MIDI MANUFACTURERS ASSOCIATION

Supporting the Arts through Technology

MMA Technical Standards Board/ AMEI MIDI Committee

Technical Note — February 2000

Bundling SMF and DLS data in an "RMID" File (RP-029)

Source: MMA TSB Item #157 (Media Archive File Format; revised 1999-2000)

Primary Authors: Jim Wright (IBM Research), Rick Cohen (Kurzweil), Todor Fay (Microsoft), Monty Schmidt (Sonic Foundry)

Abstract

This document describes a method, using the RMID file format, to effectively bundle SMF and DLS data within a single file, as recommended by the MIDI Manufacturers Association. Familiarity with RIFF formats (such as RMID) and SMF file formats is assumed. Major topics within this document include The RMID Format; DLS Data in an RMID File; Copyright Notices, Version Stamps and Other Information; RMID Files and DirectMusic™; and Comments on Encryption Issues.

Introduction

Standard MIDI Files (SMF) have been in common use for a number of years. Downloadable Sounds (DLS) files are now becoming popular. However, there is no standard means for bundling a Standard MIDI File together with the DLS data which should be used to render the file. The RMID file format, with appropriate extensions described herein, is the MMA's recommended solution for this basic need.

It should be noted, however, that the "extended RMID format" described in this document is only intended to contain a single SMF data image, a single DLS data image, plus supplemental data such as a version stamp, copyright notice and other descriptive information. In this sense, an extended RMID file will be used simply for portability of linear music — "just like a General MIDI file" — except that authors can define their own instruments, rather than using only General MIDI instruments. No standard usage is defined for multiple SMF and/or DLS chunks within a single RMID file, or for any other chunks not described in this document. Such chunks should be used with care if portability is desired.

Additional chunk types, file formats and/or recommended practices will likely be defined by the MMA in the future to cover areas of use undefined by this document, such as non-linear playback and copy protection. Software developers are advised to check the status of this ongoing work before developing their own extensions to RMID.

The RMID format

RMID is a standard RIFF file format, originally defined in the Microsoft® Windows® Multimedia Programmers Reference, Microsoft Press, 1991, p. 8-31 (now out of print). RIFF is very similar to IFF, an earlier file metaformat used as the basis for AIFF and many other formats.

RMID was introduced because a Standard MIDI File (SMF) is not, by itself, a valid IFF (or RIFF) format. While individual SMF 'MThd' and 'MTrk' chunks comply with IFF conventions, the SMF format does not specify the top-level 'FORM' chunk required in a valid IFF-based file format. RMID provides the missing elements.
An RMID form (file image) contains a Standard MIDI File data image wrapped inside a RIFF chunk.
Using standard RIFF grammar, the RMID form is defined as:

```
<RMID-form>     RIFF(     'RMID'
                          data( <MIDI-data> )
                    )
```

<MIDI-data> a Standard MIDI File, as defined in The Complete MIDI 1.0 Detailed Specification.

Note that it is perfectly legal to include additional RIFF chunks **after** the <data> chunk. For example, an INFO chunk may be added to provide copyright, author and other textual information describing the file. However, the <data> chunk must be the first chunk found within the RMID form.

A minimal RMID form contains exactly two RIFF chunks: the <RIFF'RMID'> chunk which contains everything in the file, and the <data> chunk which contains the actual SMF data. This adds a 20-byte preamble before the start of the actual SMF data. For example, assume the SMF data has a total length of 118 bytes (0x76 hex). The resulting RMID file will look like this (actual data values are shown in hex on the middle line):

(0-3)	(4-7)	(8-11)	(12-15)	(16-19)	(20-23)	(24…)
52494646	82000000	524D4944	64617461	76000000	4D546864	xxxxxx.
'RIFF'	RMID_size	'RMID'	'data'	midi_size	'MThd'	data…

(Size values are little-endian: the ckSize '82000000' represents 0x000000082 hexadecimal, or 130 decimal).
The four-byte field 'RMID_size' states the total size of all data contained within the RIFF chunk (e.g.130 bytes).
The four-byte field 'midi_size' states the size of the SMF data within the <MIDI-data> chunk (e.g. 118 bytes).
The SMF data begins at byte 20 with the chunk header 'MThd'.

In this case, the difference between RMID_size and midi_size is 12 bytes (which is the size of the 'RMID', 'data' and midi_size elements). However, if the RMID file held an 'INFO' chunk or other chunk after the 'data' chunk, the difference would reflect the size of the additional chunk(s).

“**.RMI**” is the recommended extension for an RMID file. For example: “*CompositionWithInstrumentData.RMI*.”

DLS Data in an RMID File

DLS form data may be included in an RMID file by (a) appending the complete DLS form data to the end of the RMID form and (b) updating the RMID ckSize (in bytes 4-7) to reflect the new form data size.

Note that the DLS header must **not** be modified. The RIFF format allows a single top-level RIFF form (e.g. <RIFF'RMID'>) to contain one or more lower-level form chunks (with FOURCC 'RIFF') embedded within the top-level RIFF form.

For example:

Original RMID form:

(0-3)	(4-7)	(8-11)	(12-15)	(16-19)	(20-23)	(24…)
52494646	82000000	524D4944	64617461	76000000	4D546864	xxxxxx.

'RIFF'	RMID_size	'RMID'	'data'	midi_size	'MThd'	data…

Original DLS form:

(0-3)	(4-7)	(8-11)	(12…)
52494646	98680800	444C5320	xxxxxx.
'RIFF'	DLS_size	'DLS '	DLS data…

Extended RMID form with DLS data:

(0-3)	(4-7)	(8-11)	…	(138-141)	(142-145)	(146-149)	(150…)
52494646	**1A690800**	524D4944	…	52494646	98680800	444C5320.	xxxxxx
'RIFF'	**RMID_size**	'RMID'	…	'RIFF'	DLS_size	'DLS '	DLS data…

Note: while the 'DLS ' chunk now begins at file offset 138 decimal, the DLS_size value has not changed.
(Size values are little-endian: the ckSize ' 1A690800' represents 0x0008691A hexadecimal, or 551,194 decimal).

Copyright Notices, Version Stamps and Other Information

To attach a copyright notice for the entire contents of an RMID file, insert an INFO chunk following the RMID 'data' chunk. The copyright notice text should be contained within an ICOP chunk inside the INFO chunk, as per standard RIFF conventions. Copyright notices for other content elements within an RMID file must follow the appropriate convention for that content type.

To attach a separate copyright notice for the Standard MIDI File data, use Standard MIDI File meta-event 02, as described in the Complete MIDI 1.0 Specification. This meta-event should be inserted as the first event in the first 'MTrk' chunk, at time 0. (This follows Standard MIDI file conventions, not RIFF conventions).

For DLS, the form allows a top-level copyright notice and/or individual copyright notices for various parts of the form; all such notices are attached using INFO chunks (with nested ICOP chunks) inserted at appropriate locations within the DLS form.

To indicate the version of the contents of a given RMID file, insert a version chunk as the second chunk within the enclosing RMID form chunk, immediately after the <data> chunk and before any INFO chunk. It is recommended (but not required) that all RMID files that contain DLS data also include a version chunk.

For reference, the standard definitions of INFO-list and version chunks are shown below:

The INFO-List Chunk:

An INFO list is a LIST chunk with list type INFO. The INFO list is a registered global form type that can store information that helps identify the contents of the containing chunk. This information is useful but does not affect the way a program interprets the file; examples are copyright information and comments.

The following shows a sample INFO list chunk:

```
< INFO-list >   →     LIST(  'INFO'  [<info_text-ck>]...)
```

Example:

```
< INFO-list >    →     LIST(  'INFO'
                              INAM("Distorted Tuba"Z)
                             ICMT("The tuba meets Eddie Van Halen"Z)
                         )
```

An INFO list should contain only the following chunks. New chunks may be defined, but an application should ignore any chunk it doesn't understand. The chunks listed below may only appear in an INFO list. Each chunk contains a ZSTR, or null-terminated text string.

The <info_text-ck> chunks include:

Chunk ID	Description
IARL	Archival Location. Indicates where the subject of the file is archived**.**
IART	Artist. Lists the artist (author) of the original subject of the file. For example: *Les Getalong*.
ICMS	Commissioned. Lists the name of the person or organization that commissioned the subject of the file. For example: *Acme Consolidated GameWorks*.
ICMT	Comments. Provides general comments about the file or the subject of the file. If the comment is several sentences long, end each sentence with a period. Do not include newline characters.
ICOP	Copyright. Records the copyright information for the file. For example: *Copyright Acme Consolidated GameWorks 1991*. If there are multiple copyrights, separate them by a semicolon followed by a space.
ICRD	Creation date. Specifies the date the subject of the file was created. List dates in year-month-day format, padding one-digit months and days with a zero on the left. For example: *1553-05-03* for May 3, 1553. The year should always be given using four digits.
IENG	Engineer. Stores the name of the engineer who worked on the file. If there are multiple engineers, separate the names by a semicolon and a blank. For example: *Smith, John; Adams, Joe*.
IGNR	Genre. Describes the original work, such as *jazz, classical, rock, techno, rave, neo british pop grunge metal*, etc.
IKEY	Keywords. Provides a list of keywords that refer to the file or subject of the file. Separate multiple keywords with a semicolon and a blank. For example: *FX; visitation; space alien*.
IMED	Medium. Describes the original subject of the file, such as *record, CD* and so forth.
INAM	Name. Stores the title of the subject of the file, such as *Seattle From Above*.
IPRD	Product. Specifies the name of the title the file was originally intended for, such as *Galactic Ambassadors V*.
ISBJ	Subject. Describes the contents of the file, such as *Music of the Gnu Whirled Order*.
ISFT	Software. Identifies the name of the software package used to create the file, such as *Crash Compactor, Acme Consolidated Sonic Booms*.
ISRC	Source. Identifies the name of the person or organization who supplied the original subject of the file. For example: *Acme Hysterical Media Archives*.
ISRF	Source Form. Identifies the original form of the material that was digitized, such as *record, sampling CD, TV sound track* and so forth. This is not necessarily the same as IMED.
ITCH	Technician. Identifies the technician who sampled the subject file. For example: *Smith, John*.

Version Chunk

The <vers-ck> defines an optional version stamp within a collection. It is used in DLS files, but can also be used within an RMID file. A version chunk indicates the version of the contents of the file, *not the RMID specification level*. The version stamp follows the same four number format as the version stamps used in executable code (.exe and .dll files). It is left to the discretion of the RMID file author to assign a version number scheme to a file.

0The <vers-ck> is defined as follows:

```
<vers -ck>      →      vers(
                            <dwVersionMS:DWORD>
                            <dwVersionLS:DWORD >
                       )
```

The <vers -ck> chunk:

Field	Description
dwVersionMS	Specifies the high-order 32 bits of the binary version number for the file. The value of this member is used with the value of the dwVersionLS member to form a 64-bit version number. This 32-bit value can be further broken down with HIWORD(dwVersionMS) and LOWORD(dwVersionMS) to get two 16-bit values (version info is actually 4 16-bit values.
dwVersionLS	Specifies the low-order 32 bits of the binary version number for the file. The value of this member is used with the dwVersionMS value to form a 64-bit version number. Use HIWORD(dwVersionLS) and LOWORD(dwVersionLS) to extract the sixteen bit values.

The version chunk is essentially borrowed from the dwFileVersionMS and dwFileVersionLS fields in the Microsoft® VS_FIXEDFILEINFO structure which is used to store version information in .exe and .dll files. As such, the version can be broken down into four segments: the high and low words of the MS and the high and low words of the LS. For example, "VERSION 3,10,0,61" is translated into: dwVersionMS = 0x0003000a and dwVersionLS = 0x0000003d.

A <vers-ck> version chunk is typically inserted following the <data> chunk, as the second chunk within the enclosing RMID form. Note that the <data> chunk must be the first chunk within the RMID form in order to comply with the original RMID specification.

Comments on Encryption Issues

Encryption for content protection purposes is a commonly requested feature. However, effective content protection cannot be provided solely by means of a file format. Any content protection solution requires the active participation of trusted software components permitted to access protected content, which control how the protected content may be used.

Encryption per se is relatively straightforward: given an algorithm and a key, one can readily encrypt or decrypt data. The more difficult issues relate to management and administration of encryption keys. Is a given key valid on a single system, or on multiple systems? What about consumers with multiple computers? Is a key valid for a single use? A fixed number of uses? Valid indefinitely? When a licensed DLS collection is bundled with copyrighted SMF data, do the two content chunks use the same encryption key and policy? If not, how many keys are used? In what combinations? With what policies? Different applications (and business models) will require different answers to these questions.

When encryption is needed, it can be supported in two ways:

(a) by storing an entire RMID form image in an encrypted container file (which is not, itself, a RIFF file)
OR
(b) by defining proprietary RIFF chunks (with different FOURCC IDs) to contain encrypted content within an RMID form

The first approach allows an entire set of content elements to be protected and accessed as a unit. The second approach allows different encryption methods and policies to be used for different parts of the media archive. In either case, special software components would be required to access the protected content. The specification and operation of such components is outside the scope of this technical note.

Example Code: RMID Files in DirectMusic

Microsoft DirectMusic (version 7 and above) can directly load and play an extended RMID file containing both SMF and DLS data. The following source code examples are for DirectMusic version 7:

```
/* pLoader is a previously created IDirectMusicLoader object and
   pPerformance is a previously created IDirectMusicPerformance object.
   This code reads an RMID (or .mid or .sgt) file into an IDirectMusicSegment
   object, and downloads the DLS instruments to the performance so it's ready to
play.
*/

HRESULT myLoadSegmentFile( IDirectMusicLoader       *pLoader,
      IDirectMusicPerformance               *pPerformance,
      IDirectMusicSegment                   **ppSegment,
      WCHAR                                 wszFileName )
{
      // First prepare an object descriptor...
DMUS_OBJECTDESC Desc;
Desc.guidClass     = CLSID_DirectMusicSegment;
Desc.dwSize        = sizeof( DMUS_OBJECTDESC );
wcscpy( Desc.wszFileName, wszFileName )       // Note the the filename is UNICODE.
Desc.dwValidData = DMUS_OBJ_CLASS | DMUS_OBJ_FILENAME | DMUS_OBJ_FULLPATH;

      // Then hand object descriptor to the loader to get it...
HRESULT hr = m_pLoader->GetObject(&Desc, IID_IDirectMusicSegment2,
(void**)ppSegment);
if (SUCCEEDED(hr))
   {
      // Now, download the DLS instruments...
      *ppSegment->SetParam( GUID_Download,-1,0,0,(void *) pPerformance );
   }
return hr;
}

/* Later, to play the segment */

IDirectMusicSegmentState *pPlayingSegment;
pPerformance->PlaySegment( pSegment,DMUS_SEGF_DEFAULT,0,&pPlayingSegment );
/* The SegmentState is optional, but you can use it later to stop the segment
   while other stuff continues to play.
*/
pPerformance->Stop( NULL,pPlayingSegment,0,DMUS_SEGF_DEFAULT );
```

References

The Downloadable Sounds Level 1 Specification, v. 97.1, © 1997 MIDI Manufacturers Association.**{**xe "Normative References"**}**
The Complete MIDI 1.0 Detailed Specification v. 96.1, © 1996 MIDI Manufacturers Association.
Microsoft® Windows® Multimedia Programmers Reference, © 1991, Microsoft Press