) and refer to that **Node** via **ReferenceTypeID** 3, **In-File Node**."

#### 1.2.3 Effect of Changes

No technical effects – editorial note only.

## 1.3 Allow ReferenceTypeID 5 to Refer to Nodes in Same File

### 1.3.1 Discussion

Currently ReferenceTypeID 5 uses a URI to point to a file image stored in another XMF file, by Node Name. (Syntax is "`http://theOtherXmfFilename.xmf#theNodeName`".) This should be extended to allow the URI to refer to a file image stored in the same XMF file. The syntax is just to drop the scheme and filename parts of the URI string, e.g. just the hash and node name; for example "`#theNodeName`". As a result, the name of this ReferenceTypeID should be changed from 'External XMF Resource' to 'XMF File URI' to reflect the fact that the target node is no longer necessarily external. We should also clarify that Node Names intended to be used as targets for ReferenceTypeID 5 nodes must contain only URI-legal characters.

### 1.3.2 Spec Text Changes

The text needs to be changed in two places:

1. In section **2.2.1.2.1** replace table entry for ReferenceTypeID 5:

| ID | Description | Description | Format |
|---|---|---|---|
| 5 | External XMF Resource | URI* of an external XMF file**, including specific resource name, e.g.: **`file://myOtherFile.xmf#myResource`** | XString |

With this entry:

| ID | Description | Description | Format |
|---|---|---|---|
| 5 | XMF File URI | URI* of a specific resource in an external XMF file**, e.g.: **`file://myOtherFile.xmf#myResource`** or in the same XMF file, e.g.: **`#resourceInSameXmfFile`** | XString |

2. In section 2.2.1.2.1, at ReferenceTypeID 5: External XMF Resource:,

- Replace section heading text:
  "**ReferenceTypeID 5: External XMF Resource**"
  With this text: "**ReferenceTypeID 5: XMF File URI**"
- At the end of the first descriptive paragraph, insert this text:
  "To reference a node in the same XMF file, omit the scheme and filename parts of the URI string (example: '`#nodeInMySameXmfFile`')."
- At the end of the ReferenceTypeID description, insert this text:
  "**Note:** Node Names intended to be used as targets for ReferenceTypeID 5 nodes must contain only URI-legal characters."

### 1.3.3 Effect of Changes

Existing parsers conforming to version 1.00 of the meta file format spec will not support the new feature. Existing XMF files conforming to version 1.00 of the meta file format spec will be completely readable by parsers conforming to version 1.01.

## 1.4 Clarification of Byte Ordering for 16-bit Unicode Metadata

### 1.4.1 Discussion

The version 1.00 spec incorrectly states "It is not required to start the Unicode contents with a Byte Order Mark (`0xFEFF`), because the byte order of the entire XMF file may be determined from the first two bytes of the FileHeader structure". An editorial change is needed to clarify how to produce correct Unicode contents.

### 1.4.2 Spec Text Changes

In section **3.2.2.1. StringFormatTypeID Definitions**, replace this text:

> "Unicode Format -- Unicode contents for StringFormatTypeIDs 2 and 3 must conform to version 2.0 of the Unicode standard, in UTF-16 coding. It is not required to start the Unicode contents with a Byte Order Mark (`0xFEFF`), because the byte order of the entire XMF file may be determined from the first two bytes of the FileHeader structure (section 2.1)."

With this text:

> "Unicode Format -- Unicode contents for StringFormatTypeIDs 2 and 3 must conform to version 2.0 of the Unicode standard, in UTF-16 coding with big endian byte ordering. It is not required to start the Unicode contents with a Byte Order Mark (`0xFEFF`), as StringFormatTypeIDs 2 and 3 will always be in big endian byte order. It is not required to terminate the Unicode contents with a null string terminator as the Unicode contents are preceded by a VLQ. XMF Writers on little endian platforms must byte swap StringFormatTypeID 2 and 3 contents when writing XMF files. XMF Readers on little endian platforms must byte swap StringFormatTypeID 2 and 3 contents when reading XMF files."

### 1.4.3 Effect of Changes

There is a possibility that metadata contents for some incorrect existing XMF files will be unreadable by some XMF parsers, irrespective of version; clarifying the specification now reduces the likelihood of similar problems in the future.

## 1.5 Update Specification Version Number

### 1.5.1 Discussion

The foregoing changes necessitate an increment in the version number.

### 1.5.2 Spec Text Changes

The text needs to be changed in three places:

- On title page, and all page footers, replace this text: "Version 1.00"

With this text: "Version 1.01"

- 2. In section **2.1. FileHeader Structure**, replace all occurrences of this text: "Version 1.00"

With this text: "Version 1.01"

- In **Appendix: Notes on the XMF File Format**, in the detailed description of the Node structure (page 39), replace this text: "for XMF version 1.00, but"

With this text: "for XMF metafile format versions 1.00 and 1.01, but"

### 1.5.3 Effect of Changes

No technical effect.

# 2. Updates to “Type 0 & Type 1 XMF Files” (RP-031)

There are a total of 4 changes for the XMF Meta File Format Specification, 2 technical and 2 editorial.

## 2.1 Resolve Contradiction Regarding Preload Metadata

### 2.1.1 Discussion

In section **2.4. XMF Meta-Data Standard Fields for Type 0 and Type 1 Files** the table entry for the Preload metadata field incorrectly omits the fact that a Preload metadata item may be attached to certain Folder nodes, not just File nodes.

Presently the table states:

| FieldID | Field Name and Notes | Valid for Node Types |
|---|---|---|
| 12 | Preload | File |

However, this is contradicted by section **2.4.2. Meta-Data FieldID 12: Preload** which says "This field may appear in any FolderNode without restriction, and in any FileNode that describes a DLS or SMF file image. If a Preload field appears in a FolderNode, the player should preload all SMF and DLS files in that folder and in any of its descendents."

Clearly "Folders" must be added to the table, but it is important to also avoid the impression that any arbitrary file can be marked for Preload.

### 2.1.2 Spec Text Changes

In section **2.4. XMF Meta-Data Standard Fields for Type 0 and Type 1 Files**, change the table entry for the Preload metadata item from this:

| FieldID | Field Name and Notes | Valid for Node Types |
|---|---|---|
| 12 | Preload | File |

To this:

| FieldID | Field Name and Notes | Valid for Node Types |
|---|---|---|
| 12 | Preload | File or Folder (see section 2.4.2.) |

### 2.1.3 Effect of Changes

Existing parsers conforming to revision 0 of the Type 0 & Type 1 spec may be incorrectly failing to apply inheritance for the Preload metadata field for Type 0 & Type 1 files containing Folder nodes with the Preload metadata item. Clarifying the spec now reduces the likelihood of similar problems in the future. Existing parsers correctly conforming to revision 0 of the Type 0 & Type 1 spec will properly handle all new Type 0 & Type 1 files conforming to revision 1.

## 2.2 Reflect New XMF Meta File Format Specification Version

### 2.2.1 Discussion

In section **2. File Format**, use of the **Specification for XMF Meta File Format** is required without mentioning which version should be used. It should be clarified that either v1.00 or v1.01 may be used.

### 2.2.2 Spec Text Changes

In section **2. File Format**, insert the following text just before section 2.1. About the XMF File format:

> “Note: Either v1.00 or v1.01 of the **Specification for XMF Meta File Format** may be used.”

### 2.2.3 Effect of Changes

Parsers conforming to revision 0 of the Type 0 & Type 1 spec will not be able to handle any of the new features in version 1.01 of the **Specification for XMF Meta File Format**, including ReferenceType 6. However these features only appear in revision 1 of the Type 0 & Type 1 spec, so revision 0 parsers are able to detect and reject revision 1 files. All revision 1 parsers will be able to handle all revision 0 files.

## 2.3 Reflect New ReferenceTypeID 6

### 2.3.1 Discussion

In section **2.5. VLQ Maximum Values** the table entry for the ReferenceTypeID field of the NodeContents structure states the maximum allowable value is 5, but with the introduction of ReferenceTypeID 6 in the **Specification for XMF Meta File Format v1.01**, this must be increased to 6.

### 2.3.2 Spec Text Changes

In section 2.5. VLQ Maximum Values, change the table entry for ReferenceTypeID from this:

| VLQ | Max Value | Min Bits Unsigned Integer |
|---|---|---|
| ReferenceTypeID | 5 | 3 |

To this:

| VLQ | Max Value | Min Bits Unsigned Integer |
|---|---|---|
| ReferenceTypeID | 6 | 3 |

### 2.3.3 Effect of Changes

No technical change.

## 2.4 Update Specification Version Number

### 2.4.1 Discussion

The foregoing changes necessitate an increment in the spec version number, which is reflected in the RevisionID field of the XMF File Type metadata item.

### 2.4.2 Spec Text Changes

Before the [Abstract], insert this text: "Revision 1 (RevisionID = 0x01)"

### 2.4.3 Effect of Changes

No technical effect.

# MMA Technical Standards Board/ AMEI MIDI Committee

Recommend Practice (RP-040)

## XMF Compression Definition for ZLIB

**Abstract:**

This RP defines a new XMF Standard UnpackerID for “ZLIB” compression.

**Background:**

During preparation of the initial versions of the XMF specifications, no Standard UnpackerIDs were defined. At that time, the XMF Working Group found it prohibitive to designate any Unpackers as standard, given that most of the available methods for the purpose are either proprietary, non-portable, and/or legally encumbered.

ZLIB claims that it is designed to be a free, general-purpose, legally unencumbered -- that is, not covered by any patents -- lossless data-compression library for use on virtually any computer hardware and operating system. The MMA is not warranting such claims, and is including a disclaimer as part of this Recommended Practice.

**Details:**

**DISCLAIMER**

The fact that the MMA XMF specification has a field that allows specification of certain compression algorithms is not in any way a representation, warranty, or endorsement regarding any compression algorithm, nor is it a representation or warranty that the vendor of any such compression algorithm is the owner of the intellectual or other property rights involved therein. Rather, the use of any compression algorithm to which a value has been assigned in the appropriate field in the XMF specification is solely at the user's risk.

## 1. XMF Standard UnpackerID for zlib

A Standard UnpackerID of 0x01 will indicate the Node is packed using zlib compression (refer to section 5.1 of the Specification for XMF Meta File Format).

Example:

```
0x00    // Standard UnpackerID follows
0x01    // zlib Standard UnpackerID
```

## 2. zlib Usage in XMF

The intent of zlib as an XMF Unpacker is for the data compression (size reduction) of a File Node. Refer to the Specification for XMF Meta File Format, Sections 2.2.1.1.1 and 5.1 for the prescribed usage of an Unpacker.

Note: In XMF Files, FolderNodes appearing in the Tree must not use any unpackers that would render the folder's contained Nodes illegible. This rule arises because otherwise we'd need to invent a way to describe how the Tree continues on the inside of that 'black box'. If you wish to 'black box' encode a folder, however, it is possible to store it outside the Tree, as a detached Node (see section 2.2.1.2.1 of the XMF Meta File Format), and point the FolderNode's ContentReference at the detached Node.

Along with the freely available zlib code library and efforts to create code libraries and projects for nearly all available operating systems, zlib should be fairly easy for most computer based XMF parsers to implement. The zlib source code is available in C language, which should make it quite possible for implementation in embedded systems that support C compilation.

It should be noted that zlib uses a lossless general-purpose compression algorithm, and is not optimized for audio files. For example, it will typically not perform well on .WAV files. The expected usage for zlib within XMF would be to compress SMF files and other types of included data, such as text files, that would benefit from zlib's lossless type of compression.

## 3. zlib Resources

All documentation, source code, libraries, and projects for zlib can be found on the zlib website: http://www.zlib.org

## 4. zlib License

Please refer to http://www.zlib.org/zlib_license.html for the latest version of the zlib license. For MMA members' reference at balloting time, the current license text is attached.

# MMA Technical Standards Board/ AMEI MIDI Committee

Recommended Practice — September 2004

## XMF Meta File Format 2.0 (RP-043)

**Abstract:**

The XMF Meta File Format 2.00 specification defines a new version of the XMF Meta File Format. The only difference from the v.1.01 spec is that two new fixed-size, fixed-location fields have been added to the FileHeader. These simplify determination of the XMF File Type for non-XMF parsers, for example in automatic association of MIME Media Type in web server software. These fields correspond directly to the contents of existing standard meta-data field 'XMF File Type' (FieldID 0).

**Background:**

The IETF MIME Media Type application form asks for easily findable 'magic numbers' that uniquely indicate a file's media type. Under earlier XMF Meta File Format specs, the XMF File Type is encoded as XMF meta-data, and therefore does not appear at a known offset into the file, so an XMF parser is required to access the information. This is considered unnecessarily burdensome, and for Mobile XMF (RP-042) simple MIME type determination was considered very important. This specification enhancement solves the problem in a manner that will work not just for Mobile XMF, but also for all future XMF File Types.

**Details:**

Version 2.00 of the XMF Meta File Format is identical to version 1.01, with the following exceptions:

1. In the FileHeader structure, the XmfMetaFileFormatVersion is changed from '1.01' to '2.00'
2. In the FileHeader structure, two 4-byte big-endian fields are inserted between the XmfMetaFileFormatVersion field and the FileLength field:

   – an XmfFileTypeID field, and
   – an XmfFileTypeRevisionID field.

   Together, these two fields serve the same purpose as the standard meta-data field 'XMF File Type' (standard FieldID 0), and they have the same interpretation as the contents of that meta-data field.

Designers of XMF file types using version 2.00 of the XMF Meta File Format should consider whether to allow, deprecate, or forbid use of the Standard meta-data field 'XMF File Type' (Standard FieldID 0), as it is redundant to the new FileHeader fields introduced in version 2.00.

**Example:**

Using the Mobile XMF File Type (v1.0) as an example, the revised FileHeader structure for XMF Meta File Format v2.00 is:

```
       FileID                      'XMF_' // 4 bytes
       XmfMetaFileFormatVersion    '2.00' // 4 bytes
New:   XmfFileTypeID               0x00000002  // 4 bytes: 2: XMF Type 2 file (Mobile XMF)
New:   XmfFileTypeRevisionID       0x00000001  // 4 bytes: 1: Version 1 of Mobile XMF spec
       FileLength                  <VLQ>
       MetaDataTypesTable          <length delimited data structure>
       TreeStart                   <VLQ>
       TreeEnd                     <VLQ>
```

# Mobile XMF Content Format

**Version 1.0b**

**December 2006**

**Published By:**

The MIDI Manufacturers Association

Los Angeles, CA

**Changes for Version 1.0a:**

- **Fixed “Max Value” errors in “VLQ Field Maximum Size and Values” table on Page 6.**

**Changes for Version 1.0b:**

- **Added text after Section 6.1 and before Section 6.2 (RP-042a)**
- **Reformatted page layout**

**RP-042a** Mobile XMF Content Format Specification

Printed 2007

MMA
POB 3173
La Habra CA 90632-3173

# Contents

continues…

# Contents Continued

## 1. Introduction

This document specifies in detail all aspects of containment and external content type indication that must be addressed in Mobile DLS compound documents, i.e. content that bundles SMF with Mobile DLS instruments.

## 2. Definitions of Terms

**Mobile DLS –** Specification for waveform-based musical instrument content format. See definition in [7].

**Player** – Hardware or software that reads files in the specified compound document format, parses the compound document, and plays the content.

**Renderer** – Hardware or software MIDI synthesizer or sampler for which instrument definition data is being supplied, and to which the SMF's MIDI events will be played.

**SMF** – Acronym for Standard MIDI File (see below). See definition in [5].

**SP-MIDI –** Acronym for Scalable Polyphony MIDI. See [10]

**Standard MIDI File** – Specification for MIDI message content format. See definition in [5].

**VLQ –** Variable Length Quantity, an integer number representation format used in XMF files. See definition in section 4.1 of [1]. Note that the VLQ definition for XMF is slightly different from the VLQ definition used in Standard MIDI Files (see [5]), however all SMF VLQs will be correctly interpreted by an XMF VLQ reader.

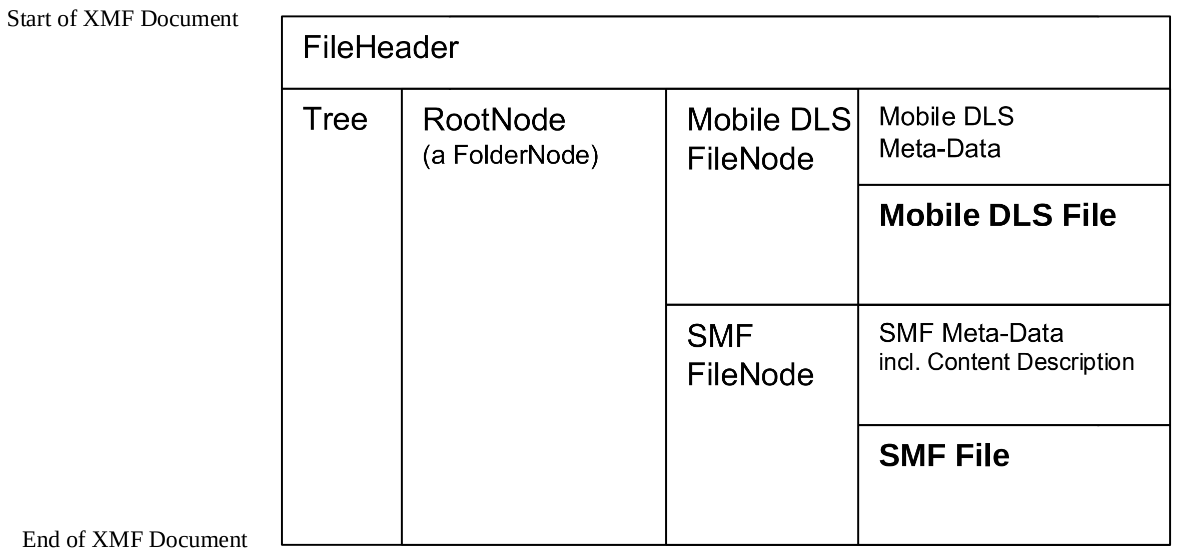
**XMF** – Acronym for eXtensible Music Format as defined in [1] and modified in [3].

**XMF File Type** – See definition in [1].

**XMF Meta File Format** – See definition in [1] and modifications in [3].

# 3. File Layout and Supported Resource Types

A Mobile XMF document must contain one SMF file, and may contain zero or one #179a Mobile DLS files. Both must be located in the Mobile XMF document's RootNode (see [1]). Any Mobile DLS file must appear before the SMF file. Detached nodes (ReferenceTypeID 3, as defined in [1]) may not be used. An example file layout is shown below.



**Notes:**

The Mobile DLS file (and the FileNode that contains it) is optional.

'Mobile DLS Meta-Data' means: One MetaDataItem for Resource Format (the ResourceFormatID must be 5, meaning Mobile DLS).

'SMF Meta-Data' means: One or more MetaDataItems for Content Description (see section 10), and one MetaDataItem for Resource Format (the ResourceFormatID may be either 0 [meaning SMF Format 0] or 1 [meaning SMF Format 1]).

## 4. Interoperable SMF Content

A Mobile XMF document must contain one SMF file [5]. The SMF file may access custom Mobile DLS instruments provided in the same Mobile XMF document, or GM instruments that the player is required to provide (see section 6). The SMF file must conform to the following requirements:

| | |
|---|---|
| **SMF Content Format** | The SMF file must be SP-MIDI-compliant [6]. |
| **SMF File Format** | The SMF file may be either Format 0 (single track), or Format 1 (multiple tracks playing in parallel). For SMF Format 1, players should support at least 16 tracks. SMF Formats 0 and 1 are specified in [5]. |
| **MIDI message support** | See SP-MIDI specification [6]. |
| **MIDI System Exclusive messages** | See SP-MIDI specification [6]. |
| **SMF Meta Events** | See SP-MIDI specification [6]. |

## 5. Interoperable Instrument Content

A Mobile XMF document may contain zero or one Mobile DLS instrument content files [7]. If present, the Mobile DLS file must conform to the following requirements:

| | |
|---|---|
| **Instrument Data Format** | The instrument data file must be compliant with the Mobile DLS specification [7]. |
| **Constraints on Instrument Content** | See the separate Mobile DLS specification document [7]. |

# 6. How Instruments are Accessed With MIDI Messages

The SMF content in the Mobile XMF document can use both programs (melodic instruments or percussion sets) contained within the same Mobile XMF document ('custom instruments') and General MIDI programs ([8], [9]) whose data is automatically supplied by the player. This section specifies the content conventions and player behaviors needed to support this functionality.

**Bank and Program Addresses –** The bank numbers used in the contained Mobile DLS content correspond directly to the bank numbers used in the contained SMF content. For example, to access custom program 12h in bank MSB=42h, LSB=04h, the SMF content would use two Control Change messages (CC0 = 42h, CC32 = 04h) followed by a Program Change message (program 12h). Certain limitations on the use of bank numbers are detailed below.

**Overriding a GM Instrument –** It is possible, if desired, for a content creator to provide temporary 'override' replacements for the player's built-in GM programs by providing custom programs in the compound document that have bank and program numbers that correspond to the desired target.

## 6.1 Content Conventions

**SMF Content Bank Numbering –** The SMF content must comply with the SP-MIDI specifications ([6], [10]), and must also follow General MIDI 2 rules for using MIDI Bank Select and Program Change messages to access the built-in GM instruments (see [9] at sections 3.2 and 3.3.1). Where the SP-MIDI and GM2 specifications diverge with respect to bank selection and program change rules, the player must observe the SP-MIDI specified behavior, not the GM2 rules. The player furnishes a set of GM percussion kit programs at bank MSB 78h LSB 00h, and a set of GM melodic instrument programs at bank MSB 79h and LSB 00h through 09h.

**Mobile DLS Content Bank Numbering –** Any custom program (melodic instrument or percussion set) may use any bank number (MSB+LSB) and any program number, however content creators should be aware of the following considerations:

Custom programs with bank MSB of 78h and bank LSB of 00h will act as temporary overrides for the player's built-in GM percussion kit programs.

Custom programs with bank MSB of 79h and bank LSB in the range 00h through 09h will act as temporary overrides for the player's built-in GM melodic instrument programs.

No two programs should use the same bank (MSB+LSB) and program number, even if the values of the ulBank bit 31 melodic/percussion bit are different. Programs with the same bank and program will result in an override, and ulBank bit 31 is ignored by the Mobile DLS synthesizer.

All other bank and program numbers are available for use by content authors, without restriction.

**Telephone Ring Vibrator Program Not Overridable**

In Mobile XMF Content, use of Mobile DLS instruments with bank MSB=79h, LSB=06h, and program=7Ch is discouraged because in the SP-MIDI 5-24 Voice Profile for 3GPP [10], this slot is exclusively reserved for the telephone ring vibrator program.

In a Mobile XMF player, the program at bank MSB=79h,LSB=06h, and program=7Ch is exclusively reserved for controlling the telephone ring vibrator, and is not considered an overridable GM/GM2 instrument. If the player supports control of the telephone ring vibrator, then this program must always control the telephone ring vibrator. This program must never produce a rendered note in the MIDI synthesizer, even if the Mobile DLS file contained in the same Mobile XMF document includes a Mobile DLS instrument with bank MSB=79h, LSB=06h, and program=7Ch.

## 6.2 Synthesizer Behaviors

**Bank Loading –** The player must map the bank select MSB/LSB values in the contained SMF content directly to the identical MSB/LSB bank locations stored in the contained Mobile DLS program (melodic instruments and percussion sets) content, without performing any shifting or other translation. For example, if the MIDI messages in the SMF indicate MSB 12h, LSB 34h, and program 56h, then the player will look for a Mobile DLS instrument with MSB 12h, LSB 34h, and program 56h.

**Program Change Processing –** When processing a MIDI Program Change message, the player must first seek the requested program (melodic instrument or percussion set) within the contained Mobile DLS content, respecting the current bank select MSB/LSB value for that MIDI channel. If the requested bank and program number is not found in the contained Mobile DLS content, and the bank select MSB+LSB is within the built-in GM instrument range (MSB 78h LSB 00h, or MSB 79h LSB 00h through 09h), then the player must go on to perform a second seek for the indicated program in the player's built-in GM bank(s), using the SP-MIDI rules given in section 2.2.3 of [10] and the GM2 rules given in section 2.6 of [9].

# 7. Use of XMF Meta File Format Features

## 7.1 Standard XMF Meta File Format Features

The following tables indicate the XMF Meta File Format features that players must support. Content should not rely upon features that players are not required to support. Players shall properly skip over all unsupported MetaDataItems.

| | |
|---|---|
| **XMF Meta File Format Version** | Players must support XMF Meta File Format version 2.00 [11], excepting certain features as detailed below. |
| **ReferenceTypeIDs** | Players are required to support only ReferenceTypeIDs 1 (In-Line Resource) and 2 (In-File Resource). Content generated according to this specification shall not rely on players to process any other ReferenceTypeIDs. |
| **UnpackerIDs** | Players are not required to support any UnpackerIDs. Content generated according to this specification shall not rely on players to process any UnpackerIDs. |
| **Standard Meta-Data Fields** | Players are required to support the following Standard Meta-Data Fields, and are not required to support any others. Content generated according to this specification shall not rely on players to process any Standard Meta-Data FieldIDs other than those listed here:<br>• **Resource Format** (Standard FieldID value 3) – Used to distinguish contained SMF files from contained Mobile DLS instrument files. This field must be attached to every Mobile DLS file and every SMF file in the XMF file. The content is a Standard ResourceFormatID, which must be one of the following three values:<br>**0: SMF Format 0** [5]<br>**1: SMF Format 1** [5]<br>**5: Mobile DLS instrument file** [7]<br>**Note:** This specification defines a new Standard ResourceFormatID for Mobile DLS instrument files, value 5, which should be added to the XMF Meta File Format specification.<br>• **Content Description** (Standard FieldID value 13) – A new field with Universal binary contents (hidden or visible), and an internal structure as described in this specification at section 10. One or more copies of this field collectively describe the playback resource consumption of an SMF in the Mobile XMF document. This MetaDataItem must appear in the FileNode for the corresponding SMF, and nowhere else in the XMF file.<br>**Note:** This specification defines a new Standard FieldID for XMF meta-data, value 13, which should be added to the XMF Meta File Format specification. |
| **Custom Meta-Data Fields** | Players are not required to support Custom Meta-Data fields. Content generated according to this specification shall not rely on players to process any Custom Meta-Data fields. |
| **International Meta-Data** | Players are not required to support International meta-data. Content generated according to this specification shall not rely on players to process any international meta-data FieldContents. |
| **VLQ Field Maximum Sizes/Values** | See following table. |

| VLQ Field Maximum Sizes and Values | | |
|---|---|---|
| **VLQ** | **Max Value** | **Min Bits Unsigned Integer** |
| **FileHeader:** | | |
| FileLength | 268,435,455 | 28 |
| TreeStart | 268,435,455 | 28 |
| TreeEnd | 268,435,455 | 28 |
| **NodeHeader:** | | |
| NodeLength | 268,435,455 | 28 |
| NodeContainedItems | 255 | 8 |
| NodeHeaderLength | 65,535 | 16 |
| **NodeMetaData:** | | |
| LengthInBytes | 65,535 | 16 |
| Standard FieldID | 255 | 8 |
| Custom Field Specifier (XString length) | 255 | 8 |
| **FieldContents:** | | |
| length for UniversalContents | 65,535 | 16 |
| NumberOfStrings for InternationalContents | 65,535 | 16 |
| MetaDataType for InternationalContents | 65,535 | 16 |
| VersionData length for InternationalContents | 65,535 | 16 |
| **NodeUnpackers:** | | |
| ListSizeInBytes | 65,535 | 16 |
| DecodedSize | 65,535 | 16 |
| **NodeContents:** | | |
| ReferenceTypeID | 5 | 3 |
| **MetaDataTypesTable:** | | |
| LengthInBytes | 65,535 | 16 |
| NumberOfEntries | 65,535 | 16 |
| MetaDataType in TypeEntry | 65,535 | 16 |
| StringFormatTypeID | 7 | 3 |
| | | |

## 7.2 Additional Restrictions on Use of XMF Meta File Format

Resource blocks (SMF file, Mobile DLS file) must always begin on a word boundary (even byte address, also known as 16-bit boundary). When the resource block start does not naturally fall on a word boundary, this alignment can be achieved in the following manner.

For In-File Resources and In-Line Resources, word alignment can be accomplished by padding, in other words by adding one pad byte (not meaningful data) between the NodeHeader and the NodeContents. It is recommended but not required to use the value 00h for the pad byte.

# 8. Content Handling Behaviors

Players are required to support the following content handling behaviors.

## 8.1 Clear Instruments from Old Mobile XMF Documents

To prevent any previously loaded Mobile XMF document instruments from being inadvertently used, and to prevent the build-up of unused instruments in the synthesizer's state, it is necessary to clear any instruments from previously loaded Mobile XMF documents when loading a new Mobile XMF document. All instrument banks should be cleared of any previously loaded instruments that originated in Mobile XMF documents before loading the compound document's Mobile DLS instrument file (if any). Previously loaded instruments that are intended to be treated as permanently installed, including General MIDI instrument permanent replacements, should not be cleared when loading a Mobile XMF document. Any further mechanism for handling persistent instrument content is beyond the scope of this specification.

## 8.2 Load the Mobile DLS File from the Mobile XMF Document

After clearing old Mobile XMF document instruments, the player must load the Mobile DLS file in the Mobile XMF document (if any) into the synthesizer in preparation for playback. If the Mobile DLS file contain any instruments with the same bank and program numbers as Mobile DLS files loaded at earlier times, then the later programs will overwrite the earlier programs.

## 8.3 Prepare to Play SMF

Note: This specification only defines playback behavior for a single SMF in a Mobile XMF document. No SMF playlist behavior is defined.

Upon loading a Mobile XMF document, its SMF file should be automatically loaded into the sequencer and prepared for playback, in case the playback has not already started.

## 8.4 Load Errors Prevent Playback

If any failure is encountered while loading the Mobile XMF document, and the SMF file has not been started, then the SMF file should never be started. If any failure is encountered while loading the Mobile XMF document, and the SMF file has already been started, then the SMF file should be stopped. This is to assure that the content will play as its creator intended. Examples of failure conditions would include: the player does not support the Mobile DLS instrument data format [7]; the Mobile DLS instrument file or SMF file contains invalid data that cannot be parsed; required storage cannot be allocated while parsing the content; etc.

# 9. Content Type Indication

## 9.1 Internal Content Type Indication

To provide content type information to XMF file parsers and generic file type recognizers, the XmfFileTypeID field in the FileHeader must indicate the proper XMF File Type number, and the XmfFileTypeRevisionID must indicate the proper spec version:

| | |
|---|---|
| **XMF File Type** | 2 |
| **Spec Revision Level** | 1 (for version 1.0 of this specification) |

## 9.2 External Content Type Indication

To provide content type information to file systems, file servers, transfer protocols, and services, a Mobile XMF document generated according to this specification must include external indication of the content type:

| | |
|---|---|
| **Filename Extension** | `.mxmf` |
| **MIME Media Type** | `audio/mobile-xmf` |

# 10. Content Description MetaDataItem

This section describes the purpose, format, and usage of the Content Description MetaDataItem, and provides a commented example.

## 10.1 Purpose

This MetaDataItem is used to characterize the set of resources needed to play an SMF file in the Mobile XMF document, so that the player is able to determine, before commencing playback and without having to analyze the contents of the SMF, whether the SMF can be correctly played using the available resources. If there are not enough resources available to play the SMF file correctly, the player should not start playback of the SMF.

## 10.2 Usage

This FieldID is valid for File nodes only, not Folder nodes. One or more Content Description MetaDataItem may be attached to any FileNode in the Mobile XMF document that contains an SMF file. There must be one MetaDataItem instance for every MIP message in the SMF file. Typically there may be several of these MetaDataItem for every SMF, all of which must be parsed to fully characterize the SMF's resource consumption.

## 10.3 Contents Format

The FieldContents format for a Content Description MetaDataItem is Universal, Binary (hidden or visible), and consists of the following sequence of parts; the data format for each part is detailed below.

1. **MIP Message Reference**: The index of the corresponding MIP message in the SMF
2. **Number of Channels**: The count of MIDI channels described in this MetaDataItem
3. **Number of Playback Resources Described** in this MetaDataItem
4. **Playback Resource List** (PRL) identifying the described resources
5. **Playback Resource Group List** (PRGL) identifying the described resources
6. **Maximum Instantaneous Resource** (MIR) **Count Table**: per-resource usage figures

The described playback resources are identified by a combination of the PRL and the PRGL; every described playback resource belongs to a group (see 5) and has a unique ID within that group (see 4). The interpretation of the MIR Count Table depends on the contents and order of the PRL and PRGL.

**Example:**

```
00h    // Corresponds to the first MIP message in the SMF
04h    // 4 MIDI channels in the MIP message
03h    // 3 resources described in this MetaDataItem
// Resource column descriptor lists for MIR Count Table:
00h, 01h,     00h, 02h,     00h, 03h  // PRL for resources 1, 2, & 3
       // 0 = Standard, 1 for Std = "Mobile DLS voices w/o Optional Control Group
                                    defined in [7] at Section 1.2"
       // 0 = Standard, 2 for Std = "Mobile DLS voices with Optional Control Group
                                    defined in [7} at Section 1.2"
       // 0 = Standard, 3 for Std = "Total wavetable memory consumption"
00h,          00h,          02h       // PRGL for resources 1, 2, & 3
// MIR Count Table: 4 channel rows, 3 resource columns
02h, 00h, 01h,          // 1st MIDI channel of the MIP message:
                        //     2 of 1st rsrc, 0 of 2nd rsrc, 1 or 3rd rsrc
03h, 00h, 01h,          // 2nd MIDI channel of the MIP message
05h, 00h, 01h,          // 3rd MIDI channel of the MIP message
05h, 02h, 01h           // 4th MIDI channel of the MIP message
```

### 10.3.1 MIP Message Reference

This is a VLQ containing the index of the MIP message in the SMF that this MetaDataItem describes. For example, the first MIP message would be index 0.

**Example:**

```
02h    // Corresponds to the third MIP message in the SMF
```

### 10.3.2 Number of MIDI Channels Described

This is a VLQ containing the number of MIDI channels described in this MetaDataItem. This count is used to determine the number of rows in the MIR Count Table. The number should agree with the number of MIP channel values present in the corresponding MIP message.

### 10.3.3 Number of Playback Resources Described

This is a VLQ containing the number of playback resources described in this MetaDataItem. This count is used to determine the number of columns in the MIR Count Table, the number of entries in the Playback Resource List, and the number of entries in the playback resource group list.

### 10.3.4 Playback Resource List

This is a list of entries that, together with the Playback Resource Groups List, uniquely identify which playback resources are being described in this MetaDataItem. There is one entry per described playback resource (see 10.3.3.). Each entry consists of one VLQ containing a ResourceTypeID prefix of 0 through 5, and a data field (whose interpretation depends on the value of the ResourceTypeID, as described below), describing the resource appearing at the corresponding index in the MIR Count Table entries.

Each defined playback resource belongs to exactly one Playback Resource Group (see 10.3.5).

**Example:**

```
00h, 01h,     // Prefix 0 = Standard,
              //     1 for Standard = “Mobile DLS voices w/o Optional Control Group
                                        defined in [7] at Section 1.2”
00h, 02h,     // Prefix 0 = Standard,
              //     2 for Standard = “Mobile DLS voices with Optional Control Group
                                        defined in [7] at Section 1.2”
00h, 03h      // Prefix 0 = Standard,
              //     3 for Standard = “Total wavetable memory consumption”
```

| ResourceTypeID (Prefix) | Description | Used with PlaybackResourceGroupIDs |
|---|---|---|
| 0 | Standard | 0, 2 |
| 1 | MMA/AMEI Manufacturer | 0, 2 |
| 2 | Registered | 0, 2 |
| 3 | Non-Registered | 0, 2 |
| 4 | Wavetable Codec wFormatTag | 1 |
| 5 | Wavetable Codec GUID | 1 |

### 10.3.4.1 Standard PlaybackResourceIDs

**Standard PlaybackResourceID**s are prefixed with **ResourceTypeID** 0 (0x00).

They are used for voice categories, wavetable memory consumption, and may be used in the future for other playback resources. Only MMA/AMEI may define Standard PlaybackResourceIDs. Each PlaybackResourceID value may be used only with its corresponding PlaybackResourceGroupID, as shown in the following table.

**Note:** Standard PlaybackResourceIDs may not be used for wavetable compression codecs (PlaybackResourceGroupID 1).

| PlaybackResourceID Value | Description | For Use with PlaybackResourceGroupID |
|---|---|---|
| **Synthesizer Voices:** | | |
| 0 | Number of General MIDI 1 voices [8]<br>**Note:** Any voice layering in the GM bank is not taken into account | 0: Synthesizer Voice |
| 1 | Number of Mobile DLS voices w/o Optional Control Group [7] | 0: Synthesizer Voice |
| 2 | Number of Mobile DLS voices with Optional Control Group [7] | 0: Synthesizer Voice |
| **Wavetable Memory Consumption:** | | |
| 3 | Total wavetable data consumption (in kilobytes) for Mobile DLS instruments using (16-bit, 8-bit) PCM samples | 2: Wavetable Memory Consumption |
| 4 | Total uncompressed wavetable data (in kilobytes) for Mobile DLS instruments using compressed samples | 2: Wavetable Memory Consumption |

**Example:**

```
00h,   // Prefix 0 = Standard
01h    // 1 for Standard = Mobile DLS voices w/o DCF & Vib. LFO
```

### *10.3.4.2 MMA/AMEI Manufacturer PlaybackResourceIDs*

**MMA/AMEI Manufacturer PlaybackResourceID**s are prefixed with ResourceTypeID 1 (0x01).

They are reserved for manufacturer-specific playback resource types. Each MMA/AMEI Manufacturer is free to create PlaybackResourceIDs based on their own MMA/AMEI Manufacturer ID, subject to the following restrictions. Manufacturers are free to either publish their PlaybackResourceIDs, or keep them proprietary.

A MMA/AMEI Manufacturer PlaybackResourceID contains two parts: first the company's assigned MMA/AMEI Manufacturer ID in ordinary 1-byte or 3-byte form, then a VLQ containing the company's internal ID.

Note that all MMA/AMEI Manufacturer IDs are either one byte long or three bytes long, as indicated by the first byte (0x00 means a three-byte ID).

**Note:** MMA/AMEI Manufacturer PlaybackResourceIDs may not be used for wavetable compression codecs (PlaybackResourceGroupID 1).

**Example:**

```
01h,                // Prefix 1 = MMA/AMEI Manufacturer,
00h, 7Ch, 7Fh,      // Electric Kazoo MMA/AMEI Mfr. ID
0Ah                 // Electric Kazoo´s playback resource type 10
```

### *10.3.4.3 Registered PlaybackResourceIDs*

**Registered PlaybackResourceID**s are prefixed with ResourceTypeID 2 (0x02).

They are reserved for custom or proprietary playback resources, but are not linked to MMA/AMEI Manufacturer IDs, and may only be defined and assigned by the MMA/AMEI. This allows non-MMA/AMEI manufacturers access to extensibility without incurring the higher byte count of Non-Registered PlaybackResourceIDs.

A Registered PlaybackResourceID consists of one integer in VLQ form.

**Note:** Registered PlaybackResourceIDs may not be used for wavetable compression codecs (PlaybackResourceGroupID 1).

**Example:**

```
02h,         // Prefix 2 = Registered
C5h          // 197 = Registered playback resource number 197
             //    (must be listed on MMA/AMEI website to be valid).
```

### *10.3.4.4 Non-Registered PlaybackResourceIDs*

**Non-Registered PlaybackResourceID**s are prefixed with ResourceTypeID 3 (0x03).

They are reserved to allow non-MMA/AMEI manufacturers to use arbitrary private playback resources without MMA/AMEI registration. To avoid collisions, a Non-Registered PlaybackResourceIDs must be generated as a Globally Unique Identifier (GUID). MMA/AMEI may also use Non-Registered PlaybackResourceIDs for specifying playback resources with GUIDs.

A Non-Registered PlaybackResourceID consists of one 16-byte GUID in binary form without any conversion (for example, to hexadecimal text form or VLQ form).

**Note:** Non-Registered PlaybackResourceIDs may not be used for wavetable compression codecs (PlaybackResourceGroupID 1).

**Example:**

```
03h,                     // Prefix 3 = Non-Registered
<16 bytes appear here>   // GUID in binary form
```

#### *10.3.4.5 Wavetable Codec wFormatTag PlaybackResourceIDs*

**Wavetable Codec wFormatTag PlaybackResourceID**s are prefixed with ResourceTypeID 4 (0x04).

They are reserved for representing wavetable compression codecs, using the same wFormatTag values as in the Format Chunk of the Mobile DLS Wave File Format. Only MMA/AMEI may register supported Wavetable Codec wFormatTag PlaybackResourceIDs for Mobile DLS and Mobile XMF.

To promote content interoperability, the MMA will maintain a public web page listing wFormatTags in use for Mobile DLS and Mobile XMF. Developers and standards bodies adding new Mobile DLS wavetable codec wFormatTags should always contact the MMA to have their wFormatTags added to this registry, using the email address on the registry web page. The MMA will promptly add all submitted wFormatTags to the registry, so long as the codec definition does not conflict with any pre-existing definition for the submitted wFormatTag value. The wFormatTag value WAVE_FORMAT_ EXTENSIBLE should not be registered, as it does not describe a specific codec. New codecs without a pre-existing wFormatTag will be registered using Wavetable Codec GUID PlaybackResourceIDs.

Each Wavetable Codec wFormatTag PlaybackResourceID consists of one VLQ for the wFormatTag value.

**Example:**

```
        04h,                        // Prefix 4 = Wavetable Codec wFormatTag
        01h                         // wFormatTag 1 = Linear PCM
```

#### *10.3.4.6 Wavetable Codec GUID PlaybackResourceIDs*

**Wavetable Codec GUID PlaybackResourceID**s are prefixed with ResourceTypeID 5 (0x05).

They are reserved for representing wavetable compression codecs, using the same WAVE_FORMAT_EXTENSIBLE GUID values as in the Format Chunk of the Mobile DLS Wave File Format. Only MMA/AMEI may register supported Wavetable Codec GUID PlaybackResourceIDs for Mobile DLS and Mobile XMF.

To promote content interoperability, the MMA will maintain a public web page listing wavetable codec GUIDs in use for Mobile DLS and Mobile XMF. Developers and standards bodies adding new Mobile DLS wavetable codec GUIDs should always contact the MMA to have their GUIDs added to this registry, using the email address on the registry web page. The MMA will promptly add all submitted GUIDs to the registry, so long as the codec definition is not the same as any pre-existing codec definition in the registry.

A Wavetable Codec GUID PlaybackResourceID consists of one 16-byte GUID in binary form without any conversion (for example, to hexadecimal text form or VLQ form).

**Example:**

```
        05h,                        // Prefix 5 = Wavetable Codec GUID
        <16 bytes appear here>      // GUID in binary form
```

### 10.3.5 Playback Resource Group List

This is a list of entries that, together with the Playback Resource List, uniquely identify which playback resources are being described in this MetaDataItem. There is one entry per described playback resource (see 10.3.3.). Each entry consists of one VLQ containing the PlaybackResourceGroupID for the resource described at the corresponding index in the MIR Count Table entries.

Each playback resource belongs to exactly one playback resource group. A group is defined by a unique PlaybackResourceGroupID, the name of the group, whether the playback resources within the group are mutually exclusive, and whether encountering an unrecognized playback resource in the group is required to cause MIDI channel masking [6].

Mutual Exclusivity means that e.g. any one synthesizer voice resource cannot be listed as a GM voice and Mobile DLS voice at the same time, and that any one wavetable can have only one wavetable compression codec. A group's Mutual Exclusivity property should be set to 'No' if the playback resources are meant to overlap within the playback resource group, in terms of counting the MIR values.

A player shall mask (not process messages on) any MIDI channel if that channel includes any unrecognized playback resources and that resource's group has its Channel Masking property set to 'Yes', unless the channel's MIR value for the unrecognized resource is zero; in that case, the unrecognized resource should not have any effect on the channel's masking.

| PlaybackResourceGroupID | Group Name | Mutual Exclusivity | Channel Masking |
|---|---|---|---|
| 0 | Synthesizer voice | Yes | Yes |
| 1 | Wavetable compression codec | Yes | Yes |
| 2 | Wavetable memory consumption | No | No |

**Mutual Exclusivity** = Are playback resources mutually exclusive inside this group

**Channel Masking** = Unrecognized playback resources in this group cause channel masking

**Example:**

```
00h, 01h,    00h, 02h,    00h, 03h      // PRL for resources 1, 2, & 3
00h,         00h,         02h           // PRGL for resources 1, 2, & 3:
             // 0 = Synthesizer voice
             // 0 = Synthesizer voice
             // 2 = Wavetable memory consumption
```

### 10.3.6 MIR Count Table

This is a table containing one MIR list for each MIDI channel described in this MetaDataItem. The channels appear in the order of decreasing channel priority, as defined by the corresponding MIP message. All MIR lists in a given MetaDataItemhave one entry per playback resource described in this MetaDataItem, a VLQs representing the MIR value for the corresponding playback resources. The MIR counts for each resource column are cumulative. For example, the MIR counts for the second MIDI channel of the MIP message are the total maximum instantaneous resource counts for the first and second MIDI channels of the MIP message combined.

**Example:**

```
// MIR Count Table: 4 channel rows, 3 resource columns:
      02h, 00h, 01h,      // 1st MIDI channel of the MIP message:
// 2 of 1st resource, 0 of 2nd resource, 1 or 3rd resource
      03h, 00h, 01h,      // 2nd MIDI channel of the MIP message – adds 1,0,0
      05h, 00h, 01h,      // 3rd MIDI channel of the MIP message – adds 2,0,0
      05h, 02h, 01h       // 4th MIDI channel of the MIP message – adds 0,2,0
```

## 10.4 Example Content Description MetaDataItem

```
// MetaDataItem --------------------------------------------
//  Field Specifier:
00h    // Standard meta-data FieldID follows
0Dh    // FieldID 13: This is a Content Description MetaDataItem
// FieldContents:
00h    // Universal contents follows
19h    // LengthInBytes 25: 25 bytes follow
06h    // Visible binary data follow
// UniversalData starts here
00h    // This MetaDataItem corresponds to
       // the first MIP message in the SMF
04h    // 4 MIDI Channels in the MIP message
03h    // 3 resources described in this MetaDataItem

// Resource column descriptor lists for MIR Count Table: PRL & PRGL
// PRL for resources 1, 2, & 3:
00h, 01h,    00h, 02h,    00h, 03h
       // 0 = Standard, 1 for Std = “Mobile DLS voices w/o Optional Control Group
                                    defined in [7] at Section 1.2”
       // 0 = Standard, 2 for Std = “Mobile DLS voices with Optional Control Group
                                    defined in [7] at Section 1.2”
       // 0 = Standard, 3 for Std = “Total wavetable memory consumption”
// PRGL for resources 1, 2, & 3:
00h,        00h,        02h
       // 0 = Synthesizer voice
       // 0 = Synthesizer voice
       // 2 = Wavetable memory consumption


// MIR Count Table: 4 channel rows, 3 resource columns:
02h, 00h, 01h// 1st MIDI channel of the MIP message:
       // 2 of 1st resource: Synth voice::Mobile DLS voices w/o Optional Control
                                            Group defined in [7] at Section 1.2
       // 0 of 2nd resource: Synth voice::Mobile DLS voices with Optional Control
                                            Group defined in [7] at Section 1.2
       // 1 of 3rd resource: Wavetable Memory::Total wavetable mem. consumption
03h, 00h, 01h// 2nd MIDI channel of the MIP message – adds 1,0,0
05h, 00h, 01h// 3rd MIDI channel of the MIP message – adds 2,0,0
05h, 02h, 01h// 4th MIDI channel of the MIP message – adds 0,2,0
```

## 11. References

[1] *“Specification for XMF Meta File Format”*, RP-030, MIDI Manufacturers Association, Los Angeles, CA, USA, 2001

[2] *“Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures”*, RFC 2048, IETF, http://www.ietf.org/rfc/rfc2048.txt, November 1996

[3] *“XMF Meta File Format Updates 1.01”*, RP-039, MIDI Manufacturers Association, Los Angeles, CA, USA, 2003

[4] *“Type 0 and Type 1 XMF Files”*, RP-031, MIDI Manufacturers Association, Los Angeles, CA, USA, 2001

[5] *“Standard MIDI Files”*, RP-001, in The Complete MIDI 1.0 Detailed Specification, Document Version 96.1, MIDI Manufacturers Association, Los Angeles, CA, USA, 1996

[6] *“Scalable Polyphony MIDI Specification, Version 1.0”*, RP-034, MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002

[7] *“Mobile DLS Specification”*, RP-041, MIDI Manufacturers Association, Los Angeles, CA, USA, 2003

[8] *“General MIDI System Level 1”*, RP-003, in The Complete MIDI 1.0 Detailed Specification, Document Version 96.1, MIDI Manufacturers Association, Los Angeles, CA, USA, 1996

[9] *“General MIDI Level 2 Specification (Recommended Practice)”*, RP-024, November 1999, MIDI Manufacturers Association, Los Angeles, CA, USA

[10] *“Scalable Polyphony MIDI Device 5–24 Note Profile For 3GPP, Version 1.0”*, RP-035, MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002

[11] *“XMF Meta File Format 2.00”*, RP-043, MIDI Manufacturers Association, Los Angeles, CA, USA, 2004

THIS PAGE IS INTENTIONALLY BLANK

# Audio Clips for Mobile XMF

February 7, 2007

**Published By:**

The MIDI Manufacturers Association

Los Angeles, CA

# PREFACE

AUDIO CLIPS FOR MOBILE XMF
RP-045

Printed 2007

MMA
PO Box 3173
La Habra CA 90632-3173

## CONTENTS

# 1. Introduction

This specification defines an extension of Mobile XMF v1.0 [2] that allows instruments to use longer mono or stereo 'audio clip' wavetables, and enables the use of various data-compressed audio encodings beyond linear PCM (for example: AAC, aacPlus, Enhanced aacPlus, AMR-WB, Extended AMR Wideband, IMA ADPCM, MP3, etc.).

## 1.1 Overview

The Mobile XMF v1.0 specification, which has been widely deployed and is referenced by at least one other standards body [6], defines a structured music and audio content format for use in resource-constrained mobile devices such as mobile phones. It allows custom wavetable-based Mobile DLS [3] instrument definitions to be bundled together with a Standard MIDI File (SMF) [7] image containing a musical performance played on a combination of the custom instruments and a standard palette of General MIDI instruments. However in Mobile XMF all wavetables must fit into the DLS synthesizer's wavetable memory, which is extremely limited (minimum of 7k or 15k bytes).

This specification allows much longer mono or stereo audio wavetables or 'audio clips' to be triggered in sync with the MIDI timeline. Audio clip instruments are intended for straight audio playback, and so do not allow most of the Mobile DLS sound shaping controls to be used. They are not polyphonic (consume 1 mono or stereo voice at most), and may not be changed in pitch, filtered, nor modulated in any way other than ordinary volume setting and, for mono wavetables rendered on stereo devices, stereo pan setting. This makes it possible to render audio clips either from within the MIDI synthesizer, or with external audio file player hardware or software.

This level of playback functionality is suitable for integrating sections of recorded music together with MIDI rendered notes, which is a common and appropriate production technique for producing full-length branded music content (as well as other content styles) at file sizes much smaller than conventionally data-compressed recorded music.

## 1.2 Technical Goals

The following technical goals governed the development of this specification:

1 - Extend Mobile XMF by adding streamed, MIDI-synchronized mono or stereo linear audio clips using data-compressed audio encodings.  This means short- to mid-length audio clips synced to a MIDI timeline, rather than long audio clips that would be continuously synchronized with the  SMF timeline as a whole.

2 - Make this extended format playable on the widest possible set of playback devices.

3 - Easy content authoring.

4 - Avoid unnecessary complexity.

5 - Place only conservative (bandwidth) demands on the audio clip data streaming channel.

6 - Need a scalable polyphony approach for audio clips for broad content interoperability

7 - It should be possible to play this content both on players that can only handle one audio clip at a time, and on players that can play more than one audio clip at a time, and still have the content do the right thing in all cases. We don't have a typical upper limit in mind for audio clip polyphony.

8 - No backwards compatibility with Mobile XMF v1.0. [Note: Upon inspection this turned out to be simply too cumbersome (codec selection, memory sizes, providing fall-back instruments, & impact on data organization).]

## 1.3 Terminology

**AMEI –** Association for Musical Electronic Industry, in Japan. AMEI includes a MIDI Committee and a Mobile Division, which serve the same functions in Japan that the MMA serves in the rest of the world.

**Audio Clip Instrument –** Mobile DLS instrument intended for use as an audio clip; not polyphonic, self-exclusive, uses mono or stereo wavetables.

**Audio Clip Polyphony** – Number of simultaneously playing audio clips, or a player's maximum capacity in terms of same.

**Audio Codec** – Audio coder/decoder technology and associated audio data encoding format, for example linear PCM, AAC, aacPlus, Enhanced aacPlus, AMR-WB, Extended AMR Wideband, IMA ADPCM, MP3 (MPEG-2 Layer 3 Audio), etc..

**Content Description Metadata (CDM) –** Additional information in a Mobile XMF file describing the use of various resources per MIDI channel. Defined in section 10 of [2].

**DLS –** Downloadable Sounds, a family of MMA/AMEI content formats and player specifications for wavetable synthesizer musical instruments. See also Mobile DLS.

**MIP Message –** MIDI message defining the Maximum Instantaneous Polyphony per MIDI channel of a piece of scalable SMF content. Defined in the SP-MIDI specification [4].

**MMA –** MIDI Manufacturers Association

**Mobile DLS –** MMA/AMEI specification for wavetable synthesizer instruments for mobile applications. See [3].

**SMF –** Standard MIDI File, see [7].

**SP-MIDI –** Scalable Polyphony MIDI, an MMA/AMEI content format specification that adapts MIDI file playback to the available device resources. See [4] and [5].

# 2. Content Format and Player Behavior

This section defines the content format for Mobile XMF content that includes Audio Clips, and the required behavior of players supporting such content.

> **IMPORTANT: Except as detailed below**, the content format and player behavior requirements of Mobile XMF content that includes Audio Clips are **identical to Mobile XMF v1.0** [2].

For easier developer reference, sections 2.1 through 2.8 of this specification correspond directly to sections 3 through 10 of the Mobile XMF v1.0 Specification [2].

## 2.1 File Layout and Supported Resource Types

The file layout and supported resource types for Mobile XMF content that includes Audio Clips is the same as for Mobile XMF v1.0 (see section 3 of [2]).

## 2.2 Interoperable SMF Content

The rules regarding interoperable SMF for Mobile XMF content that includes Audio Clips are the same as for Mobile XMF v1.0 (see section 3 of [2]). However, the operation of certain MIDI messages contained within the SMF is different when an audio clip instrument is selected, see section 2.4.

## 2.3 Interoperable Instrument & Audio Clip Content

With the following exceptions, the interoperable instrument (including audio clip instrument) content rules are the same as for Mobile XMF v1.0 (see section 5 of [2]).

### *2.3.1 Instrument Data Format*

Except as detailed below, the bundled instrument and audio clip content must be compliant with the Mobile DLS specification v1.0 [3].

### *2.3.2 Constraints on Instrument Content*

Except as detailed below, the bundled Mobile DLS instrument and audio clip content is subject to the same constraints detailed in the Mobile DLS specification v1.0 [3].

#### 2.3.2.1 Definition and Format of Audio Clip Instruments

For Mobile XMF content that includes Audio Clips, all Mobile DLS Instrument programs in MIDI bank MSB 0x7A, LSB 0x00 are considered to be audio clip instruments. The data format for audio clip instruments is identical to ordinary Mobile DLS instruments [3], however wavetables for audio clip instruments may be encoded for different audio decoders than wavetables for ordinary Mobile DLS instruments.

Banks MSB 0x7A, LSB 0x01 through MSB 0x7A, LSB 0x7F are reserved for future MMA/AMEI definition and should not be used in Mobile XMF content that includes Audio Clips.

#### 2.3.2.2 Player Behavior for Audio Clip Instruments

This section consists of normative requirements for implementers of players for Mobile XMF content that includes Audio Clips. See also the non-normative information at section 3. Player Implementation Guidelines.

The Mobile XMF player's behavior for MIDI channels on which an audio clip instrument has been selected differs from its behavior for MIDI channels on which ordinary Mobile DLS instruments have been selected in the following ways.

When handling audio clip instruments, players MUST:

- Play audio clip instruments without any pitch transposition other than sampling rate conversion from the input wavetable data sampling rate to the player audio output sample rate.
- Support at least all of the following audio output sample rates: 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, or 48 kHz.
- Take into account the fact that audio clip instruments may use larger wavetables than ordinary Mobile DLS instruments (see section 2.3.2.3).
- Take into account the fact that audio clip instruments may include wavetables encoded for different audio decoders than ordinary Mobile DLS instruments (see section 2.3.2.5).
- Take into account the fact that audio clip instruments may include stereo wavetables, and always play stereo wavetables as stereo output whenever the Mobile XMF player is capable of producing stereo output.
- Always render audio clip instruments with a maximum polyphony of 1 and self-exclusive. Re-triggering an audio clip instrument (by sending a new MIDI Note On message) must cut off wavetable playback and restart it from the beginning of the wavetable, rather than adding a second, layered, copy.
- Never loop playback of the wavetable audio data. Any loop information included in the audio wavetable in the Mobile DLS data must be ignored.
- Always treat the Velocity field of the Note On MIDI message as a Volume control.
- Follow the MMA's standard MIDI volume response curve, which is produced by the DLS 2 Concave transform. The formula for gain in terms of -dB is given at section 1.6.5.4 of [3] as:

  gain = -96 * (5/12)*log10(127/(127 - (127 - velocity)))

- Always ignore all filter and modulation connections with destinations other than Volume and, for mono wavetables rendered on stereo devices, Pan.
- For stereo source wavetables, always ignore all connections whose destination is Pan .

Notwithstanding the above differences, players must still honor the WSMP chunk's Gain Trim field.

### 2.3.2.3 Maximum Wavetable Size for Audio Clip Instruments

For audio clip instruments, Mobile DLS data used in Mobile XMF content that includes Audio Clips may use wavetables significantly larger than those the Mobile DLS v1.0 Specification allows. The actual maximum wavetable size depends on the context. For example, one product may allow a higher maximum than another product. Also, a document or profile that references this Audio Clips for Mobile XMF specification may set a specific maximum wavetable size.

#### 2.3.2.4 Maximum Wavetable Data Bit Rate for Audio Clip Instruments

For audio clip instruments, Mobile DLS data used in Mobile XMF content that includes Audio Clips may have different wavetable bit rate limitations than those the Mobile DLS v1.0 Specification allows. The actual maximum wavetable data bit rate depends on the context. For example, one product may allow a higher maximum than another product. Also, a document or profile that references this Audio Clips for Mobile XMF specification may set a specific maximum wavetable data bit rate.

#### 2.3.2.5 Audio Encodings for Audio Clip Instruments

For audio clip instruments, all required audio decoders must be indicated using the mechanism described in section 2.16.1 of [3]. In addition to defining several wFormatTags for existing audio codecs, this mechanism allows for the registration of new audio encodings via the wFormatTag value WAVE_FORMAT_EXTENSIBLE.

For audio clip instruments, Mobile DLS data used in Mobile XMF content that includes Audio Clips may use additional audio encodings not already listed in the MMA codec GUID registry referenced from section 2.16.1 of [3].

The actual set of available decoders depends on the context. For example, one product may allow a different decode, or set of decoders, than another product. Also, a document or profile that references this Audio Clips for Mobile XMF specification may specify a particular codec, or set of codecs.

#### 2.3.2.6 Rendering Tolerances for Audio Clip Instruments

Whether audio clips are rendered inside the Mobile DLS synthesizer or outside of it, implementations must observe the following tolerances:

- Instrument audio level and gain control (CC7) response for audio clip instruments should match ordinary Mobile DLS and GM instruments as closely as possible.
- Wavetable start times of audio clip instruments must be synchronized with ordinary Mobile DLS or GM instruments. In other words, two notes with start times scheduled for the same SMF tick, where one is played on an audio clip instrument and the other one is played on an ordinary Mobile DLS or GM instrument, must be started as close to simultaneously as possible. It is the player's responsibility to automatically correct for any differences in latency between the sources.
- Drift between the MIDI and audio timelines must not exceed 1 part in 1000.

## 2.4 How Instruments and Audio Clips are Accessed with MIDI Messages

With the following exceptions, the rules regarding how instruments are accessed with MIDI messages for Mobile XMF content that includes Audio Clips are the same as for Mobile XMF v1.0 (see section 6 of [2]).

### *2.4.1 Accessing Audio Clip Instruments*

Audio Clip instruments are accessed in the same way as any other Mobile DLS instrument, i.e. a MIDI channel can select the audio clip instrument by using the following sequence of MIDI messages: Bank Select MSB message, Bank Select LSB message, and finally the Program Change message. However bank MSB 0x7A, LSB 0x00 is reserved for audio clip instruments, and all audio clip instruments must be loaded only from this bank. The program number used in the Program Change message must match an audio clip instrument present in the bundled Mobile DLS content.

Banks MSB 0x7A, LSB 0x01 through MSB 0x7A, LSB 0x7F are reserved for future MMA/AMEI definition and should not be used in Mobile XMF content that includes Audio Clips.

### *2.4.2 Playing Audio Clips*

Once an audio clip instrument has been selected for a MIDI channel, the audio clip can be played on that MIDI channel by sending a MIDI Note On message. The audio clip will play until it reaches the end of the wavetable data, or until a matching MIDI Note Off message (or Note On with velocity of zero) is sent on the same MIDI channel.

Player response to MIDI messages when an audio clip instrument is selected is different from other Mobile DLS instruments in the following ways:

- The note number field of the MIDI Note On message will not affect audio clip pitch. The audio clip will always play at its natural pitch, even if the Mobile DLS instrument program calls for different pitch behavior.
- The note number field of the MIDI Note On message serves solely as a note ID. A MIDI Note Off message (or Note On with velocity of zero) terminates an audio clip note when its note number field matches the one used in a previous MIDI Note On message.
- The Velocity field of the Note On message will control the clip volume, according to the response specified for the Mobile DLS 1.0 default velocity-to-gain connection.
- If the selected audio clip is already playing on the indicated MIDI channel at the time of the MIDI Note On message, the audio clip will 'rewind' wavetable playback to the start of the wavetable. Audio clip instruments are always non-polyphonic and 'self-exclusive', even if the Mobile DLS instrument program calls for polyphonic behavior.
- The audio clip instrument will not respond to the MIDI Pitch Bend message, even if the Mobile DLS instrument program calls for it.
- The audio clip will respond to the MIDI Polyphonic Key Pressure or Channel Pressure messages only if the selected instrument program route them to a supported destination, i.e. clip volume, or for mono wavetables rendered on stereo devices, pan.
- In addition to MIDI Bank Select MSB & LSB, the audio clip instrument will respond to the MIDI Control Change message only if the controller number is either 7 (Volume) or 11 (Expression), or for mono wavetables rendered on a stereo device, 10 (Pan). No other control numbers will produce any audible effect, even if the selected Mobile DLS instrument program calls for it.

## 2.5 Use of XMF Meta File Format Features

The rules regarding the use of XMF Meta File Format features for Mobile XMF content that includes Audio Clips are the same as for Mobile XMF v1.0 (see section 7 of [2]) with the following exception. Use of the ID3 Metadata field (XMF Metadata Standard FieldID14, as defined in [8]) is allowed.

## 2.6 Content Handling Behaviors

The rules regarding content handling behavior for Mobile XMF content that includes Audio Clips are the same as for Mobile XMF v1.0 (see section 8 of [2]).

## 2.7 Content Type Identification

With the following exceptions, the rules regarding content type identification for Mobile XMF content that includes Audio Clips are the same as for Mobile XMF v1.0 (see section 9 of [2]).

### *2.7.1 Internal Content Type Indication*

To provide content type information to XMF file parsers and generic file type recognizers, the XmfFileTypeID field in the FileHeader must indicate the proper XMF File Type number, and the XmfFileTypeRevisionID must indicate the proper spec version:

| | |
|---|---|
| XMF File Type: | 3 |
| Spec Revision Level: | 0 |

### 2.7.2 External Content Type Indication

Filename extension and MIME media type are the same as for Mobile XMF 1.0 content; see section 9 of [2].

## 2.8 Content Description Metadata

With the following exceptions, Content Description Metadata (CDM) for Mobile XMF content that includes Audio Clips is the same as for Mobile XMF 1.0 content: see section 10 of [2].

For Mobile XMF content that includes Audio Clips, CDM is extended in the following ways.

The following table extends the table in section 10.3.5 Playback Resource Group List of [2]:

| PlaybackResourceGroupID | Group Name | Mutual Exclusivity | Channel Masking |
|---|---|---|---|
| 3 | Audio Clip Resources with Mutual Exclusivity | Yes | No |
| 4 | Audio Clip Resources without Mutual Exclusivity | No | No |
| 5 | Audio Clip Voices | Yes | Yes |

Five new Standard PlaybackResourceID values are available when a Mobile XMF file includes audio clip instruments The following table extends the table in section 10.3.4.1 Standard PlaybackResourceIDs of [2]:

| PlaybackResourceID Value | Description | For Use with PlaybackResourceGroupID |
|---|---|---|
| 5 | Number of Audio Clips voices | 5: Audio clip voices |
| 6 | Total encoded wavetable data (in kilobytes) for Audio Clips | 3: Audio Clip Resources with Mutual Exclusivity |
| 7 | Audio clip maximum sample rate (in kilosamples per 10 seconds) | 4: Audio Clip Resources without Mutual Exclusivity |
| 8 | Audio clip average bit rate (in kilobits per 10 seconds) | 3: Audio Clip Resources with Mutual Exclusivity |
| 9 | Audio clip maximum bit rate (in kilobits per 10 seconds) | 3: Audio Clip Resources with Mutual Exclusivity |

A MIDI Channel with a non-zero entry in a MIR column whose PRL entry is { 00h, 05h } (Audio Clip Voices) is an audio clip channel. For Mobile XMF content that includes audio clips, this column MUST always be present.  Because audio clip polyphony is limited to 1 in v1.0 of the Audio Clips for Mobile XMF specification, the maximum value in this column is 1.

**NOTE**: If any audio clips use any non-linear PCM codecs, these codec(s) MUST be indicated in MIR column(s) with PRL entry { 04x, <wFormatTag> } or { 05h, <GUID> }. A non-zero entry in such a MIR column indicates that that MIDI channel uses that codec. Because this column uses the exact same encodings as an ordinary Mobile DLS synthesizer voice, it cannot be used to distinguish between a Mobile DLS synthesizer voice and an audio clip voice; to distinguish between the two, use the separate { 00h, 05h } column instead.

For Mobile XMF content that includes audio clips, MIR columns with the following PRL entries SHOULD be used: { 00h, 06h }, { 00h, 07h }, { 00h, 08h }, and { 00h, 09h }.

## 3. Player Implementation Guidelines

This section consists of non-normative information for implementers of players for Mobile XMF content that includes Audio Clips. See also the normative requirements at section 2.3.2.2 Player Behavior for Audio Clip Instruments.

When handling audio clip instruments, players:

- May optionally render the audio clip instruments using a separate audio file playback facility (hardware or software), in other words not necessarily using the music synthesizer rendering the other Mobile DLS instruments. Players implemented in this manner must meet certain minimum performance requirements.
- May optionally play the audio clip instrument by streaming the wavetable audio data from secondary storage such as a local file system, rather than rendering directly from wavetable audio data stored wholly within the synthesizer's wavetable memory. Players offering this functionality must meet certain minimum performance requirements.

# 4. Content Authoring Guidelines

This section provides guidance to content authors producing Mobile XMF content that includes Audio Clip instruments.

## 4.1 Overview

The Audio Clips for Mobile XMF specification defines an extension of Mobile XMF v1.0 [2] that allows the use of instruments with longer mono or stereo 'audio clip' wavetables which may use various data-compressed audio encodings (for example: AAC, aacPlus, Enhanced aacPlus, AMR-WB, Extended AMR Wideband, IMA ADPCM, MP3, etc.). This means short audio snippets synced to a MIDI timeline, not very long audio playing in parallel with very long MIDI timeline. Composer control is limited to starting and stopping audio clips, plus volume and (for mono wavetables rendered on stereo devices) pan. This level of playback functionality is suitable for integrating sections of recorded music together with MIDI rendered notes, which is a common and appropriate production technique for producing full-length branded music content as well as other content styles at file sizes much smaller than conventionally data-compressed recorded music.

See also: Section 1.1. Overview.

## 4.2 Mobile DLS Content Authoring for Audio Clips

Wavetable content intended for use as audio clips goes into Mobile XMF files in the form of Mobile DLS instruments. Audio Clip instruments have exactly the same format as any other Mobile DLS instrument, and the player handles them exactly the same, except for the following rules which you should keep in mind when creating audio clip content:

- Audio clip instruments must appear in bank MSB 0x7A, LSB 0x00. Do not use banks MSB 0x7A, LSB 0x01 through MSB 0x7A, LSB 0x7F as they are reserved for future MMA/AMEI definition.
- Audio clip wavetables are never transposed to different pitches/notes, they are always played at the wavetable's natural pitch. The note number field of the MIDI Note On message does not affect pitch, the Pitch Bend message is ignored, and the Pitch Fine Tuning field of the WSMP chunk is ignored. So to minimize file size and avoid wasting space, don't include modulation connections with the Pitch destination.
- The Pitch Fine Tuning field of the WSMP chunk is ignored. If you need to tune the wavetable, use an audio editor program before importing the sample into the Mobile DLS file.
- Audio clip instruments can use significantly larger wavetables than ordinary Mobile DLS instruments. They don't have to fit into the synthesizer's wavetable memory, but can be streamed from secondary storage such as a local file system. See section 2.3.2.3 for size guidance.
- Audio clip wavetables may have data bit rate maximums. (This is distinct from the audio sampling rate.) See section 2.3.2.4 for bit rate guidance.
- Audio clip wavetables may use different audio encodings than ordinary Mobile DLS instruments, and therefore may require different decoders in the player. See section 2.3.2.5 for codec selection guidance.
- Audio clip wavetables can't loop. Any loop information included in the audio clip wavetable data will be ignored. So to minimize file size, don't include loop information in your wavetables.
- Audio clip instruments always treat the Velocity field of the MIDI Note On message as a Volume control. You can't prevent or override this by providing different modulation connections in the instrument definition.

- To trim an instrument's gain, use the WSMP chunk's Gain Trim field.
- Audio clip instruments never use the lowpass filter. So don't create instruments that depend on the filter.  And to minimize file size, don't include any connections that set or modulate the filter parameters.
- Audio clip instruments will ignore any modulation connections with destinations other than Volume, and if the wavetable is mono and the device is stereo, Pan. So to minimize file size, don't include modulation connections with any destination other than Volume and Pan, and don't use Pan if the instrument wavetable is stereo.

## 4.3 SMF File Authoring for Audio Clips

Audio clips will not be played unless the SMF inside the Mobile XMF file includes the necessary MIDI messages. To play an audio clip, you must devote a MIDI channel to it, select the Mobile DLS audio clip instrument program that references the desired audio clip wavetable, and then send MIDI Note On and Note Off messages on that MIDI channel to start and stop the clip.

### *4.3.1 Audio Clip Polyphony*

Audio clip polyphony is limited to 1. When an audio clip instrument is playing, any subsequently started audio clip will replace the first one.

### *4.3.2 Selecting Audio Clip Instruments*

See section 2.4.1.

### *4.3.3 Triggering Audio Clips*

See section 2.4.2.

## 5. Authoring Tool Guidelines

This section provides guidance to Mobile XMF authoring tool developers who wish to support content that includes Mobile DLS Audio Clip instruments.

- All Mobile DLS audio clip instruments must appear in bank MSB 0x7A, LSB 0x00. Do not use banks MSB 0x7A, LSB 0x01 through MSB 0x7A, LSB 0x7F as they are reserved for future MMA/AMEI definition.
- All wavetables used in audio clip instruments use the WAVE_FORMAT_EXTENSIBLE mechanism to identify their audio data encoding, and consequently the required decoder. However it is expected that some application domains for Audio Clips for Mobile XMF (i.e. format or standards specifications that refer to this specification but define a unique decoder set) may specify support for different sets of audio decoders for audio clips. Authoring tools should therefore guide content creators away from the use of audio decoders not appropriate for the intended end deployment of the content.
- Audio clip instruments only respond to the Volume connection, and (if the wavetable is mono and the device is stereo) the Pan connection, so to save space any connections with other destinations should be omitted (perhaps stripped) from the final Mobile DLS and/or Mobile XMF content.
- Volume and Pan have defined default connections, so if the content creator only uses default values, there is no need to include these connections in the final Mobile DLS and/or Mobile XMF content either.

## 6. References

[1] *"MIDI 1.0 Detailed Specification, Document Version 4.2"*, in "The Complete MIDI 1.0 Detailed Specification, Document Version 96.1", The MIDI Manufacturers Association Inc., Los Angeles, CA, USA, February 1996

[2] *"Mobile XMF Content Format Specification"*, RP-042, MIDI Manufacturers Association, Los Angeles, CA, USA, 2004

[3] *"Mobile DLS Specification"*, RP-041, MIDI Manufacturers Association, Los Angeles, CA, USA, 2004

[4] *"Scalable Polyphony MIDI Specification, Version 1.0"* (SP-MIDI), RP-034, MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002

[5] *"Scalable Polyphony MIDI Device 5–24 Note Profile for 3GPP, Version 1.0", RP-035,* MIDI Manufacturers Association, Los Angeles, CA, USA, February 2002

[6] *"3GPP Technical Specification 26.234 for Release 6 : End-to-end transparent streaming service; Protocols and codecs"*, http://www.3gpp.org/ftp/Specs/archive/26_series/26.234/26234-680.zip. The 3rd Generation Partnership Project, Sophia Antipolis, France, 2004

[7] *"Standard MIDI Files"*, RP-001,in The Complete MIDI 1.0 Detailed Specification, Document Version 96.1, MIDI Manufacturers Association, Los Angeles, CA, USA, 1996

[8] *"ID3 Metadata for XMF Files"*, RP-047, MIDI Manufacturers Association, Los Angeles, CA, USA, 2007

THIS PAGE IS INTENTIONALLY BLANK

# ID3 Metadata for XMF Files (RP-047)

## 0. Abstract and Background

Defines a new XMF Standard Metadata FieldID for encapsulating ID3-format metadata. As such, it could potentially be used in many different XMF File Types.

Currently Mobile XMF content is not allowed to include any XMF informational metadata. This puts Mobile XMF content at a disadvantage compared to most other music content formats. ID3 metadata is the de facto world standard for recorded music metadata. Now that the Audio Clips for Mobile XMF specification is about to make some form of recorded music in Mobile XMF files more feasible, companies in the Mobile WG feel that supporting the same metadata format the rest of the world is already using just makes sense. Players support already it (so ID3 parsers can be repurposed in part), end users are already accustomed to it (so collecting & using Mobile XMF digital audio content can be made to feel more like the MP3 or iTunes experience), and content developers already know how to provide it properly.

In this specification, all possible MP3/ID3 metadata can be added to XMF by simply defining one new XMF Standard Metadata FieldID. There would be no need to support multiple XMF informational metadata fields. This approach has the added bonus that ID3 metadata could also be used in any future XMF file type, not just Mobile XMF. MP3 to Mobile XMF conversions would be simplified as there would be no need to individually create and populate all the separate XMF standard metadata fields; instead, a tool could simply copy the whole ID3 block. Because the ID3 spec developers invite other organizations to reference their spec, there are no known intellectual property rights issues with this dependency on an external specification.

## 1. Introduction

This defines a new XMF Standard Metadata FieldID for encapsulating ID3-format metadata, version 2.4. Each XMF MetaDataItem contains one block of ID3 data. The block of ID3 metadata may contain any number of ID3 metadata frames.

The binary format and available metadata fields for ID3 metadata are defined in documents created and maintained by ID3.org. To provide a stable reference, those documents (as they exist when this is written) will be published with the XMF Meta File Format Specification [1]. These texts are included in this RP as "Annex 1" and "Annex 2".

## 2. Definition of XMF MetaData Item format

### 2.1. FieldID Definition

The following row is added to the table in section **5.2.1. Standard FieldID Assignments** of [1]:

| FieldID | FieldName and Notes | Valid for NodeTypes | Contents Format |
|---|---|---|---|
| **14** | **ID3 Metadata** Contains a block of ID3v2.4.0 Metadata | File or Folder, but in Mobile XMF files must be attached to the FileNode containing the playable SMF | Universal, Binary (hidden or visible)<br>See section 5.2.2 for **FieldContents** format. |

## 2.2. Binary Format Definition

The following text is added to section **5.2.2 Field Contents Interpretation** of [1]:

"**FieldID 14: ID3 Metadata**

The ID3 Metadata meta-data field encapsulates any amount of ID3 metadata in a single XMF MetadDataItem. ID3 is the world defacto standard for recorded music metadata and is used in MP3 files and several proprietary recorded music file formats.

The FieldContents of an ID3 Metadata MetaDataItem must consist of an entire valid ID3 metadata block as specified at http://www.id3.org/id3v2.4.0-structure.txt (reproduced in Appendix 2 of this specification). Valid frame tags and their data formats are specified at http://www.id3.org/id3v2.4.0-frames.txt (reproduced in Appendix 3 of this specification).

Example FieldContents:

```
// Start of XMF metadata item FieldContents

// Start of 10 byte ID3 header
"ID3"                     // ID3v2/file identifier

04h 00h                   // ID3v2 version

00h                       // ID3v2 flags

00h 00h 00h 26h           // ID3v2 size: 48 (dec) bytes – 10 for header = 38

// End of 10 byte ID3 header

// (Optional ID3 extended header would appear here)

// Start of First ID3 frame
"TIT2"               // Four-Char frame ID: song Title
00h 00h 00h 08h      // Size: 18 (dec) bytes in frame – 10 for header = 8
00h 00h              // Flags
// Data bytes for the 'TIT2' frame:
00h                  // Text is in ISO-8859-1 format
"Adagio"             // Text contents
00h                  // String terminator (required for ID3 v2.4)
// End of first ID3 frame

// Start of Second ID3 frame
"TCOM"               // Four-Char frame ID: Composer name
00h 00h 00h 0Ah      // Size: 20 (dec) bytes in frame – 10 for header = 10
00h 00h              // Flags
// Data bytes for the 'TCOM' frame:
00h                  // Text is in ISO-8859-1 format
"John Doe"           // Text contents
00h                  // String terminator (required for ID3 v2.4)
// End of second ID3 frame

// (Optional ID3 padding would appear here)

// (Optional ID3 footer would appear here)

// End of XMF metadata item FieldContents
```

"

## 2.3. ID3 Binary Data Format Definition References

The full text of the Annex of this document is added to [1] as Appendix 2 and Appendix 3, respectively. For consistency, the original Appendix of [1] is renamed Appendix 1.

## 3. References

[1] “Specification for XMF Meta File Format”, RP-030, MIDI Manufacturer’s Association, Los Angeles, CA, USA, 2001

## Annex 1: ID3 v2.4 Metadata Format

*Note: The text of this appendix was copied verbatim from http://www.id3.org/id3v2.4.0-structure.txt on August 7, 2006. However, for XMF usage, this Appendix is the definitive reference, not ID3.org.*

Informal standard
M. Nilsson
Document: id3v2.4.0-structure.txt
1st November 2000

ID3 tag version 2.4.0 - Main Structure

Status of this document

This document is an informal standard and replaces the ID3v2.3.0 standard [ID3v2]. A formal standard will use another revision number even if the content is identical to document. The contents in this document may change for clarifications but never for added or altered functionallity.

Distribution of this document is unlimited.

Abstract

This document describes the main structure of ID3v2.4.0, which is a revised version of the ID3v2 informal standard [ID3v2] version 2.3.0. The ID3v2 offers a flexible way of storing audio meta information within the audio file itself. The information may be technical information, such as equalisation curves, as well as title, performer, copyright etc.
ID3v2.4.0 is meant to be as close as possible to ID3v2.3.0 in order to allow for implementations to be revised as easily as possible.

1. Table of contents

2. Conventions in this document

Text within "" is a text string exactly as it appears in a tag.
Numbers preceded with $ are hexadecimal and numbers preceded with % are binary. $xx is used to indicate a byte with unknown content. %x is used to indicate a bit with unknown content. The most significant bit (MSB) of a byte is called 'bit 7' and the least significant bit (LSB) is called 'bit 0'.

A tag is the whole tag described in this document. A frame is a block of information in the tag. The tag consists of a header, frames and optional padding. A field is a piece of information; one value, a string etc. A numeric string is a string that consists of the characters "0123456789" only.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

## 3. ID3v2 overview

ID3v2 is a general tagging format for audio, which makes it possible to store meta data about the audio inside the audio file itself. The ID3 tag described in this document is mainly targeted at files encoded with MPEG-1/2 layer I, MPEG-1/2 layer II, MPEG-1/2 layer III and MPEG-2.5, but may work with other types of encoded audio or as a stand alone format for audio meta data.

ID3v2 is designed to be as flexible and expandable as possible to meet new meta information needs that might arise. To achieve that ID3v2 is constructed as a container for several information blocks, called frames, whose format need not be known to the software that encounters them. At the start of every frame is an unique and predefined identifier, a size descriptor that allows software to skip unknown frames and a flags field. The flags describes encoding details and if the frame should remain in the tag, should it be unknown to the software, if the file is altered.

The bitorder in ID3v2 is most significant bit first (MSB). The byteorder in multibyte numbers is most significant byte first (e.g. $12345678 would be encoded $12 34 56 78), also known as big endian and network byte order.

Overall tag structure:

```
+-----------------------------+
|      Header (10 bytes)      |
+-----------------------------+
|       Extended Header       |
| (variable length, OPTIONAL) |
+-----------------------------+
|   Frames (variable length)  |
+-----------------------------+
|           Padding           |
| (variable length, OPTIONAL) |
+-----------------------------+
| Footer (10 bytes, OPTIONAL) |
+-----------------------------+
```

In general, padding and footer are mutually exclusive. See details in sections 3.3, 3.4 and 5.

### 3.1. ID3v2 header

The first part of the ID3v2 tag is the 10 byte tag header, laid out as follows:

```
ID3v2/file identifier      "ID3"
ID3v2 version              $04 00
ID3v2 flags                %abcd0000
ID3v2 size             4 * %0xxxxxxx
```

The first three bytes of the tag are always "ID3", to indicate that this is an ID3v2 tag, directly followed by the two version bytes. The first byte of ID3v2 version is its major version, while the second byte is its revision number. In this case this is ID3v2.4.0. All revisions are backwards compatible while major versions are not. If software with ID3v2.4.0 and below support should encounter version five or higher it should simply ignore the whole tag. Version or revision will never be $FF.

The version is followed by the ID3v2 flags field, of which currently four flags are used.

a - Unsynchronisation

Bit 7 in the 'ID3v2 flags' indicates whether or not unsynchronisation is applied on all frames (see section 6.1 for details); a set bit indicates usage.

b - Extended header

The second bit (bit 6) indicates whether or not the header is followed by an extended header. The extended header is described in section 3.2. A set bit indicates the presence of an extended header.

c - Experimental indicator

The third bit (bit 5) is used as an 'experimental indicator'. This flag SHALL always be set when the tag is in an experimental stage.

d - Footer present

Bit 4 indicates that a footer (section 3.4) is present at the very end of the tag. A set bit indicates the presence of a footer.

All the other flags MUST be cleared. If one of these undefined flags are set, the tag might not be readable for a parser that does not know the flags function.

The ID3v2 tag size is stored as a 32 bit synchsafe integer (section 6.2), making a total of 28 effective bits (representing up to 256MB).

The ID3v2 tag size is the sum of the byte length of the extended header, the padding and the frames after unsynchronisation. If a footer is present this equals to ('total size' - 20) bytes, otherwise ('total size' - 10) bytes.

An ID3v2 tag can be detected with the following pattern:

```
  $49 44 33 yy yy xx zz zz zz zz
```

Where yy is less than $FF, xx is the 'flags' byte and zz is less than $80.

## 3.2. Extended header

The extended header contains information that can provide further  insight in the structure of the tag, but is not vital to the correct parsing of the tag information; hence the extended header is optional.

```
Extended header size   4 * %0xxxxxxx
Number of flag bytes       $01
Extended Flags             $xx
```

Where the 'Extended header size' is the size of the whole extended header, stored as a 32 bit synchsafe integer. An extended header can thus never have a size of fewer than six bytes.

The extended flags field, with its size described by 'number of flag bytes', is defined as:

```
%0bcd0000
```

Each flag that is set in the extended header has data attached, which comes in the order in which the flags are encountered (i.e. the data for flag 'b' comes before the data for flag 'c'). Unset flags cannot have any attached data. All unknown flags MUST be unset and their corresponding data removed when a tag is modified.

Every set flag's data starts with a length byte, which contains a value between 0 and 128 ($00 - $7f), followed by data that has the field length indicated by the length byte. If a flag has no attached data, the value $00 is used as length byte.

b - Tag is an update

If this flag is set, the present tag is an update of a tag found earlier in the present file or stream. If frames defined as unique are found in the present tag, they are to override any corresponding ones found in the earlier tag. This flag has no corresponding data.

```
Flag data length      $00
```

c - CRC data present

If this flag is set, a CRC-32 [ISO-3309] data is included in the extended header. The CRC is calculated on all the data between the header and footer as indicated by the header's tag length field, minus the extended header. Note that this includes the padding (if there is any), but excludes the footer. The CRC-32 is stored as an 35 bit synchsafe integer, leaving the upper four bits always zeroed.

```
Flag data length       $05
Total frame CRC    5 * %0xxxxxxx
```

d - Tag restrictions

For some applications it might be desired to restrict a tag in more ways than imposed by the ID3v2 specification. Note that the presence of these restrictions does not affect how the tag is decoded, merely how it was restricted before encoding. If this flag is set the tag is restricted as follows:

```
Flag data length       $01
Restrictions           %ppqrrstt
```

p - Tag size restrictions

```
  00   No more than 128 frames and
        1 MB total tag size.
  01   No more than 64 frames and
        128 KB total tag size.
  10   No more than 32 frames and
        40 KB total tag size.
  11   No more than 32 frames and 4 KB total tag size.
```

q - Text encoding restrictions

```
  0    No restrictions
  1    Strings are only encoded with ISO-8859-1
        [ISO-8859-1] or UTF-8 [UTF-8].
```

r - Text fields size restrictions

```
  00   No restrictions
  01   No string is longer than 1024 characters.
  10   No string is longer than 128 characters.
  11   No string is longer than 30 characters.
```

Note that nothing is said about how many bytes is used to represent those characters, since it is encoding dependent. If a text frame consists of more than one string, the sum of the strungs is restricted as stated.

s - Image encoding restrictions

```
  0   No restrictions
  1   Images are encoded only with PNG [PNG]
        or JPEG [JFIF].
```

t - Image size restrictions

```
  00  No restrictions
  01  All images are 256x256 pixels or smaller.
  10  All images are 64x64 pixels or smaller.
  11  All images are exactly 64x64 pixels,
       unless required otherwise.
```

## 3.3. Padding

It is OPTIONAL to include padding after the final frame (at the end of the ID3 tag), making the size of all the frames together smaller than the size given in the tag header. A possible purpose of this padding is to allow for adding a few additional frames or enlarge existing frames within the tag without having to rewrite the entire file. The value of the padding bytes must be $00. A tag MUST NOT have any padding between the frames or between the tag header and the frames. Furthermore it MUST NOT have any padding when a tag footer is added to the tag.

## 3.4. ID3v2 footer

To speed up the process of locating an ID3v2 tag when searching from the end of a file, a footer can be added to the tag. It is REQUIRED to add a footer to an appended tag, i.e. a tag located after all audio data. The footer is a copy of the header, but with a different identifier.

```
ID3v2 identifier           "3DI"
ID3v2 version              $04 00
ID3v2 flags                %abcd0000
ID3v2 size             4 * %0xxxxxxx
```

## 4. ID3v2 frame overview

All ID3v2 frames consists of one frame header followed by one or more fields containing the actual information. The header is always 10 bytes and laid out as follows:

```
Frame ID      $xx xx xx xx  (four characters)
Size      4 * %0xxxxxxx
Flags         $xx xx
```

The frame ID is made out of the characters capital A-Z and 0-9.
Identifiers beginning with "X", "Y" and "Z" are for experimental frames and free for everyone to use, without the need to set the experimental bit in the tag header. Bear in mind that someone else might have used the same identifier as you. All other identifiers are either used or reserved for future use.

The frame ID is followed by a size descriptor containing the size of the data in the final frame, after encryption, compression and unsynchronisation. The size is excluding the frame header ('total frame size' - 10 bytes) and stored as a 32 bit synchsafe integer.

In the frame header the size descriptor is followed by two flag bytes. These flags are described in section 4.1.

There is no fixed order of the frames' appearance in the tag, although it is desired that the frames are arranged in order of significance concerning the recognition of the file. An example of such order: UFID, TIT2, MCDI, TRCK ...

A tag MUST contain at least one frame. A frame must be at least 1 byte big, excluding the header.

If nothing else is said, strings, including numeric strings and URLs [URL], are represented as ISO-8859-1 [ISO-8859-1] characters in the range $20 - $FF. Such strings are represented in frame descriptions as <text string>, or <full text string> if newlines are allowed. If nothing else is said newline character is forbidden. In ISO-8859-1 a newline is represented, when allowed, with $0A only.

Frames that allow different types of text encoding contains a text encoding description byte. Possible encodings:

$00 ISO-8859-1 [ISO-8859-1]. Terminated with $00.
$01 UTF-16 [UTF-16] encoded Unicode [UNICODE] with BOM. All strings in the same frame SHALL have the same byteorder. Terminated with $00 00.
$02 UTF-16BE [UTF-16] encoded Unicode [UNICODE] without BOM. Terminated with $00 00.
$03 UTF-8 [UTF-8] encoded Unicode [UNICODE]. Terminated with $00.

Strings dependent on encoding are represented in frame descriptions as <text string according to encoding>, or <full text string according to encoding> if newlines are allowed. Any empty strings of type $01 which are NULL-terminated may have the Unicode BOM followed by a Unicode NULL ($FF FE 00 00 or $FE FF 00 00).

The timestamp fields are based on a subset of ISO 8601. When being as precise as possible the format of a time string is yyyy-MM-ddTHH:mm:ss (year, "-", month, "-", day, "T", hour (out of 24), ":", minutes, ":", seconds), but the precision may be reduced by removing as many time indicators as wanted. Hence valid timestamps are yyyy, yyyy-MM, yyyy-MM-dd, yyyy-MM-ddTHH, yyyy-MM-ddTHH:mm and yyyy-MM-ddTHH:mm:ss. All time stamps are UTC. For durations, use the slash character as described in 8601, and for multiple non-contiguous dates, use multiple strings, if allowed by the frame definition.

The three byte language field, present in several frames, is used to describe the language of the frame's content, according to ISO-639-2 [ISO-639-2]. The language should be represented in lower case. If the language is not known the string "XXX" should be used.

All URLs [URL] MAY be relative, e.g. "picture.png", "../doc.txt".

If a frame is longer than it should be, e.g. having more fields than specified in this document, that indicates that additions to the frame have been made in a later version of the ID3v2 standard. This is reflected by the revision number in the header of the tag.

## 4.1. Frame header flags

In the frame header the size descriptor is followed by two flag bytes. All unused flags MUST be cleared. The first byte is for 'status messages' and the second byte is a format description. If an unknown flag is set in the first byte the frame MUST NOT be changed without that bit cleared. If an unknown flag is set in the second byte the frame is likely to not be readable. Some flags in the second byte indicates that extra information is added to the header. These fields of extra information is ordered as the flags that indicates them. The flags field is defined as follows (l and o left out because of their resemblence to one and zero):

```
%0abc0000 %0h00kmnp
```

Some frame format flags indicate that additional information fields are added to the frame. This information is added after the frame header and before the frame data in the same order as the flags that indicates them. I.e. the four bytes of decompressed size will precede the encryption method byte. These additions affects the 'frame size' field, but are not subject to encryption or compression.

The default status flags setting for a frame is, unless stated otherwise, 'preserved if tag is altered' and 'preserved if file is altered', i.e. %00000000.

```
4.1.1. Frame status flags

   a - Tag alter preservation

     This flag tells the tag parser what to do with this
frame if it is unknown and the tag is altered in any way.
This applies to all kinds of alterations, including adding
more padding and reordering the frames.

     0     Frame should be preserved.
     1     Frame should be discarded.

   b - File alter preservation

     This flag tells the tag parser what to do with this
frame if it is unknown and the file, excluding the tag, is
altered. This does not apply when the audio is completely
replaced with other audio data.

     0     Frame should be preserved.
     1     Frame should be discarded.

   c - Read only

      This flag, if set, tells the software that the
contents of this frame are intended to be read only.
Changing the contents might break something, e.g. a
signature. If the contents are changed, without knowledge
of why the frame was flagged read only and without taking
the proper means to compensate, e.g. recalculating the
signature, the bit MUST be cleared.

4.1.2. Frame format flags

   h - Grouping identity

      This flag indicates whether or not this frame belongs
in a group with other frames. If set, a group identifier
byte is added to the frame. Every frame with the same group
identifier belongs to the same group.

      0     Frame does not contain group information
      1     Frame contains group information

   k - Compression

      This flag indicates whether or not the frame is
compressed.
```

```
      A 'Data Length Indicator' byte MUST be included in
the frame.

      0     Frame is not compressed.
      1     Frame is compressed using zlib [zlib]
            deflate method.
            If set, this requires the 'Data Length
            Indicator' bit to be set as well.


   m - Encryption

      This flag indicates whether or not the frame is
encrypted. If set, one byte indicating with which method it
was encrypted will be added to the frame. See description
of the ENCR frame for more information about encryption
method registration. Encryption should be done after
compression. Whether or not setting this flag requires the
presence of a 'Data Length Indicator' depends on the
specific algorithm used.

      0     Frame is not encrypted.
      1     Frame is encrypted.

   n - Unsynchronisation

      This flag indicates whether or not unsynchronisation
was applied to this frame. See section 6 for details on
unsynchronisation. If this flag is set all data from the
end of this header to the end of this frame has been
unsynchronised. Although desirable, the presence of a 'Data
Length Indicator' is not made mandatory by
unsynchronisation.

      0     Frame has not been unsynchronised.
      1     Frame has been unsyrchronised.

   p - Data length indicator

      This flag indicates that a data length indicator has
been added to the frame. The data length indicator is the
value one would write as the 'Frame length' if all of the
frame format flags were zeroed, represented as a 32 bit
synchsafe integer.

      0      There is no Data Length Indicator.
      1      A data length Indicator has been added
             to the frame.
```

## 5. Tag location

The default location of an ID3v2 tag is prepended to the audio so that players can benefit from the information when the data is streamed. It is however possible to append the tag, or make a prepend/append combination. When deciding upon where an unembedded tag should be located, the following order of preference SHOULD be considered.

1. Prepend the tag.

2. Prepend a tag with all vital information and add a second tag at the end of the file, before tags from other tagging systems. The first tag is required to have a SEEK frame.

3. Add a tag at the end of the file, before tags from other tagging systems.

In case 2 and 3 the tag can simply be appended if no other known tags are present. The suggested method to find ID3v2 tags are:

1. Look for a prepended tag using the pattern found in section 3.1.

2. If a SEEK frame was found, use its values to guide further searching.

3. Look for a tag footer, scanning from the back of the file.

For every new tag that is found, the old tag should be discarded unless the update flag in the extended header (section 3.2) is set.

## 6. Unsynchronisation

The only purpose of unsynchronisation is to make the ID3v2 tag as compatible as possible with existing software and hardware. There is no use in 'unsynchronising' tags if the file is only to be processed only by ID3v2 aware software and hardware. Unsynchronisation is only useful with tags in MPEG 1/2 layer I, II and III, MPEG 2.5 and AAC files.

### 6.1. The unsynchronisation scheme

Whenever a false synchronisation is found within the tag, one zeroed byte is inserted after the first false synchronisation byte. The format of synchronisations that should be altered by ID3 encoders is as follows:

```
%11111111 111xxxxx
```

and should be replaced with:

```
%11111111 00000000 111xxxxx
```

This has the side effect that all $FF 00 combinations have to be altered, so they will not be affected by the decoding process. Therefore all the $FF 00 combinations have to be replaced with the $FF 00 00 combination during the unsynchronisation.

To indicate usage of the unsynchronisation, the unsynchronisation flag in the frame header should be set. This bit MUST be set if the frame was altered by the unsynchronisation and SHOULD NOT be set if unaltered. If all frames in the tag are unsynchronised the unsynchronisation flag in the tag header SHOULD be set. It MUST NOT be set if the tag has a frame which is not unsynchronised.

Assume the first byte of the audio to be $FF. The special case when the last byte of the last frame is $FF and no padding nor footer is used will then introduce a false synchronisation. This can be solved by adding a footer, adding padding or unsynchronising the frame and add $00 to the end of the frame data, thus adding more byte to the frame size than a normal unsynchronisation would. Although not preferred, it is allowed to apply the last method on all frames ending with $FF.

It is preferred that the tag is either completely unsynchronised or not unsynchronised at all. A completely unsynchronised tag has no false synchonisations in it, as defined above, and does not end with $FF. A completely non-unsynchronised tag contains no unsynchronised frames, and thus the unsynchronisation flag in the header is cleared.

Do bear in mind, that if compression or encryption is used, the unsynchronisation scheme MUST be applied afterwards. When decoding an unsynchronised frame, the unsynchronisation scheme MUST be reversed first, encryption and decompression afterwards.

## 6.2. Synchsafe integers

In some parts of the tag it is inconvenient to use the unsychronisation scheme because the size of unsynchronised data is not known in advance, which is particularly problematic with size descriptors. The solution in ID3v2 is to use synchsafe integers, in which there can never be any false synchs. Synchsafe integers are integers that keep its highest bit (bit 7) zeroed, making seven bits out of eight available. Thus a 32 bit synchsafe integer can store 28 bits of information.

Example:

255 (%11111111) encoded as a 16 bit synchsafe integer is 383 (%00000001 01111111).

## 7. Copyright

Copyright (C) Martin Nilsson 2000. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that a reference to this document is included on all such copies and derivative works. However, this document itself may not be modified in any way and reissued as the original document.

The limited permissions granted above are perpetual and will not be revoked.

This document and the information contained herein is provided on an 'AS IS' basis and THE AUTHORS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## 8. References

[ID3v2] Martin Nilsson, 'ID3v2 informal standard'. <url:http://www.id3.org/id3v2.3.0.txt>

[ISO-639-2] ISO/FDIS 639-2. 'Codes for the representation of names of languages, Part 2: Alpha-3 code.' Technical committee / subcommittee: TC 37 / SC 2

[ISO-3309] ISO 3309 'Information Processing Systems--Data Communication High-Level Data Link Control Procedure--Frame Structure', IS 3309, October 1984, 3rd Edition.

[ISO-8859-1] ISO/IEC DIS 8859-1. '8-bit single-byte coded graphic character sets, Part 1: Latin alphabet No. 1.' Technical committee / subcommittee: JTC 1 / SC 2 [JFIF] 'JPEG File Interchange Format, version 1.02' <url:http://www.w3.org/Graphics/JPEG/jfif.txt>

[KEYWORDS] S. Bradner, 'Key words for use in RFCs to Indicate Requirement Levels', RFC 2119, March 1997. <url:ftp://ftp.isi.edu/in-notes/rfc2119.txt>

[MPEG] ISO/IEC 11172-3:1993. 'Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 3: Audio.' Technical committee / subcommittee: JTC 1 / SC 29
and
ISO/IEC 13818-3:1995 'Generic coding of moving pictures and associated audio information, Part 3: Audio.' Technical committee / subcommittee: JTC 1 / SC 29
and
ISO/IEC DIS 13818-3 'Generic coding of moving pictures and associated audio information, Part 3: Audio (Revision of ISO/IEC 13818-3:1995)'

[PNG] 'Portable Network Graphics, version 1.0' <url:http://www.w3.org/TR/REC-png-multi.html>

[UNICODE] The Unicode Consortium, 'The Unicode Standard Version 3.0', ISBN 0-201-61633-5. <url:http://www.unicode.org/unicode/standard/versions/Unicode3.0.htm>

[URL] T. Berners-Lee, L. Masinter & M. McCahill, 'Uniform Resource Locators (URL)', RFC 1738, December 1994. <url:ftp://ftp.isi.edu/in-notes/rfc1738.txt>

[UTF-8] F. Yergeau, 'UTF-8, a transformation format of ISO 10646', RFC 2279, January 1998.
<url:ftp://ftp.isi.edu/in-notes/rfc2279.txt>

[UTF-16] F. Yergeau, 'UTF-16, an encoding of ISO 10646', RFC 2781, February 2000.
<url:ftp://ftp.isi.edu/in-notes/rfc2781.txt>

[ZLIB] P. Deutsch, Aladdin Enterprises & J-L. Gailly, 'ZLIB Compressed Data Format Specification version 3.3', RFC 1950, May 1996.
<url:ftp://ftp.isi.edu/in-notes/rfc1950.txt>

9. Author's Address

Written by

Martin Nilsson
Rydsvägen 246 C. 30
SE-584 34 Linköping
Sweden

Email: nilsson@id3.org

## Annex 2: ID3 v2.4 Metadata Frame Definitions

Note: The text of this appendix was copied verbatim from http://www.id3.org/id3v2.4.0-frames.txt on August 7, 2006. However, for XMF usage, this Appendix is the definitive reference, not ID3.org.

$Id: id3v2.4.0-frames.txt,v 1.1 2003/07/27 18:28:34 id3 Exp $

Informal standard
M. Nilsson
Document: id3v2.4.0-frames.txt
1st November 2000

ID3 tag version 2.4.0 - Native Frames

Status of this document

This document is an informal standard and replaces the ID3v2.3.0 standard [ID3v2]. A formal standard will use another revision number even if the content is identical to document. The contents in this document may change for clarifications but never for added or altered functionallity.

Distribution of this document is unlimited.

Abstract

This document describes the frames natively supported by ID3v2.4.0, which is a revised version of the ID3v2 informal standard [ID3v2.3.0] version 2.3.0. The ID3v2 offers a flexible way of storing audio meta information within audio file itself. The information may be technical information, such as equalisation curves, as well as title, performer, copyright etc.

ID3v2.4.0 is meant to be as close as possible to ID3v2.3.0 in order to allow for implementations to be revised as easily as possible.

1. Table of contents

## 2. Conventions in this document

Text within "" is a text string exactly as it appears in a tag. Numbers preceded with $ are hexadecimal and numbers preceded with % are binary. $xx is used to indicate a byte with unknown content. %x is used to indicate a bit with unknown content. The most significant bit (MSB) of a byte is called 'bit 7' and the least significant bit (LSB) is called 'bit 0'.

A tag is the whole tag described the ID3v2 main structure document [ID3v2-strct]. A frame is a block of information in the tag. The tag consists of a header, frames and optional padding. A field is a piece of information; one value, a string etc. A numeric string is a string that consists of the characters "0123456789" only.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

## 3. Default flags

The default settings for the frames described in this document can be divided into the following classes. The flags may be set differently if found more suitable by the software.

1. Discarded if tag is altered, discarded if file is altered.

None.

2. Discarded if tag is altered, preserved if file is altered.

None.

3. Preserved if tag is altered, discarded if file is altered.

ASPI, AENC, ETCO, EQU2, MLLT, POSS, SEEK, SYLT, SYTC, RVA2, TENC, TLEN

4. Preserved if tag is altered, preserved if file is altered.

```
   The rest of the frames.


4.   Declared ID3v2 frames

  The following frames are declared in this draft.

  4.19  AENC Audio encryption
  4.14  APIC Attached picture
  4.30  ASPI Audio seek point index

  4.10  COMM Comments
  4.24  COMR Commercial frame

  4.25  ENCR Encryption method registration
  4.12  EQU2 Equalisation (2)
  4.5   ETCO Event timing codes

  4.15  GEOB General encapsulated object
  4.26  GRID Group identification registration

  4.20  LINK Linked information

  4.4   MCDI Music CD identifier
  4.6   MLLT MPEG location lookup table

  4.23  OWNE Ownership frame

  4.27  PRIV Private frame
  4.16  PCNT Play counter
  4.17  POPM Popularimeter
  4.21  POSS Position synchronisation frame

  4.18  RBUF Recommended buffer size
  4.11  RVA2 Relative volume adjustment (2)
  4.13  RVRB Reverb

  4.29  SEEK Seek frame
  4.28  SIGN Signature frame
  4.9   SYLT Synchronised lyric/text
  4.7   SYTC Synchronised tempo codes

  4.2.1 TALB Album/Movie/Show title
  4.2.3 TBPM BPM (beats per minute)
  4.2.2 TCOM Composer
  4.2.3 TCON Content type
  4.2.4 TCOP Copyright message
  4.2.5 TDEN Encoding time
  4.2.5 TDLY Playlist delay
  4.2.5 TDOR Original release time
  4.2.5 TDRC Recording time
  4.2.5 TDRL Release time
  4.2.5 TDTG Tagging time
  4.2.2 TENC Encoded by
  4.2.2 TEXT Lyricist/Text writer
  4.2.3 TFLT File type
  4.2.2 TIPL Involved people list
  4.2.1 TIT1 Content group description
  4.2.1 TIT2 Title/songname/content description
  4.2.1 TIT3 Subtitle/Description refinement
  4.2.3 TKEY Initial key
  4.2.3 TLAN Language(s)
  4.2.3 TLEN Length
  4.2.2 TMCL Musician credits list
  4.2.3 TMED Media type
  4.2.3 TMOO Mood
  4.2.1 TOAL Original album/movie/show title
  4.2.5 TOFN Original filename
  4.2.2 TOLY Original lyricist(s)/text writer(s)
  4.2.2 TOPE Original artist(s)/performer(s)
  4.2.4 TOWN File owner/licensee
  4.2.2 TPE1 Lead performer(s)/Soloist(s)
  4.2.2 TPE2 Band/orchestra/accompaniment
  4.2.2 TPE3 Conductor/performer refinement
  4.2.2 TPE4 Interpreted, remixed, or otherwise modified by
  4.2.1 TPOS Part of a set
  4.2.4 TPRO Produced notice
  4.2.4 TPUB Publisher
  4.2.1 TRCK Track number/Position in set
  4.2.4 TRSN Internet radio station name
  4.2.4 TRSO Internet radio station owner
  4.2.5 TSOA Album sort order
  4.2.5 TSOP Performer sort order
  4.2.5 TSOT Title sort order
  4.2.1 TSRC ISRC (international standard recording code)
  4.2.5 TSSE Software/Hardware and settings
             used for encoding
  4.2.1 TSST Set subtitle
  4.2.2 TXXX User defined text information frame

  4.1   UFID Unique file identifier
  4.22  USER Terms of use
  4.8   USLT Unsynchronised lyric/text transcription

  4.3.1 WCOM Commercial information
```

4.3.1 WCOP Copyright/Legal information
4.3.1 WOAF Official audio file webpage
4.3.1 WOAR Official artist/performer webpage
4.3.1 WOAS Official audio source webpage
4.3.1 WORS Official Internet radio station homepage
4.3.1 WPAY Payment
4.3.1 WPUB Publishers official webpage
4.3.2 WXXX User defined URL link frame

## 4.1. Unique file identifier

This frame's purpose is to be able to identify the audio file in a database, that may provide more information relevant to the content.

Since standardisation of such a database is beyond this document, all UFID frames begin with an 'owner identifier' field. It is a null-terminated string with a URL [URL] containing an email address, or a link to a location where an email address can be found, that belongs to the organisation responsible for this specific database implementation. Questions regarding the database should be sent to the indicated email address. The URL should not be used for the actual database queries. The string "http://www.id3.org/dummy/ufid.html" should be used for tests. The 'Owner identifier' must be non-empty (more than just a termination).

The 'Owner identifier' is then followed by the actual identifier, which may be up to 64 bytes. There may be more than one "UFID" frame in a tag, but only one with the same 'Owner identifier'.

```
<Header for 'Unique file identifier', ID: "UFID">
Owner identifier        <text string> $00
Identifier              <up to 64 bytes binary data>
```

## 4.2. Text information frames

The text information frames are often the most important frames, containing information like artist, album and more. There may only be one text information frame of its kind in an tag. All text information frames supports multiple strings, stored as a null separated list, where null is reperesented by the termination code for the charater encoding. All text frame identifiers begin with "T".

Only text frame identifiers begin with "T", with the exception of the "TXXX" frame. All the text information frames have the following format:

```
<Header for 'Text information frame', ID: "T000" - "TZZZ",
excluding "TXXX" described in 4.2.6.>
     Text encoding     $xx
     Information       <text string(s) according to
                        encoding>
```

### 4.2.1. Identification frames

TIT1
The 'Content group description' frame is used if the sound belongs to a larger category of sounds/music. For example, classical music is often sorted in different musical sections (e.g. "Piano Concerto", "Weather - Hurricane").

TIT2
The 'Title/Songname/Content description' frame is the actual name of the piece (e.g. "Adagio", "Hurricane Donna").

TIT3
The 'Subtitle/Description refinement' frame is used for information directly related to the contents title (e.g. "Op. 16" or "Performed live at Wembley").

TALB
The 'Album/Movie/Show title' frame is intended for the title of the recording (or source of sound) from which the audio in the file is taken.

TOAL
The 'Original album/movie/show title' frame is intended for the title of the original recording (or source of sound), if for example the music in the file should be a cover of a previously released song.

TRCK
The 'Track number/Position in set' frame is a numeric string containing the order number of the audio-file on its original recording. This MAY be extended with a "/" character and a numeric string containing the total number of tracks/elements on the original recording. E.g. "4/9".

TPOS
The 'Part of a set' frame is a numeric string that describes which part of a set the audio came from. This frame is used if the source described in the "TALB" frame is divided into several mediums, e.g. a double CD. The value MAY be extended with a "/" character and a numeric string containing the total number of parts in the set. E.g. "1/2".

TSST
The 'Set subtitle' frame is intended for the subtitle of the part of a set this track belongs to.

TSRC
The 'ISRC' frame should contain the International Standard Recording Code [ISRC] (12 characters).

### 4.2.2. Involved persons frames

TPE1
The 'Lead artist/Lead performer/Soloist/Performing group' is used for the main artist.

TPE2
The 'Band/Orchestra/Accompaniment' frame is used for additional information about the performers in the recording.

TPE3
The 'Conductor' frame is used for the name of the conductor.

TPE4
The 'Interpreted, remixed, or otherwise modified by' frame contains more information about the people behind a remix and similar interpretations of another existing piece.

TOPE
The 'Original artist/performer' frame is intended for the performer of the original recording, if for example the music in the file should be a cover of a previously released song.

TEXT
The 'Lyricist/Text writer' frame is intended for the writer of the text or lyrics in the recording.

TOLY
The 'Original lyricist/text writer' frame is intended for the text writer of the original recording, if for example the music in the file should be a cover of a previously released song.

TCOM
The 'Composer' frame is intended for the name of the composer.

TMCL
The 'Musician credits list' is intended as a mapping between instruments and the musician that played it. Every odd field is an instrument and every even is an artist or a comma delimited list of artists.

TIPL
The 'Involved people list' is very similar to the musician credits list, but maps between functions, like producer, and names.

TENC
The 'Encoded by' frame contains the name of the person or organisation that encoded the audio file. This field may contain a copyright message, if the audio file also is copyrighted by the encoder.

### 4.2.3. Derived and subjective properties frames

TBPM
The 'BPM' frame contains the number of beats per minute in the main part of the audio. The BPM is an integer and represented as a numerical string.

TLEN
The 'Length' frame contains the length of the audio file in milliseconds, represented as a numeric string.

TKEY
The 'Initial key' frame contains the musical key in which the sound starts. It is represented as a string with a maximum length of three characters. The ground keys are represented with "A","B","C","D","E", "F" and "G" and half keys represented with "b" and "#". Minor is represented as "m", e.g. "Dbm" $00. Off key is represented with an "o" only.

TLAN
The 'Language' frame should contain the languages of the text or lyrics spoken or sung in the audio. The language is represented with three characters according to ISO-639-2 [ISO-639-2]. If more than one language is used in the text

their language codes should follow according to the amount of their usage, e.g. "eng" $00 "sve" $00.

TCON

The 'Content type', which ID3v1 was stored as a one byte numeric value only, is now a string. You may use one or several of the ID3v1 types as numerical strings, or, since the category list would be impossible to maintain with accurate and up to date categories, define your own. Example: "21" $00 "Eurodisco" $00

You may also use any of the following keywords:

```
RX  Remix
CR  Cover
```

TFLT

The 'File type' frame indicates which type of audio this tag defines.

The following types and refinements are defined:

```
MIME   MIME type follows
MPG    MPEG Audio
  /1     MPEG 1/2 layer I
  /2     MPEG 1/2 layer II
  /3     MPEG 1/2 layer III
  /2.5   MPEG 2.5
  /AAC   Advanced audio compression
VQF    Transform-domain Weighted Interleave Vector
       Quantisation
PCM    Pulse Code Modulated audio
```

but other types may be used, but not for these types though. This is used in a similar way to the predefined types in the "TMED" frame, but without parentheses. If this frame is not present audio type is assumed to be "MPG".

TMED

The 'Media type' frame describes from which media the sound originated. This may be a text string or a reference to the predefined media types found in the list below. Example: "VID/PAL/VHS" $00.

```
DIG    Other digital media
  /A    Analogue transfer from media

ANA    Other analogue media
  /WAC  Wax cylinder
  /8CA  8-track tape cassette

CD     CD
  /A    Analogue transfer from media
  /DD   DDD
  /AD   ADD
  /AA   AAD

LD     Laserdisc

TT     Turntable records
  /33    33.33 rpm
  /45    45 rpm
  /71    71.29 rpm
  /76    76.59 rpm
  /78    78.26 rpm
  /80    80 rpm

MD     MiniDisc
  /A    Analogue transfer from media

DAT    DAT
  /A    Analogue transfer from media
  /1    standard, 48 kHz/16 bits, linear
  /2    mode 2, 32 kHz/16 bits, linear
  /3    mode 3, 32 kHz/12 bits, non-linear, low speed
  /4    mode 4, 32 kHz/12 bits, 4 channels
  /5    mode 5, 44.1 kHz/16 bits, linear
  /6    mode 6, 44.1 kHz/16 bits, 'wide track' play

DCC    DCC
  /A    Analogue transfer from media

DVD    DVD
  /A    Analogue transfer from media

TV     Television
  /PAL    PAL
  /NTSC   NTSC
  /SECAM  SECAM

VID    Video
  /PAL    PAL
  /NTSC   NTSC
  /SECAM  SECAM
  /VHS    VHS
  /SVHS   S-VHS
  /BETA   BETAMAX
RAD    Radio
/FM   FM
  /AM   AM
  /LW   LW
  /MW   MW

TEL    Telephone
/I    ISDN

MC     MC (normal cassette)
  /4    4.75 cm/s (normal speed for a two sided
        cassette)
  /9    9.5 cm/s
  /I    Type I cassette (ferric/normal)
  /II   Type II cassette (chrome)
  /III  Type III cassette (ferric chrome)
  /IV   Type IV cassette (metal)

  REE    Reel
  /9    9.5 cm/s
/19   19 cm/s
  /38   38 cm/s
  /76   76 cm/s
  /I    Type I cassette (ferric/normal)
  /II   Type II cassette (chrome)
  /III  Type III cassette (ferric chrome)
  /IV   Type IV cassette (metal)
```

TMOO

The 'Mood' frame is intended to reflect the mood of the audio with a few keywords, e.g. "Romantic" or "Sad".

## 4.2.4. Rights and license frames

TCOP

The 'Copyright message' frame, in which the string must begin with a year and a space character (making five characters), is intended for the copyright holder of the original sound, not the audio file itself. The absence of this frame means only that the copyright information is unavailable or has been removed, and must not be interpreted to mean that the audio is public domain. Every time this field is displayed the field must be preceded with "Copyright " (C) " ", where (C) is one character showing a C in a circle.

TPRO

The 'Produced notice' frame, in which the string must begin with a year and a space character (making five characters), is intended for the production copyright holder of the original sound, not the audio file itself. The absence of this frame means only that the production copyright information is unavailable or has been removed, and must not be interpreted to mean that the audio is public domain. Every time this field is displayed the field must be preceded with "Produced " (P) " ", where (P) is one character showing a P in a circle.

TPUB

The 'Publisher' frame simply contains the name of the label or publisher.

TOWN

The 'File owner/licensee' frame contains the name of the owner or licensee of the file and it's contents.

TRSN

The 'Internet radio station name' frame contains the name of the internet radio station from which the audio is streamed.

TRSO

The 'Internet radio station owner' frame contains the name of the owner of the internet radio station from which the audio is streamed.

## 4.2.5. Other text frames

TOFN

The 'Original filename' frame contains the preferred filename for the file, since some media doesn't allow the desired length of the filename. The filename is case sensitive and includes its suffix.

TDLY

The 'Playlist delay' defines the numbers of milliseconds of silence that should be inserted before this audio. The value zero indicates that this is a part of a multifile audio track that should be played continuously.

TDEN

The 'Encoding time' frame contains a timestamp describing when the audio was encoded. Timestamp format is described in the ID3v2 structure document [ID3v2-strct].

TDOR
The 'Original release time' frame contains a timestamp describing when the original recording of the audio was released. Timestamp format is described in the ID3v2 structure document [ID3v2-strct].

TDRC
The 'Recording time' frame contains a timestamp describing when the audio was recorded. Timestamp format is described in the ID3v2 structure document [ID3v2-strct].

TDRL
The 'Release time' frame contains a timestamp describing when the audio was first released. Timestamp format is described in the ID3v2 structure document [ID3v2-strct].

TDTG
The 'Tagging time' frame contains a timestamp describing then the audio was tagged. Timestamp format is described in the ID3v2 structure document [ID3v2-strct].

TSSE
The 'Software/Hardware and settings used for encoding' frame includes the used audio encoder and its settings when the file was encoded. Hardware refers to hardware encoders, not the computer on which a program was run.

TSOA
The 'Album sort order' frame defines a string which should be used instead of the album name (TALB) for sorting purposes. E.g. an album named "A Soundtrack" might preferably be sorted as "Soundtrack".

TSOP
The 'Performer sort order' frame defines a string which should be used instead of the performer (TPE2) for sorting purposes.

TSOT
The 'Title sort order' frame defines a string which should be used instead of the title (TIT2) for sorting purposes.

## 4.2.6. User defined text information frame

This frame is intended for one-string text information concerning the audio file in a similar way to the other "T"-frames. The frame body consists of a description of the string, represented as a terminated string, followed by the actual string. There may be more than one "TXXX" frame in each tag, but only one with the same description.

```
<Header for 'User defined text information frame',
 ID: "TXXX">
Text encoding  $xx
Description    <text string according to
                 encoding> $00 (00)
Value          <text string according to encoding>
```

## 4.3. URL link frames

With these frames dynamic data such as webpages with touring information, price information or plain ordinary news can be added to the tag. There may only be one URL [URL] link frame of its kind in an tag, except when stated otherwise in the frame description. If the text string is followed by a string termination, all the following information should be ignored and not be displayed. All URL link frame identifiers begins with "W". Only URL link frame identifiers begins with "W", except for "WXXX". All URL link frames have the  following format:

```
<Header for 'URL link frame', ID: "W000" - "WZZZ",
  excluding "WXXX" described in 4.3.2.>
URL          <text string>
```

### 4.3.1. URL link frames - details

WCOM
The 'Commercial information' frame is a URL pointing at a webpage with information such as where the album can be bought. There may be more than one "WCOM" frame in a tag, but not with the same content.

WCOP
The 'Copyright/Legal information' frame is a URL pointing at a webpage where the terms of use and ownership of the file is
described.

WOAF
The 'Official audio file webpage' frame is a URL pointing at a file specific webpage.

WOAR
The 'Official artist/performer webpage' frame is a URL pointing at the artists official webpage. There may be more than one "WOAR" frame in a tag if the audio contains more than one performer, but not with the same content.

WOAS
The 'Official audio source webpage' frame is a URL pointing at the official webpage for the source of the audio file, e.g. a movie.

WORS
The 'Official Internet radio station homepage' contains a URL pointing at the homepage of the internet radio station.

WPAY
The 'Payment' frame is a URL pointing at a webpage that will handle the process of paying for this file.

WPUB
The 'Publishers official webpage' frame is a URL pointing at the official webpage for the publisher.

### 4.3.2. User defined URL link frame

This frame is intended for URL [URL] links concerning the audio file in a similar way to the other "W"-frames. The frame body consists of a description of the string, represented as a terminated string, followed by the actual URL. The URL is always encoded with ISO-8859-1 [ISO-8859-1]. There may be more than one "WXXX" frame in each tag, but only one with the same description.

```
<Header for 'User defined URL link frame', ID: "WXXX">
Text encoding     $xx
Description       <text string according to
                   encoding> $00 (00)
URL               <text string>
```

### 4.4. Music CD identifier

This frame is intended for music that comes from a CD, so that the CD can be identified in databases such as the CDDB [CDDB]. The frame consists of a binary dump of the Table Of Contents, TOC, from the CD, which is a header of 4 bytes and then 8 bytes/track on the CD plus 8 bytes for the 'lead out', making a maximum of 804 bytes. The offset to the beginning of every track on the CD should be described with a four bytes absolute CD-frame address per track, and not with absolute time. When this frame is used the presence of a valid "TRCK" frame is REQUIRED, even if the CD's only got one track. It is recommended that this frame is always added to tags originating from CDs. There may only be one "MCDI" frame in each tag.

```
<Header for 'Music CD identifier', ID: "MCDI">
CD TOC         <binary data>
```

### 4.5. Event timing codes

This frame allows synchronisation with key events in the audio. The header is:

```
<Header for 'Event timing codes', ID: "ETCO">
Time stamp format    $xx
```

Where time stamp format is:

```
$01  Absolute time, 32 bit sized,
     using MPEG [MPEG] frames as unit
$02  Absolute time, 32 bit sized,
     using milliseconds as unit
```

Absolute time means that every stamp contains the time from the beginning of the file.

Followed by a list of key events in the following format:

```
Type of event   $xx
Time stamp      $xx (xx ...)
```

The 'Time stamp' is set to zero if directly at the beginning of the sound or after the previous event. All events MUST be sorted in chronological order. The type of event is as follows:

```
$00  padding (has no meaning)
$01  end of initial silence
$02  intro start
$03  main part start
$04  outro start
```

```
$05  outro end
$06  verse start
$07  refrain start
$08  interlude start
$09  theme start
$0A  variation start
$0B  key change
$0C  time change
$0D  momentary unwanted noise (Snap, Crackle & Pop)
$0E  sustained noise
$0F  sustained noise end
$10  intro end
$11  main part end
$12  verse end
$13  refrain end
$14  theme end
$15  profanity
$16  profanity end

$17-$DF  reserved for future use

$E0-$EF  not predefined synch 0-F

$F0-$FC  reserved for future use

$FD  audio end (start of silence)
$FE  audio file ends
$FF  one more byte of events follows (all the
     following bytes with the value $FF have the
     same function)
```

Terminating the start events such as "intro start" is OPTIONAL. The 'Not predefined synch's ($E0-EF) are for user events. You might want to synchronise your music to something, like setting off an explosion on-stage, activating a screensaver etc.

There may only be one "ETCO" frame in each tag.

## 4.6. MPEG location lookup table

To increase performance and accuracy of jumps within a MPEG [MPEG] audio file, frames with time codes in different locations in the file might be useful. This ID3v2 frame includes references that the software can use to calculate positions in the file. After the frame header follows a descriptor of how much the 'frame counter' should be increased for every reference. If this value is two then the first reference points out the second frame, the 2nd reference the 4th frame, the 3rd reference the 6th frame etc. In a similar way the 'bytes between reference' and 'milliseconds between reference' points out bytes and milliseconds respectively.

Each reference consists of two parts; a certain number of bits, as defined in 'bits for bytes deviation', that describes the difference between what is said in 'bytes between reference' and the reality and a certain number of bits, as defined in 'bits for milliseconds deviation', that describes the difference between what is said in 'milliseconds between reference' and the reality. The number of bits in every reference, i.e. 'bits for bytes deviation'+'bits for milliseconds deviation', must be a multiple of four. There may only be one "MLLT" frame in each tag.

```
<Header for 'Location lookup table', ID: "MLLT">
MPEG frames between reference  $xx xx
Bytes between reference        $xx xx xx
Milliseconds between reference $xx xx xx
Bits for bytes deviation       $xx
Bits for milliseconds dev.     $xx
```

Then for every reference the following data is included;

```
Deviation in bytes         %xxx....
Deviation in milliseconds  %xxx....
```

## 4.7. Synchronised tempo codes

For a more accurate description of the tempo of a musical piece, this frame might be used. After the header follows one byte describing which time stamp format should be used. Then follows one or more tempo codes. Each tempo code consists of one tempo part and one time part. The tempo is in BPM described with one or two bytes. If the first byte has the value $FF, one more byte follows, which is added to the first giving a range from 2 - 510 BPM, since $00 and $01 is reserved. $00 is used to describe a beat-free time period, which is not the same as a music-free time period. $01 is used to indicate one single beat-stroke followed by a beat-free period.

The tempo descriptor is followed by a time stamp. Every time the tempo in the music changes, a tempo descriptor may indicate this for the player. All tempo descriptors MUST be sorted in chronological order. The first beat-stroke in a time-period is at the same time as the beat description occurs. There may only be one "SYTC" frame in each tag.

```
   <Header for 'Synchronised tempo codes', ID: "SYTC">
   Time stamp format   $xx
   Tempo data          <binary data>
```

Where time stamp format is:

```
   $01  Absolute time, 32 bit sized, using MPEG [MPEG]
        frames as unit
   $02  Absolute time, 32 bit sized, using milliseconds
        as unit
```

Absolute time means that every stamp contains the time from the beginning of the file.

## 4.8. Unsynchronised lyrics/text transcription

This frame contains the lyrics of the song or a text transcription of other vocal activities. The head includes an encoding descriptor and a content descriptor. The body consists of the actual text. The 'Content descriptor' is a terminated string. If no descriptor is entered, 'Content descriptor' is $00 (00) only. Newline characters are allowed in the text. There may be more than one synchronized lyrics/text transcription' frame in each tag, but only one with the same language and content descriptor.

```
  <Header for 'Unsynchronised lyrics/text transcription',
    ID: "USLT">
  Text encoding        $xx
  Language             $xx xx xx
  Content descriptor   <text string according to
                       encoding> $00 (00)
  Lyrics/text          <full text string according to
                       encoding>
```

## 4.9. Synchronised lyrics/text

This is another way of incorporating the words, said or sung lyrics, in the audio file as text, this time, however, in sync with the audio. It might also be used to describing events e.g. occurring on a stage or on the screen in sync with the audio. The header includes a content descriptor, represented with as terminated text string. If no descriptor is entered, 'Content descriptor' is $00 (00) only.

```
   <Header for 'Synchronised lyrics/text', ID: "SYLT">
   Text encoding        $xx
   Language             $xx xx xx
   Time stamp format    $xx
   Content type         $xx
   Content descriptor   <text string according
                         to encoding> $00 (00)

 Content type:   $00 is other
                 $01 is lyrics
                 $02 is text transcription
                 $03 is movement/part name
                   (e.g. "Adagio")
                 $04 is events
                  (e.g. "Don Quijote enters the stage")
                 $05 is chord (e.g. "Bb F Fsus")
                 $06 is trivia/'pop up' information
                 $07 is URLs to webpages
                 $08 is URLs to images
```

Time stamp format:

```
   $01  Absolute time, 32 bit sized,
         using MPEG [MPEG] frames as unit
   $02  Absolute time, 32 bit sized,
         using milliseconds as unit
```

Absolute time means that every stamp contains the time from the beginning of the file.

The text that follows the frame header differs from that of the unsynchronised lyrics/text transcription in one major way. Each syllable (or whatever size of text is considered to be convenient by the encoder) is a null terminated string followed by a time stamp denoting where in the sound file it belongs. Each sync thus has the following structure:

```
   Terminated text to be synced (typically a syllable)
   Sync identifier (terminator to above string)  $00 (00)
   Time stamp                                    $xx (xx ...)
```

The 'time stamp' is set to zero or the whole sync is omitted if located directly at the beginning of the sound. All time stamps should be sorted in chronological order. The sync can be considered as a validator of the subsequent string.

Newline characters are allowed in all "SYLT" frames and MUST be used after every entry (name, event etc.) in a frame with the content type $03 - $04.

A few considerations regarding whitespace characters: Whitespace separating words should mark the beginning of a new word, thus occurring in front of the first syllable of a new word. This is also valid for new line characters. A syllable followed by a comma should not be broken apart with a sync (both the syllable and the comma should be before the sync).

An example: The "USLT" passage

```
  "Strangers in the night" $0A "Exchanging glances"
```

would be "SYLT" encoded as:

```
  "Strang" $00 xx xx
  "ers" $00 xx xx
  " in" $00 xx xx
  " the" $00 xx xx
  " night" $00 xx xx
  0A
  "Ex" $00 xx xx
  "chang" $00 xx xx
  "ing" $00 xx xx
  "glan" $00 xx xx
  "ces" $00 xx xx
```

There may be more than one "SYLT" frame in each tag, but only one with the same language and content descriptor.

## 4.10. Comments

This frame is intended for any kind of full text information that does not fit in any other frame. It consists of a frame header followed by encoding, language and content descriptors and is ended with the actual comment as a text string. Newline characters are allowed in the comment text string. There may be more than one comment frame in each tag, but only one with the same language and content descriptor.

```
  <Header for 'Comment', ID: "COMM">
  Text encoding          $xx
  Language               $xx xx xx
  Short content descrip. <text string according
                           to encoding> $00 (00)
  The actual text        <full text string
                           according to encoding>
```

## 4.11. Relative volume adjustment (2)

This is a more subjective frame than the previous ones. It allows the user to say how much he wants to increase/decrease the volume on each channel when the file is played. The purpose is to be able to align all files to a reference volume, so that you don't have to change the volume constantly. This frame may also be used to balance adjust the audio. The volume adjustment is encoded as a fixed point decibel value, 16 bit signed integer representing (adjustment*512), giving +/- 64 dB with a precision of 0.001953125 dB. E.g. +2 dB is stored as $04 00 and -2 dB is $FC 00. There may be more than one "RVA2" frame in each tag, but only one with the same identification string.

```
  <Header for 'Relative volume adjustment (2)', ID:
"RVA2">
  Identification          <text string> $00
```

The 'identification' string is used to identify the situation and/or device where this adjustment should apply. The following is then repeated for every channel

```
  Type of channel         $xx
  Volume adjustment       $xx xx
  Bits representing peak  $xx
  Peak volume             $xx (xx ...)

Type of channel:  $00  Other
                  $01  Master volume
                  $02  Front right
                  $03  Front left
                  $04  Back right
                  $05  Back left
                  $06  Front centre
                  $07  Back centre
                  $08  Subwoofer
```

Bits representing peak can be any number between 0 and 255. 0 means that there is no peak volume field. The peak volume field is always padded to whole bytes, setting the most significant bits to zero.

## 4.12. Equalisation (2)

This is another subjective, alignment frame. It allows the user to predefine an equalisation curve within the audio file. There may be more than one "EQU2" frame in each tag, but only one with the same identification string.

```
<Header of 'Equalisation (2)', ID: "EQU2">
Interpolation method  $xx
Identification        <text string> $00
```

The 'interpolation method' describes which method is preferred when an interpolation between the adjustment point that follows. The following methods are currently defined:

```
$00  Band
     No interpolation is made. A jump from one
     adjustment level to another occurs in the middle
     between two adjustment points.
$01  Linear
     Interpolation between adjustment points is
     linear.
```

The 'identification' string is used to identify the situation and/or device where this adjustment should apply. The following is then repeated for every adjustment point

```
Frequency          $xx xx
Volume adjustment  $xx xx
```

The frequency is stored in units of 1/2 Hz, giving it a range from 0 to 32767 Hz.

The volume adjustment is encoded as a fixed point decibel value, 16 bit signed integer representing (adjustment*512), giving +/- 64 dB with a precision of 0.001953125 dB. E.g. +2 dB is stored as $04 00 and -2 dB is $FC 00.

Adjustment points should be ordered by frequency and one frequency should only be described once in the frame.

## 4.13. Reverb

Yet another subjective frame, with which you can adjust echoes of different kinds. Reverb left/right is the delay between every bounce in ms. Reverb bounces left/right is the number of bounces that should be made. $FF equals an infinite number of bounces. Feedback is the amount of volume that should be returned to the next echo bounce. $00 is 0%, $FF is 100%. If this value were $7F, there would be 50% volume reduction on the first bounce, 50% of that on the second and so on.
Left to left means the sound from the left bounce to be played in the left speaker, while left to right means sound from the left bounce to be played in the right speaker.

'Premix left to right' is the amount of left sound to be mixed in the right before any reverb is applied, where $00 id 0% and $FF is 100%.
'Premix right to left' does the same thing, but right to left.
Setting both premix to $FF would result in a mono output (if the reverb is applied symmetric). There may only be one "RVRB" frame in each tag.

```
<Header for 'Reverb', ID: "RVRB">
Reverb left (ms)                 $xx xx
Reverb right (ms)                $xx xx
Reverb bounces, left             $xx
Reverb bounces, right            $xx
Reverb feedback, left to left    $xx
Reverb feedback, left to right   $xx
Reverb feedback, right to right  $xx
Reverb feedback, right to left   $xx
Premix left to right             $xx
Premix right to left             $xx
```

## 4.14. Attached picture

This frame contains a picture directly related to the audio file.

Image format is the MIME type and subtype [MIME] for the image. In the event that the MIME media type name is omitted, "image/" will be implied. The "image/png" [PNG] or "image/jpeg" [JFIF] picture format should be used when interoperability is wanted. Description is a short description of the picture, represented as a terminated text string. There may be several pictures attached to one

file, each in their individual "APIC" frame, but only one with the same content descriptor. There may only be one picture with the picture type declared as picture type $01 and $02 respectively. There is the possibility to put only a link to the image file by using the 'MIME type' "-->" and having a complete URL [URL] instead of picture data.
   The use of linked files should however be used sparingly since there is the risk of separation of files.

```
     <Header for 'Attached picture', ID: "APIC">
     Text encoding      $xx
     MIME type          <text string> $00
     Picture type       $xx
     Description        <text string according to
                          encoding> $00 (00)
     Picture data       <binary data>
```

| Picture type: | | |
|---|---|---|
| | $00 | Other |
| | $01 | 32x32 pixels 'file icon' (PNG only) |
| | $02 | Other file icon |
| | $03 | Cover (front) |
| | $04 | Cover (back) |
| | $05 | Leaflet page |
| | $06 | Media (e.g. label side of CD) |
| | $07 | Lead artist/lead performer/soloist |
| | $08 | Artist/performer |
| | $09 | Conductor |
| | $0A | Band/Orchestra |
| | $0B | Composer |
| | $0C | Lyricist/text writer |
| | $0D | Recording Location |
| | $0E | During recording |
| | $0F | During performance |
| | $10 | Movie/video screen capture |
| | $11 | A bright coloured fish |
| | $12 | Illustration |
| | $13 | Band/artist logotype |
| | $14 | Publisher/Studio logotype |

## 4.15. General encapsulated object

In this frame any type of file can be encapsulated. After the header, 'Frame size' and 'Encoding' follows 'MIME type' [MIME] represented as as a terminated string encoded with ISO 8859-1 [ISO-8859-1]. The filename is case sensitive and is encoded as 'Encoding'. Then follows a content description as terminated string, encoded as 'Encoding'.

The last thing in the frame is the actual object. The first two strings may be omitted, leaving only their terminations. MIME type is always an ISO-8859-1 text string. There may be more than one "GEOB" frame in each tag, but only one with the same content descriptor.

```
     <Header for 'General encapsulated object', ID: "GEOB">
     Text encoding          $xx
     MIME type              <text string> $00
     Filename               <text string according to
                              encoding> $00 (00)
     Content description    <text string according to
                              encoding> $00 (00)
     Encapsulated object    <binary data>
```

## 4.16. Play counter

This is simply a counter of the number of times a file has been played. The value is increased by one every time the file begins to play. There may only be one "PCNT" frame in each tag. When the counter reaches all one's, one byte is inserted in front of the counter thus making the counter eight bits bigger.  The counter must be at least 32-bits long to begin with.

```
     <Header for 'Play counter', ID: "PCNT">
     Counter        $xx xx xx xx (xx ...)
```

## 4.17. Popularimeter

The purpose of this frame is to specify how good an audio file is.
   Many interesting applications could be found to this frame such as a playlist that features better audio files more often than others or it could be used to profile a person's taste and find other 'good' files by comparing people's profiles. The frame contains the email address to the user, one rating byte and a four byte play counter, intended to be increased with one for every time the file is played.
   The email is a terminated string. The rating is 1-255 where 1 is worst and 255 is best. 0 is unknown. If no personal counter is wanted it may be omitted. When the counter reaches all one's, one byte is inserted in front of the counter thus making the counter eight bits bigger in the same away as the play counter ("PCNT"). There may be

more than one "POPM" frame in each tag, but only one with the same email address.

```
     <Header for 'Popularimeter', ID: "POPM">
     Email to user   <text string> $00
     Rating          $xx
     Counter         $xx xx xx xx (xx ...)
```

## 4.18. Recommended buffer size

Sometimes the server from which an audio file is streamed is aware of transmission or coding problems resulting in interruptions in the audio stream. In these cases, the size of the buffer can be recommended by the server using this frame. If the 'embedded info flag' is true (1) then this indicates that an ID3 tag with the
maximum size described in 'Buffer size' may occur in the audio stream. In such case the tag should reside between two MPEG [MPEG] frames, if the audio is MPEG encoded. If the position of the next tag is known, 'offset to next tag' may be used. The offset is calculated from the end of tag in which this frame resides to the first byte of the header in the next. This field may be omitted. Embedded tags are
generally not recommended since this could render unpredictable behaviour from present software/hardware.

For applications like streaming audio it might be an idea to embed tags into the audio stream though. If the clients connects to individual connections like HTTP and there is a possibility to begin every transmission with a tag, then this tag should include a 'recommended buffer size' frame. If the client is connected to a arbitrary point in the stream, such as radio or multicast, then the
'recommended buffer size' frame SHOULD be included in every tag.

The 'Buffer size' should be kept to a minimum. There may only be one "RBUF" frame in each tag.

```
     <Header for 'Recommended buffer size', ID: "RBUF">
     Buffer size               $xx xx xx
     Embedded info flag        %0000000x
     Offset to next tag        $xx xx xx xx
```

## 4.19. Audio encryption

This frame indicates if the actual audio stream is encrypted, and by whom. Since standardisation of such encryption scheme is beyond this document, all "AENC" frames begin with a terminated string with a URL containing an email address, or a link to a location where an email address can be found, that belongs to the organization responsible for this specific encrypted audio file. Questions regarding the encrypted audio should be sent to the email address specified. If a $00 is found directly after the 'Frame size' and the
audio file indeed is encrypted, the whole file may be considered useless.

After the 'Owner identifier', a pointer to an unencrypted part of the audio can be specified. The 'Preview start' and 'Preview length' is described in frames. If no part is unencrypted, these fields should be left zeroed. After the 'preview length' field follows optionally a data block required for decryption of the audio. There may be more than one "AENC" frames in a tag, but only one with the same 'Owner identifier'.

```
     <Header for 'Audio encryption', ID: "AENC">
     Owner identifier   <text string> $00
     Preview start      $xx xx
     Preview length     $xx xx
     Encryption info    <binary data>
```

## 4.20. Linked information

To keep information duplication as low as possible this frame may be used to link information from another ID3v2 tag that might reside in another audio file or alone in a binary file. It is RECOMMENDED that this method is only used when the files are stored on a CD-ROM or other circumstances when the risk of file separation is low. The frame contains a frame identifier, which is the frame that should be linked into this tag, a URL [URL] field, where a reference to the file where the frame is given, and additional ID data, if needed.

Data should be retrieved from the first tag found in the file to which this link points. There may be more than one "LINK" frame in a tag, but only one with the same contents. A linked frame is to be considered as part of the tag and has the same restrictions as if it was a physical part of the tag (i.e. only one "RVRB" frame allowed, whether it's linked or not).

```
     <Header for 'Linked information', ID: "LINK">
     Frame identifier        $xx xx xx xx
     URL                     <text string> $00
     ID and additional data  <text string(s)>
```

Frames that may be linked and need no additional data are "ASPI", "ETCO", "EQU2", "MCID", "MLLT", "OWNE", "RVA2", "RVRB", "SYTC", the text information frames and the URL link frames.

The "AENC", "APIC", "GEOB" and "TXXX" frames may be linked with the content descriptor as additional ID data.

The "USER" frame may be linked with the language field as additional ID data.

The "PRIV" frame may be linked with the owner identifier as additional ID data.

The "COMM", "SYLT" and "USLT" frames may be linked with three bytes of language descriptor directly followed by a content descriptor as additional ID data.

## 4.21. Position synchronisation frame

This frame delivers information to the listener of how far into the audio stream he picked up; in effect, it states the time offset from the first frame in the stream. The frame layout is:

```
     <Head for 'Position synchronisation', ID: "POSS">
     Time stamp format         $xx
     Position                  $xx (xx ...)
```

Where time stamp format is:

```
     $01  Absolute time, 32 bit sized,
          using MPEG frames as unit
     $02  Absolute time, 32 bit sized,
          using milliseconds as unit
```

and position is where in the audio the listener starts to receive, i.e. the beginning of the next frame. If this frame is used in the beginning of a file the value is always 0. There may only be one "POSS" frame in each tag.

## 4.22. Terms of use frame

This frame contains a brief description of the terms of use and ownership of the file. More detailed information concerning the legal terms might be available through the "WCOP" frame. Newlines are allowed in the text. There may be more than one 'Terms of use' frame in a tag, but only one with the same 'Language'.

```
     <Header for 'Terms of use frame', ID: "USER">
     Text encoding        $xx
     Language             $xx xx xx
     The actual text      <text string according to
encoding>
```

## 4.23. Ownership frame

The ownership frame might be used as a reminder of a made transaction or, if signed, as proof. Note that the "USER" and "TOWN" frames are good to use in conjunction with this one. The frame begins, after the frame ID, size and encoding fields, with a 'price paid' field. The first three characters of this field contains the currency used for the transaction, encoded according to ISO 4217 [ISO-4217] alphabetic currency code. Concatenated to this is the actual price paid, as a numerical string using "." as the decimal separator. Next is an 8  character date string (YYYYMMDD) followed by a string with the name of the seller as the last field in the frame. There may only be one "OWNE" frame in a tag.

```
     <Header for 'Ownership frame', ID: "OWNE">
     Text encoding     $xx
     Price paid        <text string> $00
     Date of purch.    <text string>
     Seller            <text string according to encoding>
```

## 4.24. Commercial frame

This frame enables several competing offers in the same tag by bundling all needed information. That makes this frame rather complex but it's an easier solution than if one tries to achieve the same result with several frames. The frame begins, after the frame ID, size and encoding fields, with a price string field. A price is constructed by one three character currency code, encoded according to

ISO 4217 [ISO-4217] alphabetic currency code, followed by a numerical value where "." is used as decimal separator. In the price string several prices may be concatenated, separated by a "/"
character, but there may only be one currency of each type.

The price string is followed by an 8 character date string in the format YYYYMMDD, describing for how long the price is valid. After that is a contact URL, with which the user can contact the seller, followed by a one byte 'received as' field. It describes how the audio is delivered when bought according to the following list:

```
        $00  Other
        $01  Standard CD album with other songs
        $02  Compressed audio on CD
        $03  File over the Internet
        $04  Stream over the Internet
        $05  As note sheets
        $06  As note sheets in a book with other sheets
        $07  Music on other media
        $08  Non-musical merchandise
```

Next follows a terminated string with the name of the seller followed by a terminated string with a short description of the product. The last thing is the ability to include a company logotype. The first of them is the 'Picture MIME type' field containing information about
which picture format is used. In the event that the MIME media type name is omitted, "image/" will be implied. Currently only "image/png" and "image/jpeg" are allowed. This format string is followed by the binary picture data. This two last fields may be omitted if no picture is attached. There may be more than one 'commercial frame' in
a tag, but no two may be identical.

```
     <Header for 'Commercial frame', ID: "COMR">
     Text encoding      $xx
     Price string       <text string> $00
     Valid until        <text string>
     Contact URL        <text string> $00
     Received as        $xx
     Name of seller     <text string according
                          to encoding> $00 (00)
     Description        <text string according
                          to encoding> $00 (00)
     Picture MIME type  <string> $00
     Seller logo        <binary data>
```

## 4.25. Encryption method registration

To identify with which method a frame has been encrypted the
encryption method must be registered in the tag with this frame. The 'Owner identifier' is a null-terminated string with a URL [URL] containing an email address, or a link to a location where an email address can be found, that belongs to the organisation responsible for this specific encryption method. Questions regarding the encryption method should be sent to the indicated email address. The 'Method symbol' contains a value that is associated with this method
throughout the whole tag, in the range $80-F0. All other values are reserved. The 'Method symbol' may optionally be followed by
encryption specific data. There may be several "ENCR" frames in a tag but only one containing the same symbol and only one containing the same owner identifier. The method must be used somewhere in the tag.

See the description of the frame encryption flag in the ID3v2 structure document [ID3v2-strct] for more information.

```
     <Header for 'Encryption method registration', ID:
"ENCR">
     Owner identifier    <text string> $00
     Method symbol       $xx
     Encryption data     <binary data>
```

## 4.26. Group identification registration

This frame enables grouping of otherwise unrelated frames. This can be used when some frames are to be signed. To identify which frames belongs to a set of frames a group identifier must be registered in the tag with this frame. The 'Owner identifier' is a null-terminated string with a URL [URL] containing an email address, or a link to a location where an email address can be found, that belongs to the organisation responsible for this grouping. Questions regarding the grouping should be sent to the indicated email address. The 'Group symbol' contains a value that associates the frame with this group throughout the whole tag, in the range $80-F0. All other values are reserved. The 'Group symbol' may optionally be followed by some group specific data, e.g. a digital signature. There may be several "GRID" frames in a tag but only one containing the same symbol and only one containing the same owner identifier. The group symbol must be used somewhere in the tag. See the description of the frame grouping flag in the ID3v2 structure document [ID3v2-strct] for more information.

```
     <Header for 'Group ID registration', ID: "GRID">
     Owner identifier      <text string> $00
     Group symbol          $xx
     Group dependent data  <binary data>
```

## 4.27. Private frame

This frame is used to contain information from a software producer that its program uses and does not fit into the other frames. The frame consists of an 'Owner identifier' string and the binary data.
The 'Owner identifier' is a null-terminated string with a URL [URL] containing an email address, or a link to a location where an email address can be found, that belongs to the organisation responsible for the frame. Questions regarding the frame should be sent to the indicated email address. The tag may contain more than one "PRIV" frame but only with different contents.

```
     <Header for 'Private frame', ID: "PRIV">
     Owner identifier      <text string> $00
     The private data      <binary data>
```

## 4.28. Signature frame

This frame enables a group of frames, grouped with the 'Group identification registration', to be signed. Although signatures can reside inside the registration frame, it might be desired to store the signature elsewhere, e.g. in watermarks. There may be more than one 'signature frame' in a tag, but no two may be identical.

```
     <Header for 'Signature frame', ID: "SIGN">
     Group symbol      $xx
     Signature         <binary data>
```

## 4.29. Seek frame

This frame indicates where other tags in a file/stream can be found.
The 'minimum offset to next tag' is calculated from the end of this tag to the beginning of the next. There may only be one 'seek frame' in a tag.

```
   <Header for 'Seek frame', ID: "SEEK">
   Minimum offset to next tag       $xx xx xx xx
```

## 4.30. Audio seek point index

Audio files with variable bit rates are intrinsically difficult to deal with in the case of seeking within the file. The ASPI frame makes seeking easier by providing a list a seek points within the audio file. The seek points are a fractional offset within the audio data, providing a starting point from which to find an appropriate point to start decoding. The presence of an ASPI frame requires the existence of a TLEN frame, indicating the duration of the file in milliseconds. There may only be one 'audio seek point index' frame in a tag.

```
     <Header for 'Seek Point Index', ID: "ASPI">
     Indexed data start (S)         $xx xx xx xx
     Indexed data length (L)        $xx xx xx xx
     Number of index points (N)     $xx xx
     Bits per index point (b)       $xx
```

Then for every index point the following data is included;

```
     Fraction at index (Fi)          $xx (xx)
```

'Indexed data start' is a byte offset from the beginning of the file.
'Indexed data length' is the byte length of the audio data being indexed. 'Number of index points' is the number of in dex points, as the name implies. The recommended number is 100.
'Bits per index point' is 8 or 16, depending on the chosen precision. 8 bits works well for short files (less than 5 minutes of audio), while 16 bits is advantageous for long files. 'Fraction at index' is the numerator of the fraction representing a relative position in the data. The denominator is 2 to the power of b.

Here are the algorithms to be used in the calculation. The known data must be the offset of the start of the indexed data (S), the offset of the end of the indexed data (E), the number of index points (N), the offset at index i (Oi). We calculate the fraction at index I (Fi).

Oi is the offset of the frame whose start is soonest after the point for which the time offset is (i/N * duration).

The frame data should be calculated as follows:

```
  Fi = Oi/L * 2^b    (rounded down to the nearest
                      integer)
```

Offset calculation should be calculated as follows from data in the frame:

```
  Oi = (Fi/2^b)*L    (rounded up to the nearest integer)
```

## 5. Copyright

## 6. References

[CDDB] Compact Disc Data Base
<url:http://www.cddb.com>

[ID3v2.3.0] Martin Nilsson, "ID3v2 informal standard".
<url:http://www.id3.org/id3v2.3.0.txt>

[ID3v2-strct] Martin Nilsson,
"ID3 tag version 2.4.0 - Main Structure"
<url:http//www.id3.org/id3v2.4.0-structure.txt>

[ISO-639-2] ISO/FDIS 639-2.
Codes for the representation of names of languages, Part 2: Alpha-3 code. Technical committee / subcommittee: TC 37 / SC 2

[ISO-4217] ISO 4217:1995.
Codes for the representation of currencies and funds.
Technical committee / subcommittee: TC 68

[ISO-8859-1] ISO/IEC DIS 8859-1.
8-bit single-byte coded graphic character sets, Part 1: Latin alphabet No. 1. Technical committee / subcommittee: JTC 1 / SC 2

[ISRC] ISO 3901:1986
International Standard Recording Code (ISRC).
Technical committee / subcommittee: TC 46 / SC 9

[JFIF] JPEG File Interchange Format, version 1.02
<url:http://www.w3.org/Graphics/JPEG/jfif.txt>

[KEYWORDS] S. Bradner, 'Key words for use in RFCs to Indicate Requirement Levels', RFC 2119, March 1997.
<url:ftp://ftp.isi.edu/in-notes/rfc2119.txt>

[MIME] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
<url:ftp://ftp.isi.edu/in-notes/rfc2045.txt>

[MPEG] ISO/IEC 11172-3:1993.
Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 3: Audio.
Technical committee / subcommittee: JTC 1 / SC 29
and
ISO/IEC 13818-3:1995
Generic coding of moving pictures and associated audio information, Part 3: Audio.
Technical committee / subcommittee: JTC 1 / SC 29
and
ISO/IEC DIS 13818-3
Generic coding of moving pictures and associated audio information, Part 3: Audio (Revision of ISO/IEC 13818-3:1995)

[PNG] Portable Network Graphics, version 1.0
<url:http://www.w3.org/TR/REC-png-multi.html>

[URL] T. Berners-Lee, L. Masinter & M. McCahill, "Uniform Resource Locators (URL).", RFC 1738, December 1994.
<url:ftp://ftp.isi.edu/in-notes/rfc1738.txt>

[ZLIB] P. Deutsch, Aladdin Enterprises & J-L. Gailly, "ZLIB Compressed
Data Format Specification version 3.3", RFC 1950, May 1996.
<url:ftp://ftp.isi.edu/in-notes/rfc1950.txt>

## 7. Appendix

## A. Appendix A - Genre List from ID3v1

The following genres are defined in ID3v1

| | | |
|---|---|---|
| 0.Blues | 28.Vocal | 56.Southern Rock |
| 1.Classic Rock | 29.Jazz+Funk | 57.Comedy |
| 2.Country | 30.Fusion | 58.Cult |
| 3.Dance | 31.Trance | 59.Gangsta |
| 4.Disco | 32.Classical | 60.Top 40 |
| 5.Funk | 33.Instrumental | 61.Christian Rap |
| 6.Grunge | 34.Acid | 62.Pop/Funk |
| 7.Hip-Hop | 35.House | 63.Jungle |
| 8.Jazz | 36.Game | 64.Native American |
| 9.Metal | 37.Sound Clip | 65.Cabaret |
| 10.New Age | 38.Gospel | 66.New Wave |
| 11.Oldies | 39.Noise | 67.Psychadelic |
| 12.Other | 40.AlternRock | 68.Rave |
| 13.Pop | 41.Bass | 69.Showtunes |
| 14.R&B | 42.Soul | 70.Trailer |
| 15.Rap | 43.Punk | 71.Lo-Fi |
| 16.Reggae | 44.Space | 72.Tribal |
| 17.Rock | 45.Meditative | 73.Acid Punk |
| 18.Techno | 46.Instrumental Pop | 74.Acid Jazz |
| 19.Industrial | 47.Instrumental Rock | 75.Polka |
| 20.Alternative | 48.Ethnic | 76.Retro |
| 21.Ska | 49.Gothic | 77.Musical |
| 22.Death Metal | 50.Darkwave | 78.Rock & Roll |
| 23.Pranks | 51.Techno-Industrial | 79.Hard Rock |
| 24.Soundtrack | 52.Electronic | |
| 25.Euro-Techno | 53.Pop-Folk | |
| 26.Ambient | 54.Eurodance | |
| 27.Trip-Hop | 55.Dream | |

## 8. Author's Address

Written by

Martin Nilsson
Rydsvägen 246 C. 30
SE-584 34 Linköping
Sweden
Email: nilsson@id3.org

..end..