Preview: Interactive XMF

A Standardized Interchange File Format for Advanced Interactive Audio Content

Chris Grigg
MIDI Manufacturers Association, Los Angeles, CA, USA
Beatnik Inc., San Mateo, CA, USA
Control-G, Oakland, CA, USA
gdc04@chrisgrigg.org

Attribution Note: Interactive XMF is a continuation of previous work done at Project Bar-B-Q [4, 5, 6], all of which was highly collaborative, involving too many participants to list here, all of them invaluable; kindly consult the references. George A. Sanger, The Fat Man, provided most of Section 4 of this paper.

Abstract

The Interactive XMF (iXMF) working group in the Interactive Audio SIG of the MIDI Manufacturers Association has produced a draft specification for an open public standard file format supporting the interchange of advanced interactive audio soundtracks. It uses a cue-oriented model, is not tied to any particular authoring or playback platform, is programming language-neutral, and intended to be used without license agreements or royalty payments. It is technically extensible in several dimensions. This paper describes the iXMF file format and the model for the underlying soundtrack engine.

Contents

1. Background and Motivation

Perhaps uniquely in the interactive media world, the field of interactive audio (IA) has never experienced the benefits of a public standard file format to represent what they do. This has resulted in a substantial duplication of effort, as each game developer or OS vendor has had to develop their own proprietary tools, which can be seen as a highly inefficient use of the industry resources invested in the IA field. Equally problematic is that as a side-effect of the parallel development, all of these IA systems are mutually incompatible, with little or no interoperability, or import/export capability, due to large or small differences in file formats, underlying models, and content editing methods. Further, since the number of users for each

of these systems is small, the incentive to invest in tool development is small, so tools in general tend to be primitive and difficult to use.

These problems present barriers both to the quality of any given soundtrack, and to professional development for interactive audio artists, since the technical/creative learning curve must be scaled anew every time a new target platform needs to be dealt with, and every time a given audio artist works with a new game developer.

By way of comparison, imagine how the progress rate of the 3D graphics world, including the quality of model design and animation, would have been retarded had public open standards such as OpenGL never materialized.

Ironically, despite the many apparent differences, the underlying fundamental capabilities and operating models of most of these proprietary IA systems tend to be highly similar. In IA circles, there is now a sense that there exists a mature and stable set of required basic features for any advanced IA system.

The key element that has been missing is a non-proprietary standard for IA content. A standard format creates a single market for infrastructure (playback technologies and content creation tools), encouraging greater investment and better quality. A standard format also provides a unified conceptual model for IA practitioners, defragmenting the field in general and encouraging the sharing of information with regard to technique, style, and similar professional development-related communication.

In an attempt to forge this missing link, the Interactive XMF (eXtensible Music Format) working group (iXMF-WG) of the Interactive Audio SIG (IASIG, an activity of the MIDI Manufacturers Association [MMA]) has produced a draft specification for just such a public standard – a file format for the interchange of professional advanced interactive audio soundtracks.

The proposed iXMF format is a binary format not bound to any particular playback platform or programming language, free of any license and royalty encumbrances, and is extensible in several dimensions both for future standardization and for custom purposes. It uses the XMF Meta File Format [1], an extensible container technology previously standardized by the MMA.

An iXMF file contains both playable soundtrack media (audio and MIDI), plus data expressing a set of rules governing exactly how that media will be played in reaction to events. In other words, the dynamic aspect of the soundtrack is data-driven, by contrast to many previous IA systems that require coding in C++ or similar high-level programming languages to achieve real-time soundtrack adaptation. This is an important shift from a product development management perspective, as it decouples the audio creative department from dependencies on the engineering department, which in a typical game development setting tends to be highly overburdened and thus unable to devote sufficient time to fully and successfully cooperate on achieving the audio artist's desired creative effects.

Data structures in the iXMF file give the IA artist extensive facilities for assembling blocks of pre-authored sound media into a continuous soundtrack, including detailed control over the timing and crossfade characteristics of every assembly transition. Through the combination of a simple event mechanism and a simple scripting language, iXMF makes it possible for an IA artist to control, at a fine level of detail, what the soundtrack's response to any given real-time event will be, including altering the order in which the blocks play, controlling the muting of tracks within the blocks, and dynamically controlling any available continuous DSP parameters, from volume to 3D spatial position. These real-time events may be triggered either from the game (or other application) hosting the soundtrack engine, or else from markers placed in the sound media. A callback mechanism allows the soundtrack to signal the host when particular points are reached, or certain conditions occur.

The draft specification provides a detailed model of the underlying playback engine, with full explanation of several extensibility mechanisms allowing for future growth of the format. A set of terminology describing the data and code entities in the model is defined.

2. Architecture

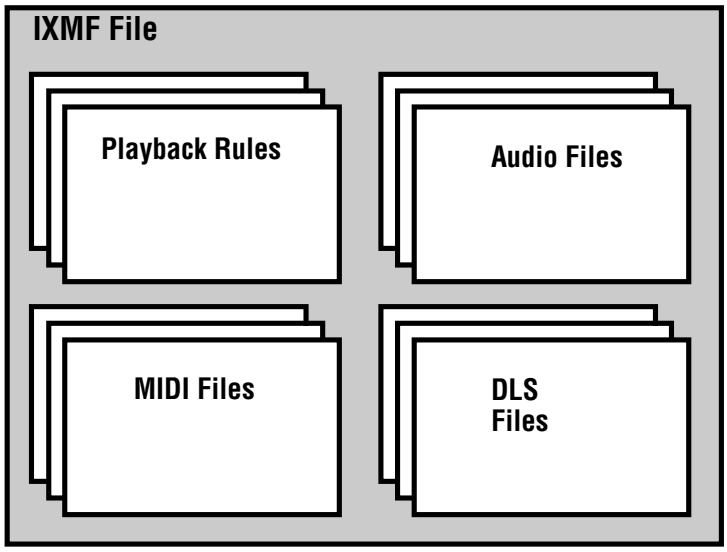
Underlying the design of the iXMF data file, and the model for the software that reads iXMF files, are some general principles gleaned from extensive practical experience developing game soundtracks. This experience dictated the iXMF system's architectural design.

There is much more to a game soundtrack than just the collection of sound files that get played. The thing that creates the desired effect in the mind of the gamer is not just the sounds themselves, however excellent a job the audio artist may have done in creating them, but rather the way the sounds are used in reaction to gameplay events, and the way the individual sounds combine to create a compelling soundtrack. The downsides of the traditional practice of giving the game programmer a completely free hand to set the compositional rules for these real-time audio collages are now widely acknowledged, and it has become more or less standard practice in advanced IA systems to provide the audio artist with some mechanism for controlling what happens in the soundtrack, and when, and how the soundtrack responds to real-time changes.

Therefore, an iXMF file contains not just the playable sound files, but also data that captures the semantics for when each file will be played, and sometimes how it will be manipulated as it plays.

2.1 XMF Structured Storage

This compound characteristic leads to a requirement that the data file be able to contain multiple data resources.

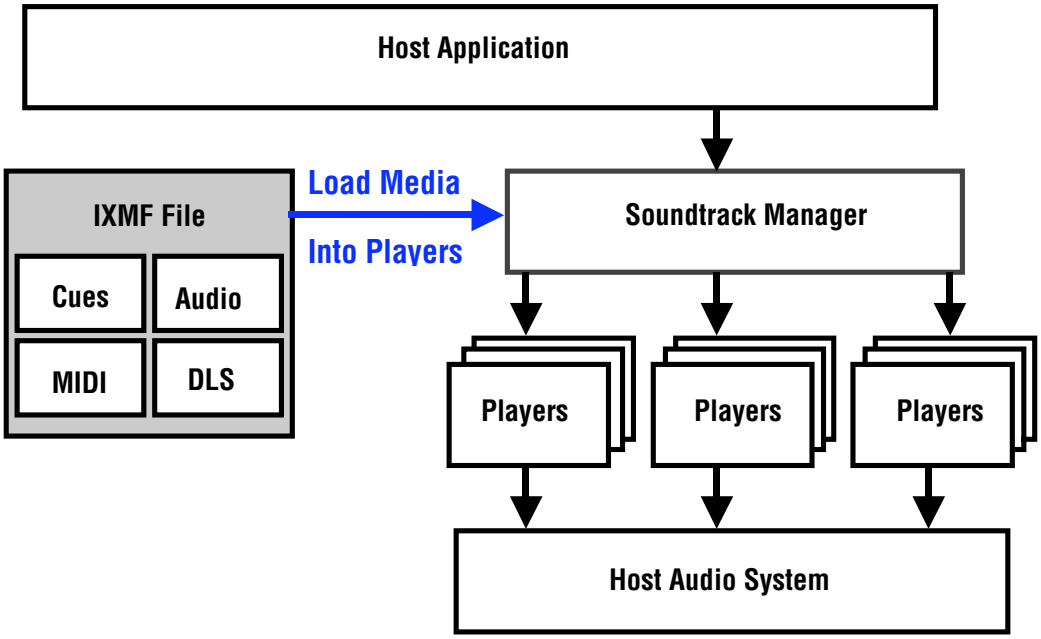


The iXMF-WG chose XMF technology for the container because of its flexible folder-tree orientation and its ability to accommodate future standardized or custom extensions, because it is free to use, and because it was also created and is maintained by the MMA. The XMF container technology is specified in [1].

In the current spec draft, media files are limited to audio files, Standard MIDI Files, and DLS-family instrument files. The set of available media formats can be extended by future standards, or with proprietary or open-source formats, though this would require authoring tool support.

2.2 Soundtrack Manager

The iXMF architecture assumes the existence of a Soundtrack Manager, a software layer that reads audio content from iXMF files and manages its interactive playback via any required number of Players.



Native Players for all the iXMF media types are assumed to exist for any given target platform. Depending on the platform, Players may take the form of (on computers) media playback API objects, or else of (on game consoles) a fixed set of available audio channels. The Soundtrack Manager also sets up and handles all Player callbacks (or interrupts) for markers embedded in the audio media, as well as any other position-based callbacks.

> **Note:** As a side project to encourage the proliferation of iXMF, the iXMF Working Group is considering creating an open-source, platform-independent C++ Soundtrack Manager implementation. This implementation would define a common API for an abstract 'Player adapter layer' object to glue the Soundtrack Manager to the pre-existing platform-native Player facilities, with the hope that bringing iXMF up on a new platform could be quickly done by simply writing an implementation of the adapter layer. Any such implementation would not be encumbered by any license, including the GPL or any of its variants.

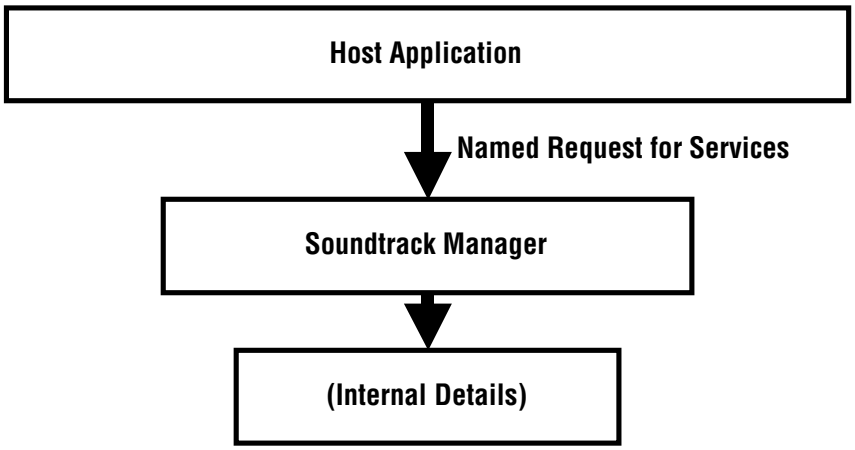
2.3 Cues

The structure of iXMF files and of the underlying Soundtrack Manager model is based on the premise that the primary semantic content unit of a soundtrack is a Cue. The draft specification defines Cue thus:

> "…the game (or other host application) requests interactive audio services using named, high-level events [called Cues], and the system's response to each event is determined primarily by the audio artist. The soundtrack's response to a Cue may be simple, such as playing or halting a single audio file, or it may be complex, such as controlling [the playback of] a system of several related files and dynamically manipulating various playback parameters over time."

A soundtrack is therefore constructed of some number of named Cues, as determined by negotiation between the game programmer(s) who must call the Cues and the audio artist(s) who must create them. For each required Cue there is a Cue definition data structure in an iXMF file, with some number of rules and some number of pointers to Chunks (region or whole file) of playable sound data. When the host requests a given Cue by name, the Soundtrack Manager searches its loaded iXMF files for a Cue with that name, and executes the appropriate rule, in most cases causing some of the linked sound files to play.

```
startCue("mainTitlesMusic");
```



There are no restrictions on the ASCII text strings used for Cue names, so developers are free to invent any desired naming convention. When the game requests a Cue, the Soundtrack Manager returns an ID that can be used for any needed subsequent communication with that Cue.

Example Cue Requests:

```
Dialog-Danny-Line0233

Music-Level22-Intro

Sfx-CarExplosion( 3 )
```

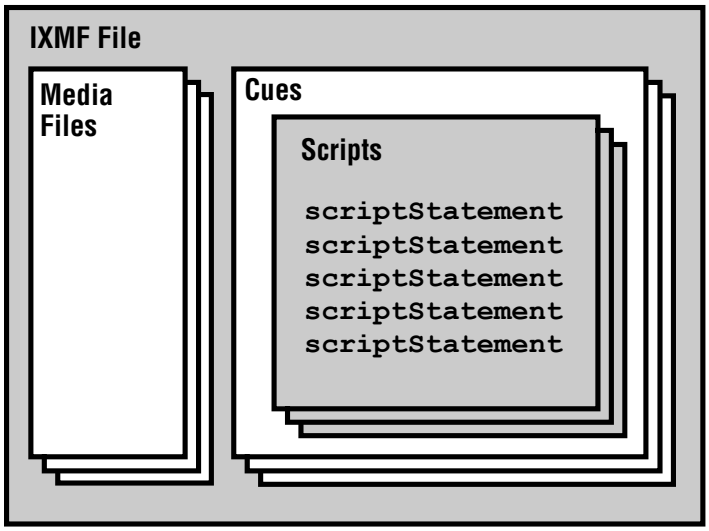
The corresponding iXMF File would contain definitions for the same three Cue names, Dialog-Danny-Line0233, Music-Level22-Intro, and Sfx-CarExplosion.

Note that the last example above shows a parameter. At the API level, Cue requests can optionally include one or more numeric parameters, which the behavior rules in a Cue (explained in the next section) can interrogate, for example to select a variation, set a volume level, or otherwise adapt to game state. Use of these parameters of course requires specific cooperation and understanding between game programmer and audio artist.

Cue definitions can be seen as iXMF's internal, sound-artist-created handlers for events that originate outside of the Soundtrack Manager domain; later, you will see the Callback Handler mechanism, which allows the outside world to provide handlers for events that originate within the Soundtrack Manager domain.

2.4 Behavior Rules: Events and Scripts

The Cue rules described earlier take the form of some number of Scripts stored within each Cue definition structure.



Scripts are named, and a Script is invoked when an event with the same name occurs while that Cue is running. Scripts can be seen as handlers for events that originate within the Soundtrack Manager domain, specifically within that same Cue.

A Script is a simple program in binary opcode format, not source code format, which is crafted by the audio artist to implement some aspect of the real-time assembly of the desired soundtrack. A Script consists of one or more simple statements. The set of available Script statements can be extended by future standards, or with proprietary or open-source extensions, though this would require authoring tool support.

Example Script Statements:

```
conditionalBranch( expression, labelID )

preloadCueMedia( cueID )

setDspParameter( dspParamID, value )
```

Certain Script names are predefined, and will be invoked by the Soundtrack Manager under pre-determined conditions, such as:

`CueStart` – This Cue is about to be started (set up instance)

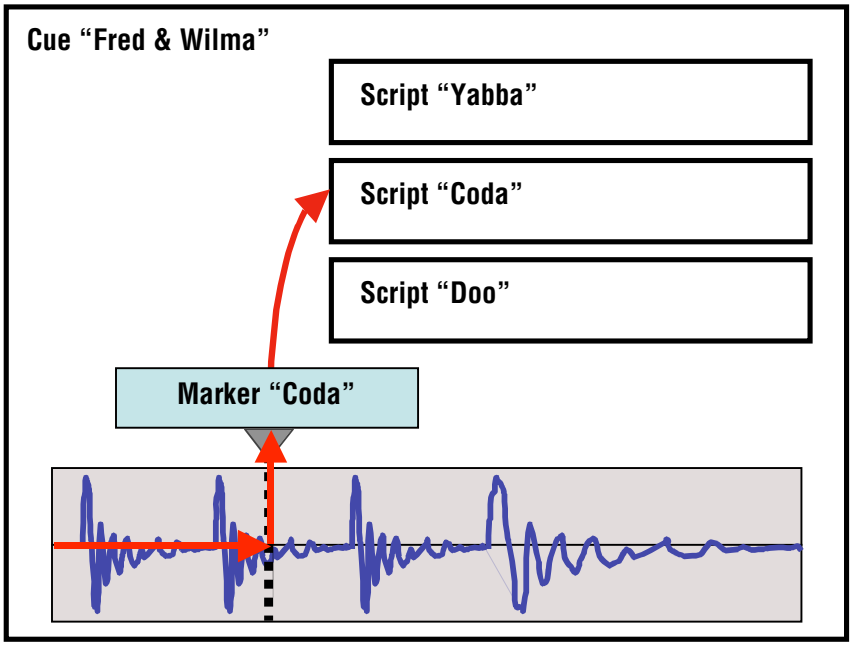
`CueCanceled` – This Cue has been canceled by the host (tear down)

`XmfFileLoaded` – The iXMF file containing this Cue has just been loaded (set up statics)

`ChunkEnded` – One of the Chunks in this Cue has just ended (prepare for next Chunk)

`XmfUnloaded` – The iXMF file containing this Cue is about to be unloaded (tear down statics)

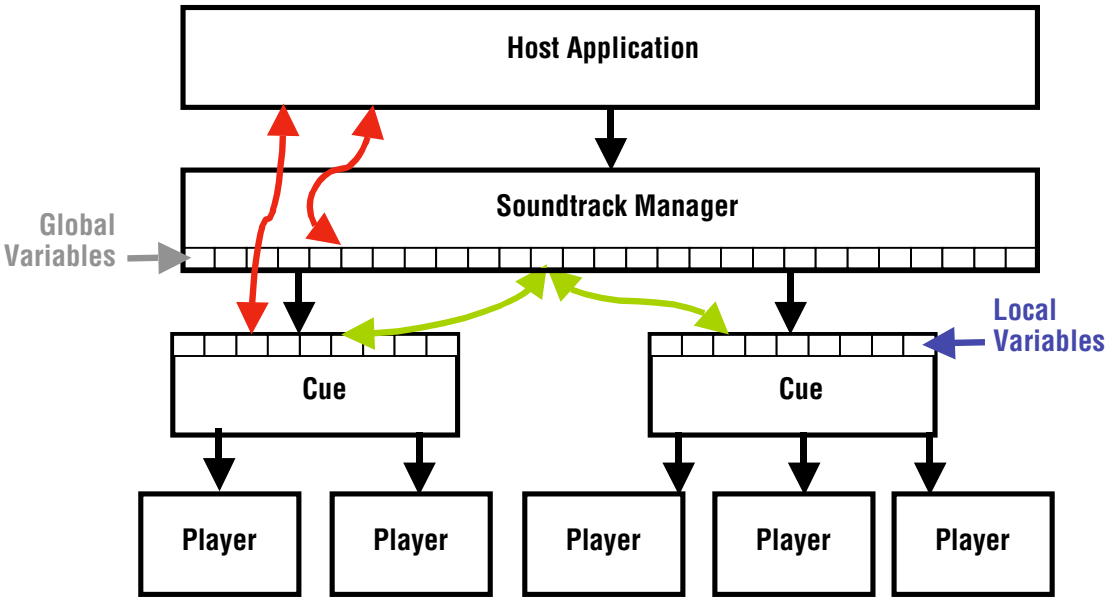
All other Script names are available for use by the audio artist. Each of theseScripts will be invoked by the Soundtrack Manager whenever a marker with the same name is encountered during Chunk playback within the same Cue.



For example, if the audio artist places a marker named “Coda” in an audio file, and that audio file is played in a given Cue that includes a Script named “Coda”, then that Script will be called when playback passes through that marker. This mechanism gives the audio artist great flexibility in synchronizing soundtrack elements to one another, and in signaling soundtrack events to the host game.

2.5 Variables

To maintain state, count loops, and facilitate communication with the host and other Cues, each Cue is given an array of local variables. With Script statements, local variables can be read, set, modified, and tested in conditional branches. No Cue can see any other Cue’s local variables. The Soundtrack Manager also maintains a separate array of global variables that can be used in the same ways, but are visible to all Cues simultaneously; this makes one-way or two-way communication among multiple running Cues possible.



All local and global variables can also be set and read by the host game; this makes one-way or two-way communication between a running Cue and the game possible, although since the variables are only indexed and not named, this requires a certain amount of planning and cooperation between the game programmer and the audio artist.

Example Script Statements Using Variables:

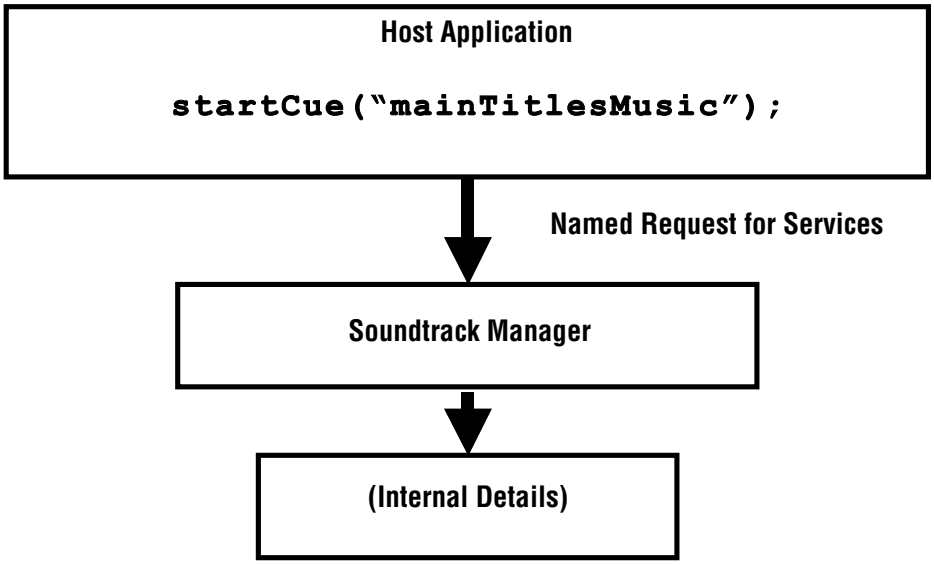
```
setVariable( localOrGlobal, variableID, value )
val = getVariable( localOrGlobal, variableID )
```

3. Functionality Summary

The architectural elements of Cues, Scripts, Variables, and playable Chunks have been designed so that when combined in various ways, a comprehensive set of advanced IA functionality is made available. This section summarizes that functionality, as broken down into several subject areas.

3.1 Implementation of Cue Interface

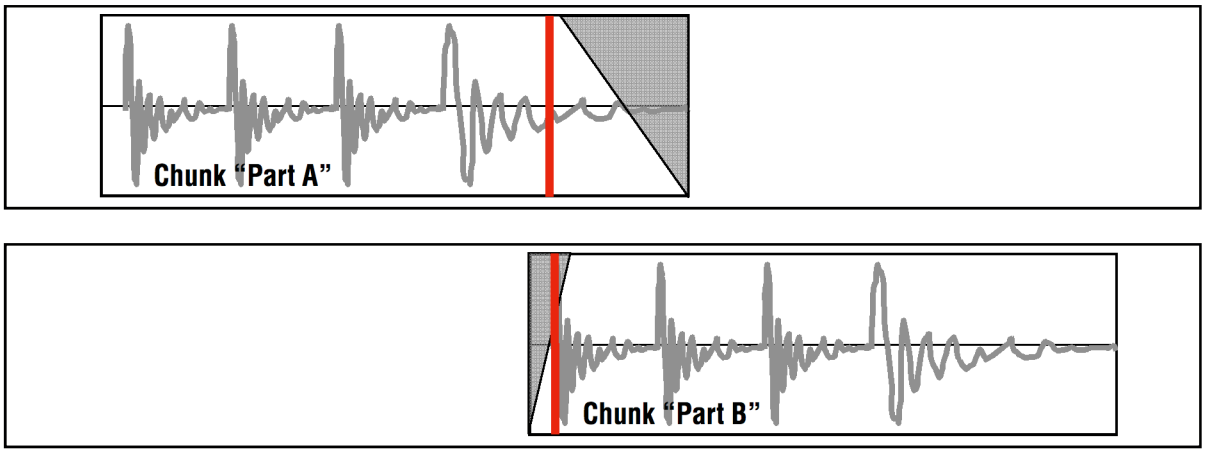
Most basically, the Soundtrack Manager receives Cue requests from the host game, and responds by fetching the corresponding data from the iXMF file, and executing it. In most cases this means playing appropriate sound media, sometimes influenced by game state. The Cue interface 'gets the ball rolling,' in the sense that once a Cue has been started, its internal actions as defined by the audio artist are able to take over all internal management of the Cue.



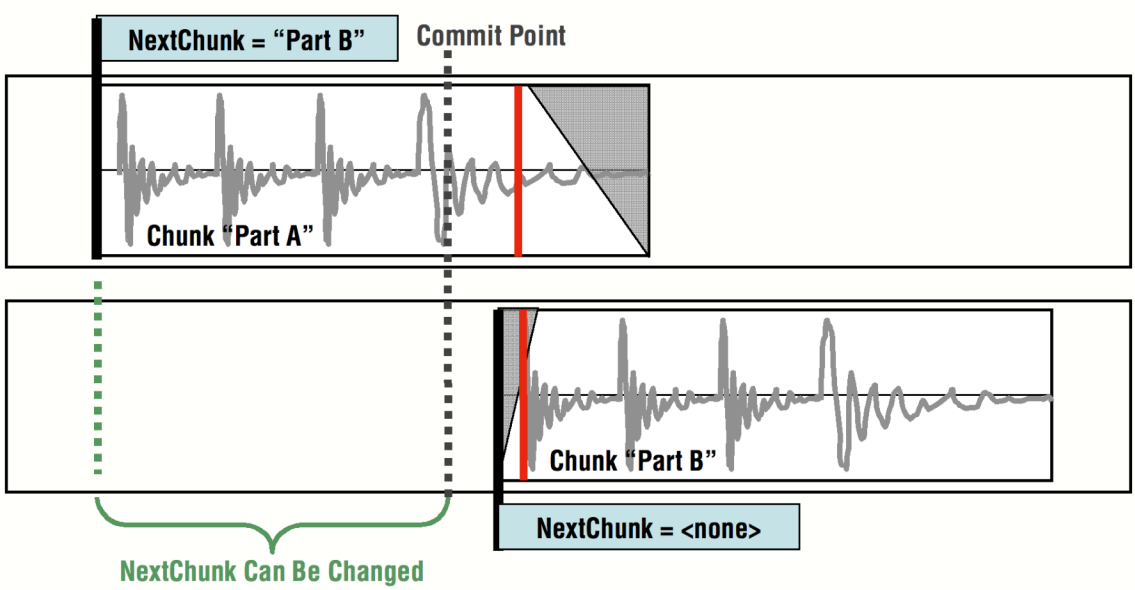
3.2 Assembling Continuous Sounds

Perhaps the most basic function of any IA sound system is the runtime assembly of each apparently continuous soundtrack element (such as a music bed, or an ongoing background sound effect) from discrete media Chunks, without producing audible gaps or seams at the transitions. This generally requires the use of at least two Players and crossfading, as well as a mechanism for loading and pre-queuing the Players with the appropriate Chunks. This section explains the iXMF Soundtrack Manager's mechanism for achieving this assembly, using a technique that simultaneously allows for both dynamic branching and preserving artist-defined sync relationships among the Chunks.

For smooth transitions between Chunks, two Players are used for each transition. This allows an outgoing Chunk to be overlapped with an incoming Chunk, and cross-faded, rather than producing a disruptive sudden cut-out:

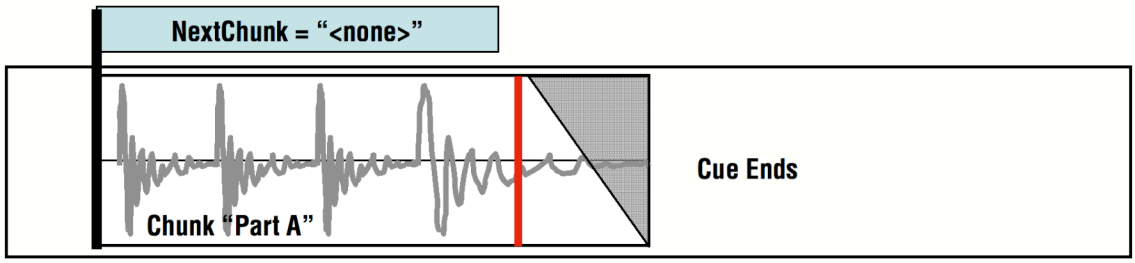


However since this is generic functionality, it says nothing about the mechanism for determining which of the audio artist's many media Chunks will be played next. For simplest implementation and maximum flexibility, iXMF uses the concept of a NextChunk variable. The value of NextChunk can be set at any time, however it is only used and acted on at the last possible moment before the upcoming transition, when the Soundtrack Manager needs to commit to the transition:

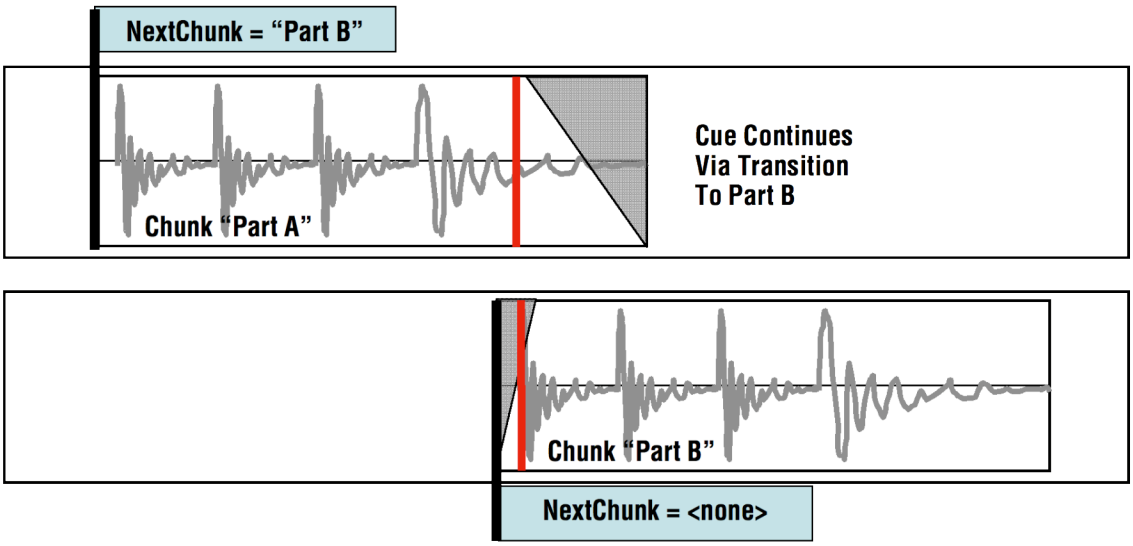


Among other things (see next section), this approach allows the audio artist to give each Chunk in an iXMF file the ID of the Chunk that is supposed to follow it, setting up a chain of Chunks without having to do any scripting at all. Here’s how.

If NextChunk is empty, then playback stops at the end of the current Chunk and this Player is released; if this is the last Chunk for that Cue, then the Cue will also expire and be released.



However, if NextChunk isn’t empty, but instead points at another, valid Chunk, then at an appropriately minimal interval before the current Chunk’s exit sync point, the Soundtrack Manager prepares a second player with the indicated NextChunk, and then starts the second player at whatever time is needed to synchronize the new Chunk’s entry sync point to the current Chunk’s exit sync point. This achieves the desired smooth transition to the new Chunk:



Note that these entry and exit sync points are not required to coincide with the Chunk start and end; in this example, the red lines are the sync points and might represent musically interesting times, perhaps a bar line.

iXMF supports the concept of different SyncTypes, for example ‘match marker names,’ ‘match beats,’ ‘match bars,’ ‘match phrases,’ ‘match any marker,’ and so forth. For music, the iXMF-WG is currently working on a separate sync scheme for algorithmically determined synchronization that would not depend on placing markers in the playable sound media, but rather would automatically align bars and beats based on tempo maps in MIDI format, which the audio artist would associate with each audio music file.

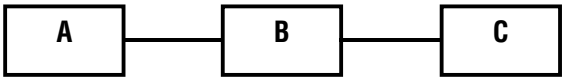
Since it is common to need more than one kind of continuous sound element at once, iXMF also supports the concept of Sync Groups. Every Cue (and, via Scripts, optionally every Chunk in a Cue) can be assigned to a named Sync Group, and it will then only synchronize to other elements also belonging to the same Sync Group. Since Sync Groups span Cues, this makes it possible to trigger new Cues or Chunks on the fly, even if not originally authored as part of the element that’s currently playing, and have them automatically time-align themselves.

With regard to the selection of which Chunk will be used as NextChunk, iXMF allows continuous sound elements to be constructed either via static playlists of Chunks, or else through the use of dynamic rules.

3.3 Static Chunk Ordering

With regard to the selection of which Chunk will be used as NextChunk, iXMF allows continuous sound elements to be constructed either via static, pre-defined sequence of Chunks, or else through the use of dynamic rules.

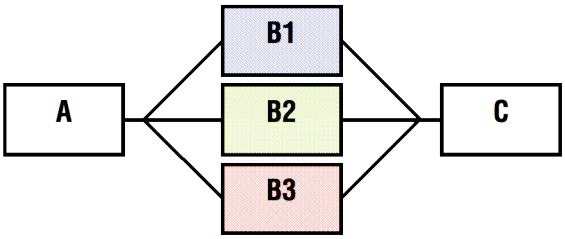
There are two ways to set up a static Chunk order:



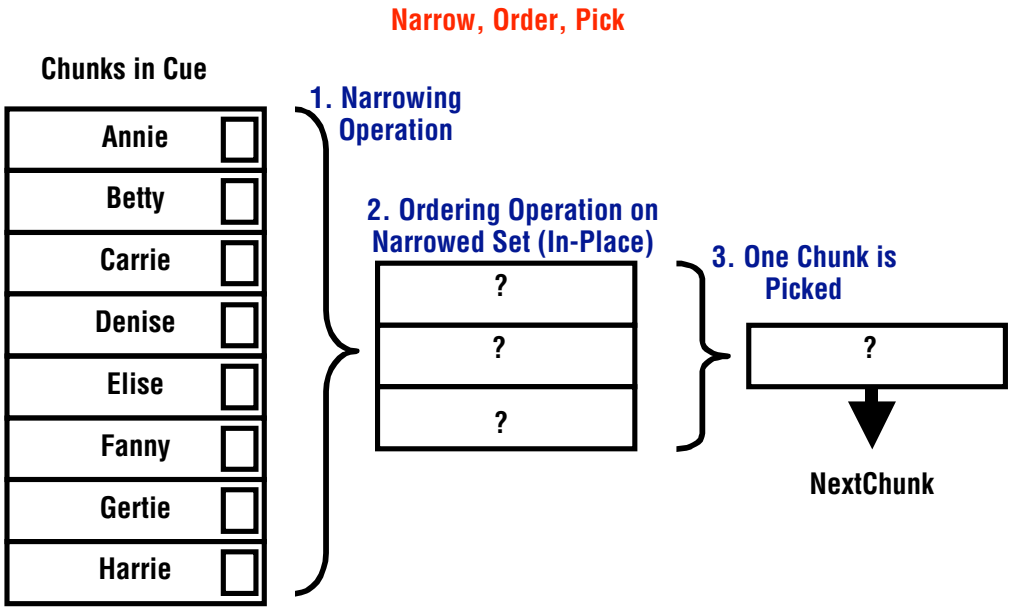
- As described above, the audio artist can set a default NextChunk for each Chunk in the iXMF file. This will cause the Chunks to chain one to the next, using two Players. This is a good technique for many applications, however it has the limitation that each Chunk only has one default Next Chunk, which limits re-usage options.
- Alternatively, the audio artist can use Scripts, specifically the setNextChunk statement, instead of relying on the default NextChunk ID stored with each Chunk.

3.4 Dynamic Chunk Selection and Ordering

Dynamic ordering, or selection of which media Chunks to play, is both much more interesting and much more open-ended. Dynamic ordering may be desirable in order to reflect changes in game state, and/or in introducing variations in an attempt to reduce the listener's sense of repetition in the available audio material. For example, in a piece with simple A-B-C structure, the audio artist may want to provide three alternative versions of the B section and select one randomly:

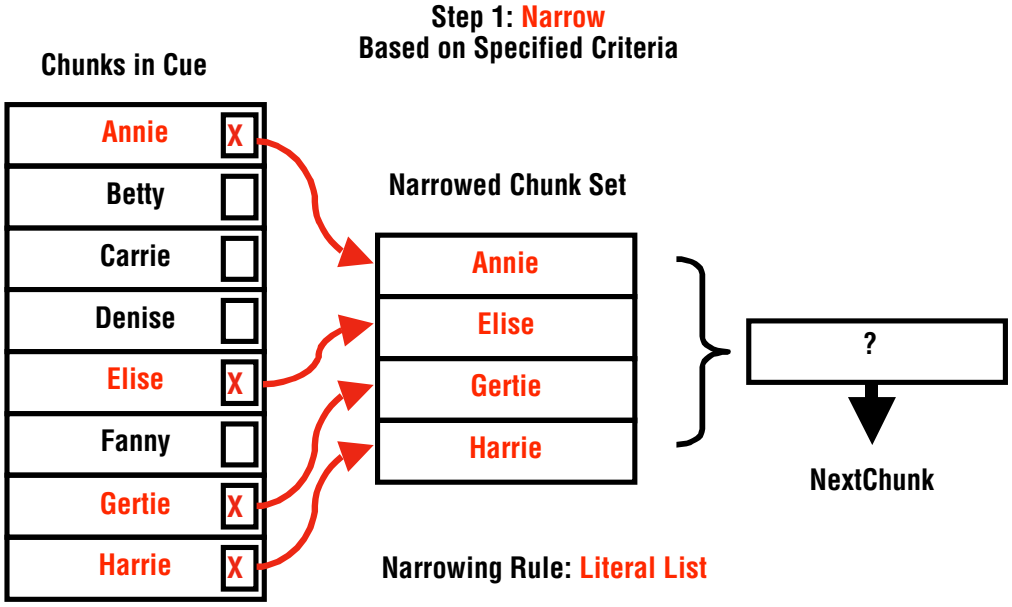


In iXMF, dynamic chunk ordering is accomplished via Scripts, either by means of conventional conditional branches in Scripts that the audio artist creates, or else by using a 3-step process using Script statements called Narrow, Order, and Pick:

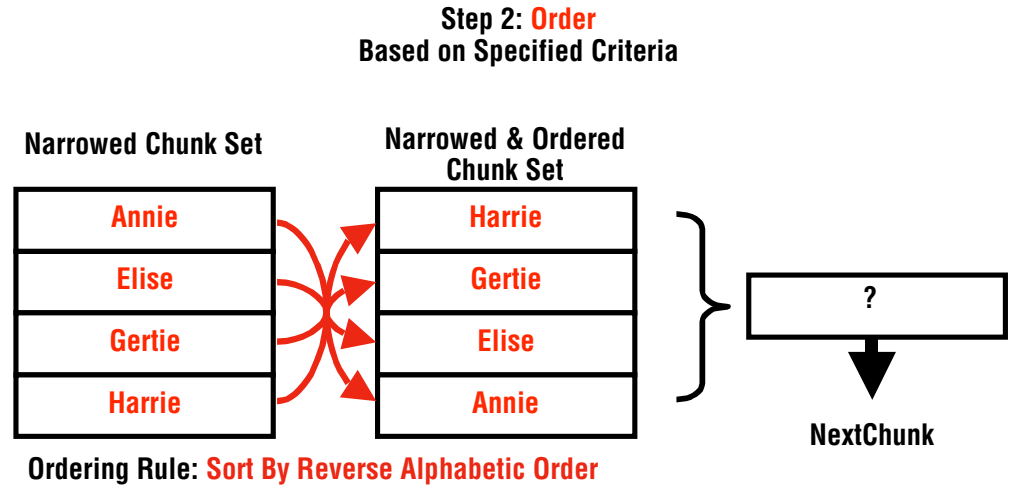


The 'Narrow, Order, Pick' mechanism is designed to be the simplest possible way to preserve the full range of selection, exclusion, and ordering criteria that have been found to be useful in game sound practice.

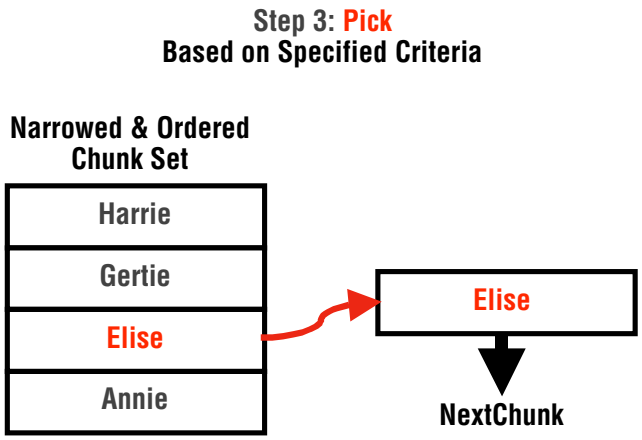
Step 1 is 'Narrowing.' This means creating a sub-pool of one or more of the Chunks used by the Cue. This is achieved either by supplying a literal list, or else according to a criterion such as least recently used or a metadata query.



Step 2 is 'Ordering.' This means to sort the narrowed pool according to some specific criterion, for example: name, order of most recent use, order of appearance in the iXMF file, or value of a given metadata field.



Step 3 is 'Pick.' This means to select a single Chunk from the narrowed and ordered pool, according to a specific criterion; for example: index, random, or next.

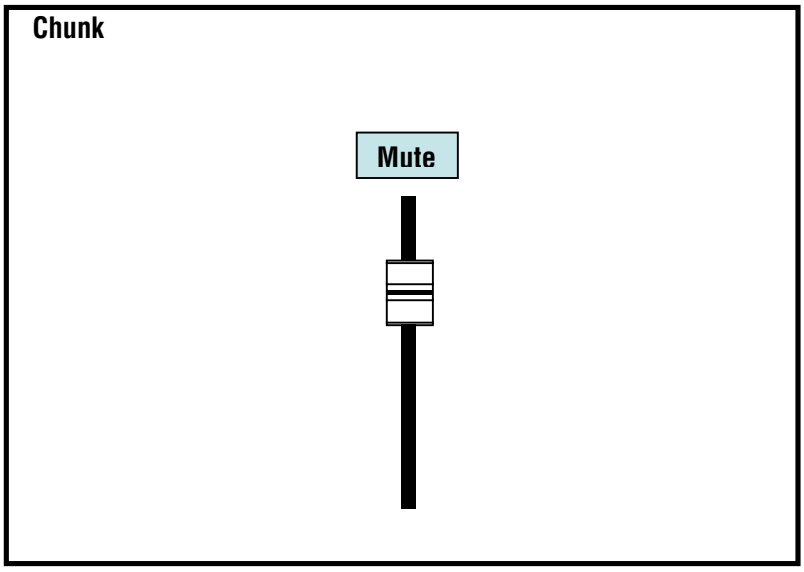


Once a Script has identified a particular Chunk as the next one to be played, it uses the **`SetNextChunk`** statement. Then, at the appropriate time, the Soundtrack Manager will play that Chunk.

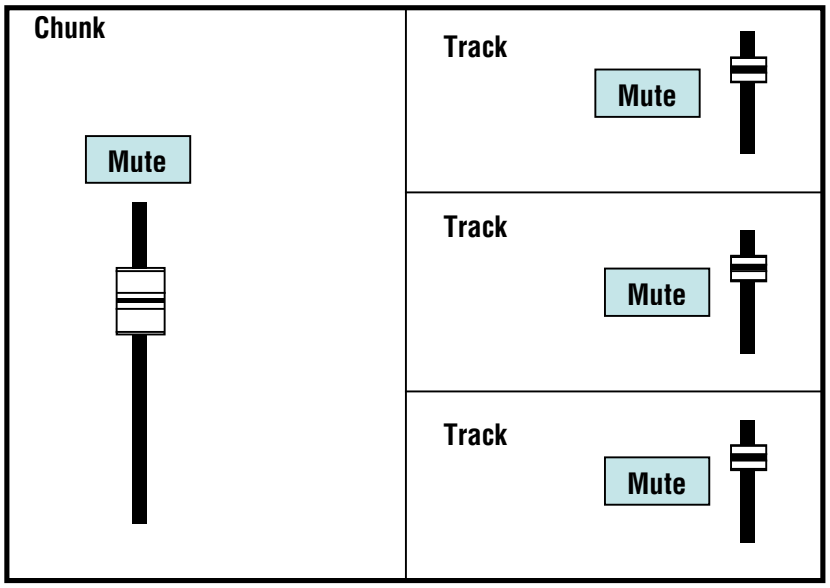
3.5 Mixing and Muting

For practical and expressive reasons, iXMF uses a 4-level hierarchical model for mixing and muting (Track, Chunk, Cue, and MixGroups).

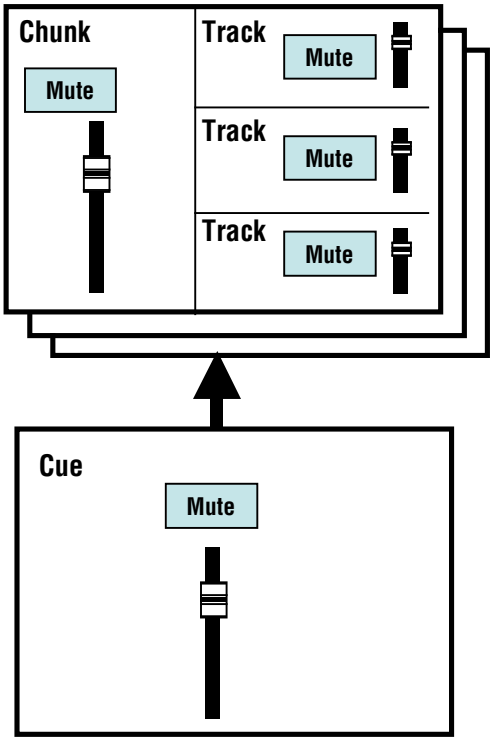
With Script statements, the audio artist can set or ramp a fader level, or mute or unmute, any currently playing Chunk:



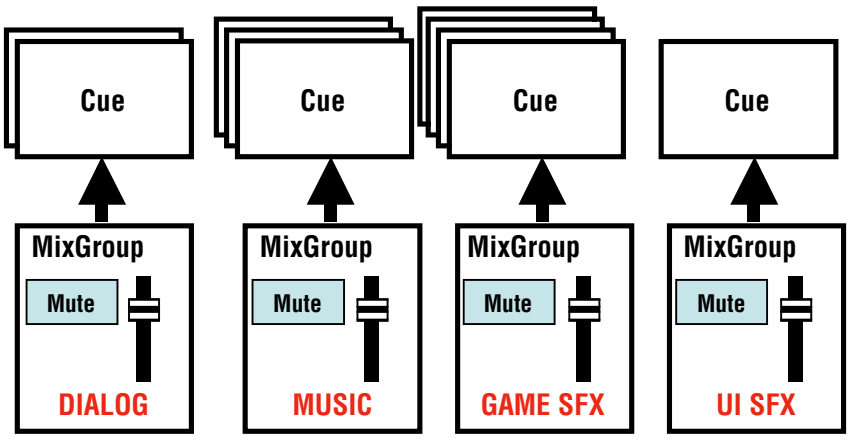
Chunks also have a separate volume and mute for each Track:



Further, Cues have master volumes that apply to all their Chunks:



Finally, for easier mixing of semantically related soundtrack elements, Cues (and optionally their Chunks) can additionally be assigned to artist-named MixGroups, each of which has a master fader level and a mute switch. In typical game applications there may be MixGroups for Dialog, Music, Gameplay Sfx, and UI Sfx:



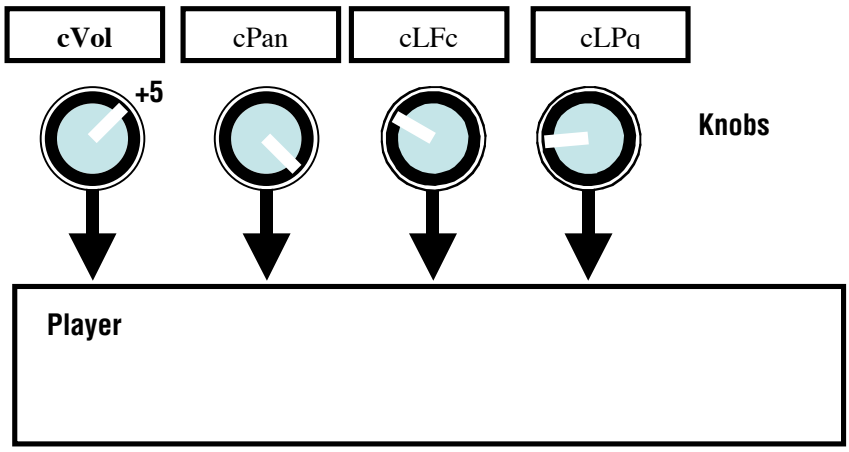
All of these virtual gain stages are specified similar to a VCA subgroup, where although the master fader affects all members of the group, there is no requirement for the audio engine to actually combine the signals. All that is required is to sum all the relevant faders and apply a single net volume at the individual track level.

Since Scripts can be triggered by markers placed in the playable media, or by Cues sent from the game host, and since the game can communicate through Variables, this mixing and muting mechanism is very flexible from a creative point of view.

3.6 Dynamic DSP Parameters

Script statements allow for continuous dynamic control of any available DSP parameter offered by the native playback engine. Typically these include, at a minimum, volume and stereo pan, and on some platforms may include such things as spatial location. The draft specification includes a short list of proposed four-character IDs for common DSP parameters, however it is hoped that in the future some global common registry could be established for this purpose, perhaps through cooperation between the IA-SIG, the MIDI Manufacturers Association, and the Audio Engineering Society.

```
setDspParameter("cVol",  +5  );
```



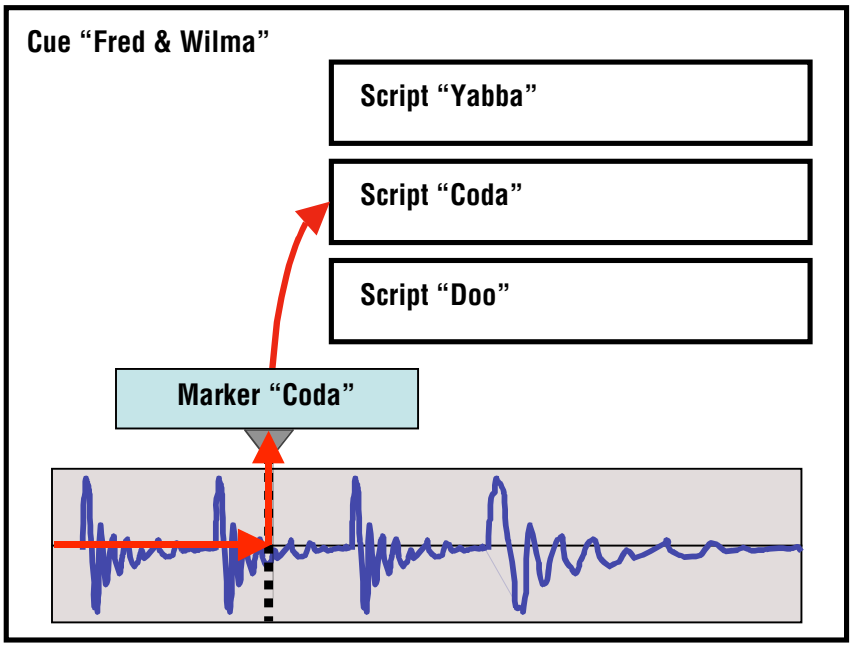
Again because of the architecture's flexibility, these settings can vary over time, for example in response to game state influence, or else as scripted by the IA artist in order to reduce the listener's sense of repetition.

3.7 Media Handling

For each playable media file used in the iXMF file, the audio artist has control over where the file is stored (local disk or network), and whether the file will be played by streaming from storage through a small buffer, or else by loading the whole file into memory and playing from the loaded copy.

3.8 Callback Scripts

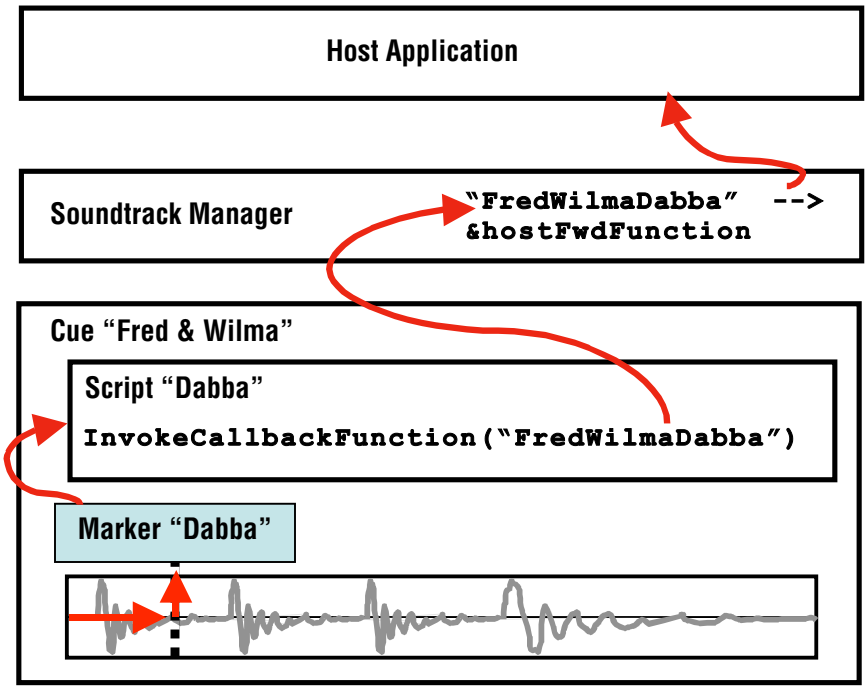
The audio artist can place named markers at any position in the playable sound media files; when playback passes through a marker, any Script with the same name as the marker will be executed. This allows any media files to trigger any Script at any time.



Because Scripts are very flexible, this allows the audio artist to tie practically any response to any given marker. For example, a callback Script could signal to the host game when certain points (time offsets) in the chunks are reached (typically: sound effect hit points, or musical accents or phrase boundaries).

3.9 Host Game Callbacks

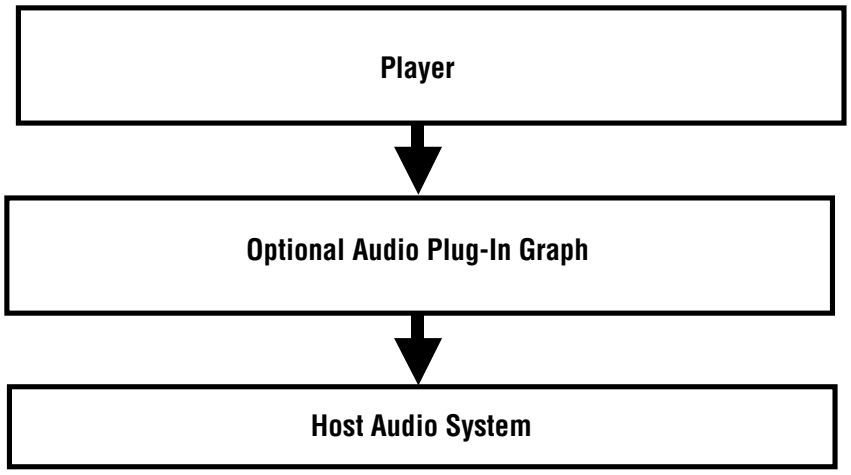
The audio artist can invoke any registered host game callback function from any Script by using the Script statement `InvokeCallbackFunction("callbackFnName")`. However, making good use of this feature requires prior planning and cooperation between the game programmer and the audio artist.



The defined callbacks are named, and represented by data stored in the iXMF file.

3.10 Audio Signal Chain Management & Plug-Ins

While the current draft specification does not include a specific design for this area, future versions of iXMF are expected to address management of the audio signal chain between the Players and the final system mixer, including optional graphs of user DSP plug-ins and the control of their parameters.



There is hope that the MMA's Generalized Music Plug-In Interface (GMPI) project [2] will be successful and have a role in this area, as it is the first plug-in API project that addresses both audio and music in a standardized and platform-independent manner.

3.11 Priorities and Scalability

The audio artist can assign a default priority for each Cue, which the host game can optionally override when requesting the Cue. When the Soundtrack Manager is requesting more resources (memory, number of

players, CPU bandwidth, synth voices [e.g. SP-MIDI], etc.) than the host platform is able to furnish, Cues with lower priorities will be canceled to allow those with higher priorities to play.

On appropriate platforms, such as mobile devices, Scalable Polyphony MIDI players could be used in an iXMF system, supporting content that adapts the musical arrangement to available number of synthesizer voices.

4. Content Development

This section describes how the use of iXMF affects the development of titles, including both the interactive audio content and the software development effort for the product. In general, the audio artist creates and edits iXMF files with an authoring tool, according to guidance from the game designers, and then the game engineers integrate the iXMF files into game builds.

4.1 Audio Artist Experience / Game Development Team Relations

[This section contributed by George A. Sanger, 'The Fat Man']

It is typical for an artist who is attempting to create IA to experience great challenges. The greatest challenges lie in the areas of (1) availability of good tools and (2) efficient artistic interaction with the development team. It is anticipated that wide adoption of iXMF as a standard will alleviate many of those challenges.

Standard file formats such as MIDI, .WAV, .MP3, .DLS, etc. for linear (non-interactive) sound have been around for years. As a result, there are plenty of adequate if not downright wonderful tools in place, provided by the Musical Instrument industry, that enable an audio artist to make linear music and sounds effectively. Movies, TV, and in-game cinematic scenes are all relatively easy to score using commercially available software and hardware. In the game world, however, the artist's job is to map those linear sounds, and possibly other sounds that might be created "on the fly," into the interactive context of the game.

Millions of dollars and thousands of man-years of effort have been put into building and mastering audio tools for such interactive uses that have not lived up to expectations and become obsolete, and when those tools, platforms, and companies have gone away, most of the lessons learned in the process of creating and using them have gone, too. In addition, as elaborated upon in Section 1, there are only a lucky few who have had access to these tools, so it's been logical for the toolmaker to put proportionally little effort put into them, resulting in less than perfect interfaces. Each of the tools created in the past has forced the user to create works that can only be edited by that tool, again severely limiting the reasons to even begin to use each system. As a result, for the independent game audio creator, there is no fully developed tool. The independent audio artist working on games is pretty much a blacksmith without an anvil.

The iXMF format will allow tools to be developed for general use independently of specific platforms and companies. The result will be that tools and GUI's will not need to be scrapped every time a company or platform or preferred method of editing changes. Efforts put into building tools will not be wasted: That means it will make more sense for tool creators to invest effort into those tools, and it will make more sense for a content creator to invest in learning the tools. It also means that the best parts of tools will be seen and used by enough people that we will likely see layered improvement of GUI's, features and functionality. Building better and better tools will finally make sense, and the usefulness of each tool will increase significantly.

Aside from the non-availability of tools, the other area in which audio artists see significant challenges is in the interface between his work and that of the development team. All too often, programmers are forced to make artistic decisions, shouting in frustration that music needs to be a bit more "green," or more like Terminator II, or in an effort to hear dialog clearly, a developer will brutally mix music completely out of a scene. Worse, sometimes artists are forced to program, and the results of that are predictably slow, riddled with errors, and otherwise awful. Good fences make good neighbors, and iXMF is designed to sit precisely where the fence is needed between audio artists and game developers. All of the issues that matter to the audio artist and that should be under his control (and no more), such as volume, fade speed, and the name of the file that plays in a certain gamestate, are available to be edited by him and auditioned using iXMF-

savvy tools, the design of which can make the process as simple or as detailed as desired on an individual basis. All of the issues that fall under the expertise of the game developer (and no more), such as the timing of when the player enters a certain gamestate, or such as a tension-related variable, are available on the programming side of the fence, to be passed at his will as "Cues" and Variables to the iXMF Soundtrack Manager. One might say that iXMF mimics the appropriate dialogue between game designer and audio designer. The Game Development side says, "At this point, the game is doing *this* (Cue), and we want some appropriate sound behavior," and the audio side says, "Very well, at that (Cue) point, I'll cause the sound to behave *this* way. When shall we cue the next behavior?"

As a result, huge increases in efficiency are expected under iXMF, leading to increased resources available for artistic effort on the part of artists, and programming effort on the part of programmers. Increased stability in audio engines can be expected, leading to more reliable products and less time in testing.

Efficient game production plus higher customer satisfaction equals bigger profits.

It doesn't stop there. The standardization of the file format enables software from various sources to interact in a modular way with other software, which could lead to unpredicted uses of iXMF. iXMF could, for example, even be used to control the flow of music from Internet radio stations in appropriate response to workers' productivity or the weather, or it could serve as a tool to allow video authors to create interactive movies that respond to the output of a MIDI glove.

The most remarkable benefits will be artistic, though this may only become apparent when masses of people get their hands on interactive media editing software, and start creating things of which the readers of this document never dreamed.

4.2 Authoring Tools

An iXMF content authoring tool would be a software application with a GUI of moderate complexity. At a minimum, the GUI would include the following elements:

- A list of defined Cues
- A list of playable sound files & their defined regions ('Chunks')
- A Cue Editor window, for adjustment of all data field in the Cue, including a list of available Scripts
- A Script Editor window, for inserting, deleting, and editing Script statements in any given Script
- A list of defined Callbacks for the host game to register, and that can be invoked via a Script statement

Additionally, the inherent extensibility of the XMF container technology makes it possible to add notes, comments, source control information, and other desired metadata to the whole iXMF file, or to any piece of it.

> **Note:** iXMF-WG members have prototyped such a GUI, and hope to release this work along with the final iXMF specification. This GUI prototype may be presented in the GDC session.

5. Notes

A few unobvious characteristics of the iXMF architecture should be pointed out:

- While the most typical application of IA is in game soundtracks, similar requirements are also present in other applications including themed sonic environments and improvised electronic music performance, one variant of which would be DJ-style performance. With an appropriate GUI (and perhaps a MIDI-driven) software application containing just a Soundtrack Manager, iXMF media could serve as an appropriate 'session file' for these types of musical performance uses.
- While this paper has focused exclusively on music and sound applications of iXMF technology, it should be noted that nothing in the framework itself prevents the 'Soundtrack Manager' and iXMF

file format specification from being extended to handle other media types, including images, movies, sprite animations, etc. The path to completion would be fairly short: glue code to manage and control Players for the other media types would need to be written, and media type tokens would need to be defined.

- Due to the underlying general similarity of many existing IA systems, it may be feasible to treat iXMF as an importable format, rather than a directly playable format for some of those systems. This would allow those systems to take advantage of iXMF's tool advantages, and help address their users' desires to port IA content that originates on other platforms.
- The playable sound files in the iXMF file could optionally be represented as links to external files, or links to files stored in other XMF files, through the use of XMF container technology features. This makes several things possible, including very small iXMF files, storage of content on remote servers via the Internet where they can be dynamically updated by the content provider, and sharing a single copy of any needed sound file among any number of iXMF files.

6. Standardization Status

At the time this paper was submitted for publication, it was not known with certainty when the iXMF-WG will be completing and releasing a final specification. The draft specification is undergoing internal review and revision, with a number of design detail issues as yet unresolved. This is not surprising since, even more than most standards efforts, iXMF is a volunteer-driven project. Until the final specification is released, the best way to monitor this work and evaluate its detailed technical design is to join the iXMF working group; see [3].

Updated status information will be presented during this session at GDC.

7. References

[1] *"Specification for XMF Meta File Format."* RP-030, MIDI Manufacturers Association, Los Angeles, CA, USA, http://www.midi.org/xmf

[2] Generalized Music Plug-In (GMPI-WG) working group, MIDI Manufacturers Association, Los Angeles, CA, USA, http://www.midi.org

[3] Interactive XMF (iXMF) working group (IXWG), MIDI Manufacturers Association Interactive Audio Special Interest Group (IASIG), Los Angeles, CA, USA, http://www.iasig.org

[4] *"Report of 'Big Picture' Group."* October 1999, Project Bar-B-Q, Austin, TX, USA. http://www.projectbarbq.com/bbq99/bbq99r4.htm

[5] *"Report of 'General Interactive Audio' Group."* October 2000, Project Bar-B-Q, Austin, TX, USA. http://www.projectbarbq.com/bbq00/bbq00r7.htm

[6] *"Report of 'Towards Interactive XMF' Group."* October 2001, Project Bar-B-Q, Austin, TX, USA. http://www.projectbarbq.com/bbq01/bbq01r5.htm