Network Working Group
Request for Comments: 4695
Category: Standards Track

J. Lazzaro
J. Wawrzynek
UC Berkeley

November 2006

RTP Payload Format for MIDI

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The IETF Trust (2006).

Abstract

This memo describes a Real-time Transport Protocol (RTP) payload format for the MIDI (Musical Instrument Digital Interface) command language. The format encodes all commands that may legally appear on a MIDI 1.0 DIN cable. The format is suitable for interactive applications (such as network musical performance) and content-delivery applications (such as file streaming). The format may be used over unicast and multicast UDP and TCP, and it defines tools for graceful recovery from packet loss. Stream behavior, including the MIDI rendering method, may be customized during session setup. The format also serves as a mode for the mpeg4-generic format, to support the MPEG 4 Audio Object Types for General MIDI, Downloadable Sounds Level 2, and Structured Audio.

Table of Contents

1. Introduction

The Internet Engineering Task Force (IETF) has developed a set of focused tools for multimedia networking ([RFC3550] [RFC4566] [RFC3261] [RFC2326]). These tools can be combined in different ways to support a variety of real-time applications over Internet Protocol (IP) networks.

For example, a telephony application might use the Session Initiation Protocol (SIP, [RFC3261]) to set up a phone call. Call setup would include negotiations to agree on a common audio codec [RFC3264]. Negotiations would use the Session Description Protocol (SDP, [RFC4566]) to describe candidate codecs.

After a call is set up, audio data would flow between the parties using the Real Time Protocol (RTP, [RFC3550]) under any applicable profile (for example, the Audio/Visual Profile (AVP, [RFC3551])). The tools used in this telephony example (SIP, SDP, RTP) might be combined in a different way to support a content streaming application, perhaps in conjunction with other tools, such as the Real Time Streaming Protocol (RTSP, [RFC2326]).

The MIDI (Musical Instrument Digital Interface) command language [MIDI] is widely used in musical applications that are analogous to the examples described above. On stage and in the recording studio, MIDI is used for the interactive remote control of musical instruments, an application similar in spirit to telephony. On web pages, Standard MIDI Files (SMFs, [MIDI]) rendered using the General MIDI standard [MIDI] provide a low-bandwidth substitute for audio streaming.

This memo is motivated by a simple premise: if MIDI performances could be sent as RTP streams that are managed by IETF session tools, a hybridization of the MIDI and IETF application domains may occur.

For example, interoperable MIDI networking may foster network music performance applications, in which a group of musicians, located at different physical locations, interact over a network to perform as they would if they were located in the same room [NMP]. As a second example, the streaming community may begin to use MIDI for low-bitrate audio coding, perhaps in conjunction with normative sound synthesis methods [MPEGSA].

To enable MIDI applications to use RTP, this memo defines an RTP payload format and its media type. Sections 2-5 and Appendices A-B define the RTP payload format. Section 6 and Appendices C-D define the media types identifying the payload format, the parameters needed for configuration, and how the parameters are utilized in SDP.

Appendix C also includes interoperability guidelines for the example applications described above: network musical performance using SIP (Appendix C.7.2) and content-streaming using RTSP (Appendix C.7.1).

Another potential application area for RTP MIDI is MIDI networking for professional audio equipment and electronic musical instruments. We do not offer interoperability guidelines for this application in this memo. However, RTP MIDI has been designed with stage and studio applications in mind, and we expect that efforts to define a stage and studio framework will rely on RTP MIDI for MIDI transport services.

Some applications may require MIDI media delivery at a certain service quality level (latency, jitter, packet loss, etc). RTP itself does not provide service guarantees. However, applications may use lower-layer network protocols to configure the quality of the transport services that RTP uses. These protocols may act to reserve network resources for RTP flows [RFC2205] or may simply direct RTP traffic onto a dedicated "media network" in a local installation. Note that RTP and the MIDI payload format do provide tools that applications may use to achieve the best possible real-time performance at a given service level.

This memo normatively defines the syntax and semantics of the MIDI payload format. However, this memo does not define algorithms for sending and receiving packets. An ancillary document [RFC4696] provides informative guidance on algorithms. Supplemental information may be found in related conference publications [NMP] [GRAME].

Throughout this memo, the phrase "native stream" refers to a stream that uses the rtp-midi media type. The phrase "mpeg4-generic stream" refers to a stream that uses the mpeg4-generic media type (in mode rtp-midi) to operate in an MPEG 4 environment [RFC3640]. Section 6 describes this distinction in detail.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

1.2. Bitfield Conventions

In this document, the packet bitfields that share a common name often have identical semantics. As most of these bitfields appear in Appendices A-B, we define the common bitfield names in Appendix A.1.

However, a few of these common names also appear in the main text of this document. For convenience, we list these definitions below:

- R flag bit. R flag bits are reserved for future use. Senders MUST set R bits to 0. Receivers MUST ignore R bit values.

- LENGTH field. All fields named LENGTH (as distinct from LEN) code the number of octets in the structure that contains it, including the header it resides in and all hierarchical levels below it. If a structure contains a LENGTH field, a receiver MUST use the LENGTH field value to advance past the structure during parsing, rather than use knowledge about the internal format of the structure.

2. Packet Format

In this section, we introduce the format of RTP MIDI packets. The description includes some background information on RTP, for the benefit of MIDI implementors new to IETF tools. Implementors should consult [RFC3550] for an authoritative description of RTP.

This memo assumes that the reader is familiar with MIDI syntax and semantics. Appendix E provides a MIDI overview, at a level of detail sufficient to understand most of this memo. Implementors should consult [MIDI] for an authoritative description of MIDI.

The MIDI payload format maps a MIDI command stream (16 voice channels + systems) onto an RTP stream. An RTP media stream is a sequence of logical packets that share a common format. Each packet consists of two parts: the RTP header and the MIDI payload. Figure 1 shows this format (vertical space delineates the header and payload).

We describe RTP packets as "logical" packets to highlight the fact that RTP itself is not a network-layer protocol. Instead, RTP packets are mapped onto network protocols (such as unicast UDP, multicast UDP, or TCP) by an application [ALF]. The interleaved mode of the Real Time Streaming Protocol (RTSP, [RFC2326]) is an example of an RTP mapping to TCP transport, as is [RFC4571].

2.1. RTP Header

[RFC3550] provides a complete description of the RTP header fields. In this section, we clarify the role of a few RTP header fields for MIDI applications. All fields are coded in network byte order (big-endian).

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | V |P|X|  CC   |M|     PT      |        Sequence number        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                           Timestamp                           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                             SSRC                              |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                     MIDI command section ...                  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                       Journal section ...                     |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1 -- Packet format

The behavior of the 1-bit M field depends on the media type of the stream. For native streams, the M bit MUST be set to 1 if the MIDI command section has a non-zero LEN field, and MUST be set to 0 otherwise. For mpeg4-generic streams, the M bit MUST be set to 1 for all packets (to conform to [RFC3640]).

In an RTP MIDI stream, the 16-bit sequence number field is initialized to a randomly chosen value and is incremented by one (modulo 2^16) for each packet sent in the stream. A related quantity, the 32-bit extended packet sequence number, may be computed by tracking rollovers of the 16-bit sequence number. Note that different receivers of the same stream may compute different extended packet sequence numbers, depending on when the receiver joined the session.

The 32-bit timestamp field sets the base timestamp value for the packet. The payload codes MIDI command timing relative to this value. The timestamp units are set by the clock rate parameter. For example, if the clock rate has a value of 44100 Hz, two packets whose base timestamp values differ by 2 seconds have RTP timestamp fields that differ by 88200.

Note that the clock rate parameter is not encoded within each RTP MIDI packet. A receiver of an RTP MIDI stream becomes aware of the clock rate as part of the session setup process. For example, if a session management tool uses the Session Description Protocol (SDP, [RFC4566]) to describe a media session, the clock rate parameter is set using the rtpmap attribute. We show examples of session setup in Section 6.

For RTP MIDI streams destined to be rendered into audio, the clock rate SHOULD be an audio sample rate of 32 KHz or higher. This recommendation is due to the sensitivity of human musical perception to small timing errors in musical note sequences, and due to the timbral changes that occur when two near-simultaneous MIDI NoteOns are rendered with a different timing than that desired by the content author due to clock rate quantization. RTP MIDI streams that are not destined for audio rendering (such as MIDI streams that control stage lighting) MAY use a lower clock rate but SHOULD use a clock rate high enough to avoid timing artifacts in the application.

For RTP MIDI streams destined to be rendered into audio, the clock rate SHOULD be chosen from rates in common use in professional audio applications or in consumer audio distribution. At the time of this writing, these rates include 32 KHz, 44.1 KHz, 48 KHz, 64 KHz, 88.2 KHz, 96 KHz, 176.4 KHz, and 192 KHz. If the RTP MIDI session is a part of a synchronized media session that includes another (non-MIDI) RTP audio stream with a clock rate of 32 KHz or higher, the RTP MIDI stream SHOULD use a clock rate that matches the clock rate of the other audio stream. However, if the RTP MIDI stream is destined to be rendered into audio, the RTP MIDI stream SHOULD NOT use a clock rate lower than 32 KHz, even if this second stream has a clock rate less than 32 KHz.

Timestamps of consecutive packets do not necessarily increment at a fixed rate, because RTP MIDI packets are not necessarily sent at a fixed rate. The degree of packet transmission regularity reflects the underlying application dynamics. Interactive applications may vary the packet sending rate to track the gestural rate of a human performer, whereas content-streaming applications may send packets at a fixed rate.

Therefore, the timestamps for two sequential RTP packets may be identical, or the second packet may have a timestamp arbitrarily larger than the first packet (modulo 2^32). Section 3 places additional restrictions on the RTP timestamps for two sequential RTP packets, as does the guardtime parameter (Appendix C.4.2).

We use the term "media time" to denote the temporal duration of the media coded by an RTP packet. The media time coded by a packet is computed by subtracting the last command timestamp in the MIDI command section from the RTP timestamp (modulo 2^32). If the MIDI list of the MIDI command section of a packet is empty, the media time coded by the packet is 0 ms. Appendix C.4.1 discusses media time issues in detail.

We now define RTP session semantics, in the context of sessions specified using the session description protocol [RFC4566]. A session description media line ("m=") specifies an RTP session. An RTP session has an independent space of 2^32 synchronization sources. Synchronization source identifiers are coded in the SSRC header field of RTP session packets. The payload types that may appear in the PT header field of RTP session packets are listed at the end of the media line.

Several RTP MIDI streams may appear in an RTP session. Each stream is distinguished by a unique SSRC value and has a unique sequence number and RTP timestamp space. Multiple streams in the RTP session may be sent by a single party. Multiple parties may send streams in the RTP session. An RTP MIDI stream encodes data for a single MIDI command name space (16 voice channels + Systems).

Streams in an RTP session may use different payload types, or they may use the same payload type. However, each party may send, at most, one RTP MIDI stream for each payload type mapped to an RTP MIDI payload format in an RTP session. Recall that dynamic binding of payload type numbers in [RFC4566] lets a party map many payload type numbers to the RTP MIDI payload format; thus a party may send many RTP MIDI streams in a single RTP session. Pairs of streams (unicast or multicast) that communicate between two parties in an RTP session and that share a payload type have the same association as a MIDI cable pair that cross-connects two devices in a MIDI 1.0 DIN network.

The RTP session architecture described above is efficient in its use of network ports, as one RTP session (using a port pair per party) supports the transport of many MIDI name spaces (16 MIDI channels + systems). We define tools for grouping and labelling MIDI name spaces across streams and sessions in Appendix C.5 of this memo.

The RTP header timestamps for each stream in an RTP session have separately and randomly chosen initialization values. Receivers use the timing fields encoded in the RTP control protocol (RTCP, [RFC3550]) sender reports to synchronize the streams sent by a party. The SSRC values for each stream in an RTP session are also separately and randomly chosen, as described in [RFC3550]. Receivers use the CNAME field encoded in RTCP sender reports to verify that streams were sent by the same party, and to detect SSRC collisions, as described in [RFC3550].

In some applications, a receiver renders MIDI commands into audio (or into control actions, such as the rewind of a tape deck or the dimming of stage lights). In other applications, a receiver presents a MIDI stream to software programs via an Application Programmer Interface (API). Appendix C.6 defines session configuration tools to specify what receivers should do with a MIDI command stream.

If a multimedia session uses different RTP MIDI streams to send different classes of media, the streams MUST be sent over different RTP sessions. For example, if a multimedia session uses one MIDI stream for audio and a second MIDI stream to control a lighting system, the audio and lighting streams MUST be sent over different RTP sessions, each with its own media line.

Session description tools defined in Appendix C.5 let a sending party split a single MIDI name space (16 voice channels + systems) over several RTP MIDI streams. Split transport of a MIDI command stream is a delicate task, because correct command stream reconstruction by a receiver depends on exact timing synchronization across the streams.

To support split name spaces, we define the following requirements:

- A party MUST NOT send several RTP MIDI streams that share a MIDI name space in the same RTP session. Instead, each stream MUST be sent from a different RTP session.

- If several RTP MIDI streams sent by a party share a MIDI name space, all streams MUST use the same SSRC value and MUST use the same randomly chosen RTP timestamp initialization value.

These rules let a receiver identify streams that share a MIDI name space (by matching SSRC values) and also let a receiver accurately reconstruct the source MIDI command stream (by using RTP timestamps to interleave commands from the two streams). Care MUST be taken by senders to ensure that SSRC changes due to collisions are reflected in both streams. Receivers MUST regularly examine the RTCP CNAME fields associated with the linked streams, to ensure that the assumed link is legitimate and not the result of an SSRC collision by another sender.

Except for the special cases described above, a party may send many RTP MIDI streams in the same session. However, it is sometimes advantageous for two RTP MIDI streams to be sent over different RTP sessions. For example, two streams may need different values for RTP session-level attributes (such as the sendonly and recvonly attributes). As a second example, two RTP sessions may be needed to send two unicast streams in a multimedia session that originate on different computers (with different IP numbers). Two RTP sessions are needed in this case because transport addresses are specified on the RTP-session or multimedia-session level, not on a payload type level.

On a final note, in some uses of MIDI, parties send bidirectional traffic to conduct transactions (such as file exchange). These commands were designed to work over MIDI 1.0 DIN cable networks may be configured in a multicast topology, which use pure "party-line" signalling. Thus, if a multimedia session ensures a multicast connection between all parties, bidirectional MIDI commands will work without additional support from the RTP MIDI payload format.

2.2. MIDI Payload

The payload (Figure 1) MUST begin with the MIDI command section. The MIDI command section codes a (possibly empty) list of timestamped MIDI commands, and provides the essential service of the payload format.

The payload MAY also contain a journal section. The journal section provides resiliency by coding the recent history of the stream. A flag in the MIDI command section codes the presence of a journal section in the payload.

Section 3 defines the MIDI command section. Sections 4-5 and Appendices A-B define the recovery journal, the default format for the journal section. Here, we describe how these payload sections operate in a stream in an RTP session.

The journalling method for a stream is set at the start of a session and MUST NOT be changed thereafter. A stream may be set to use the recovery journal, to use an alternative journal format (none are defined in this memo), or not to use a journal.

The default journalling method of a stream is inferred from its transport type. Streams that use unreliable transport (such as UDP) default to using the recovery journal. Streams that use reliable transport (such as TCP) default to not using a journal. Appendix C.2.1 defines session configuration tools for overriding these defaults. For all types of transport, a sender MUST transmit an RTP packet stream with consecutive sequence numbers (modulo 2^16).

If a stream uses the recovery journal, every payload in the stream MUST include a journal section. If a stream does not use journalling, a journal section MUST NOT appear in a stream payload. If a stream uses an alternative journal format, the specification for the journal format defines an inclusion policy.

If a stream is sent over UDP transport, the Maximum Transmission Unit (MTU) of the underlying network limits the practical size of the payload section (for example, an Ethernet MTU is 1500 octets), for applications where predictable and minimal packet transmission latency is critical. A sender SHOULD NOT create RTP MIDI UDP packets whose size exceeds the MTU of the underlying network. Instead, the sender SHOULD take steps to keep the maximum packet size under the MTU limit.

These steps may take many forms. The default closed-loop recovery journal sending policy (defined in Appendix C.2.2.2) uses RTP control protocol (RTCP, [RFC3550]) feedback to manage the RTP MIDI packet size. In addition, Section 3.2 and Appendix B.5.2 provide specific tools for managing the size of packets that code MIDI System Exclusive (0xF0) commands. Appendix C.5 defines session configuration tools that may be used to split a dense MIDI name space into several UDP streams (each sent in a different RTP session, per Section 2.1) so that the payload fits comfortably into an MTU. Another option is to use TCP. Section 4.3 of [RFC4696] provides non-normative advice for packet size management.

3. MIDI Command Section

Figure 2 shows the format of the MIDI command section.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|B|J|Z|P|LEN... |  MIDI list ...                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2 -- MIDI command section

The MIDI command section begins with a variable-length header.

The header field LEN codes the number of octets in the MIDI list that follow the header. If the header flag B is 0, the header is one octet long, and LEN is a 4-bit field, supporting a maximum MIDI list length of 15 octets.

If B is 1, the header is two octets long, and LEN is a 12-bit field, supporting a maximum MIDI list length of 4095 octets. LEN is coded in network byte order (big-endian): the 4 bits of LEN that appear in the first header octet code the most significant 4 bits of the 12-bit LEN value.

A LEN value of 0 is legal, and it codes an empty MIDI list.

If the J header bit is set to 1, a journal section MUST appear after the MIDI command section in the payload. If the J header bit is set to 0, the payload MUST NOT contain a journal section.

We define the semantics of the P header bit in Section 3.2.

If the LEN header field is nonzero, the MIDI list has the structure shown in Figure 3.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Delta Time 0     (1-4 octets long, or 0 octets if Z = 1)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  MIDI Command 0   (1 or more octets long)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Delta Time 1     (1-4 octets long)                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  MIDI Command 1   (1 or more octets long)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Delta Time N     (1-4 octets long)                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  MIDI Command N   (0 or more octets long)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3 -- MIDI list structure

If the header flag Z is 1, the MIDI list begins with a complete MIDI command (coded in the MIDI Command 0 field, in Figure 3) preceded by a delta time (coded in the Delta Time 0 field). If Z is 0, the Delta Time 0 field is not present in the MIDI list, and the command coded in the MIDI Command 0 field has an implicit delta time of 0.

The MIDI list structure may also optionally encode a list of N additional complete MIDI commands, each coded in a MIDI Command K field. Each additional command MUST be preceded by a Delta Time K field, which codes the command’s delta time. We discuss exceptions to the "command fields code complete MIDI commands" rule in Section 3.2.

The final MIDI command field (i.e., the MIDI Command N field, shown in Figure 3) in the MIDI list MAY be empty. Moreover, a MIDI list MAY consist a single delta time (encoded in the Delta Time 0 field) without an associated command (which would have been encoded in the MIDI Command 0 field). These rules enable MIDI coding features that are explained in Section 3.1. We delay the explanations because an understanding of RTP MIDI timestamps is necessary to describe the features.

3.1. Timestamps

In this section, we describe how RTP MIDI encodes a timestamp for each MIDI list command. Command timestamps have the same units as RTP packet header timestamps (described in Section 2.1 and [RFC3550]). Recall that RTP timestamps have units of seconds, whose scaling is set during session configuration (see Section 6.1 and [RFC4566]).

As shown in Figure 3, the MIDI list encodes time using a compact delta-time format. The RTP MIDI delta time syntax is a modified form of the MIDI File delta time syntax [MIDI]. RTP MIDI delta times use 1-4 octet fields to encode 32-bit unsigned integers. Figure 4 shows the encoded and decoded forms of delta times. Note that delta time values may be legally encoded in multiple formats; for example, there are four legal ways to encode the zero delta time (0x00, 0x8000, 0x808000, 0x80808000).

RTP MIDI uses delta times to encode a timestamp for each MIDI command. The timestamp for MIDI Command K is the summation (modulo 2^32) of the RTP timestamp and decoded delta times 0 through K. This cumulative coding technique, borrowed from MIDI File delta time coding, is efficient because it reduces the number of multi-octet delta times.

All command timestamps in a packet MUST be less than or equal to the RTP timestamp of the next packet in the stream (modulo 2^32).

This restriction ensures that a particular RTP MIDI packet in a stream is uniquely responsible for encoding time starting at the moment after the RTP timestamp encoded in the RTP packet header, and ending at the moment before the final command timestamp encoded in the MIDI list. The "moment before" and "moment after" qualifiers acknowledge the "less than or equal" semantics (as opposed to "strictly less than") in the sentence above this paragraph.

Note that it is possible to "pad" the end of an RTP MIDI packet with time that is guaranteed to be void of MIDI commands, by setting the "Delta Time N" field of the MIDI list to the end of the void time, and by omitting its corresponding "MIDI Command N" field (a syntactic construction the preamble of Section 3 expressly made legal).

In addition, it is possible to code an RTP MIDI packet to express that a period of time in the stream is void of MIDI commands. The RTP timestamp in the header would code the start of the void time. The MIDI list of this packet would consist of a "Delta Time 0" field that coded the end of the void time. No other fields would be present in the MIDI list (a syntactic construction the preamble of Section 3 also expressly made legal).

By default, a command timestamp indicates the execution time for the command. The difference between two timestamps indicates the time delay between the execution of the commands. This difference may be zero, coding simultaneous execution. In this memo, we refer to this interpretation of timestamps as "comex" (COMmand EXecution) semantics. We formally define comex semantics in Appendix C.3.

The comex interpretation of timestamps works well for transcoding a Standard MIDI File (SMF) into an RTP MIDI stream, as SMFs code a timestamp for each MIDI command stored in the file. To transcode an SMF that uses metric time markers, use the SMF tempo map (encoded in the SMF as meta-events) to convert metric SMF timestamp units into seconds-based RTP timestamp units.

The comex interpretation also works well for MIDI hardware controllers that are coding raw sensor data directly onto an RTP MIDI stream. Note that this controller design is preferable to a design that converts raw sensor data into a MIDI 1.0 cable command stream and then transcodes the stream onto an RTP MIDI stream.

The comex interpretation of timestamps is usually not the best timestamp interpretation for transcoding a MIDI source that uses implicit command timing (such as MIDI 1.0 DIN cables) into an RTP MIDI stream. Appendix C.3 defines alternatives to comex semantics and describes session configuration tools for selecting the timestamp interpretation semantics for a stream.

```
      One-Octet Delta Time:

         Encoded form: 0ddddddd
         Decoded form: 00000000 00000000 00000000 0ddddddd

      Two-Octet Delta Time:

         Encoded form: 1ccccccc 0ddddddd
         Decoded form: 00000000 00000000 00cccccc cddddddd

      Three-Octet Delta Time:

         Encoded form: 1bbbbbbb 1ccccccc 0ddddddd
         Decoded form: 00000000 000bbbbb bbcccccc cddddddd

      Four-Octet Delta Time:

         Encoded form: 1aaaaaaa 1bbbbbbb 1ccccccc 0ddddddd
         Decoded form: 0000aaaa aaabbbbb bbcccccc cddddddd
```

Figure 4 -- Decoding delta time formats

3.2. Command Coding

Each non-empty MIDI Command field in the MIDI list codes one of the MIDI command types that may legally appear on a MIDI 1.0 DIN cable. Standard MIDI File meta-events do not fit this definition and MUST NOT appear in the MIDI list. As a rule, each MIDI Command field codes a complete command, in the binary command format defined in [MIDI]. In the remainder of this section, we describe exceptions to this rule.

The first MIDI channel command in the MIDI list MUST include a status octet. Running status coding, as defined in [MIDI], MAY be used for all subsequent MIDI channel commands in the list. As in [MIDI], System Common and System Exclusive messages (0xF0 ... 0xF7) cancel the running status state, but System Real-time messages (0xF8 ... 0xFF) do not affect the running status state. All System commands in the MIDI list MUST include a status octet.

As we note above, the first channel command in the MIDI list MUST include a status octet. However, the corresponding command in the original MIDI source data stream might not have a status octet (in this case, the source would be coding the command using running status). If the status octet of the first channel command in the MIDI list does not appear in the source data stream, the P (phantom) header bit MUST be set to 1. In all other cases, the P bit MUST be set to 0.

Note that the P bit describes the MIDI source data stream, not the MIDI list encoding; regardless of the state of the P bit, the MIDI list MUST include the status octet.

As receivers MUST be able to decode running status, sender implementors should feel free to use running status to improve bandwidth efficiency. However, senders SHOULD NOT introduce timing jitter into an existing MIDI command stream through an inappropriate use or removal of running status coding. This warning primarily applies to senders whose RTP MIDI streams may be transcoded onto a MIDI 1.0 DIN cable [MIDI] by the receiver: both the timestamps and the command coding (running status or not) must comply with the physical restrictions of implicit time coding over a slow serial line.

On a MIDI 1.0 DIN cable [MIDI], a System Real-time command may be embedded inside of another "host" MIDI command. This syntactic construction is not supported in the payload format: a MIDI Command field in the MIDI list codes exactly one MIDI command (partially or completely).

To encode an embedded System Real-time command, senders MUST extract the command from its host and code it in the MIDI list as a separate command. The host command and System Real-time command SHOULD appear in the same MIDI list. The delta time of the System Real-time command SHOULD result in a command timestamp that encodes the System Real-time command placement in its original embedded position.

Two methods are provided for encoding MIDI System Exclusive (SysEx) commands in the MIDI list. A SysEx command may be encoded in a MIDI Command field verbatim: a 0xF0 octet, followed by an arbitrary number of data octets, followed by a 0xF7 octet.

Alternatively, a SysEx command may be encoded as multiple segments. The command is divided into two or more SysEx command segments; each segment is encoded in its own MIDI Command field in the MIDI list.

The payload format supports segmentation in order to encode SysEx commands that encode information in the temporal pattern of data octets. By encoding these commands as a series of segments, each data octet may be associated with a distinct delta time. Segmentation also supports the coding of large SysEx commands across several packets.

To segment a SysEx command, first partition its data octet list into two or more sublists. The last sublist MAY be empty (i.e., contain no octets); all other sublists MUST contain at least one data octet. To complete the segmentation, add the status octets defined in Figure

5 to the head and tail of the first, last, and any "middle" sublists. Figure 6 shows example segmentations of a SysEx command.

A sender MAY cancel a segmented SysEx command transmission that is in progress, by sending the "cancel" sublist shown in Figure 5. A "cancel" sublist MAY follow a "first" or "middle" sublist in the transmission, but MUST NOT follow a "last" sublist. The cancel MUST be empty (thus, 0xF7 0xF4 is the only legal cancel sublist).

The cancellation feature is needed because Appendix C.1 defines configuration tools that let session parties exclude certain SysEx commands in the stream. Senders that transcode a MIDI source onto an RTP MIDI stream under these constraints have the responsibility of excluding undesired commands from the RTP MIDI stream.

The cancellation feature lets a sender start the transmission of a command before the MIDI source has sent the entire command. If a sender determines that the command whose transmission is in progress should not appear on the RTP stream, it cancels the command. Without a method for cancelling a SysEx command transmission, senders would be forced to use a high-latency store-and-forward approach to transcoding SysEx commands onto RTP MIDI packets, in order to validate each SysEx command before transmission.

The recommended receiver reaction to a cancellation depends on the capabilities of the receiver. For example, a sound synthesizer that is directly parsing RTP MIDI packets and rendering them to audio will be aware of the fact that SysEx commands may be cancelled in RTP MIDI. These receivers SHOULD detect a SysEx cancellation in the MIDI list and act as if they had never received the SysEx command.

As a second example, a synthesizer may be receiving MIDI data from an RTP MIDI stream via a MIDI DIN cable (or a software API emulation of a MIDI DIN cable). In this case, an RTP-MIDI-aware system receives the RTP MIDI stream and transcodes it onto the MIDI DIN cable (or its emulation). Upon the receipt of the cancel sublist, the RTP-MIDI-aware transcoder might have already sent the first part of the SysEx command on the MIDI DIN cable to the receiver.

Unfortunately, the MIDI DIN cable protocol cannot directly code "cancel SysEx in progress" semantics. However, MIDI DIN cable receivers begin SysEx processing after the complete command arrives. The receiver checks to see if it recognizes the command (coded in the first few octets) and then checks to see if the command is the correct length. Thus, in practice, a transcoder can cancel a SysEx command by sending an 0xF7 to (prematurely) end the SysEx command -- the receiver will detect the incorrect command length and discard the command.

Appendix C.1 defines configuration tools that may be used to prohibit SysEx command cancellation.

The relative ordering of SysEx command segments in a MIDI list must match the relative ordering of the sublists in the original SysEx command. By default, commands other than System Real-time MIDI commands MUST NOT appear between SysEx command segments (Appendix C.1 defines configuration tools to change this default, to let other commands types appear between segments). If the command segments of a SysEx command are placed in the MIDI lists of two or more RTP packets, the segment ordering rules apply to the concatenation of all affected MIDI lists.

Sublist Position	Head Status Octet	Tail Status Octet
first	0xF0	0xF0
middle	0xF7	0xF0
last	0xF7	0xF7
cancel	0xF7	0xF4

Figure 5 -- Command segmentation status octets

[MIDI] permits 0xF7 octets that are not part of a (0xF0, 0xF7) pair to appear on a MIDI 1.0 DIN cable. Unpaired 0xF7 octets have no semantic meaning in MIDI, apart from cancelling running status.

Unpaired 0xF7 octets MUST NOT appear in the MIDI list of the MIDI Command section. We impose this restriction to avoid interference with the command segmentation coding defined in Figure 5.

SysEx commands carried on a MIDI 1.0 DIN cable may use the "dropped 0xF7" construction [MIDI]. In this coding method, the 0xF7 octet is dropped from the end of the SysEx command, and the status octet of the next MIDI command acts both to terminate the SysEx command and start the next command. To encode this construction in the payload format, follow these steps:

- Determine the appropriate delta times for the SysEx command and the command that follows the SysEx command.

- Insert the "dropped" 0xF7 octet at the end of the SysEx command, to form the standard SysEx syntax.

- Code both commands into the MIDI list using the rules above.

- Replace the 0xF7 octet that terminates the verbatim SysEx encoding or the last segment of the segmented SysEx encoding with a 0xF5 octet. This substitution informs the receiver of the original dropped 0xF7 coding.

[MIDI] reserves the undefined System Common commands 0xF4 and 0xF5 and the undefined System Real-time commands 0xF9 and 0xFD for future use. By default, undefined commands MUST NOT appear in a MIDI Command field in the MIDI list, with the exception of the 0xF5 octets used to code the "dropped 0xF7" construction and the 0xF4 octets used by SysEx "cancel" sublists.

During session configuration, a stream may be customized to transport undefined commands (Appendix C.1). For this case, we now define how senders encode undefined commands in the MIDI list.

An undefined System Real-time command MUST be coded using the System Real-time rules.

If the undefined System Common commands are put to use in a future version of [MIDI], the command will begin with an 0xF4 or 0xF5 status octet, followed by an arbitrary number of data octets (i.e., zero or more data bytes). To encode these commands, senders MUST terminate the command with an 0xF7 octet and place the modified command into the MIDI Command field.

Unfortunately, non-compliant uses of the undefined System Common commands may appear in MIDI implementations. To model these commands, we assume that the command begins with an 0xF4 or 0xF5 status octet, followed by zero or more data octets, followed by zero or more trailing 0xF7 status octets. To encode the command, senders MUST first remove all trailing 0xF7 status octets from the command. Then, senders MUST terminate the command with an 0xF7 octet and place the modified command into the MIDI Command field.

Note that we include the trailing octets in our model as a cautionary measure: if such commands appeared in a non-compliant use of an undefined System Common command, an RTP MIDI encoding of the command that did not remove trailing octets could be mistaken for an encoding of "middle" or "last" sublist of a segmented SysEx commands (Figure 5) under certain packet loss conditions.

```
 Original SysEx command:

     0xF0 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0xF7

 A two-segment segmentation:

     0xF0 0x01 0x02 0x03 0x04 0xF0

     0xF7 0x05 0x06 0x07 0x08 0xF7

 A different two-segment segmentation:

     0xF0 0x01 0xF0

     0xF7 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0xF7

 A three-segment segmentation:

     0xF0 0x01 0x02 0xF0

     0xF7 0x03 0x04 0xF0

     0xF7 0x05 0x06 0x07 0x08 0xF7

The segmentation with the largest number of segments:

     0xF0 0x01 0xF0

     0xF7 0x02 0xF0

     0xF7 0x03 0xF0

     0xF7 0x04 0xF0

     0xF7 0x05 0xF0

     0xF7 0x06 0xF0

     0xF7 0x07 0xF0

     0xF7 0x08 0xF0

     0xF7 0xF7
```

Figure 6 -- Example segmentations

4. The Recovery Journal System

The recovery journal is the default resiliency tool for unreliable transport. In this section, we normatively define the roles that senders and receivers play in the recovery journal system.

MIDI is a fragile code. A single lost command in a MIDI command stream may produce an artifact in the rendered performance. We normatively classify rendering artifacts into two categories:

- Transient artifacts. Transient artifacts produce immediate but short-term glitches in the performance. For example, a lost NoteOn (0x9) command produces a transient artifact: one note fails to play, but the artifact does not extend beyond the end of that note.

- Indefinite artifacts. Indefinite artifacts produce long-lasting errors in the rendered performance. For example, a lost NoteOff (0x8) command may produce an indefinite artifact: the note that should have been ended by the lost NoteOff command may sustain indefinitely. As a second example, the loss of a Control Change (0xB) command for controller number 7 (Channel Volume) may produce an indefinite artifact: after the loss, all notes on the channel may play too softly or too loudly.

The purpose of the recovery journal system is to satisfy the recovery journal mandate: the MIDI performance rendered from an RTP MIDI stream sent over unreliable transport MUST NOT contain indefinite artifacts.

The recovery journal system does not use packet retransmission to satisfy this mandate. Instead, each packet includes a special section, called the recovery journal.

The recovery journal codes the history of the stream, back to an earlier packet called the checkpoint packet. The range of coverage for the journal is called the checkpoint history. The recovery journal codes the information necessary to recover from the loss of an arbitrary number of packets in the checkpoint history. Appendix A.1 normatively defines the checkpoint packet and the checkpoint history.

When a receiver detects a packet loss, it compares its own knowledge about the history of the stream with the history information coded in the recovery journal of the packet that ends the loss event. By noting the differences in these two versions of the past, a receiver is able to transform all indefinite artifacts in the rendered performance into transient artifacts, by executing MIDI commands to repair the stream.

We now state the normative role for senders in the recovery journal system.

Senders prepare a recovery journal for every packet in the stream. In doing so, senders choose the checkpoint packet identity for the journal. Senders make this choice by applying a sending policy. Appendix C.2.2 normatively defines three sending policies: "closed-loop", "open-loop", and "anchor".

By default, senders MUST use the closed-loop sending policy. If the session description overrides this default policy, by using the parameter j_update defined in Appendix C.2.2, senders MUST use the specified policy.

After choosing the checkpoint packet identity for a packet, the sender creates the recovery journal. By default, this journal MUST conform to the normative semantics in Section 5 and Appendices A-B in this memo. In Appendix C.2.3, we define parameters that modify the normative semantics for recovery journals. If the session description uses these parameters, the journal created by the sender MUST conform to the modified semantics.

Next, we state the normative role for receivers in the recovery journal system.

A receiver MUST detect each RTP sequence number break in a stream. If the sequence number break is due to a packet loss event (as defined in [RFC3550]), the receiver MUST repair all indefinite artifacts in the rendered MIDI performance caused by the loss. If the sequence number break is due to an out-of-order packet (as defined in [RFC3550]), the receiver MUST NOT take actions that introduce indefinite artifacts (ignoring the out-of-order packet is a safe option).

Receivers take special precautions when entering or exiting a session. A receiver MUST process the first received packet in a stream as if it were a packet that ends a loss event. Upon exiting a session, a receiver MUST ensure that the rendered MIDI performance does not end with indefinite artifacts.

Receivers are under no obligation to perform indefinite artifact repairs at the moment a packet arrives. A receiver that uses a playout buffer may choose to wait until the moment of rendering before processing the recovery journal, as the "lost" packet may be a late packet that arrives in time to use.

Next, we state the normative role for the creator of the session description in the recovery journal system. Depending on the application, the sender, the receivers, and other parties may take part in creating or approving the session description.

A session description that specifies the default closed-loop sending policy and the default recovery journal semantics satisfies the recovery journal mandate. However, these default behaviors may not be appropriate for all sessions. If the creators of a session description use the parameters defined in Appendix C.2 to override these defaults, the creators MUST ensure that the parameters define a system that satisfies the recovery journal mandate.

Finally, we note that this memo does not specify sender or receiver recovery journal algorithms. Implementations are free to use any algorithm that conforms to the requirements in this section. The non-normative [RFC4696] discusses sender and receiver algorithm design.

5. Recovery Journal Format

This section introduces the structure of the recovery journal and defines the bitfields of recovery journal headers. Appendices A-B complete the bitfield definition of the recovery journal.

The recovery journal has a three-level structure:

- Top-level header.
- Channel and system journal headers. These headers encode recovery information for a single voice channel (channel journal) or for all systems commands (system journal).
- Chapters. Chapters describe recovery information for a single MIDI command type.

Figure 7 shows the top-level structure of the recovery journal. The recovery journals consists of a 3-octet header, followed by an optional system journal (labeled S-journal in Figure 7) and an optional list of channel journals. Figure 8 shows the recovery journal header format.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Recovery journal header            | S-journal ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Channel journals ...                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7 -- Top-level recovery journal format

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|Y|A|H|TOTCHAN|   Checkpoint Packet Seqnum    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8 -- Recovery journal header

If the Y header bit is set to 1, the system journal appears in the recovery journal, directly following the recovery journal header.

If the A header bit is set to 1, the recovery journal ends with a list of (TOTCHAN + 1) channel journals (the 4-bit TOTCHAN header field is interpreted as an unsigned integer).

A MIDI channel MAY be represented by (at most) one channel journal in a recovery journal. Channel journals MUST appear in the recovery journal in ascending channel-number order.

If A and Y are both zero, the recovery journal only contains its 3-octet header and is considered to be an "empty" journal.

The S (single-packet loss) bit appears in most recovery journal structures, including the recovery journal header. The S bit helps receivers efficiently parse the recovery journal in the common case of the loss of a single packet. Appendix A.1 defines S bit semantics.

The H bit indicates if MIDI channels in the stream have been configured to use the enhanced Chapter C encoding (Appendix A.3.3).

By default, the payload format does not use enhanced Chapter C encoding. In this default case, the H bit MUST be set to 0 for all packets in the stream.

If the stream has been configured so that controller numbers for one or more MIDI channels use enhanced Chapter C encoding, the H bit MUST be set to 1 in all packets in the stream. In Appendix C.2.3, we show how to configure a stream to use enhanced Chapter C encoding.

The 16-bit Checkpoint Packet Seqnum header field codes the sequence number of the checkpoint packet for this journal, in network byte order (big-endian). The choice of the checkpoint packet sets the depth of the checkpoint history for the journal (defined in Appendix A.1).

Receivers may use the Checkpoint Packet Seqnum field of the packet that ends a loss event to verify that the journal checkpoint history covers the entire loss event. The checkpoint history covers the loss event if the Checkpoint Packet Seqnum field is less than or equal to one plus the highest RTP sequence number previously received on the stream (modulo 2^16).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S| CHAN  |H|      LENGTH       |P|C|M|W|N|E|T|A|  Chapters ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9 -- Channel journal format

Figure 9 shows the structure of a channel journal: a 3-octet header, followed by a list of leaf elements called channel chapters. A channel journal encodes information about MIDI commands on the MIDI channel coded by the 4-bit CHAN header field. Note that CHAN uses the same bit encoding as the channel nibble in MIDI Channel Messages (the cccc field in Figure E.1 of Appendix E).

The 10-bit LENGTH field codes the length of the channel journal. The semantics for LENGTH fields are uniform throughout the recovery journal, and are defined in Appendix A.1.

The third octet of the channel journal header is the Table of Contents (TOC) of the channel journal. The TOC is a set of bits that encode the presence of a chapter in the journal. Each chapter contains information about a certain class of MIDI channel command:

- Chapter P: MIDI Program Change (0xC)
- Chapter C: MIDI Control Change (0xB)
- Chapter M: MIDI Parameter System (part of 0xB)
- Chapter W: MIDI Pitch Wheel (0xE)
- Chapter N: MIDI NoteOff (0x8), NoteOn (0x9)
- Chapter E: MIDI Note Command Extras (0x8, 0x9)

- Chapter T: MIDI Channel Aftertouch (0xD)
- Chapter A: MIDI Poly Aftertouch (0xA)

Chapters appear in a list following the header, in order of their appearance in the TOC. Appendices A.2-9 describe the bitfield format for each chapter, and define the conditions under which a chapter type MUST appear in the recovery journal. If any chapter types are required for a channel, an associated channel journal MUST appear in the recovery journal.

The H bit indicates if controller numbers on a MIDI channel have been configured to use the enhanced Chapter C encoding (Appendix A.3.3).

By default, controller numbers on a MIDI channel do not use enhanced Chapter C encoding. In this default case, the H bit MUST be set to 0 for all channel journal headers for the channel in the recovery journal, for all packets in the stream.

However, if at least one controller number for a MIDI channel has been configured to use the enhanced Chapter C encoding, the H bit for its channel journal MUST be set to 1, for all packets in the stream.

In Appendix C.2.3, we show how to configure a controller number to use enhanced Chapter C encoding.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|D|V|Q|F|X|      LENGTH       |  System chapters ...          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10 -- System journal format

Figure 10 shows the structure of the system journal: a 2-octet header, followed by a list of system chapters. Each chapter codes information about a specific class of MIDI Systems command:

- Chapter D: Song Select (0xF3), Tune Request (0xF6), Reset (0xFF), undefined System commands (0xF4, 0xF5, 0xF9, 0xFD)
- Chapter V: Active Sense (0xFE)
- Chapter Q: Sequencer State (0xF2, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC)
- Chapter F: MTC Tape Position (0xF1, 0xF0 0x7F 0xcc 0x01 0x01)
- Chapter X: System Exclusive (all other 0xF0)

The 10-bit LENGTH field codes the size of the system journal and conforms to semantics described in Appendix A.1.

The D, V, Q, F, and X header bits form a Table of Contents (TOC) for the system journal. A TOC bit that is set to 1 codes the presence of a chapter in the journal. Chapters appear in a list following the header, in the order of their appearance in the TOC.

Appendix B describes the bitfield format for the system chapters and defines the conditions under which a chapter type MUST appear in the recovery journal. If any system chapter type is required to appear in the recovery journal, the system journal MUST appear in the recovery journal.

6. Session Description Protocol

RTP does not perform session management. Instead, RTP works together with session management tools, such as the Session Initiation Protocol (SIP, [RFC3261]) and the Real Time Streaming Protocol (RTSP, [RFC2326]).

RTP payload formats define media type parameters for use in session management (for example, this memo defines "rtp-midi" as the media type for native RTP MIDI streams).

In most cases, session management tools use the media type parameters via another standard, the Session Description Protocol (SDP, [RFC4566]).

SDP is a textual format for specifying session descriptions. Session descriptions specify the network transport and media encoding for RTP sessions. Session management tools coordinate the exchange of session descriptions between participants ("parties").

Some session management tools use SDP to negotiate details of media transport (network addresses, ports, etc.). We refer to this use of SDP as "negotiated usage". One example of negotiated usage is the Offer/Answer protocol ([RFC3264] and Appendix C.7.2 in this memo) as used by SIP.

Other session management tools use SDP to declare the media encoding for the session but use other techniques to negotiate network transport. We refer to this use of SDP as "declarative usage". One example of declarative usage is RTSP ([RFC2326] and Appendix C.7.1 in this memo).

Below, we show session description examples for native (Section 6.1) and mpeg4-generic (Section 6.2) streams. In Section 6.3, we introduce session configuration tools that may be used to customize streams.

6.1. Session Descriptions for Native Streams

The session description below defines a unicast UDP RTP session (via a media ("m=") line) whose sole payload type (96) is mapped to a minimal native RTP MIDI stream.

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP4 192.0.2.94
a=rtpmap:96 rtp-midi/44100
```

The rtpmap attribute line uses the "rtp-midi" media type to specify an RTP MIDI native stream. The clock rate specified on the rtpmap line (in the example above, 44100 Hz) sets the scaling for the RTP timestamp header field (see Section 2.1, and also [RFC3550]).

Note that this document does not specify a default clock rate value for RTP MIDI. When RTP MIDI is used with SDP, parties MUST use the rtpmap line to communicate the clock rate. Guidance for selecting the RTP MIDI clock rate value appears in Section 2.1.

We consider the RTP MIDI stream shown above to be "minimal" because the session description does not customize the stream with parameters. Without such customization, a native RTP MIDI stream has these characteristics:

1. If the stream uses unreliable transport (unicast UDP, multicast UDP, etc.), the recovery journal system is in use, and the RTP payload contains both the MIDI command section and the journal section. If the stream uses reliable transport (such as TCP), the stream does not use journalling, and the payload contains only the MIDI command section (Section 2.2).

2. If the stream uses the recovery journal system, the recovery journal system uses the default sending policy and the default journal semantics (Section 4).

3. In the MIDI command section of the payload, command timestamps use the default "comex" semantics (Section 3).

4. The recommended temporal duration ("media time") of an RTP packet ranges from 0 to 200 ms, and the RTP timestamp difference between sequential packets in the stream may be arbitrarily large (Section 2.1).

5. If more than one minimal rtp-midi stream appears in a session, the MIDI name spaces for these streams are independent: channel 1 in the first stream does not reference the same MIDI channel as channel 1 in the second stream (see Appendix C.5 for a discussion of the independence of minimal rtp-midi streams).

6. The rendering method for the stream is not specified. What the receiver "does" with a minimal native MIDI stream is "out of scope" of this memo. For example, in content creation environments, a user may manually configure client software to render the stream with a specific software package.

As in standard in RTP, RTP sessions managed by SIP are sendrecv by default (parties send and receive MIDI), and RTP sessions managed by RTSP are recvonly by default (server sends and client receives).

In sendrecv RTP MIDI sessions for the session description shown above, the 16 voice channel + systems MIDI name space is unique for each sender. Thus, in a two-party session, the voice channel 0 sent by one party is distinct from the voice channel 0 sent by the other party.

This behavior corresponds to what occurs when two MIDI 1.0 DIN devices are cross-connected with two MIDI cables (one cable routing MIDI Out from the first device into MIDI In of the second device, a second cable routing MIDI In from the first device into MIDI Out of the second device). We define this "association" formally in Section 2.1.

MIDI 1.0 DIN networks may be configured in a "party-line" multicast topology. For these networks, the MIDI protocol itself provides tools for addressing specific devices in transactions on a multicast network, and for device discovery. Thus, apart from providing a 1-to-many forward path and a many-to-1 reverse path, IETF protocols do not need to provide any special support for MIDI multicast networking.

6.2. Session Descriptions for mpeg4-generic Streams

An mpeg4-generic [RFC3640] RTP MIDI stream uses an MPEG 4 Audio Object Type to render MIDI into audio. Three Audio Object Types accept MIDI input:

- General MIDI (Audio Object Type ID 15), based on the General MIDI rendering standard [MIDI].

- Wavetable Synthesis (Audio Object Type ID 14), based on the Downloadable Sounds Level 2 (DLS 2) rendering standard [DLS2].

- Main Synthetic (Audio Object Type ID 13), based on Structured Audio and the programming language SAOL [MPEGSA].

The primary service of an mpeg4-generic stream is to code Access Units (AUs). We define the mpeg4-generic RTP MIDI AU as the MIDI payload shown in Figure 1 of Section 2.1 of this memo: a MIDI command section optionally followed by a journal section.

Exactly one RTP MIDI AU MUST be mapped to one mpeg4-generic RTP MIDI packet. The mpeg4-generic options for placing several AUs in an RTP packet MUST NOT be used with RTP MIDI. The mpeg4-generic options for fragmenting and interleaving AUs MUST NOT be used with RTP MIDI. The mpeg4-generic RTP packet payload (Figure 1 in [RFC3640]) MUST contain empty AU Header and Auxiliary sections. These rules yield mpeg4-generic packets that are structurally identical to native RTP MIDI packets, an essential property for the correct operation of the payload format.

The session description that follows defines a unicast UDP RTP session (via a media ("m=") line) whose sole payload type (96) is mapped to a minimal mpeg4-generic RTP MIDI stream. This example uses the General MIDI Audio Object Type under Synthesis Profile @ Level 2.

```
v=0
o=lazzaro 2520644554 2838152170 IN IP6 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP6 2001:DB80::7F2E:172A:1E24
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; profile-level-id=12;
config=7A0A0000001A4D546864000000060000000100604D54726B0000
000600FF2F000
```

(The a=fmtp line has been wrapped to fit the page to accommodate memo formatting restrictions; it comprises a single line in SDP.)

The fmtp attribute line codes the four parameters (streamtype, mode, profile-level-id, and config) that are required in all mpeg4-generic session descriptions [RFC3640]. For RTP MIDI streams, the streamtype parameter MUST be set to 5, the "mode" parameter MUST be set to "rtp-midi", and the "profile-level-id" parameter MUST be set to the MPEG-4 Profile Level for the stream. For the Synthesis Profile, legal profile-level-id values are 11, 12, and 13, coding low (11), medium (12), or high (13) decoder computational complexity, as defined by MPEG conformance tests.

In a minimal RTP MIDI session description, the config value MUST be a hexadecimal encoding [RFC3640] of the AudioSpecificConfig data block [MPEGAUDIO] for the stream. AudioSpecificConfig encodes the Audio Object Type for the stream and also encodes initialization data (SAOL programs, DLS 2 wave tables, etc.). Standard MIDI Files encoded in AudioSpecificConfig in a minimal session description MUST be ignored by the receiver.

Receivers determine the rendering algorithm for the session by interpreting the first 5 bits of AudioSpecificConfig as an unsigned integer that codes the Audio Object Type. In our example above, the leading config string nibbles "7A" yield the Audio Object Type 15 (General MIDI). In Appendix E.4, we derive the config string value in the session description shown above; the starting point of the derivation is the MPEG bitstreams defined in [MPEGSA] and [MPEGAUDIO].

We consider the stream to be "minimal" because the session description does not customize the stream through the use of parameters, other than the 4 required mpeg4-generic parameters described above. In Section 6.1, we describe the behavior of a minimal native stream, as a numbered list of characteristics. Items 1-4 on that list also describe the minimal mpeg4-generic stream, but items 5 and 6 require restatements, as listed below:

5. If more than one minimal mpeg4-generic stream appears in a session, each stream uses an independent instance of the Audio Object Type coded in the config parameter value.

6. A minimal mpeg4-generic stream encodes the AudioSpecificConfig as an inline hexadecimal constant. If a session description is sent over UDP, it may be impossible to transport large AudioSpecificConfig blocks within the Maximum Transmission Size (MTU) of the underlying network (for Ethernet, the MTU is 1500 octets). In some cases, the AudioSpecificConfig block may exceed the maximum size of the UDP packet itself.

The comments in Section 6.1 on SIP and RTSP stream directional defaults, sendrecv MIDI channel usage, and MIDI 1.0 DIN multicast networks also apply to mpeg4-generic RTP MIDI sessions.

In sendrecv sessions, each party’s session description MUST use identical values for the mpeg4-generic parameters (including the required streamtype, mode, profile-level-id, and config parameters). As a consequence, each party uses an identically configured MPEG 4 Audio Object Type to render MIDI commands into audio. The preamble to Appendix C discusses a way to create "virtual sendrecv" sessions that do not have this restriction.

6.3. Parameters

This section introduces parameters for session configuration for RTP MIDI streams. In session descriptions, parameters modify the semantics of a payload type. Parameters are specified on an fmtp attribute line. See the session description example in Section 6.2 for an example of a fmtp attribute line.

The parameters add features to the minimal streams described in Sections 6.1-2, and support several types of services:

- Stream subsetting. By default, all MIDI commands that are legal to appear on a MIDI 1.0 DIN cable may appear in an RTP MIDI stream. The cm_unused parameter overrides this default by prohibiting certain commands from appearing in the stream. The cm_used parameter is used in conjunction with cm_unused, to simplify the specification of complex exclusion rules. We describe cm_unused and cm_used in Appendix C.1.

- Journal customization. The j_sec and j_update parameters configure the use of the journal section. The ch_default, ch_never, and ch_anchor parameters configure the semantics of the recovery journal chapters. These parameters are described in Appendix C.2 and override the default stream behaviors 1 and 2, listed in Section 6.1 and referenced in Section 6.2.

- MIDI command timestamp semantics. The tsmode, octpos, mperiod, and linerate parameters customize the semantics of timestamps in the MIDI command section. These parameters let RTP MIDI accurately encode the implicit time coding of MIDI 1.0 DIN cables. These parameters are described in Appendix C.3 and override default stream behavior 3, listed in Section 6.1 and referenced in Section 6.2

- Media time. The rtp_ptime and rtp_maxptime parameters define the temporal duration ("media time") of an RTP MIDI packet. The guardtime parameter sets the minimum sending rate of stream packets. These parameters are described in Appendix C.4 and override default stream behavior 4, listed in Section 6.1 and referenced in Section 6.2.

- Stream description. The musicport parameter labels the MIDI name space of RTP streams in a multimedia session. Musicport is described in Appendix C.5. The musicport parameter overrides default stream behavior 5, in Sections 6.1 and 6.2.

- MIDI rendering. Several parameters specify the MIDI rendering method of a stream. These parameters are described in Appendix C.6 and override default stream behavior 6, in Sections 6.1 and 6.2.

In Appendix C.7, we specify interoperability guidelines for two RTP MIDI application areas: content-streaming using RTSP (Appendix C.7.1) and network musical performance using SIP (Appendix C.7.2).

7. Extensibility

The payload format defined in this memo exclusively encodes all commands that may legally appear on a MIDI 1.0 DIN cable.

Many worthy uses of MIDI over RTP do not fall within the narrow scope of the payload format. For example, the payload format does not support the direct transport of Standard MIDI File (SMF) meta-event and metric timing data. As a second example, the payload format does not define transport tools for user-defined commands (apart from tools to support System Exclusive commands [MIDI]).

The payload format does not provide an extension mechanism to support new features of this nature, by design. Instead, we encourage the development of new payload formats for specialized musical applications. The IETF session management tools [RFC3264] [RFC2326] support codec negotiation, to facilitate the use of new payload formats in a backward-compatible way.

However, the payload format does provide several extensibility tools, which we list below:

- Journalling. As described in Appendix C.2, new token values for the j_sec and j_update parameters may be defined in IETF standards-track documents. This mechanism supports the design of new journal formats and the definition of new journal sending policies.

- Rendering. The payload format may be extended to support new MIDI renderers (Appendix C.6.2). Certain general aspects of the RTP MIDI rendering process may also be extended, via the definition of new token values for the render (Appendix C.6) and smf_info (Appendix C.6.4.1) parameters.

- Undefined commands. [MIDI] reserves 4 MIDI System commands for future use (0xF4, 0xF5, 0xF9, 0xFD). If updates to [MIDI] define the reserved commands, IETF standards-track documents may be defined to provide resiliency support for the commands.

Opaque LEGAL fields appear in System Chapter D for this purpose (Appendix B.1.1).

A final form of extensibility involves the inclusion of the payload format in framework documents. Framework documents describe how to combine protocols to form a platform for interoperable applications. For example, a stage and studio framework might define how to use SIP [RFC3261], RTSP [RFC2326], SDP [RFC4566], and RTP [RFC3550] to support media networking for professional audio equipment and electronic musical instruments.

8. Congestion Control

The RTP congestion control requirements defined in [RFC3550] apply to RTP MIDI sessions, and implementors should carefully read the congestion control section in [RFC3550]. As noted in [RFC3550], all transport protocols used on the Internet need to address congestion control in some way, and RTP is not an exception.

In addition, the congestion control requirements defined in [RFC3551] applies to RTP MIDI sessions run under applicable profiles. The basic congestion control requirement defined in [RFC3551] is that RTP sessions that use UDP transport should monitor packet loss (via RTCP or other means) to ensure that the RTP stream competes fairly with TCP flows that share the network.

Finally, RTP MIDI has congestion control issues that are unique for an audio RTP payload format. In applications such as network musical performance [NMP], the packet rate is linked to the gestural rate of a human performer. Senders MUST monitor the MIDI command source for patterns that result in excessive packet rates and take actions during RTP transcoding to reduce the RTP packet rate. [RFC4696] offers implementation guidance on this issue.

9. Security Considerations

Implementors should carefully read the Security Considerations sections of the RTP [RFC3550], AVP [RFC3551], and other RTP profile documents, as the issues discussed in these sections directly apply to RTP MIDI streams. Implementors should also review the Secure Real-time Transport Protocol (SRTP, [RFC3711]), an RTP profile that addresses the security issues discussed in [RFC3550] and [RFC3551].

Here, we discuss security issues that are unique to the RTP MIDI payload format.

When using RTP MIDI, authentication of incoming RTP and RTCP packets is RECOMMENDED. Per-packet authentication may be provided by SRTP or by other means. Without the use of authentication, attackers could forge MIDI commands into an ongoing stream, damaging speakers and eardrums. An attacker could also craft RTP and RTCP packets to exploit known bugs in the client and take effective control of a client machine.

Session management tools (such as SIP [RFC3261]) SHOULD use authentication during the transport of all session descriptions containing RTP MIDI media streams. For SIP, the Security Considerations section in [RFC3261] provides an overview of possible authentication mechanisms. RTP MIDI session descriptions should use authentication because the session descriptions may code initialization data using the parameters described in Appendix C. If an attacker inserts bogus initialization data into a session description, he can corrupt the session or forge an client attack.

Session descriptions may also code renderer initialization data by reference, via the url (Appendix C.6.3) and smf_url (Appendix C.6.4.2) parameters. If the coded URL is spoofed, both session and client are open to attack, even if the session description itself is authenticated. Therefore, URLs specified in url and smf_url parameters SHOULD use [RFC2818].

Section 2.1 allows streams sent by a party in two RTP sessions to have the same SSRC value and the same RTP timestamp initialization value, under certain circumstances. Normally, these values are randomly chosen for each stream in a session, to make plaintext guessing harder to do if the payloads are encrypted. Thus, Section 2.1 weakens this aspect of RTP security.

10. Acknowledgements

We thank the networking, media compression, and computer music community members who have commented or contributed to the effort, including Kurt B, Cynthia Bruyns, Steve Casner, Paul Davis, Robin Davies, Joanne Dow, Tobias Erichsen, Nicolas Falquet, Dominique Fober, Philippe Gentric, Michael Godfrey, Chris Grigg, Todd Hager, Michel Jullian, Phil Kerr, Young-Kwon Lim, Jessica Little, Jan van der Meer, Colin Perkins, Charlie Richmond, Herbie Robinson, Larry Rowe, Eric Scheirer, Dave Singer, Martijn Sipkema, William Stewart, Kent Terry, Magnus Westerlund, Tom White, Jim Wright, Doug Wyatt, and Giorgio Zoia. We also thank the members of the San Francisco Bay Area music and audio community for creating the context for the work, including Don Buchla, Chris Chafe, Richard Duda, Dan Ellis, Adrian Freed, Ben Gold, Jaron Lanier, Roger Linn, Richard Lyon, Dana Massie, Max Mathews, Keith McMillen, Carver Mead, Nelson Morgan, Tom Oberheim, Malcolm Slaney, Dave Smith, Julius Smith, David Wessel, and Matt Wright.

11. IANA Considerations

This section makes a series of requests to IANA. The IANA has completed registration/assignments of the below requests.

The sub-sections that follow hold the actual, detailed requests. All registrations in this section are in the IETF tree and follow the rules of [RFC4288] and [RFC3555], as appropriate.

In Section 11.1, we request the registration of a new media type: "audio/rtp-midi". Paired with this request is a request for a repository for new values for several parameters associated with "audio/rtp-midi". We request this repository in Section 11.1.1.

In Section 11.2, we request the registration of a new value ("rtp-midi") for the "mode" parameter of the "mpeg4-generic" media type. The "mpeg4-generic" media type is defined in [RFC3640], and [RFC3640] defines a repository for the "mode" parameter. However, we believe we are the first to request the registration of a "mode" value, so we believe the registry for "mode" has not yet been created by IANA.

Paired with our "mode" parameter value request for "mpeg4-generic" is a request for a repository for new values for several parameters we have defined for use with the "rtp-midi" mode value. We request this repository in Section 11.2.1.

In Section 11.3, we request the registration of a new media type: "audio/asc". No repository request is associated with this request.

11.1. rtp-midi Media Type Registration

This section requests the registration of the "rtp-midi" subtype for the "audio" media type. We request the registration of the parameters listed in the "optional parameters" section below (both the "non-extensible parameters" and the "extensible parameters" lists). We also request the creation of repositories for the "extensible parameters"; the details of this request appear in Section 11.1.1, below.

Media type name:

audio

Subtype name:

rtp-midi

Required parameters:

rate: The RTP timestamp clock rate. See Sections 2.1 and 6.1 for usage details.

Optional parameters:

Non-extensible parameters:

ch_anchor: See Appendix C.2.3 for usage details.
ch_default: See Appendix C.2.3 for usage details.
ch_never: See Appendix C.2.3 for usage details.
cm_unused: See Appendix C.1 for usage details.
cm_used: See Appendix C.1 for usage details.
chanmask: See Appendix C.6.4.3 for usage details.
cid: See Appendix C.6.3 for usage details.
guardtime: See Appendix C.4.2 for usage details.
inline: See Appendix C.6.3 for usage details.
linerate: See Appendix C.3 for usage details.
mperiod: See Appendix C.3 for usage details.
multimode: See Appendix C.6.1 for usage details.
musicport: See Appendix C.5 for usage details.
octpos: See Appendix C.3 for usage details.
rinit: See Appendix C.6.3 for usage details.
rtp_maxptime: See Appendix C.4.1 for usage details.
rtp_ptime: See Appendix C.4.1 for usage details.
smf_cid: See Appendix C.6.4.2 for usage details.
smf_inline: See Appendix C.6.4.2 for usage details.
smf_url: See Appendix C.6.4.2 for usage details.
tsmode: See Appendix C.3 for usage details.
url: See Appendix C.6.3 for usage details.

Extensible parameters:

j_sec: See Appendix C.2.1 for usage details. See Section 11.1.1 for repository details.
j_update: See Appendix C.2.2 for usage details. See Section 11.1.1 for repository details.
render: See Appendix C.6 for usage details. See Section 11.1.1 for repository details.
subrender: See Appendix C.6.2 for usage details. See Section 11.1.1 for repository details.
smf_info: See Appendix C.6.4.1 for usage details. See Section 11.1.1 for repository details.

Encoding considerations:

The format for this type is framed and binary.

Restrictions on usage:

This type is only defined for real-time transfers of MIDI streams via RTP. Stored-file semantics for rtp-midi may be defined in the future.

Security considerations:

See Section 9 of this memo.

Interoperability considerations:

None.

Published specification:

This memo and [MIDI] serve as the normative specification. In addition, references [NMP], [GRAME], and [RFC4696] provide non-normative implementation guidance.

Applications that use this media type:

Audio content-creation hardware, such as MIDI controller piano keyboards and MIDI audio synthesizers. Audio content-creation software, such as music sequencers, digital audio workstations, and soft synthesizers. Computer operating systems, for network support of MIDI Application Programmer Interfaces. Content distribution servers and terminals may use this media type for low bit-rate music coding.

Additional information:

None.

Person & email address to contact for further information:

John Lazzaro <lazzaro@cs.berkeley.edu>

Intended usage:

COMMON.

Author:

John Lazzaro <lazzaro@cs.berkeley.edu>

Change controller:

IETF Audio/Video Transport Working Group delegated from the IESG.

11.1.1. Repository Request for "audio/rtp-midi"

For the "rtp-midi" subtype, we request the creation of repositories for extensions to the following parameters (which are those listed as "extensible parameters" in Section 11.1).

j_sec:

Registrations for this repository may only occur via an IETF standards-track document. Appendix C.2.1 of this memo describes appropriate registrations for this repository.

Initial values for this repository appear below:

"none": Defined in Appendix C.2.1 of this memo.
"recj": Defined in Appendix C.2.1 of this memo.

j_update:

Registrations for this repository may only occur via an IETF standards-track document. Appendix C.2.2 of this memo describes appropriate registrations for this repository.

Initial values for this repository appear below:

"anchor": Defined in Appendix C.2.2 of this memo.
"open-loop": Defined in Appendix C.2.2 of this memo.
"closed-loop": Defined in Appendix C.2.2 of this memo.

render:

Registrations for this repository MUST include a specification of the usage of the proposed value. See text in the preamble of Appendix C.6 for details (the paragraph that begins "Other render token ...").

Initial values for this repository appear below:

"unknown": Defined in Appendix C.6 of this memo.
"synthetic": Defined in Appendix C.6 of this memo.
"api": Defined in Appendix C.6 of this memo.
"null": Defined in Appendix C.6 of this memo.

subrender:

Registrations for this repository MUST include a specification of the usage of the proposed value. See text Appendix C.6.2 for details (the paragraph that begins "Other subrender token ...").

Initial values for this repository appear below:

"default": Defined in Appendix C.6.2 of this memo.

smf_info:

Registrations for this repository MUST include a specification of the usage of the proposed value. See text in Appendix C.6.4.1 for details (the paragraph that begins "Other smf_info token ...").

Initial values for this repository appear below:

"ignore": Defined in Appendix C.6.4.1 of this memo.
"sdp_start": Defined in Appendix C.6.4.1 of this memo.
"identity": Defined in Appendix C.6.4.1 of this memo.

11.2. mpeg4-generic Media Type Registration

This section requests the registration of the "rtp-midi" value for the "mode" parameter of the "mpeg4-generic" media type. The "mpeg4-generic" media type is defined in [RFC3640], and [RFC3640] defines a repository for the "mode" parameter. We are registering mode rtp-midi to support the MPEG Audio codecs [MPEGSA] that use MIDI.

In conjunction with this registration request, we request the registration of the parameters listed in the "optional parameters" section below (both the "non-extensible parameters" and the "extensible parameters" lists). We also request the creation of repositories for the "extensible parameters"; the details of this request appear in Appendix 11.2.1, below.

Media type name:

audio

Subtype name:

mpeg4-generic

Required parameters:

The "mode" parameter is required by [RFC3640]. [RFC3640] requests a repository for "mode", so that new values for mode may be added. We request that the value "rtp-midi" be added to the "mode" repository.

In mode rtp-midi, the mpeg4-generic parameter rate is a required parameter. Rate specifies the RTP timestamp clock rate. See Sections 2.1 and 6.2 for usage details of rate in mode rtp-midi.

Optional parameters:

We request registration of the following parameters for use in mode rtp-midi for mpeg4-generic.

Non-extensible parameters:

ch_anchor: See Appendix C.2.3 for usage details.
ch_default: See Appendix C.2.3 for usage details.
ch_never: See Appendix C.2.3 for usage details.
cm_unused: See Appendix C.1 for usage details.
cm_used: See Appendix C.1 for usage details.
chanmask: See Appendix C.6.4.3 for usage details.
cid: See Appendix C.6.3 for usage details.
guardtime: See Appendix C.4.2 for usage details.
inline: See Appendix C.6.3 for usage details.
linerate: See Appendix C.3 for usage details.
mperiod: See Appendix C.3 for usage details.
multimode: See Appendix C.6.1 for usage details.
musicport: See Appendix C.5 for usage details.
octpos: See Appendix C.3 for usage details.
rinit: See Appendix C.6.3 for usage details.
rtp_maxptime: See Appendix C.4.1 for usage details.
rtp_ptime: See Appendix C.4.1 for usage details.
smf_cid: See Appendix C.6.4.2 for usage details.
smf_inline: See Appendix C.6.4.2 for usage details.

smf_url: See Appendix C.6.4.2 for usage details.
tsmode: See Appendix C.3 for usage details.
url: See Appendix C.6.3 for usage details.

Extensible parameters:

j_sec: See Appendix C.2.1 for usage details. See Section 11.2.1 for repository details.
j_update: See Appendix C.2.2 for usage details. See Section 11.2.1 for repository details.
render: See Appendix C.6 for usage details. See Section 11.2.1 for repository details.
subrender: See Appendix C.6.2 for usage details. See Section 11.2.1 for repository details.
smf_info: See Appendix C.6.4.1 for usage details. See Section 11.2.1 for repository details.

Encoding considerations:

The format for this type is framed and binary.

Restrictions on usage:

Only defined for real-time transfers of audio/mpeg4-generic RTP streams with mode=rtp-midi.

Security considerations:

See Section 9 of this memo.

Interoperability considerations:

Except for the marker bit (Section 2.1), the packet formats for audio/rtp-midi and audio/mpeg4-generic (mode rtp-midi) are identical. The formats differ in use: audio/mpeg4-generic is for MPEG work, and audio/rtp-midi is for all other work.

Published specification:

This memo, [MIDI], and [MPEGSA] are the normative references. In addition, references [NMP], [GRAME], and [RFC4696] provide non-normative implementation guidance.

Applications that use this media type:

MPEG 4 servers and terminals that support [MPEGSA].

Additional information:

None.

Person & email address to contact for further information:

John Lazzaro <lazzaro@cs.berkeley.edu>

Intended usage:

COMMON.

Author:

John Lazzaro <lazzaro@cs.berkeley.edu>

Change controller:

IETF Audio/Video Transport Working Group delegated
from the IESG.

11.2.1. Repository Request for Mode rtp-midi for mpeg4-generic

For mode rtp-midi of the mpeg4-generic subtype, we request the creation of repositories for extensions to the following parameters (which are those listed as "extensible parameters" in Section 11.2).

j_sec:

Registrations for this repository may only occur via an IETF standards-track document. Appendix C.2.1 of this memo describes appropriate registrations for this repository.

Initial values for this repository appear below:

"none": Defined in Appendix C.2.1 of this memo.
"recj": Defined in Appendix C.2.1 of this memo.

j_update:

Registrations for this repository may only occur via an IETF standards-track document. Appendix C.2.2 of this memo describes appropriate registrations for this repository.

Initial values for this repository appear below:

"anchor": Defined in Appendix C.2.2 of this memo.
"open-loop": Defined in Appendix C.2.2 of this memo.
"closed-loop": Defined in Appendix C.2.2 of this memo.

render:

Registrations for this repository MUST include a specification of the usage of the proposed value. See text in the preamble of Appendix C.6 for details (the paragraph that begins "Other render token ...").

Initial values for this repository appear below:

"unknown": Defined in Appendix C.6 of this memo.
"synthetic": Defined in Appendix C.6 of this memo.
"null": Defined in Appendix C.6 of this memo.

subrender:

Registrations for this repository MUST include a specification of the usage of the proposed value. See text Appendix C.6.2 for details (the paragraph that begins "Other subrender token ..." and subsequent paragraphs). Note that the text in Appendix C.6.2 contains restrictions on subrender registrations for mpeg4-generic ("Registrations for mpeg4-generic subrender values ...").

Initial values for this repository appear below:

"default": Defined in Appendix C.6.2 of this memo.

smf_info:

Registrations for this repository MUST include a specification of the usage of the proposed value. See text in Appendix C.6.4.1 for details (the paragraph that begins "Other smf_info token ...").

Initial values for this repository appear below:

"ignore": Defined in Appendix C.6.4.1 of this memo.
"sdp_start": Defined in Appendix C.6.4.1 of this memo.
"identity": Defined in Appendix C.6.4.1 of this memo.

11.3. asc Media Type Registration

This section registers "asc" as a subtype for the "audio" media type. We register this subtype to support the remote transfer of the "config" parameter of the mpeg4-generic media type [RFC3640] when it is used with mpeg4-generic mode rtp-midi (registered in Appendix 11.2 above). We explain the mechanics of using "audio/asc" to set the config parameter in Section 6.2 and Appendix C.6.5 of this document.

Note that this registration is a new subtype registration and is not an addition to a repository defined by MPEG-related memos (such as [RFC3640]). Also note that this request for "audio/asc" does not register parameters, and does not request the creation of a repository.

Media type name:

audio

Subtype name:

asc

Required parameters:

None.

Optional parameters:

None.

Encoding considerations:

The native form of the data object is binary data, zero-padded to an octet boundary.

Restrictions on usage:

This type is only defined for data object (stored file) transfer. The most common transports for the type are HTTP and SMTP.

Security considerations:

See Section 9 of this memo.

Interoperability considerations:

None.

Published specification:

The audio/asc data object is the AudioSpecificConfig binary data structure, which is normatively defined in [MPEGAUDIO].

Applications that use this media type:

MPEG 4 Audio servers and terminals that support audio/mpeg4-generic RTP streams for mode rtp-midi.

Additional information:

None.

Person & email address to contact for further information:

John Lazzaro <lazzaro@cs.berkeley.edu>

Intended usage:

COMMON.

Author:

John Lazzaro <lazzaro@cs.berkeley.edu>

Change controller:

IETF Audio/Video Transport Working Group delegated from the IESG.

A. The Recovery Journal Channel Chapters

A.1. Recovery Journal Definitions

This appendix defines the terminology and the coding idioms that are used in the recovery journal bitfield descriptions in Section 5 (journal header structure), Appendices A.2 to A.9 (channel journal chapters) and Appendices B.1 to B.5 (system journal chapters).

We assume that the recovery journal resides in the journal section of an RTP packet with sequence number I ("packet I") and that the Checkpoint Packet Seqnum field in the top-level recovery journal header refers to a previous packet with sequence number C (an exception is the self-referential C = I case). Unless stated otherwise, algorithms are assumed to use modulo 2^16 arithmetic for calculations on 16-bit sequence numbers and modulo 2^32 arithmetic for calculations on 32-bit extended sequence numbers.

Several bitfield coding idioms appear throughout the recovery journal system, with consistent semantics. Most recovery journal elements begin with an "S" (Single-packet loss) bit. S bits are designed to help receivers efficiently parse through the recovery journal hierarchy in the common case of the loss of a single packet.

As a rule, S bits MUST be set to 1. However, an exception applies if a recovery journal element in packet I encodes data about a command stored in the MIDI command section of packet I - 1. In this case, the S bit of the recovery journal element MUST be set to 0. If a recovery journal element has its S bit set to 0, all higher-level recovery journal elements that contain it MUST also have S bits that are set to 0, including the top-level recovery journal header.

Other consistent bitfield coding idioms are described below:

- R flag bit. R flag bits are reserved for future use. Senders MUST set R bits to 0. Receivers MUST ignore R bit values.

- LENGTH field. All fields named LENGTH (as distinct from LEN) code the number of octets in the structure that contains it, including the header it resides in and all hierarchical levels below it. If a structure contains a LENGTH field, a receiver MUST use the LENGTH field value to advance past the structure during parsing, rather than use knowledge about the internal format of the structure.

We now define normative terms used to describe recovery journal semantics.

- Checkpoint history. The checkpoint history of a recovery journal is the concatenation of the MIDI command sections of packets C through I - 1. The final command in the MIDI command section for packet I - 1 is considered the most recent command; the first command in the MIDI command section for packet C is the oldest command. If command X is less recent than command Y, X is considered to be "before Y". A checkpoint history with no commands is considered to be empty. The checkpoint history never contains the MIDI command section of packet I (the packet containing the recovery journal), so if C == I, the checkpoint history is empty by definition.

- Session history. The session history of a recovery journal is the concatenation of MIDI command sections from the first packet of the session up to packet I - 1. The definitions of command recency and history emptiness follow those in the checkpoint history. The session history never contains the MIDI command section of packet I, and so the session history of the first packet in the session is empty by definition.

- Finished/unfinished commands. If all octets of a MIDI command appear in the session history, the command is defined as being finished. If some but not all octets of a command appear in the session history, the command is defined as being unfinished. Unfinished commands occur if segments of a SysEx command appear in several RTP packets. For example, if a SysEx command is coded as 3 segments, with segment 1 in packet K, segment 2 in packet K + 1, and segment 3 in packet K + 2, the session histories for packets K + 1 and K + 2 contain unfinished versions of the command. A session history contains a finished version of a cancelled SysEx command if the history contains the cancel sublist for the command.

- Reset State commands. Reset State (RS) commands reset renderers to an initialized "powerup" condition. The RS commands are: System Reset (0xFF), General MIDI System Enable (0xF0 0x7E 0xcc 0x09 0x01 0xF7), General MIDI 2 System Enable (0xF0 0x7E 0xcc 0x09 0x03 0xF7), General MIDI System Disable (0xF0 0x7E 0xcc 0x09 0x00 0xF7), Turn DLS On (0xF0 0x7E 0xcc 0x0A 0x01 0xF7), and Turn DLS Off (0xF0 0x7E 0xcc 0x0A 0x02 0xF7). Registrations of subrender parameter token values (Appendix C.6.2) and IETF standards-track documents MAY specify additional RS commands.

- Active commands. Active command are MIDI commands that do not appear before a Reset State command in the session history.

- N-active commands. N-active commands are MIDI commands that do not appear before one of the following commands in the session history: MIDI Control Change numbers 123-127 (numbers with All Notes Off semantics) or 120 (All Sound Off), and any Reset State command.

- C-active commands. C-active commands are MIDI commands that do not appear before one of the following commands in the session history: MIDI Control Change number 121 (Reset All Controllers) and any Reset State command.

- Oldest-first ordering rule. Several recovery journal chapters contain a list of elements, where each element is associated with a MIDI command that appears in the session history. In most cases, the chapter definition requires that list elements be ordered in accordance with the "oldest-first ordering rule". Below, we normatively define this rule:

  Elements associated with the most recent command in the session history coded in the list MUST appear at the end of the list.

  Elements associated with the oldest command in the session history coded in the list MUST appear at the start of the list.

  All other list elements MUST be arranged with respect to these boundary elements, to produce a list ordering that strictly reflects the relative session history recency of the commands coded by the elements in the list.

- Parameter system. A MIDI feature that provides two sets of 16,384 parameters to expand the 0-127 controller number space. The Registered Parameter Names (RPN) system and the Non-Registered Parameter Names (NRPN) system each provides 16,384 parameters.

- Parameter system transaction. The value of RPNs and NRPNs are changed by a series of Control Change commands that form a parameter system transaction. A canonical transaction begins with two Control Change commands to set the parameter number (controller numbers 99 and 98 for NRPNs, controller numbers 101 and 100 for RPNs). The transaction continues with an arbitrary number of Data Entry (controller numbers 6 and 38), Data Increment (controller number 96), and Data Decrement (controller number 97) Control Change commands to set the parameter value. The transaction ends with a second pair of (99, 98) or (101, 100) Control Change commands that specify the null parameter (MSB value 0x7F, LSB value 0x7F).

Several variants of the canonical transaction sequence are possible. Most commonly, the terminal pair of (99, 98) or (101, 100) Control Change commands may specify a parameter other than the null parameter. In this case, the command pair terminates the first transaction and starts a second transaction. The command pair is considered to be a part of both transactions. This variant is legal and recommended in [MIDI]. We refer to this variant as a "type 1 variant".

Less commonly, the MSB (99 or 101) or LSB (98 or 100) command of a (99, 98) or (101, 100) Control Change pair may be omitted.

If the MSB command is omitted, the transaction uses the MSB value of the most recent C-active Control Change command for controller number 99 or 101 that appears in the session history. We refer to this variant as a "type 2 variant".

If the LSB command is omitted, the LSB value 0x00 is assumed. We refer to this variant as a "type 3 variant". The type 2 and type 3 variants are defined as legal, but are not recommended, in [MIDI].

System real-time commands may appear at any point during a transaction (even between octets of individual commands in the transaction). More generally, [MIDI] does not forbid the appearance of unrelated MIDI commands during an open transaction. As a rule, these commands are considered to be "outside" the transaction and do not affect the status of the transaction in any way. Exceptions to this rule are commands whose semantics act to terminate transactions: Reset State commands, and Control Change (0xB) for controller number 121 (Reset All Controllers) [RP015].

- Initiated parameter system transaction. A canonical parameter system transaction whose (99, 98) or (101, 100) initial Control Change command pair appears in the session history is considered to be an initiated parameter system transaction. This definition also holds for type 1 variants. For type 2 variants (dropped MSB), a transaction whose initial LSB Control Change command appears in the session history is an initiated transaction. For type 3 variants (dropped LSB), a transaction is considered to be initiated if at least one transaction command follows the initial MSB (99 or 101) Control Change command in the session history. The completion of a transaction does not nullify its "initiated" status.

- Session history reference counts. Several recovery journal chapters include a reference count field, which codes the total number of commands of a type that appear in the session history. Examples include the Reset and Tune Request command logs (Chapter D, Appendix B.1) and the Active Sense command (Chapter V, Appendix B.2). Upon the detection of a loss event, reference count fields let a receiver deduce if any instances of the command have been lost, by comparing the journal reference count with its own reference count. Thus, a reference count field makes sense, even for command types in which knowing the NUMBER of lost commands is irrelevant (as is true with all of the example commands mentioned above).

The chapter definitions in Appendices A.2 to A.9 and B.1 to B.5 reflect the default recovery journal behavior. The ch_default, ch_never, and ch_anchor parameters modify these definitions, as described in Appendix C.2.3.

The chapter definitions specify if data MUST be present in the journal. Senders MAY also include non-required data in the journal. This optional data MUST comply with the normative chapter definition. For example, if a chapter definition states that a field codes data from the most recent active command in the session history, the sender MUST NOT code inactive commands or older commands in the field.

Finally, we note that a channel journal only encodes information about MIDI commands appearing on the MIDI channel the journal protects. All references to MIDI commands in Appendices A.2 to A.9 should be read as "MIDI commands appearing on this channel."

A.2. Chapter P: MIDI Program Change

A channel journal MUST contain Chapter P if an active Program Change (0xC) command appears in the checkpoint history. Figure A.2.1 shows the format for Chapter P.

```
 0                   1                   2
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|   PROGRAM   |B|   BANK-MSB  |X|  BANK-LSB   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.2.1 -- Chapter P format

The chapter has a fixed size of 24 bits. The PROGRAM field indicates the data value of the most recent active Program Change command in the session history. By default, the B, BANK-MSB, X, and BANK-LSB fields MUST be set to 0. Below, we define exceptions to this default condition.

If an active Control Change (0xB) command for controller number 0 (Bank Select MSB) appears before the Program Change command in the session history, the B bit MUST be set to 1, and the BANK-MSB field MUST code the data value of the Control Change command.

If B is set to 1, the BANK-LSB field MUST code the data value of the most recent Control Change command for controller number 32 (Bank Select LSB) that preceded the Program Change command coded in the PROGRAM field and followed the Control Change command coded in the BANK-MSB field. If no such Control Change command exists, the BANK-LSB field MUST be set to 0.

If B is set to 1, and if a Control Change command for controller number 121 (Reset All Controllers) appears in the MIDI stream between the Control Change command coded by the BANK-MSB field and the Program Change command coded by the PROGRAM field, the X bit MUST be set to 1.

Note that [RP015] specifies that Reset All Controllers does not reset the values of controller numbers 0 (Bank Select MSB) and 32 (Bank Select LSB). Thus, the X bit does not effect how receivers will use the BANK-LSB and BANK-MSB values when recovering from a lost Program Change command. The X bit serves to aid recovery in MIDI applications where controller numbers 0 and 32 are used in a non-standard way.

A.3. Chapter C: MIDI Control Change

Figure A.3.1 shows the format for Chapter C.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 8 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|     LEN     |S|   NUMBER    |A|  VALUE/ALT  |S|   NUMBER    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|  VALUE/ALT  |  ....                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.3.1 -- Chapter C format

The chapter consists of a 1-octet header, followed by a variable length list of 2-octet controller logs. The list MUST contain at least one controller log. The 7-bit LEN field codes the number of controller logs in the list, minus one. We define the semantics of the controller log fields in Appendix A.3.2.

A channel journal MUST contain Chapter C if the rules defined in this appendix require that one or more controller logs appear in the list.

A.3.1. Log Inclusion Rules

A controller log encodes information about a particular Control Change command in the session history.

In the default use of the payload format, list logs MUST encode information about the most recent active command in the session history for a controller number. Logs encoding earlier commands MUST NOT appear in the list.

Also, as a rule, the list MUST contain a log for the most recent active command for a controller number that appears in the checkpoint history. Below, we define exceptions to this rule:

- MIDI streams may transmit 14-bit controller values using paired Most Significant Byte (MSB, controller numbers 0-31, 99, 101) and Least Significant Byte (LSB, controller numbers 32-63, 98, 100) Control Change commands [MIDI].

  If the most recent active Control Change command in the session history for a 14-bit controller pair uses the MSB number, Chapter C MAY omit the controller log for the most recent active Control Change command for the associated LSB number, as the command ordering makes this LSB value irrelevant. However, this exception MUST NOT be applied if the sender is not certain that the MIDI source uses 14-bit semantics for the controller number pair. Note that some MIDI sources ignore 14-bit controller semantics and use the LSB controller numbers as independent 7-bit controllers.

- If active Control Change commands for controller numbers 0 (Bank Select MSB) or 32 (Bank Select LSB) appear in the checkpoint history, and if the command instances are also coded in the BANK-MSB and BANK-LSB fields of the Chapter P (Appendix A.2), Chapter C MAY omit the controller logs for the commands.

- Several controller number pairs are defined to be mutually exclusive. Controller numbers 124 (Omni Off) and 125 (Omni On) form a mutually exclusive pair, as do controller numbers 126 (Mono) and 127 (Poly).

  If active Control Change commands for one or both members of a mutually exclusive pair appear in the checkpoint history, a log for the controller number of the most recent command for the pair in the checkpoint history MUST appear in the controller list. However, the list MAY omit the controller log for the most recent active command for the other number in the pair.

If active Control Change commands for one or both members of a mutually exclusive pair appear in the session history, and if a log for the controller number of the most recent command for the pair does not appear in the controller list, a log for the most recent command for the other number of the pair MUST NOT appear in the controller list.

- If an active Control Change command for controller number 121 (Reset All Controllers) appears in the session history, the controller list MAY omit logs for Control Change commands that precede the Reset All Controllers command in the session history, under certain conditions.

  Namely, a log MAY be omitted if the sender is certain that a command stream follows the Reset All Controllers semantics defined in [RP015], and if the log codes a controller number for which [RP015] specifies a reset value.

  For example, [RP015] specifies that controller number 1 (Modulation Wheel) is reset to the value 0, and thus a controller log for Modulation Wheel MAY be omitted from the controller log list. In contrast, [RP015] specifies that controller number 7 (Channel Volume) is not reset, and thus a controller log for Channel Volume MUST NOT be omitted from the controller log list.

- Appendix A.3.4 defines exception rules for the MIDI Parameter System controller numbers 6, 38, and 96-101.

A.3.2. Controller Log Format

Figure A.3.2 shows the controller log structure of Chapter C.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|    NUMBER   |A|  VALUE/ALT  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.3.2 -- Chapter C controller log

The 7-bit NUMBER field identifies the controller number of the coded command. The 7-bit VALUE/ALT field codes recovery information for the command. The A bit sets the format of the VALUE/ALT field.

A log encodes recovery information using one of the following tools: the value tool, the toggle tool, or the count tool.

A log uses the value tool if the A bit is set to 0. The value tool codes the 7-bit data value of a command in the VALUE/ALT field. The value tool works best for controllers that code a continuous quantity, such as number 1 (Modulation Wheel).

The A bit is set to 1 to code the toggle or count tool. These tools work best for controllers that code discrete actions. Figure A.3.3 shows the controller log for these tools.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|    NUMBER   |1|T|    ALT    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.3.3 -- Controller log for ALT tools

A log uses the toggle tool if the T bit is set to 0. A log uses the count tool if the T bit is set to 1. Both methods use the 6-bit ALT field as an unsigned integer.

The toggle tool works best for controllers that act as on/off switches, such as 64 (Damper Pedal (Sustain)). These controllers code the "off" state with control values 0-63 and the "on" state with 64-127.

For the toggle tool, the ALT field codes the total number of toggles (off->on and on->off) due to Control Change commands in the session history, up to and including a toggle caused by the command coded by the log. The toggle count includes toggles caused by Control Change commands for controller number 121 (Reset All Controllers).

Toggle counting is performed modulo 64. The toggle count is reset at the start of a session, and whenever a Reset State command (Appendix A.1) appears in the session history. When these reset events occur, the toggle count for a controller is set to 0 (for controllers whose default value is 0-63) or 1 (for controllers whose default value is 64-127).

The Damper Pedal (Sustain) controller illustrates the benefits of the toggle tool over the value tool for switch controllers. As often used in piano applications, the "on" state of the controller lets notes resonate, while the "off" state immediately damps notes to silence. The loss of the "off" command in an "on->off->on" sequence results in ringing notes that should have been damped silent. The toggle tool lets receivers detect this lost "off" command, but the value tool does not.

The count tool is conceptually similar to the toggle tool. For the count tool, the ALT field codes the total number of Control Change commands in the session history, up to and including the command coded by the log. Command counting is performed modulo 64. The command count is set to 0 at the start of the session and is reset to 0 whenever a Reset State command (Appendix A.1) appears in the session history.

Because the count tool ignores the data value, it is a good match for controllers whose controller value is ignored, such as number 123 (All Notes Off). More generally, the count tool may be used to code a (modulo 64) identification number for a command.

A.3.3. Log List Coding Rules

In this section, we describe the organization of controller logs in the Chapter C log list.

A log encodes information about a particular Control Change command in the session history. In most cases, a command SHOULD be coded by a single tool (and, thus, a single log). If a number is coded with a single tool and this tool is the count tool, recovery Control Change commands generated by a receiver SHOULD use the default control value for the controller.

However, a command MAY be coded by several tool types (and, thus, several logs, each using a different tool). This technique may improve recovery performance for controllers with complex semantics, such as controller number 84 (Portamento Control) or controller number 121 (Reset All Controllers) when used with a non-zero data octet (with the semantics described in [DLS2]).

If a command is encoded by multiple tools, the logs MUST be placed in the list in the following order: count tool log (if any), followed by value tool log (if any), followed by toggle tool log (if any).

The Chapter C log list MUST obey the oldest-first ordering rule (defined in Appendix A.1). Note that this ordering preserves the information necessary for the recovery of 14-bit controller values, without precluding the use of MSB and LSB controller pairs as independent 7-bit controllers.

In the default use of the payload format, all logs that appear in the list for a controller number encode information about one Control Change command -- namely, the most recent active Control Change command in the session history for the number.

This coding scheme provides good recovery performance for the standard uses of Control Change commands defined in [MIDI]. However, not all MIDI applications restrict the use of Control Change commands to those defined in [MIDI].

For example, consider the common MIDI encoding of rotary encoders ("infinite" rotation knobs). The mixing console MIDI convention defined in [LCP] codes the position of rotary encoders as a series of Control Change commands. Each command encodes a relative change of knob position from the last update (expressed as a clockwise or counter-clockwise knob turning angle).

As the knob position is encoded incrementally over a series of Control Change commands, the best recovery performance is obtained if the log list encodes all Control Change commands for encoder controller numbers that appear in the checkpoint history, not only the most recent command.

To support application areas that use Control Change commands in this way, Chapter C may be configured to encode information about several Control Change commands for a controller number. We use the term "enhanced" to describe this encoding method, which we describe below.

In Appendix C.2.3, we show how to configure a stream to use enhanced Chapter C encoding for specific controller numbers. In Section 5 in the main text, we show how the H bits in the recovery journal header (Figure 8) and in the channel journal header (Figure 9) indicate the use of enhanced Chapter C encoding.

Here, we define how to encode a Chapter C log list that uses the enhanced encoding method.

Senders that use the enhanced encoding method for a controller number MUST obey the rules below. These rules let a receiver determine which logs in the list correspond to lost commands. Note that these rules override the exceptions listed in Appendix A.3.1.

- If N commands for a controller number are encoded in the list, the commands MUST be the N most recent commands for the controller number in the session history. For example, for N = 2, the sender MUST encode the most recent command and the second most recent command, not the most recent command and the third most recent command.

- If a controller number uses enhanced encoding, the encoding of the least-recent command for the controller number in the log list MUST include a count tool log. In addition, if commands are encoded for the controller number whose logs have S bits set to 0, the encoding of the least-recent command with S = 0 logs MUST include a count tool log.

  The count tool is OPTIONAL for the other commands for the controller number encoded in the list, as a receiver is able to efficiently deduce the count tool value for these commands, for both single-packet and multi-packet loss events.

- The use of the value and toggle tools MUST be identical for all commands for a controller number encoded in the list. For example, a value tool log either MUST appear for all commands for the controller number coded in the list, or alternatively, value tool logs for the controller number MUST NOT appear in the list. Likewise, a toggle tool log either MUST appear for all commands for the controller number coded in the list, or alternatively, toggle tool logs for the controller number MUST NOT appear in the list.

- If a command is encoded by multiple tools, the logs MUST be placed in the list in the following order: count tool log (if any), followed by value tool log (if any), followed by toggle tool log (if any).

These rules permit a receiver recovering from a packet loss to use the count tool log to match the commands encoded in the list with its own history of the stream, as we describe below. Note that the text below describes a non-normative algorithm; receivers are free to use any algorithm to match its history with the log list.

In a typical implementation of the enhanced encoding method, a receiver computes and stores count, value, and toggle tool data field values for the most recent Control Change command it has received for a controller number.

After a loss event, a receiver parses the Chapter C list and processes list logs for a controller number that uses enhanced encoding as follows.

The receiver compares the count tool ALT field for the least-recent command for the controller number in the list against its stored count data for the controller number, to determine if recovery is necessary for the command coded in the list. The value and toggle tool logs (if any) that directly follow the count tool log are associated with this least-recent command.

To check more-recent commands for the controller, the receiver detects additional value and/or toggle tool logs for the controller number in the list and infers count tool data for the command coded by these logs. This inferred data is used to determine if recovery is necessary for the command coded by the value and/or toggle tool logs.

In this way, a receiver is able to execute only lost commands, without executing a command twice. While recovering from a single packet loss, a receiver may skip through S = 1 logs in the list, as the first S = 0 log for an enhanced controller number is always a count tool log.

Note that the requirements in Appendix C.2.2.2 for protective sender and receiver actions during session startup for multicast operation are of particular importance for enhanced encoding, as receivers need to initialize its count tool data structures with recovery journal data in order to match commands correctly after a loss event.

Finally, we note in passing that in some applications of rotary encoders, a good user experience may be possible without the use of enhanced encoding. These applications are distinguished by visual feedback of encoding position that is driven by the post-recovery rotary encoding stream, and relatively low packet loss. In these domains, recovery performance may be acceptable for rotary encoders if the log list encodes only the most recent command, if both count and value logs appear for the command.

A.3.4. The Parameter System

Readers may wish to review the Appendix A.1 definitions of "parameter system", "parameter system transaction", and "initiated parameter system transaction" before reading this section.

Parameter system transactions update a MIDI Registered Parameter Number (RPN) or Non-Registered Parameter Number (NRPN) value. A parameter system transaction is a sequence of Control Change commands that may use the following controllers numbers:

- Data Entry MSB (6)
- Data Entry LSB (38)
- Data Increment (96)
- Data Decrement (97)
- Non-Registered Parameter Number (NRPN) LSB (98)
- Non-Registered Parameter Number (NRPN) MSB (99)
- Registered Parameter Number (RPN) LSB (100)
- Registered Parameter Number (RPN) MSB (101)

Control Change commands that are a part of a parameter system transaction MUST NOT be coded in Chapter C controller logs. Instead, these commands are coded in Chapter M, the MIDI Parameter chapter defined in Appendix A.4.

However, Control Change commands that use the listed controllers as general-purpose controllers (i.e., outside of a parameter system transaction) MUST NOT be coded in Chapter M.

Instead, the controllers are coded in Chapter C controller logs. The controller logs follow the coding rules stated in Appendix A.3.2 and A.3.3. The rules for coding paired LSB and MSB controllers, as defined in Appendix A.3.1, apply to the pairs (6, 38), (99, 98), and (101, 100) when coded in Chapter C.

If active Control Change commands for controller numbers 6, 38, or 96-101 appear in the checkpoint history, and these commands are used as general-purpose controllers, the most recent general-purpose command instance for these controller numbers MUST appear as entries in the Chapter C controller list.

MIDI syntax permits a source to use controllers 6, 38, 96, and 97 as parameter-system controllers and general-purpose controllers in the same stream. An RTP MIDI sender MUST deduce the role of each Control Change command for these controller numbers by noting the placement of the command in the stream and MUST use this information to code the command in Chapter C or Chapter M, as appropriate.

Specifically, active Control Change commands for controllers 6, 38, 96, and 97 act in a general-purpose way when

- no active Control Change commands that set an RPN or NRPN parameter number appear in the session history, or

- the most recent active Control Change commands in the session history that set an RPN or NRPN parameter number code the null parameter (MSB value 0x7F, LSB value 0x7F), or

- a Control Change command for controller number 121 (Reset All Controllers) appears more recently in the session history than all active Control Change commands that set an RPN or NRPN parameter number (see [RP015] for details).

Finally, we note that a MIDI source that follows the recommendations of [MIDI] exclusively uses numbers 98-101 as parameter system controllers. Alternatively, a MIDI source may exclusively use 98-101 as general-purpose controllers and lose the ability perform parameter system transactions in a stream.

In the language of [MIDI], the general-purpose use of controllers 98-101 constitutes a non-standard controller assignment. As most real-world MIDI sources use the standard controller assignment for controller numbers 98-101, an RTP MIDI sender SHOULD assume these controllers act as parameter system controllers, unless it knows that a MIDI source uses controller numbers 98-101 in a general-purpose way.

A.4. Chapter M: MIDI Parameter System

Readers may wish to review the Appendix A.1 definitions for "C-active", "parameter system", "parameter system transaction", and "initiated parameter system transaction" before reading this appendix.

Chapter M protects parameter system transactions for Registered Parameter Number (RPN) and Non-Registered Parameter Number (NRPN) values. Figure A.4.1 shows the format for Chapter M.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|P|E|U|W|Z|      LENGTH       |Q|  PENDING    |  Log list ... |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.4.1 -- Top-level Chapter M format

Chapter M begins with a 2-octet header. If the P header bit is set to 1, a 1-octet field follows the header, coding the 7-bit PENDING value and its associated Q bit.

The 10-bit LENGTH field codes the size of Chapter M and conforms to semantics described in Appendix A.1.

Chapter M ends with a list of zero or more variable-length parameter logs. Appendix A.4.2 defines the bitfield format of a parameter log. Appendix A.4.1 defines the inclusion semantics of the log list.

A channel journal MUST contain Chapter M if the rules defined in Appendix A.4.1 require that one or more parameter logs appear in the list.

A channel journal also MUST contain Chapter M if the most recent C-active Control Change command involved in a parameter system transaction in the checkpoint history is

- o an RPN MSB (101) or NRPN MSB (99) controller, or

- an RPN LSB (100) or NRPN LSB (98) controller that completes the coding of the null parameter (MSB value 0x7F, LSB value 0x7F).

This rule provides loss protection for partially transmitted parameter numbers and for the null parameter numbers.

If the most recent C-active Control Change command involved in a parameter system transaction in the session history is for the RPN MSB or NRPN MSB controller, the P header bit MUST be set to 1, and the PENDING field (and its associated Q bit) MUST follow the Chapter M header. Otherwise, the P header bit MUST be set to 0, and the PENDING field and Q bit MUST NOT appear in Chapter M.

If PENDING codes an NRPN MSB, the Q bit MUST be set to 1. If PENDING codes an RPN MSB, the Q bit MUST be set to 0.

The E header bit codes the current transaction state of the MIDI stream. If E = 1, an initiated transaction is in progress. Below, we define the rules for setting the E header bit:

- If no C-active parameter system transaction Control Change commands appear in the session history, the E bit MUST be set to 0.

- If the P header bit is set to 1, the E bit MUST be set to 0.

- If the most recent C-active parameter system transaction Control Change command in the session history is for the NRPN LSB or RPN LSB controller number, and if this command acts to complete the coding of the null parameter (MSB value 0x7F, LSB value 0x7F), the E bit MUST be set to 0.

- Otherwise, an initiated transaction is in progress, and the E bit MUST be set to 1.

The U, W, and Z header bits code properties that are shared by all parameter logs in the list. If these properties are set, parameter logs may be coded with improved efficiency (we explain how in A.4.1).

By default, the U, W, and Z bits MUST be set to 0. If all parameter logs in the list code RPN parameters, the U bit MAY be set to 1. If all parameter logs in the list code NRPN parameters, the W bit MAY be set to 1. If the parameter numbers of all RPN and NRPN logs in the list lie in the range 0-127 (and thus have an MSB value of 0), the Z bit MAY be set to 1.

Note that C-active semantics appear in the preceding paragraphs because [RP015] specifies that pending Parameter System transactions are closed by a Control Change command for controller number 121 (Reset All Controllers).

A.4.1. Log Inclusion Rules

Parameter logs code recovery information for a specific RPN or NRPN parameter.

A parameter log MUST appear in the list if an active Control Change command that forms a part of an initiated transaction for the parameter appears in the checkpoint history.

An exception to this rule applies if the checkpoint history only contains transaction Control Change commands for controller numbers 98-101 that act to terminate the transaction. In this case, a log for the parameter MAY be omitted from the list.

A log MAY appear in the list if an active Control Change command that forms a part of an initiated transaction for the parameter appears in the session history. Otherwise, a log for the parameter MUST NOT appear in the list.

Multiple logs for the same RPN or NRPN parameter MUST NOT appear in the log list.

The parameter log list MUST obey the oldest-first ordering rule (defined in Appendix A.1), with the phrase "parameter transaction" replacing the word "command" in the rule definition.

Parameter logs associated with the RPN or NRPN null parameter (LSB = 0x7F, MSB = 0x7F) MUST NOT appear in the log list. Chapter M uses the E header bit (Figure A.4.1) and the log list ordering rules to code null parameter semantics.

Note that "active" semantics (rather than "C-active" semantics) appear in the preceding paragraphs because [RP015] specifies that pending Parameter System transactions are not reset by a Control Change command for controller number 121 (Reset All Controllers). However, the rule that follows uses C-active semantics, because it concerns the protection of the transaction system itself, and [RP015] specifies that Reset All Controllers acts to close a transaction in progress.

In most cases, parameter logs for RPN and NRPN parameters that are assigned to the ch_never parameter (Appendix C.2.3) MAY be omitted from the list. An exception applies if

- the log codes the most recent initiated transaction in the session history, and
- a C-active command that forms a part of the transaction appears in the checkpoint history, and
- the E header bit for the top-level Chapter M header (Figure A.4.1) is set to 1.

In this case, a log for the parameter MUST appear in the list. This log informs receivers recovering from a loss that a transaction is in progress, so that the receiver is able to correctly interpret RPN or NRPN Control Change commands that follow the loss event.

A.4.2. Log Coding Rules

Figure A.4.2 shows the parameter log structure of Chapter M.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 8 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |S|  PNUM-LSB   |Q|  PNUM-MSB   |J|K|L|M|N|T|V|R|   Fields ...  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.4.2 -- Parameter log format

The log begins with a header, whose default size (as shown in Figure A.4.2) is 3 octets. If the Q header bit is set to 0, the log encodes an RPN parameter. If Q = 1, the log encodes an NRPN parameter. The 7-bit PNUM-MSB and PNUM-LSB fields code the parameter number and reflect the Control Change command data values for controllers 99 and 98 (for NRPNs) or 101 and 100 (for RPNs).

The J, K, L, M, and N header bits form a Table of Contents (TOC) for the log and signal the presence of fixed-sized fields that follow the header. A header bit that is set to 1 codes the presence of a field in the log. The ordering of fields in the log follows the ordering of the header bits in the TOC. Appendices A.4.2.1-2 define the fields associated with each TOC header bit.

The T and V header bits code information about the parameter log but are not part of the TOC. A set T or V bit does not signal the presence of any parameter log field.

If the rules in Appendix A.4.1 state that a log for a given parameter MUST appear in Chapter M, the log MUST code sufficient information to protect the parameter from the loss of active parameter transaction Control Change commands in the checkpoint history.

This rule does not apply if the parameter coded by the log is assigned to the ch_never parameter (Appendix C.2.3). In this case, senders MAY choose to set the J, K, L, M, and N TOC bits to 0, coding a parameter log with no fields.

Note that logs to protect parameters that are assigned to ch_never are REQUIRED under certain conditions (see Appendix A.4.1). The purpose of the log is to inform receivers recovering from a loss that a transaction is in progress, so that the receiver is able to correctly interpret RPN or NRPN Control Change commands that follow the loss event.

Parameter logs provide two tools for parameter protection: the value tool and the count tool. Depending on the semantics of the parameter, senders may use either tool, both tools, or neither tool to protect a given parameter.

The value tool codes information a receiver may use to determine the current value of an RPN or NRPN parameter. If a parameter log uses the value tool, the V header bit MUST be set to 1, and the semantics defined in Appendices A.4.2.1 for setting the J, K, L, and M TOC bits MUST be followed. If a parameter log does not use the value tool, the V bit MUST be set to 0, and the J, K, L, and M TOC bits MUST also be set to 0.

The count tool codes the number of transactions for an RPN or NRPN parameter. If a parameter log uses the count tool, the T header bit MUST be set to 1, and the semantics defined in Appendices A.4.2.2 for setting the N TOC bit MUST be followed. If a parameter log does not use the count tool, the T bit and the N TOC bit MUST be set to 0.

Note that V and T are set if the sender uses value (V) or count (T) tool for the log on an ongoing basis. Thus, V may be set even if J = K = L = M = 0, and T may be set even if N = 0.

In many cases, all parameters coded in the log list are of one type (RPN and NRPN), and all parameter numbers lie in the range 0-127. As described in Appendix A.4.1, senders MAY signal this condition by setting the top-level Chapter M header bit Z to 1 (to code the restricted range) and by setting the U or W bit to 1 (to code the parameter type).

If the top-level Chapter M header codes Z = 1 and either U = 1 or W = 1, all logs in the parameter log list MUST use a modified header format. This modification deletes bits 8-15 of the bitfield shown in Figure A.4.2, to yield a 2-octet header. The values of the deleted PNUM-MSB and Q fields may be inferred from the U, W, and Z bit values.

A.4.2.1. The Value Tool

The value tool uses several fields to track the value of an RPN or NRPN parameter.

The J TOC bit codes the presence of the octet shown in Figure A.4.3 in the field list.

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|X|  ENTRY-MSB  |
+-+-+-+-+-+-+-+-+
```

Figure A.4.3 -- ENTRY-MSB field

The 7-bit ENTRY-MSB field codes the data value of the most recent active Control Change command for controller number 6 (Data Entry MSB) in the session history that appears in a transaction for the log parameter.

The X bit MUST be set to 1 if the command coded by ENTRY-MSB precedes the most recent Control Change command for controller 121 (Reset All Controllers) in the session history. Otherwise, the X bit MUST be set to 0.

A parameter log that uses the value tool MUST include the ENTRY-MSB field if an active Control Change command for controller number 6 appears in the checkpoint history.

Note that [RP015] specifies that Control Change commands for controller 121 (Reset All Controllers) do not reset RPN and NRPN values, and thus the X bit would not play a recovery role for MIDI systems that comply with [RP015].

However, certain renderers (such as DLS 2 [DLS2]) specify that certain RPN values are reset for some uses of Reset All Controllers. The X bit (and other bitfield features of this nature in this appendix) plays a role in recovery for renderers of this type.

The K TOC bit codes the presence of the octet shown in Figure A.4.4 in the field list.

```
                                 0
                                 0 1 2 3 4 5 6 7
                                +-+-+-+-+-+-+-+-+
                                |X|  ENTRY-LSB  |
                                +-+-+-+-+-+-+-+-+
```

Figure A.4.4 -- ENTRY-LSB field

The 7-bit ENTRY-LSB field codes the data value of the most recent active Control Change command for controller number 38 (Data Entry LSB) in the session history that appears in a transaction for the log parameter.

The X bit MUST be set to 1 if the command coded by ENTRY-LSB precedes the most recent Control Change command for controller 121 (Reset All Controllers) in the session history. Otherwise, the X bit MUST be set to 0.

As a rule, a parameter log that uses the value tool MUST include the ENTRY-LSB field if an active Control Change command for controller number 38 appears in the checkpoint history. However, the ENTRY-LSB field MUST NOT appear in a parameter log if the Control Change command associated with the ENTRY-LSB precedes a Control Change command for controller number 6 (Data Entry MSB) that appears in a transaction for the log parameter in the session history.

The L TOC bit codes the presence of the octets shown in Figure A.4.5 in the field list.

```
                           0                   1
                           0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
                          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                          |G|X|       A-BUTTON            |
                          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.4.5 -- A-BUTTON field

The 14-bit A-BUTTON field codes a count of the number of active Control Change commands for controller numbers 96 and 97 (Data Increment and Data Decrement) in the session history that appear in a transaction for the log parameter.

The M TOC bit codes the presence of the octets shown in Figure A.4.6 in the field list.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|G|R|       C-BUTTON            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.4.6 -- C-BUTTON field

The 14-bit C-BUTTON field has semantics identical to A-BUTTON, except that Data Increment and Data Decrement Control Change commands that precede the most recent Control Change command for controller 121 (Reset All Controllers) in the session history are not counted.

For both A-BUTTON and C-BUTTON, Data Increment and Data Decrement Control Change commands are not counted if they precede Control Changes commands for controller numbers 6 (Data Entry MSB) or 38 (Data Entry LSB) that appear in a transaction for the log parameter in the session history.

The A-BUTTON and C-BUTTON fields are interpreted as unsigned integers, and the G bit associated the field codes the sign of the integer (G = 0 for positive or zero, G = 1 for negative).

To compute and code the count value, initialize the count value to 0, add 1 for each qualifying Data Increment command, and subtract 1 for each qualifying Data Decrement command. After each add or subtract, limit the count magnitude to 16383. The G bit codes the sign of the count, and the A-BUTTON or C-BUTTON field codes the count magnitude.

For the A-BUTTON field, if the most recent qualified Data Increment or Data Decrement command precedes the most recent Control Change command for controller 121 (Reset All Controllers) in the session history, the X bit associated with A-BUTTON field MUST be set to 1. Otherwise, the X bit MUST be set to 0.

A parameter log that uses the value tool MUST include the A-BUTTON and C-BUTTON fields if an active Control Change command for controller numbers 96 or 97 appears in the checkpoint history. However, to improve coding efficiency, this rule has several exceptions:

- If the log includes the A-BUTTON field, and if the X bit of the A-BUTTON field is set to 1, the C-BUTTON field (and its associated R and G bits) MAY be omitted from the log.

- If the log includes the A-BUTTON field, and if the A-BUTTON and C-BUTTON fields (and their associated G bits) code identical values, the C-BUTTON field (and its associated R and G bits) MAY be omitted from the log.

A.4.2.2. The Count Tool

The count tool tracks the number of transactions for an RPN or NRPN parameter. The N TOC bit codes the presence of the octet shown in Figure A.4.7 in the field list.

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|X|    COUNT    |
+-+-+-+-+-+-+-+-+
```

Figure A.4.7 -- COUNT field

The 7-bit COUNT codes the number of initiated transactions for the log parameter that appear in the session history. Initiated transactions are counted if they contain one or more active Control Change commands, including commands for controllers 98-101 that initiate the parameter transaction.

If the most recent counted transaction precedes the most recent Control Change command for controller 121 (Reset All Controllers) in the session history, the X bit associated with the COUNT field MUST be set to 1. Otherwise, the X bit MUST be set to 0.

Transaction counting is performed modulo 128. The transaction count is set to 0 at the start of a session and is reset to 0 whenever a Reset State command (Appendix A.1) appears in the session history.

A parameter log that uses the count tool MUST include the COUNT field if an active command that increments the transaction count (modulo 128) appears in the checkpoint history.

A.5. Chapter W: MIDI Pitch Wheel

A channel journal MUST contain Chapter W if a C-active MIDI Pitch Wheel (0xE) command appears in the checkpoint history. Figure A.5.1 shows the format for Chapter W.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|     FIRST   |R|    SECOND   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.5.1 -- Chapter W format

The chapter has a fixed size of 16 bits. The FIRST and SECOND fields are the 7-bit values of the first and second data octets of the most recent active Pitch Wheel command in the session history.

Note that Chapter W encodes C-active commands and thus does not encode active commands that are not C-active (see the second-to-last paragraph of Appendix A.1 for an explanation of chapter inclusion text in this regard).

Chapter W does not encode "active but not C-active" commands because [RP015] declares that Control Change commands for controller number 121 (Reset All Controllers) act to reset the Pitch Wheel value to 0. If Chapter W encoded "active but not C-active" commands, a repair operation following a Reset All Controllers command could incorrectly repair the stream with a stale Pitch Wheel value.

A.6. Chapter N: MIDI NoteOff and NoteOn

In this appendix, we consider NoteOn commands with zero velocity to be NoteOff commands. Readers may wish to review the Appendix A.1 definition of "N-active commands" before reading this appendix.

Chapter N completely protects note commands in streams that alternate between NoteOn and NoteOff commands for a particular note number. However, in rare applications, multiple overlapping NoteOn commands may appear for a note number. Chapter E, described in Appendix A.7, augments Chapter N to completely protect these streams.

A channel journal MUST contain Chapter N if an N-active MIDI NoteOn (0x9) or NoteOff (0x8) command appears in the checkpoint history. Figure A.6.1 shows the format for Chapter N.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 8 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |B|     LEN     |  LOW  | HIGH  |S|   NOTENUM   |Y|  VELOCITY   |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |S|   NOTENUM   |Y|  VELOCITY   |             ....              |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |    OFFBITS    |    OFFBITS    |     ....      |    OFFBITS    |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.6.1 -- Chapter N format

Chapter N consists of a 2-octet header, followed by at least one of the following data structures:

- o A list of note logs to code NoteOn commands.
- o A NoteOff bitfield structure to code NoteOff commands.

We define the header bitfield semantics in Appendix A.6.1. We define the note log semantics and the NoteOff bitfield semantics in Appendix A.6.2.

If one or more N-active NoteOn or NoteOff commands in the checkpoint history reference a note number, the note number MUST be coded in either the note log list or the NoteOff bitfield structure.

The note log list MUST contain an entry for all note numbers whose most recent checkpoint history appearance is in an N-active NoteOn command. The NoteOff bitfield structure MUST contain a set bit for all note numbers whose most recent checkpoint history appearance is in an N-active NoteOff command.

A note number MUST NOT be coded in both structures.

All note logs and NoteOff bitfield set bits MUST code the most recent N-active NoteOn or NoteOff reference to a note number in the session history.

The note log list MUST obey the oldest-first ordering rule (defined in Appendix A.1).

A.6.1. Header Structure

The header for Chapter N, shown in Figure A.6.2, codes the size of the note list and bitfield structures.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|B|     LEN     |  LOW  | HIGH  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.6.2 -- Chapter N header

The LEN field, a 7-bit integer value, codes the number of 2-octet note logs in the note list. Zero is a valid value for LEN and codes an empty note list.

The 4-bit LOW and HIGH fields code the number of OFFBITS octets that follow the note log list. LOW and HIGH are unsigned integer values. If LOW <= HIGH, there are (HIGH - LOW + 1) OFFBITS octets in the chapter. The value pairs (LOW = 15, HIGH = 0) and (LOW = 15, HIGH = 1) code an empty NoteOff bitfield structure (i.e., no OFFBITS octets). Other (LOW > HIGH) value pairs MUST NOT appear in the header.

The B bit provides S-bit functionality (Appendix A.1) for the NoteOff bitfield structure. By default, the B bit MUST be set to 1. However, if the MIDI command section of the previous packet (packet I - 1, with I as defined in Appendix A.1) includes a NoteOff command for the channel, the B bit MUST be set to 0. If the B bit is set to 0, the higher-level recovery journal elements that contain Chapter N MUST have S bits that are set to 0, including the top-level journal header.

The LEN value of 127 codes a note list length of 127 or 128 note logs, depending on the values of LOW and HIGH. If LEN = 127, LOW = 15, and HIGH = 0, the note list holds 128 note logs, and the NoteOff bitfield structure is empty. For other values of LOW and HIGH, LEN = 127 codes that the note list contains 127 note logs. In this case, the chapter has (HIGH - LOW + 1) NoteOff OFFBITS octets if LOW <= HIGH and has no OFFBITS octets if LOW = 15 and HIGH = 1.

A.6.2. Note Structures

Figure A.6.3 shows the 2-octet note log structure.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|   NOTENUM   |Y|  VELOCITY   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.6.3 -- Chapter N note log

The 7-bit NOTENUM field codes the note number for the log. A note number MUST NOT be represented by multiple note logs in the note list.

The 7-bit VELOCITY field codes the velocity value for the most recent N-active NoteOn command for the note number in the session history. Multiple overlapping NoteOns for a given note number may be coded using Chapter E, as discussed in Appendix A.7.

VELOCITY is never zero; NoteOn commands with zero velocity are coded as NoteOff commands in the NoteOff bitfield structure.

The note log does not code the execution time of the NoteOn command. However, the Y bit codes a hint from the sender about the NoteOn execution time. The Y bit codes a recommendation to play (Y = 1) or skip (Y = 0) the NoteOn command recovered from the note log. See Section 4.2 of [RFC4696] for non-normative guidance on the use of the Y bit.

Figure A.6.1 shows the NoteOff bitfield structure, as the list of OFFBITS octets at the end of the chapter. A NoteOff OFFBITS octet codes NoteOff information for eight consecutive MIDI note numbers, with the most-significant bit representing the lowest note number. The most-significant bit of the first OFFBITS octet codes the note number 8*LOW; the most-significant bit of the last OFFBITS octet codes the note number 8*HIGH.

A set bit codes a NoteOff command for the note number. In the most efficient coding for the NoteOff bitfield structure, the first and last octets of the structure contain at least one set bit. Note that Chapter N does not code NoteOff velocity data.

Note that in the general case, the recovery journal does not code the relative placement of a NoteOff command and a Change Control command for controller 64 (Damper Pedal (Sustain)). In many cases, a receiver processing a loss event may deduce this relative placement from the history of the stream and thus determine if a NoteOff note is sustained by the pedal. If such a determination is not possible, receivers SHOULD err on the side of silencing pedal sustains, as erroneously sustained notes may produce unpleasant (albeit transient) artifacts.

A.7. Chapter E: MIDI Note Command Extras

Readers may wish to review the Appendix A.1 definition of "N-active commands" before reading this appendix. In this appendix, a NoteOn command with a velocity of 0 is considered to be a NoteOff command with a release velocity value of 64.

Chapter E encodes recovery information about MIDI NoteOn (0x9) and NoteOff (0x8) command features that rarely appear in MIDI streams. Receivers use Chapter E to reduce transient artifacts for streams where several NoteOn commands appear for a note number without an intervening NoteOff. Receivers also use Chapter E to reduce transient artifacts for streams that use NoteOff release velocity. Chapter E supplements the note information coded in Chapter N (Appendix A.6).

Figure A.7.1 shows the format for Chapter E.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 8 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|     LEN     |S|   NOTENUM   |V|  COUNT/VEL  |S|  NOTENUM    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V|  COUNT/VEL  |  ....                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.7.1 -- Chapter E format

The chapter consists of a 1-octet header, followed by a variable-length list of 2-octet note logs. Appendix A.7.1 defines the bitfield format for a note log.

The log list MUST contain at least one note log. The 7-bit LEN header field codes the number of note logs in the list, minus one. A channel journal MUST contain Chapter E if the rules defined in this appendix require that one or more note logs appear in the list. The note log list MUST obey the oldest-first ordering rule (defined in Appendix A.1).

A.7.1. Note Log Format

Figure A.7.2 reproduces the note log structure of Chapter E.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|   NOTENUM   |V|  COUNT/VEL  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.7.2 -- Chapter E note log

A note log codes information about the MIDI note number coded by the 7-bit NOTENUM field. The nature of the information depends on the value of the V flag bit.

If the V bit is set to 1, the COUNT/VEL field codes the release velocity value for the most recent N-active NoteOff command for the note number that appears in the session history.

If the V bit is set to 0, the COUNT/VEL field codes a reference count of the number of NoteOn and NoteOff commands for the note number that appear in the session history.

The reference count is set to 0 at the start of the session. NoteOn commands increment the count by 1. NoteOff commands decrement the count by 1. However, a decrement that generates a negative count value is not performed.

If the reference count is in the range 0-126, the 7-bit COUNT/VEL field codes an unsigned integer representation of the count. If the count is greater than or equal to 127, COUNT/VEL is set to 127.

By default, the count is reset to 0 whenever a Reset State command (Appendix A.1) appears in the session history, and whenever MIDI Control Change commands for controller numbers 123-127 (numbers with All Notes Off semantics) or 120 (All Sound Off) appear in the session history.

A.7.2. Log Inclusion Rules

If the most recent N-active NoteOn or NoteOff command for a note number in the checkpoint history is a NoteOff command with a release velocity value other than 64, a note log whose V bit is set to 1 MUST appear in Chapter E for the note number.

If the most recent N-active NoteOn or NoteOff command for a note number in the checkpoint history is a NoteOff command, and if the reference count for the note number is greater than 0, a note log whose V bit is set to 0 MUST appear in Chapter E for the note number.

If the most recent N-active NoteOn or NoteOff command for a note number in the checkpoint history is a NoteOn command, and if the reference count for the note number is greater than 1, a note log whose V bit is set to 0 MUST appear in Chapter E for the note number.

At most, two note logs MAY appear in Chapter E for a note number: one log whose V bit is set to 0, and one log whose V bit is set to 1.

Chapter E codes a maximum of 128 note logs. If the log inclusion rules yield more than 128 REQUIRED logs, note logs whose V bit is set to 1 MUST be dropped from Chapter E in order to reach the 128-log limit. Note logs whose V bit is set to 0 MUST NOT be dropped.

Most MIDI streams do not use NoteOn and NoteOff commands in ways that would trigger the log inclusion rules. For these streams, Chapter E would never be REQUIRED to appear in a channel journal.

The ch_never parameter (Appendix C.2.3) may be used to configure the log inclusion rules for Chapter E.

A.8. Chapter T: MIDI Channel Aftertouch

A channel journal MUST contain Chapter T if an N-active and C-active MIDI Channel Aftertouch (0xD) command appears in the checkpoint history. Figure A.8.1 shows the format for Chapter T.

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|S|   PRESSURE  |
+-+-+-+-+-+-+-+-+
```

Figure A.8.1 -- Chapter T format

The chapter has a fixed size of 8 bits. The 7-bit PRESSURE field holds the pressure value of the most recent N-active and C-active Channel Aftertouch command in the session history.

Chapter T only encodes commands that are C-active and N-active. We define a C-active restriction because [RP015] declares that a Control Change command for controller 121 (Reset All Controllers) acts to reset the channel pressure to 0 (see the discussion at the end of Appendix A.5 for a more complete rationale).

We define an N-active restriction on the assumption that aftertouch commands are linked to note activity, and thus Channel Aftertouch commands that are not N-active are stale and should not be used to repair a stream.

A.9. Chapter A: MIDI Poly Aftertouch

A channel journal MUST contain Chapter A if a C-active Poly Aftertouch (0xA) command appears in the checkpoint history. Figure A.9.1 shows the format for Chapter A.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 8 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|     LEN     |S|   NOTENUM   |X|  PRESSURE   |S|   NOTENUM   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|X|  PRESSURE   |  ....                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.9.1 -- Chapter A format

The chapter consists of a 1-octet header, followed by a variable-length list of 2-octet note logs. A note log MUST appear for a note number if a C-active Poly Aftertouch command for the note number appears in the checkpoint history. A note number MUST NOT be represented by multiple note logs in the note list. The note log list MUST obey the oldest-first ordering rule (defined in Appendix A.1).

The 7-bit LEN field codes the number of note logs in the list, minus one. Figure A.9.2 reproduces the note log structure of Chapter A.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|   NOTENUM   |X|  PRESSURE   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure A.9.2 -- Chapter A note log

The 7-bit PRESSURE field codes the pressure value of the most recent C-active Poly Aftertouch command in the session history for the MIDI note number coded in the 7-bit NOTENUM field.

As a rule, the X bit MUST be set to 0. However, the X bit MUST be set to 1 if the command coded by the log appears before one of the following commands in the session history: MIDI Control Change numbers 123-127 (numbers with All Notes Off semantics) or 120 (All Sound Off).

We define C-active restrictions for Chapter A because [RP015] declares that a Control Change command for controller 121 (Reset All Controllers) acts to reset the polyphonic pressure to 0 (see the discussion at the end of Appendix A.5 for a more complete rationale).

B. The Recovery Journal System Chapters

B.1. System Chapter D: Simple System Commands

The system journal MUST contain Chapter D if an active MIDI Reset (0xFF), MIDI Tune Request (0xF6), MIDI Song Select (0xF3), undefined MIDI System Common (0xF4 and 0xF5), or undefined MIDI System Real-time (0xF9 and 0xFD) command appears in the checkpoint history.

Figure B.1.1 shows the variable-length format for Chapter D.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|B|G|H|J|K|Y|Z|  Command logs ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.1.1 -- System Chapter D format

The chapter consists of a 1-octet header, followed by one or more command logs. Header flag bits indicate the presence of command logs for the Reset (B = 1), Tune Request (G = 1), Song Select (H = 1), undefined System Common 0xF4 (J = 1), undefined System Common 0xF5 (K = 1), undefined System Real-time 0xF9 (Y = 1), or undefined System Real-time 0xFD (Z = 1) commands.

Command logs appear in a list following the header, in the order that the flag bits appear in the header.

Figure B.1.2 shows the 1-octet command log format for the Reset and Tune Request commands.

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|S|    COUNT    |
+-+-+-+-+-+-+-+-+
```

Figure B.1.2 -- Command log for Reset and Tune Request

Chapter D MUST contain the Reset command log if an active Reset command appears in the checkpoint history. The 7-bit COUNT field codes the total number of Reset commands (modulo 128) present in the session history.

Chapter D MUST contain the Tune Request command log if an active Tune Request command appears in the checkpoint history. The 7-bit COUNT field codes the total number of Tune Request commands (modulo 128) present in the session history.

For these commands, the COUNT field acts as a reference count. See the definition of "session history reference counts" in Appendix A.1 for more information.

Figure B.1.3 shows the 1-octet command log format for the Song Select command.

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|S|    VALUE    |
+-+-+-+-+-+-+-+-+
```

Figure B.1.3 -- Song Select command log format

Chapter D MUST contain the Song Select command log if an active Song Select command appears in the checkpoint history. The 7-bit VALUE field codes the song number of the most recent active Song Select command in the session history.

B.1.1. Undefined System Commands

In this section, we define the Chapter D command logs for the undefined System commands. [MIDI] reserves the undefined System commands 0xF4, 0xF5, 0xF9, and 0xFD for future use. At the time of this writing, any MIDI command stream that uses these commands is non-compliant with [MIDI]. However, future versions of [MIDI] may define these commands, and a few products do use these commands in a non-compliant manner.

Figure B.1.4 shows the variable-length command log format for the undefined System Common commands (0xF4 and 0xF5).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|C|V|L|DSZ|      LENGTH       |    COUNT      |  VALUE ...    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  LEGAL ...                                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.1.4 -- Undefined System Common command log format

The command log codes a single command type (0xF4 or 0xF5, not both). Chapter D MUST contain a command log if an active 0xF4 command appears in the checkpoint history and MUST contain an independent command log if an active 0xF5 command appears in the checkpoint history.

Chapter D consists of a two-octet header followed by a variable number of data fields. Header flag bits indicate the presence of the COUNT field (C = 1), the VALUE field (V = 1), and the LEGAL field (L = 1). The 10-bit LENGTH field codes the size of the command log and conforms to semantics described in Appendix A.1.

The 2-bit DSZ field codes the number of data octets in the command instance that appears most recently in the session history. If DSZ = 0-2, the command has 0-2 data octets. If DSZ = 3, the command has 3 or more command data octets.

We now define the default rules for the use of the COUNT, VALUE, and LEGAL fields. The session configuration tools defined in Appendix C.2.3 may be used to override this behavior.

By default, if the DSZ field is set to 0, the command log MUST include the COUNT field. The 8-bit COUNT field codes the total number of commands of the type coded by the log (0xF4 or 0xF5) present in the session history, modulo 256.

By default, if the DSZ field is set to 1-3, the command log MUST include the VALUE field. The variable-length VALUE field codes a verbatim copy the data octets for the most recent use of the command type coded by the log (0xF4 or 0xF5) in the session history. The most-significant bit of the final data octet MUST be set to 1, and the most-significant bit of all other data octets MUST be set to 0.

The LEGAL field is reserved for future use. If an update to [MIDI] defines the 0xF4 or 0xF5 command, an IETF standards-track document may define the LEGAL field. Until such a document appears, senders MUST NOT use the LEGAL field, and receivers MUST use the LENGTH field to skip over the LEGAL field. The LEGAL field would be defined by the IETF if the semantics of the new 0xF4 or 0xF5 command could not be protected from packet loss via the use of the COUNT and VALUE fields.

Figure B.1.5 shows the variable-length command log format for the undefined System Real-time commands (0xF9 and 0xFD).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|C|L| LENGTH  |     COUNT     |  LEGAL ...                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.1.5 -- Undefined System Real-time command log format

The command log codes a single command type (0xF9 or 0xFD, not both). Chapter D MUST contain a command log if an active 0xF9 command appears in the checkpoint history and MUST contain an independent command log if an active 0xFD command appears in the checkpoint history.

Chapter D consists of a one-octet header followed by a variable number of data fields. Header flag bits indicate the presence of the COUNT field (C = 1) and the LEGAL field (L = 1). The 5-bit LENGTH field codes the size of the command log and conforms to semantics described in Appendix A.1.

We now define the default rules for the use of the COUNT and LEGAL fields. The session configuration tools defined in Appendix C.2.3 may be used to override this behavior.

The 8-bit COUNT field codes the total number of commands of the type coded by the log present in the session history, modulo 256. By default, the COUNT field MUST be present in the command log.

The LEGAL field is reserved for future use. If an update to [MIDI] defines the 0xF9 or 0xFD command, an IETF standards-track document may define the LEGAL field to protect the command. Until such a document appears, senders MUST NOT use the LEGAL field, and receivers MUST use the LENGTH field to skip over the LEGAL field. The LEGAL field would be defined by the IETF if the semantics of the new 0xF9 or 0xFD command could not be protected from packet loss via the use of the COUNT field.

Finally, we note that some non-standard uses of the undefined System Real-time commands act to implement non-compliant variants of the MIDI sequencer system. In Appendix B.3.1, we describe resiliency tools for the MIDI sequencer system that provide some protection in this case.

B.2. System Chapter V: Active Sense Command

The system journal MUST contain Chapter V if an active MIDI Active Sense (0xFE) command appears in the checkpoint history. Figure B.2.1 shows the format for Chapter V.

```
                               0
                               0 1 2 3 4 5 6 7
                              +-+-+-+-+-+-+-+-+
                              |S|    COUNT    |
                              +-+-+-+-+-+-+-+-+
```

Figure B.2.1 -- System Chapter V format

The 7-bit COUNT field codes the total number of Active Sense commands (modulo 128) present in the session history. The COUNT field acts as a reference count. See the definition of "session history reference counts" in Appendix A.1 for more information.

B.3. System Chapter Q: Sequencer State Commands

This appendix describes Chapter Q, the system chapter for the MIDI sequencer commands.

The system journal MUST contain Chapter Q if an active MIDI Song Position Pointer (0xF2), MIDI Clock (0xF8), MIDI Start (0xFA), MIDI Continue (0xFB), or MIDI Stop (0xFC) command appears in the checkpoint history, and if the rules defined in this appendix require a change in the Chapter Q bitfield contents because of the command appearance.

Figure B.3.1 shows the variable-length format for Chapter Q.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |S|N|D|C|T| TOP |            CLOCK              | TIMETOOLS ... |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |              ...              |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.3.1 -- System Chapter Q format

Chapter Q consists of a 1-octet header followed by several optional fields, in the order shown in Figure B.3.1.

Header flag bits signal the presence of the 16-bit CLOCK field (C = 1) and the 24-bit TIMETOOLS field (T = 1). The 3-bit TOP header field is interpreted as an unsigned integer, as are CLOCK and TIMETOOLS. We describe the TIMETOOLS field in Appendix B.3.1.

Chapter Q encodes the most recent state of the sequencer system. Receivers use the chapter to re-synchronize the sequencer after a packet loss episode. Chapter fields encode the on/off state of the sequencer, the current position in the song, and the downbeat.

The N header bit encodes the relative occurrence of the Start, Stop, and Continue commands in the session history. If an active Start or Continue command appears most recently, the N bit MUST be set to 1. If an active Stop appears most recently, or if no active Start, Stop, or Continue commands appear in the session history, the N bit MUST be set to 0.

The C header flag, the TOP header field, and the CLOCK field act to code the current position in the sequence:

- If C = 1, the 3-bit TOP header field and the 16-bit CLOCK field are combined to form the 19-bit unsigned quantity 65536*TOP + CLOCK. This value encodes the song position in units of MIDI Clocks (24 clocks per quarter note), modulo 524288. Note that the maximum song position value that may be coded by the Song Position Pointer command is 98303 clocks (which may be coded with 17 bits), and that MIDI-coded songs are generally constructed to avoid durations longer than this value. However, the 19-bit size may be useful for real-time applications, such as a drum machine MIDI output that is sending clock commands for long periods of time.

- If C = 0, the song position is the start of the song. The C = 0 position is identical to the position coded by C = 1, TOP = 0, and CLOCK = 0, for the case where the song position is less than 524288 MIDI clocks. In certain situations (defined later in this section), normative text may require the C = 0 or the C = 1, TOP = 0, CLOCK = 0 encoding of the start of the song.

The C, TOP, and CLOCK fields MUST be set to code the current song position, for both N = 0 and N = 1 conditions. If C = 0, the TOP field MUST be set to 0. See [MIDI] for a precise definition of a song position.

The D header bit encodes information about the downbeat and acts to qualify the song position coded by the C, TOP, and CLOCK fields.

If the D bit is set to 1, the song position represents the most recent position in the sequence that has played. If D = 1, the next Clock command (if N = 1) or the next (Continue, Clock) pair (if N = 0) acts to increment the song position by one clock, and to play the updated position.

If the D bit is set to 0, the song position represents a position in the sequence that has not yet been played. If D = 0, the next Clock command (if N = 1) or the next (Continue, Clock) pair (if N = 0) acts to play the point in the song coded by the song position. The song position is not incremented.

An example of a stream that uses D = 0 coding is one whose most recent sequence command is a Start or Song Position Pointer command (both N = 1 conditions). However, it is also possible to construct examples where D = 0 and N = 0. A Start command immediately followed by a Stop command is coded in Chapter Q by setting C = 0, D = 0, N = 0, TOP = 0.

If N = 1 (coding Start or Continue), D = 0 (coding that the downbeat has yet to be played), and the song position is at the start of the song, the C = 0 song position encoding MUST be used if a Start command occurs more recently than a Continue command in the session history, and the C = 1, TOP = 0, CLOCK = 0 song position encoding MUST be used if a Continue command occurs more recently than a Start command in the session history.

B.3.1. Non-compliant Sequencers

The Chapter Q description in this appendix assumes that the sequencer system counts off time with Clock commands, as mandated in [MIDI]. However, a few non-compliant products do not use Clock commands to count off time, but instead use non-standard methods.

Chapter Q uses the TIMETOOLS field to provide resiliency support for these non-standard products. By default, the TIMETOOLS field MUST NOT appear in Chapter Q, and the T header bit MUST be set to 0. The session configuration tools described in Appendix C.2.3 may be used to select TIMETOOLS coding.

Figure B.3.2 shows the format of the 24-bit TIMETOOLS field.

```
                0                   1                   2
                0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
               |                   TIME                        |
               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.3.2 -- TIMETOOLS format

The TIME field is a 24-bit unsigned integer quantity, with units of milliseconds. TIME codes an additive correction term for the song position coded by the TOP, CLOCK, and C fields. TIME is coded in network byte order (big-endian).

A receiver computes the correct song position by converting TIME into units of MIDI clocks and adding it to 65536*TOP + CLOCK (assuming C = 1). Alternatively, a receiver may convert 65536*TOP + CLOCK into milliseconds (assuming C = 1) and add it to TIME. The downbeat (D header bit) semantics defined in Appendix B.3 apply to the corrected song position.

B.4. System Chapter F: MIDI Time Code Tape Position

This appendix describes Chapter F, the system chapter for the MIDI Time Code (MTC) commands. Readers may wish to review the Appendix A.1 definition of "finished/unfinished commands" before reading this appendix.

The system journal MUST contain Chapter F if an active System Common Quarter Frame command (0xF1) or an active finished System Exclusive (Universal Real Time) MTC Full Frame command (F0 7F cc 01 01 hr mn sc fr F7) appears in the checkpoint history. Otherwise, the system journal MUST NOT contain Chapter F.

Figure B.4.1 shows the variable-length format for Chapter F.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |S|C|P|Q|D|POINT|  COMPLETE ...                                 |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |     ...       |  PARTIAL  ...                                 |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |     ...       |
      +-+-+-+-+-+-+-+-+
```

Figure B.4.1 -- System Chapter F format

Chapter F holds information about recent MTC tape positions coded in the session history. Receivers use Chapter F to re-synchronize the MTC system after a packet loss episode.

Chapter F consists of a 1-octet header followed by several optional fields, in the order shown in Figure B.4.1. The C and P header bits form a Table of Contents (TOC) and signal the presence of the 32-bit COMPLETE field (C = 1) and the 32-bit PARTIAL field (P = 1).

The Q header bit codes information about the COMPLETE field format. If Chapter F does not contain a COMPLETE field, Q MUST be set to 0.

The D header bit codes the tape movement direction. If the tape is moving forward, or if the tape direction is indeterminate, the D bit MUST be set to 0. If the tape is moving in the reverse direction, the D bit MUST be set to 1. In most cases, the ordering of commands in the session history clearly defines the tape direction. However, a few command sequences have an indeterminate direction (such as a session history consisting of one Full Frame command).

The 3-bit POINT header field is interpreted as an unsigned integer. Appendix B.4.1 defines how the POINT field codes information about the contents of the PARTIAL field. If Chapter F does not contain a PARTIAL field, POINT MUST be set to 7 (if D = 0) or 0 (if D = 1).

Chapter F MUST include the COMPLETE field if an active finished Full Frame command appears in the checkpoint history, or if an active Quarter Frame command that completes the encoding of a frame value appears in the checkpoint history.

The COMPLETE field encodes the most recent active complete MTC frame value that appears in the session history. This frame value may take the form of a series of 8 active Quarter Frame commands (0xF1 0x0n through 0xF1 0x7n for forward tape movement, 0xF1 0x7n through 0xF1 0x0n for reverse tape movement) or may take the form of an active finished Full Frame command.

If the COMPLETE field encodes a Quarter Frame command series, the Q header bit MUST be set to 1, and the COMPLETE field MUST have the format shown in Figure B.4.2. The 4-bit fields MT0 through MT7 code the data (lower) nibble for the Quarter Frame commands for Message Type 0 through Message Type 7 [MIDI]. These nibbles encode a complete frame value, in addition to fields reserved for future use by [MIDI].

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  MT0  |  MT1  |  MT2  |  MT3  |  MT4  |  MT5  |  MT6  |  MT7  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.4.2 -- COMPLETE field format, Q = 1

In this usage, the frame value encoded in the COMPLETE field MUST be offset by 2 frames (relative to the frame value encoded in the Quarter Frame commands) if the frame value codes a 0xF1 0x0n through 0xF1 0x7n command sequence. This offset compensates for the two-frame latency of the Quarter Frame encoding for forward tape movement. No offset is applied if the frame value codes a 0xF1 0x7n through 0xF1 0x0n Quarter Frame command sequence.

The most recent active complete MTC frame value may alternatively be encoded by an active finished Full Frame command. In this case, the Q header bit MUST be set to 0, and the COMPLETE field MUST have format shown in Figure B.4.3. The HR, MN, SC, and FR fields correspond to the hr, mn, sc, and fr data octets of the Full Frame command.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |      HR       |      MN       |      SC       |      FR       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.4.3 -- COMPLETE field format, Q = 0

B.4.1. Partial Frames

The most recent active session history command that encodes MTC frame value data may be a Quarter Frame command other than a forward-moving 0xF1 0x7n command (which completes a frame value for forward tape movement) or a reverse-moving 0xF1 0x1n command (which completes a frame value for reverse tape movement).

We consider this type of Quarter Frame command to be associated with a partial frame value. The Quarter Frame sequence that defines a partial frame value MUST either start at Message Type 0 and increment contiguously to an intermediate Message Type less than 7, or start at Message Type 7 and decrement contiguously to an intermediate Message type greater than 0. A Quarter Frame command sequence that does not follow this pattern is not associated with a partial frame value.

Chapter F MUST include a PARTIAL field if the most recent active command in the checkpoint history that encodes MTC frame value data is a Quarter Frame command that is associated with a partial frame value. Otherwise, Chapter F MUST NOT include a PARTIAL field.

The partial frame value consists of the data (lower) nibbles of the Quarter Frame command sequence. The PARTIAL field codes the partial frame value, using the format shown in Figure B.4.2. Message Type fields that are not associated with a Quarter Frame command MUST be set to 0.

The POINT header field indicates the Message Type fields in the PARTIAL field code valid data. If P = 1, the POINT field MUST encode the unsigned integer value formed by the lower 3 bits of the upper nibble of the data value of the most recent active Quarter Frame command in the session history. If D = 0 and P = 1, POINT MUST take on a value in the range 0-6. If D = 1 and P = 1, POINT MUST take on a value in the range 1-7.

If D = 0, MT fields (Figure B.4.2) in the inclusive range from 0 up to and including the POINT value encode the partial frame value. If D = 1, MT fields in the inclusive range from 7 down to and including the POINT value encode the partial frame value. Note that, unlike the COMPLETE field encoding, senders MUST NOT add a 2-frame offset to the partial frame value encoded in PARTIAL.

For the default semantics, if a recovery journal contains Chapter F, and if the session history codes a legal [MIDI] series of Quarter Frame and Full Frame commands, the chapter always contains a COMPLETE or a PARTIAL field (and may contain both fields). Thus, a one-octet Chapter F (C = P = 0) always codes the presence of an illegal command sequence in the session history (under some conditions, the C = 1, P = 0 condition may also code the presence of an illegal command sequence). The illegal command sequence conditions are transient in nature and usually indicate that a Quarter Frame command sequence began with an intermediate Message Type.

B.5. System Chapter X: System Exclusive

This appendix describes Chapter X, the system chapter for MIDI System Exclusive (SysEx) commands (0xF0). Readers may wish to review the Appendix A.1 definition of "finished/unfinished commands" before reading this appendix.

Chapter X consists of a list of one or more command logs. Each log in the list codes information about a specific finished or unfinished SysEx command that appears in the session history. The system journal MUST contain Chapter X if the rules defined in Appendix B.5.2 require that one or more logs appear in the list.

The log list is not preceded by a header. Instead, each log implicitly encodes its own length. Given the length of the N'th list log, the presence of the (N+1)'th list log may be inferred from the LENGTH field of the system journal header (Figure 10 in Section 5 of the main text). The log list MUST obey the oldest-first ordering rule (defined in Appendix A.1).

B.5.1. Chapter Format

Figure B.5.1 shows the bitfield format for the Chapter X command log.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|T|C|F|D|L|STA|    TCOUNT     |     COUNT     |  FIRST ...    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  DATA ...                                                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure B.5.1 -- Chapter X command log format

A Chapter X command log consists of a 1-octet header, followed by the optional TCOUNT, COUNT, FIRST, and DATA fields.

The T, C, F, and D header bits act as a Table of Contents (TOC) for the log. If T is set to 1, the 1-octet TCOUNT field appears in the log. If C is set to 1, the 1-octet COUNT field appears in the log. If F is set to 1, the variable-length FIRST field appears in the log. If D is set to 1, the variable-length DATA field appears in the log.

The L header bit sets the coding tool for the log. We define the log coding tools in Appendix B.5.2.

The STA field codes the status of the command coded by the log. The 2-bit STA value is interpreted as an unsigned integer. If STA is 0, the log codes an unfinished command. Non-zero STA values code different classes of finished commands. An STA value of 1 codes a cancelled command, an STA value of 2 codes a command that uses the "dropped F7" construction, and an STA value of 3 codes all other finished commands. Section 3.2 in the main text describes cancelled and "dropped F7" commands.

The S bit (Appendix A.1) of the first log in the list acts as the S bit for Chapter X. For the other logs in the list, the S bit refers to the log itself. The value of the "phantom" S bit associated with the first log is defined by the following rules:

- If the list codes one log, the phantom S-bit value is the same as the Chapter X S-bit value.
- If the list codes multiple logs, the phantom S-bit value is the logical OR of the S-bit value of the first and second command logs in the list.

In all other respects, the S bit follows the semantics defined in Appendix A.1.

The FIRST field (present if F = 1) encodes a variable-length unsigned integer value that sets the coverage of the DATA field.

The FIRST field (present if F = 1) encodes a variable-length unsigned integer value that specifies which SysEx data bytes are encoded in the DATA field of the log. The FIRST field consists of an octet whose most-significant bit is set to 0, optionally preceded by one or more octets whose most-significant bit is set to 1. The algorithm shown in Figure B.5.2 decodes this format into an unsigned integer, to yield the value dec(FIRST). FIRST uses a variable-length encoding because dec(FIRST) references a data octet in a SysEx command, and a SysEx command may contain an arbitrary number of data octets.

```
One-Octet FIRST value:

   Encoded form: 0ddddddd
   Decoded form: 00000000 00000000 00000000 0ddddddd

Two-Octet FIRST value:

   Encoded form: 1ccccccc 0ddddddd
   Decoded form: 00000000 00000000 00cccccc cddddddd

Three-Octet FIRST value:

   Encoded form: 1bbbbbbb 1ccccccc 0ddddddd
   Decoded form: 00000000 000bbbbb bbcccccc cddddddd

Four-Octet FIRST value:

   Encoded form: 1aaaaaaa 1bbbbbbb 1ccccccc 0ddddddd
   Decoded form: 0000aaaa aaabbbbb bbcccccc cddddddd
```

Figure B.5.2 -- Decoding FIRST field formats

The DATA field (present if D = 1) encodes a modified version of the data octets of the SysEx command coded by the log. Status octets MUST NOT be coded in the DATA field.

If F = 0, the DATA field begins with the first data octet of the SysEx command and includes all subsequent data octets for the command that appear in the session history. If F = 1, the DATA field begins with the (dec(FIRST) + 1)’th data octet of the SysEx command and includes all subsequent data octets for the command that appear in the session history. Note that the word "command" in the descriptions above refers to the original SysEx command as it appears in the source MIDI data stream, not to a particular MIDI list SysEx command segment.

The length of the DATA field is coded implicitly, using the most-significant bit of each octet. The most-significant bit of the final octet of the DATA field MUST be set to 1. The most-significant bit of all other DATA octets MUST be set to 0. This coding method relies on the fact that the most-significant bit of a MIDI data octet is 0 by definition. Apart from this length-coding modification, the DATA field encodes a verbatim copy of all data octets it encodes.

B.5.2. Log Inclusion Semantics

Chapter X offers two tools to protect SysEx commands: the "recency" tool and the "list" tool. The tool definitions use the concept of the "SysEx type" of a command, which we now define.

Each SysEx command instance in a session, excepting MTC Full Frame commands, is said to have a "SysEx type". Types are used in equality comparisons: two SysEx commands in a session are said to have "the same SysEx type" or "different SysEx types".

If efficiency is not a concern, a sender may follow a simple typing rule: every SysEx command in the session history has a different SysEx type, and thus no two commands in the session have the same type.

To improve efficiency, senders MAY implement exceptions to this rule. These exceptions declare that certain sets of SysEx command instances have the same SysEx type. Any command not covered by an exception follows the simple rule. We list exceptions below:

- All commands with identical data octet fields (same number of data octets, same value for each data octet) have the same type. This rule MUST be applied to all SysEx commands in the session, or not at all. Note that the implementation of this exception requires no sender knowledge of the format and semantics of the

SysEx commands in the stream, merely the ability to count and compare octets.

- Two instances of the same command whose semantics set or report the value of the same "parameter" have the same type. The implementation of this exception requires specific knowledge of the format and semantics of SysEx commands. In practice, a sender implementation chooses to support this exception for certain classes of commands (such as the Universal System Exclusive commands defined in [MIDI]). If a sender supports this exception for a particular command in a class (for example, the Universal Real Time System Exclusive message for Master Volume, F0 F7 cc 04 01 vv vv F7, defined in [MIDI]), it MUST support the exception to all instances of this particular command in the session.

We now use this definition of "SysEx type" to define the "recency" tool and the "list" tool for Chapter X.

By default, the Chapter X log list MUST code sufficient information to protect the rendered MIDI performance from indefinite artifacts caused by the loss of all finished or unfinished active SysEx commands that appear in the checkpoint history (excluding finished MTC Full Frame commands, which are coded in Chapter F (Appendix B.4)).

To protect a command of a specific SysEx type with the recency tool, senders MUST code a log in the log list for the most recent finished active instance of the SysEx type that appears in the checkpoint history. Additionally, if an unfinished active instance of the SysEx type appears in the checkpoint history, senders MUST code a log in the log list for the unfinished command instance. The L header bit of both command logs MUST be set to 0.

To protect a command of a specific SysEx type with the list tool, senders MUST code a log in the Chapter X log list for each finished or unfinished active instance of the SysEx type that appears in the checkpoint history. The L header bit of list tool command logs MUST be set to 1.

As a rule, a log REQUIRED by the list or recency tool MUST include a DATA field that codes all data octets that appear in the checkpoint history for the SysEx command instance associated with the log. The FIRST field MAY be used to configure a DATA field that minimally meets this requirement.

An exception to this rule applies to cancelled commands (defined in Section 3.2). REQUIRED command logs associated with cancelled commands MAY be coded with no DATA field. However, if DATA appears in the log, DATA MUST code all data octets that appear in the checkpoint history for the command associated with the log.

As defined by the preceding text in this section, by default all finished or unfinished active SysEx commands that appear in the checkpoint history (excluding finished MTC Full Frame commands) MUST be protected by the list tool or the recency tool.

For some MIDI source streams, this default yields a Chapter X whose size is too large. For example, imagine that a sender begins to transcode a SysEx command with 10,000 data octets onto a UDP RTP stream "on the fly", by sending SysEx command segments as soon as data octets are delivered by the MIDI source. After 1000 octets have been sent, the expansion of Chapter X yields an RTP packet that is too large to fit in the Maximum Transmission Unit (MTU) for the stream.

In this situation, if a sender uses the closed-loop sending policy for SysEx commands, the RTP packet size may always be capped by stalling the stream. In a stream stall, once the packet reaches a maximum size, the sender refrains from sending new packets with non-empty MIDI Command Sections until receiver feedback permits the trimming of Chapter X. If the stream permits arbitrary commands to appear between SysEx segments (selectable during configuration using the tools defined in Appendix C.1), the sender may stall the SysEx segment stream but continue to code other commands in the MIDI list.

Stalls are a workable but sub-optimal solution to Chapter X size issues. As an alternative to stalls, senders SHOULD take preemptive action during session configuration to reduce the anticipated size of Chapter X, using the methods described below:

- o Partitioned transport. Appendix C.5 provides tools for sending a MIDI name space over several RTP streams. Senders may use these tools to map a MIDI source into a low-latency UDP RTP stream (for channel commands and short SysEx commands) and a reliable [RFC4571] TCP stream (for bulk-data SysEx commands). The cm_unused and cm_used parameters (Appendix C.1) may be used to communicate the nature of the SysEx command partition. As TCP is reliable, the RTP MIDI TCP stream would not use the recovery journal. To minimize transmission latency for short SysEx commands, senders may begin segmental transmission for all SysEx commands over the UDP stream and then cancel the UDP transmission of long commands (using tools described in Section 3.2) and resend the commands over the TCP stream.

- o Selective protection. Journal protection may not be necessary for all SysEx commands in a stream. The ch_never parameter (Appendix C.2) may be used to communicate which SysEx commands are excluded from Chapter X.

B.5.3. TCOUNT and COUNT Fields

If the T header bit is set to 1, the 8-bit TCOUNT field appears in the command log. If the C header bit is set to 1, the 8-bit COUNT field appears in the command log. TCOUNT and COUNT are interpreted as unsigned integers.

The TCOUNT field codes the total number of SysEx commands of the SysEx type coded by the log that appear in the session history, at the moment after the (finished or unfinished) command coded by the log enters the session history.

The COUNT field codes the total number of SysEx commands that appear in the session history, excluding commands that are excluded from Chapter X via the ch_never parameter (Appendix C.2), at the moment after the (finished or unfinished) command coded by the log enters the session history.

Command counting for TCOUNT and COUNT uses modulo-256 arithmetic. MTC Full Frame command instances (Appendix B.4) are included in command counting if the TCOUNT and COUNT definitions warrant their inclusion, as are cancelled commands (Section 3.2).

Senders use the TCOUNT and COUNT fields to track the identity and (for TCOUNT) the sequence position of a command instance. Senders MUST use the TCOUNT or COUNT fields if identity or sequence information is necessary to protect the command type coded by the log.

If a sender uses the COUNT field in a session, the final command log in every Chapter X in the stream MUST code the COUNT field. This rule lets receivers resynchronize the COUNT value after a packet loss.

C. Session Configuration Tools

In Sections 6.1-2 of the main text, we show session descriptions for minimal native and mpeg4-generic RTP MIDI streams. Minimal streams lack the flexibility to support some applications. In this appendix, we describe how to customize stream behavior through the use of the payload format parameters.

The appendix begins with 6 sections, each devoted to parameters that affect a particular aspect of stream behavior:

- o Appendix C.1 describes the stream subsetting system (cm_unused and cm_used).
- o Appendix C.2 describes the journalling system (ch_anchor, ch_default, ch_never, j_sec, j_update).
- o Appendix C.3 describes MIDI command timestamp semantics (linerate, mperiod, octpos, tsmode).
- o Appendix C.4 describes the temporal duration ("media time") of an RTP MIDI packet (guardtime, rtp_maxptime, rtp_ptime).
- o Appendix C.5 concerns stream description (musicport).
- o Appendix C.6 describes MIDI rendering (chanmask, cid, inline, multimode, render, rinit, subrender, smf_cid, smf_info, smf_inline, smf_url, url).

The parameters listed above may optionally appear in session descriptions of RTP MIDI streams. If these parameters are used in an SDP session description, the parameters appear on an fmtp attribute line. This attribute line applies to the payload type associated with the fmtp line.

The parameters listed above add extra functionality ("features") to minimal RTP MIDI streams. In Appendix C.7, we show how to use these features to support two classes of applications: content-streaming using RTSP (Appendix C.7.1) and network musical performance using SIP (Appendix C.7.2).

The participants in a multimedia session MUST share a common view of all of the RTP MIDI streams that appear in an RTP session, as defined by a single media (m=) line. In some RTP MIDI applications, the "common view" restriction makes it difficult to use sendrecv streams (all parties send and receive), as each party has its own requirements. For example, a two-party network musical performance application may wish to customize the renderer on each host to match the CPU performance of the host [NMP].

We solve this problem by using two RTP MIDI streams -- one sendonly, one recvonly -- in lieu of one sendrecv stream. The data flows in the two streams travel in opposite directions, to control receivers configured to use different renderers. In the third example in Appendix C.5, we show how the musicport parameter may be used to define virtual sendrecv streams.

As a general rule, the RTP MIDI protocol does not handle parameter changes during a session well, because the parameters describe heavyweight or stateful configuration that is not easily changed once a session has begun. Thus, parties SHOULD NOT expect that parameter change requests during a session will be accepted by other parties. However, implementors SHOULD support in-session parameter changes that are easy to handle (for example, the guardtime parameter defined in Appendix C.4) and SHOULD be capable of accepting requests for changes of those parameters, as received by its session management protocol (for example, re-offers in SIP [RFC3264]).

Appendix D defines the Augmented Backus-Naur Form (ABNF, [RFC4234]) syntax for the payload parameters. Section 11 provides information to the Internet Assigned Numbers Authority (IANA) on the media types and parameters defined in this document.

Appendix C.6.5 defines the media type "audio/asc", a stored object for initializing mpeg4-generic renderers. As described in Appendix C.6, the audio/asc media type is assigned to the "rinit" parameter to specify an initialization data object for the default mpeg4-generic renderer. Note that RTP stream semantics are not defined for "audio/asc". Therefore, the "asc" subtype MUST NOT appear on the rtpmap line of a session description.

C.1. Configuration Tools: Stream Subsetting

As defined in Section 3.2 in the main text, the MIDI list of an RTP MIDI packet may encode any MIDI command that may legally appear on a MIDI 1.0 DIN cable.

In this appendix, we define two parameters (cm_unused and cm_used) that modify this default condition, by excluding certain types of MIDI commands from the MIDI list of all packets in a stream. For example, if a multimedia session partitions a MIDI name space into two RTP MIDI streams, the parameters may be used to define which commands appear in each stream.

In this appendix, we define a simple language for specifying MIDI command types. If a command type is assigned to cm_unused, the commands coded by the string MUST NOT appear in the MIDI list. If a command type is assigned to cm_used, the commands coded by the string MAY appear in the MIDI list.

The parameter list may code multiple assignments to cm_used and cm_unused. Assignments have a cumulative effect and are applied in the order of appearance in the parameter list. A later assignment of a command type to the same parameter expands the scope of the earlier assignment. A later assignment of a command type to the opposite parameter cancels (partially or completely) the effect of an earlier assignment.

To initialize the stream subsetting system, "implicit" assignments to cm_unused and cm_used are processed before processing the actual assignments that appear in the parameter list. The System Common undefined commands (0xF4, 0xF5) and the System Real-Time Undefined commands (0xF9, 0xFD) are implicitly assigned to cm_unused. All other command types are implicitly assigned to cm_used.

Note that the implicit assignments code the default behavior of an RTP MIDI stream as defined in Section 3.2 in the main text (namely, that all commands that may legally appear on a MIDI 1.0 DIN cable may appear in the stream). Also note that assignments of the System Common undefined commands (0xF4, 0xF5) apply to the use of these commands in the MIDI source command stream, not the special use of 0xF4 and 0xF5 in SysEx segment encoding defined in Section 3.2 in the main text.

As a rule, parameter assignments obey the following syntax (see Appendix D for ABNF):

```
<parameter> = [channel list]<command-type list>[field list]
```

The command-type list is mandatory; the channel and field lists are optional.

The command-type list specifies the MIDI command types for which the parameter applies. The command-type list is a concatenated sequence of one or more of the letters (ABCFGHJKMNPQTVWXYZ). The letters code the following command types:

- A: Poly Aftertouch (0xA)
- B: System Reset (0xFF)
- C: Control Change (0xB)
- F: System Time Code (0xF1)
- G: System Tune Request (0xF6)
- H: System Song Select (0xF3)
- J: System Common Undefined (0xF4)
- K: System Common Undefined (0xF5)
- N: NoteOff (0x8), NoteOn (0x9)
- P: Program Change (0xC)
- Q: System Sequencer (0xF2, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC)
- T: Channel Aftertouch (0xD)
- V: System Active Sense (0xFE)
- W: Pitch Wheel (0xE)

- X: SysEx (0xF0)
- Y: System Real-Time Undefined (0xF9)
- Z: System Real-Time Undefined (0xFD)

In addition to the letters above, the letter M may also appear in the command-type list. The letter M refers to the MIDI parameter system (see definition in Appendix A.1 and in [MIDI]). An assignment of M to cm_unused codes that no RPN or NRPN transactions may appear in the MIDI list.

Note that if cm_unused is assigned the letter M, Control Change (0xB) commands for the controller numbers in the standard controller assignment might still appear in the MIDI list. For an explanation, see Appendix A.3.4 for a discussion of the "general-purpose" use of parameter system controller numbers.

In the text below, rules that apply to "MIDI voice channel commands" also apply to the letter M.

The letters in the command-type list MUST be uppercase and MUST appear in alphabetical order. Letters other than (ABCFGHJKMNPQTVWXYZ) that appear in the list MUST be ignored.

For MIDI voice channel commands, the channel list specifies the MIDI channels for which the parameter applies. If no channel list is provided, the parameter applies to all MIDI channels (0-15). The channel list takes the form of a list of channel numbers (0 through 15) and dash-separated channel number ranges (i.e., 0-5, 8-12, etc). Dots (i.e., "." characters) separate elements in the channel list.

Recall that System commands do not have a MIDI channel associated with them. Thus, for most command-type letters that code System commands (B, F, G, H, J, K, Q, V, Y, and Z), the channel list is ignored.

For the command-type letter X, the appearance of certain numbers in the channel list codes special semantics.

- The digit 0 codes that SysEx "cancel" sublists (Section 3.2 in the main text) MUST NOT appear in the MIDI list.

- The digit 1 codes that cancel sublists MAY appear in the MIDI list (the default condition).

- The digit 2 codes that commands other than System Real-time MIDI commands MUST NOT appear between SysEx command segments in the MIDI list (the default condition).

- The digit 3 codes that any MIDI command type may appear between SysEx command segments in the MIDI list, with the exception of the segmented encoding of a second SysEx command (verbatim SysEx commands are OK).

For command-type X, the channel list MUST NOT contain both digits 0 and 1, and it MUST NOT contain both digits 2 and 3. For command-type X, channel list numbers other than the numbers defined above are ignored. If X does not have a channel list, the semantics marked "the default condition" in the list above apply.

The syntax for field lists in a parameter assignment follows the syntax for channel lists. If no field list is provided, the parameter applies to all controller or note numbers.

For command-type C (Control Change), the field list codes the controller numbers (0-255) for which the parameter applies.

For command-type M (Parameter System), the field list codes the Registered Parameter Numbers (RPNs) and Non-Registered Parameter Numbers (NRPNs) for which the parameter applies. The number range 0-16383 specifies RPNs, the number range 16384-32767 specifies NRPNs (16384 corresponds to NRPN 0, 32767 corresponds to NRPN 16383).

For command-types N (NoteOn and NoteOff) and A (Poly Aftertouch), the field list codes the note numbers for which the parameter applies.

For command-types J and K (System Common Undefined), the field list consists of a single digit, which specifies the number of data octets that follow the command octet.

For command-type X (SysEx), the field list codes the number of data octets that may appear in a SysEx command. Thus, the field list 0-255 specifies SysEx commands with 255 or fewer data octets, the field list 256-4294967295 specifies SysEx commands with more than 255 data octets but excludes commands with 255 or fewer data octets, and the field list 0 excludes all commands.

A secondary parameter assignment syntax customizes command-type X (see Appendix D for complete ABNF):

```
<parameter> = "__" <h-list> ["_" <h-list>] "__"
```

The assignment defines the class of SysEx commands that obeys the semantics of the assigned parameter. The command class is specified by listing the permitted values of the first N data octets that follow the SysEx 0xF0 command octet. Any SysEx command whose first N data octets match the list is a member of the class.

Each <h-list> defines a data octet of the command, as a dot-separated (".") list of one or more hexadecimal constants (such as "7F") or dash-separated hexadecimal ranges (such as "01-1F"). Underscores ("_") separate each <h-list>. Double-underscores ("__") delineate the data octet list.

Using this syntax, each assignment specifies a single SysEx command class. Session descriptions may use several assignments to cm_used and cm_unused to specify complex behaviors.

The example session description below illustrates the use of the stream subsetting parameters:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP6 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP6 2001:DB80::7F2E:172A:1E24
a=rtpmap:96 rtp-midi/44100
a=fmtp:96 cm_unused=ACGHJKNMPTVWXYZ; cm_used=__7F_00-7F_01_01__
```

The session description configures the stream for use in clock applications. All voice channels are unused, as are all System Commands except those used for MIDI Time Code (command-type F, and the Full Frame SysEx command that is matched by the string assigned to cm_used), the System Sequencer commands (command-type Q), and System Reset (command-type B).

C.2. Configuration Tools: The Journalling System

In this appendix, we define the payload format parameters that configure stream journalling and the recovery journal system.

The j_sec parameter (Appendix C.2.1) sets the journalling method for the stream. The j_update parameter (Appendix C.2.2) sets the recovery journal sending policy for the stream. Appendix C.2.2 also defines the sending policies of the recovery journal system.

Appendix C.2.3 defines several parameters that modify the recovery journal semantics. These parameters change the default recovery journal semantics as defined in Section 5 and Appendices A-B.

The journalling method for a stream is set at the start of a session and MUST NOT be changed thereafter. This requirement forbids changes to the j_sec parameter once a session has begun.

A related requirement, defined in the appendix sections below, forbids the acceptance of parameter values that would violate the recovery journal mandate. In many cases, a change in one of the parameters defined in this appendix during an ongoing session would result in a violation of the recovery journal mandate for an implementation; in this case, the parameter change MUST NOT be accepted.

C.2.1. The j_sec Parameter

Section 2.2 defines the default journalling method for a stream. Streams that use unreliable transport (such as UDP) default to using the recovery journal. Streams that use reliable transport (such as TCP) default to not using a journal.

The parameter j_sec may be used to override this default. This memo defines two symbolic values for j_sec: "none", to indicate that all stream payloads MUST NOT contain a journal section, and "recj", to indicate that all stream payloads MUST contain a journal section that uses the recovery journal format.

For example, the j_sec parameter might be set to "none" for a UDP stream that travels between two hosts on a local network that is known to provide reliable datagram delivery.

The session description below configures a UDP stream that does not use the recovery journal:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP4 192.0.2.94
a=rtpmap:96 rtp-midi/44100
a=fmtp:96 j_sec=none
```

Other IETF standards-track documents may define alternative journal formats. These documents MUST define new symbolic values for the j_sec parameter to signal the use of the format.

Parties MUST NOT accept a j_sec value that violates the recovery journal mandate (see Section 4 for details). If a session description uses a j_sec value unknown to the recipient, the recipient MUST NOT accept the description.

Special j_sec issues arise when sessions are managed by session management tools (like RTSP, [RFC2326]) that use SDP for "declarative usage" purposes (see the preamble of Section 6 for details). For these session management tools, SDP does not code transport details (such as UDP or TCP) for the session. Instead, server and client negotiate transport details via other means (for RTSP, the SETUP method).

In this scenario, the use of the j_sec parameter may be ill-advised, as the creator of the session description may not yet know the transport type for the session. In this case, the session description SHOULD configure the journalling system using the parameters defined in the remainder of Appendix C.2, but it SHOULD NOT use j_sec to set the journalling status. Recall that if j_sec does not appear in the session description, the default method for choosing the journalling method is in effect (no journal for reliable transport, recovery journal for unreliable transport).

However, in declarative usage situations where the creator of the session description knows that journalling is always required or never required, the session description SHOULD use the j_sec parameter.

C.2.2. The j_update Parameter

In Section 4, we use the term "sending policy" to describe the method a sender uses to choose the checkpoint packet identity for each recovery journal in a stream. In the sub-sections that follow, we normatively define three sending policies: anchor, closed-loop, and open-loop.

As stated in Section 4, the default sending policy for a stream is the closed-loop policy. The j_update parameter may be used to override this default.

We define three symbolic values for j_update: "anchor", to indicate that the stream uses the anchor sending policy, "open-loop", to indicate that the stream uses the open-loop sending policy, and "closed-loop", to indicate that the stream uses the closed-loop sending policy. See Appendix C.2.3 for examples session descriptions that use the j_update parameter.

Parties MUST NOT accept a j_update value that violates the recovery journal mandate (Section 4).

Other IETF standards-track documents may define additional sending policies for the recovery journal system. These documents MUST define new symbolic values for the j_update parameter to signal the use of the new policy. If a session description uses a j_update value unknown to the recipient, the recipient MUST NOT accept the description.

C.2.2.1. The anchor Sending Policy

In the anchor policy, the sender uses the first packet in the stream as the checkpoint packet for all packets in the stream. The anchor policy satisfies the recovery journal mandate (Section 4), as the checkpoint history always covers the entire stream.

The anchor policy does not require the use of the RTP control protocol (RTCP, [RFC3550]) or other feedback from receiver to sender. Senders do not need to take special actions to ensure that received streams start up free of artifacts, as the recovery journal always covers the entire history of the stream. Receivers are relieved of the responsibility of tracking the changing identity of the checkpoint packet, because the checkpoint packet never changes.

The main drawback of the anchor policy is bandwidth efficiency. Because the checkpoint history covers the entire stream, the size of the recovery journals produced by this policy usually exceeds the journal size of alternative policies. For single-channel MIDI data streams, the bandwidth overhead of the anchor policy is often acceptable (see Appendix A.4 of [NMP]). For dense streams, the closed-loop or open-loop policies may be more appropriate.

C.2.2.2. The closed-loop Sending Policy

The closed-loop policy is the default policy of the recovery journal system. For each packet in the stream, the policy lets senders choose the smallest possible checkpoint history that satisfies the recovery journal mandate. As smaller checkpoint histories generally yield smaller recovery journals, the closed-loop policy reduces the bandwidth of a stream, relative to the anchor policy.

The closed-loop policy relies on feedback from receiver to sender. The policy assumes that a receiver periodically informs the sender of the highest sequence number it has seen so far in the stream, coded in the 32-bit extension format defined in [RFC3550]. For RTCP, receivers transmit this information in the Extended Highest Sequence Number Received (EHSNR) field of Receiver Reports. RTCP Sender or Receiver Reports MUST be sent by any participant in a session with closed loop sending policy, unless another feedback mechanism has been agreed upon.

The sender may safely use receiver sequence number feedback to guide checkpoint history management, because Section 4 requires that receivers repair indefinite artifacts whenever a packet loss event occur.

We now normatively define the closed-loop policy. At the moment a sender prepares an RTP packet for transmission, the sender is aware of R >= 0 receivers for the stream. Senders may become aware of a receiver via RTCP traffic from the receiver, via RTP packets from a paired stream sent by the receiver to the sender, via messages from a session management tool, or by other means. As receivers join and leave a session, the value of R changes.

Each known receiver k (1 <= k <= R) is associated with a 32-bit extended packet sequence number M(k), where the extension reflects the sequence number rollover count of the sender.

If the sender has received at least one feedback report from receiver k, M(k) is the most recent report of the highest RTP packet sequence number seen by the receiver, normalized to reflect the rollover count of the sender.

If the sender has not received a feedback report from the receiver, M(k) is the extended sequence number of the last packet the sender transmitted before it became aware of the receiver. If the sender became aware of this receiver before it sent the first packet in the stream, M(k) is the extended sequence number of the first packet in the stream.

Given this definition of M(), we now state the closed-loop policy. When preparing a new packet for transmission, a sender MUST choose a checkpoint packet with extended sequence number N, such that M(k) >= (N - 1) for all k, 1 <= k <= R, where R >= 1. The policy does not restrict sender behavior in the R == 0 (no known receivers) case.

Under the closed-loop policy as defined above, a sender may transmit packets whose checkpoint history is shorter than the session history (as defined in Appendix A.1). In this event, a new receiver that joins the stream may experience indefinite artifacts.

For example, if a Control Change (0xB) command for Channel Volume (controller number 7) was sent early in a stream, and later a new receiver joins the session, the closed-loop policy may permit all packets sent to the new receiver to use a checkpoint history that does not include the Channel Volume Control Change command. As a result, the new receiver experiences an indefinite artifact, and plays all notes on a channel too loudly or too softly.

To address this issue, the closed-loop policy states that whenever a sender becomes aware of a new receiver, the sender MUST determine if the receiver would be subject to indefinite artifacts under the closed-loop policy. If so, the sender MUST ensure that the receiver starts the session free of indefinite artifacts.

For example, to solve the Channel Volume issue described above, the sender may code the current state of the Channel Volume controller numbers in the recovery journal Chapter C, until it receives the first RTCP RR report that signals that a packet containing this Chapter C has been received.

In satisfying this requirement, senders MAY infer the initial MIDI state of the receiver from the session description. For example, the stream example in Section 6.2 has the initial state defined in [MIDI] for General MIDI.

In a unicast RTP session, a receiver may safely assume that the sender is aware of its presence of a receiver from the first packet sent in the RTP stream. However, in other types of RTP sessions (multicast, conference focus, RTP translator/mixer), a receiver is often not able to determine if the sender is initially aware of its presence as a receiver.

To address this issue, the closed-loop policy states that if a receiver participates in a session where it may have access to a stream whose sender is not aware of the receiver, the receiver MUST take actions to ensure that its rendered MIDI performance does not contain indefinite artifacts. These protections will be necessarily incomplete. For example, a receiver may monitor the Checkpoint Packet Seqnum for uncovered loss events, and "err on the side of caution" with respect to handling stuck notes due to lost MIDI NoteOff commands, but the receiver is not able to compensate for the lack of Channel Volume initialization data in the recovery journal.

The receiver MUST NOT discontinue these protective actions until it is certain that the sender is aware of its presence. If a receiver is not able to ascertain sender awareness, the receiver MUST continue these protective actions for the duration of the session.

Note that in a multicast session where all parties are expected to send and receive, the reception of RTCP receiver reports from the sender about the RTP stream a receiver is multicasting is evidence of the sender’s awareness that the RTP stream multicast by the sender is being monitored by the receiver. Receivers may also obtain sender awareness evidence from session management tools, or by other means. In practice, ongoing observation of the Checkpoint Packet Seqnum to determine if the sender is taking actions to prevent loss events for a receiver is a good indication of sender awareness, as is the sudden appearance of recovery journal chapters with numerous Control Change controller data that was not foreshadowed by recent commands coded in the MIDI list shortly after sending an RTCP RR.

The final set of normative closed-loop policy requirements concern how senders and receivers handle unplanned disruptions of RTCP feedback from a receiver to a sender. By "unplanned", we refer to disruptions that are not due to the signalled termination of an RTP stream, via an RTCP BYE or via session management tools.

As defined earlier in this section, the closed-loop policy states that a sender MUST choose a checkpoint packet with extended sequence number N, such that M(k) >= (N - 1) for all k, 1 <= k <= R, where R >= 1. If the sender has received at least one feedback report from receiver k, M(k) is the most recent report of the highest RTP packet sequence number seen by the receiver, normalized to reflect the rollover count of the sender.

If this receiver k stops sending feedback to the sender, the M(k) value used by the sender reflects the last feedback report from the receiver. As time progresses without feedback from receiver k, this fixed M(k) value forces the sender to increase the size of the checkpoint history, and thus increases the bandwidth of the stream.

At some point, the sender may need to take action in order to limit the bandwidth of the stream. In most envisioned uses of RTP MIDI, long before this point is reached, the SSRC time-out mechanism defined in [RFC3550] will remove the uncooperative receiver from the session (note that the closed-loop policy does not suggest or require any special sender behavior upon an SSRC time-out, other than the sender actions related to changing R, described earlier in this section).

However, in rare situations, the bandwidth of the stream (due to a lack of feedback reports from the sender) may become too large to continue sending the stream to the receiver before the SSRC time-out occurs for the receiver. In this case, the closed-loop policy states that the sender should invoke the SSRC time-out for the receiver early.

We now discuss receiver responsibilities in the case of unplanned disruptions of RTCP feedback from receiver to sender.

In the unicast case, if a sender invokes the SSRC time-out mechanism for a receiver, the receiver stops receiving packets from the sender. The sender behavior imposed by the guardtime parameter (Appendix

C.4.2) lets the receiver conclude that an SSRC time-out has occurred in a reasonable time period.

In this case of a time-out, a receiver MUST keep sending RTCP feedback, in order to re-establish the RTP flow from the sender. Unless the receiver expects a prompt recovery of the RTP flow, the receiver MUST take actions to ensure that the rendered MIDI performance does not exhibit "very long transient artifacts" (for example, by silencing NoteOns to prevent stuck notes) while awaiting reconnection of the flow.

In the multicast case, if a sender invokes the SSRC time-out mechanism for a receiver, the receiver may continue to receive packets, but the sender will no longer be using the M(k) feedback from the receiver to choose each checkpoint packet. If the receiver does not have additional information that precludes an SSRC time-out (such as RTCP Receiver Reports from the sender about an RTP stream the receiver is multicasting back to the sender), the receiver MUST monitor the Checkpoint Packet Seqnum to detect an SSRC time-out. If an SSRC time-out is detected, the receiver MUST follow the instructions for SSRC time-outs described for the unicast case above.

Finally, we note that the closed-loop policy is suitable for use in RTP/RTCP sessions that use multicast transport. However, aspects of the closed-loop policy do not scale well to sessions with large numbers of participants. The sender state scales linearly with the number of receivers, as the sender needs to track the identity and M(k) value for each receiver k. The average recovery journal size is not independent of the number of receivers, as the RTCP reporting interval backoff slows down the rate of a full update of M(k) values. The backoff algorithm may also increase the amount of ancillary state used by implementations of the normative sender and receiver behaviors defined in Section 4.

C.2.2.3. The open-loop Sending Policy

The open-loop policy is suitable for sessions that are not able to implement the receiver-to-sender feedback required by the closed-loop policy, and that are also not able to use the anchor policy because of bandwidth constraints.

The open-loop policy does not place constraints on how a sender chooses the checkpoint packet for each packet in the stream. In the absence of such constraints, a receiver may find that the recovery journal in the packet that ends a loss event has a checkpoint history that does not cover the entire loss event. We refer to loss events of this type as uncovered loss events.

To ensure that uncovered loss events do not compromise the recovery journal mandate, the open-loop policy assigns specific recovery tasks to senders, receivers, and the creators of session descriptions. The underlying premise of the open-loop policy is that the indefinite artifacts produced during uncovered loss events fall into two classes.

One class of artifacts is recoverable indefinite artifacts. Receivers are able to repair recoverable artifacts that occur during an uncovered loss event without intervention from the sender, at the potential cost of unpleasant transient artifacts.

For example, after an uncovered loss event, receivers are able to repair indefinite artifacts due to NoteOff (0x8) commands that may have occurred during the loss event, by executing NoteOff commands for all active NoteOns commands. This action causes a transient artifact (a sudden silent period in the performance), but ensures that no stuck notes sound indefinitely. We refer to MIDI commands that are amenable to repair in this fashion as recoverable MIDI commands.

A second class of artifacts is unrecoverable indefinite artifacts. If this class of artifact occurs during an uncovered loss event, the receiver is not able to repair the stream.

For example, after an uncovered loss event, receivers are not able to repair indefinite artifacts due to Control Change (0xB) Channel Volume (controller number 7) commands that have occurred during the loss event. A repair is impossible because the receiver has no way of determining the data value of a lost Channel Volume command. We refer to MIDI commands that are fragile in this way as unrecoverable MIDI commands.

The open-loop policy does not specify how to partition the MIDI command set into recoverable and unrecoverable commands. Instead, it assumes that the creators of the session descriptions are able to come to agreement on a suitable recoverable/unrecoverable MIDI command partition for an application.

Given these definitions, we now state the normative requirements for the open-loop policy.

In the open-loop policy, the creators of the session description MUST use the ch_anchor parameter (defined in Appendix C.2.3) to protect all unrecoverable MIDI command types from indefinite artifacts, or alternatively MUST use the cm_unused parameter (defined in Appendix

C.1) to exclude the command types from the stream. These options act to shield command types from artifacts during an uncovered loss event.

In the open-loop policy, receivers MUST examine the Checkpoint Packet Seqnum field of the recovery journal header after every loss event, to check if the loss event is an uncovered loss event. Section 5 shows how to perform this check. If an uncovered loss event has occurred, a receiver MUST perform indefinite artifact recovery for all MIDI command types that are not shielded by ch_anchor and cm_unused parameter assignments in the session description.

The open-loop policy does not place specific constraints on the sender. However, the open-loop policy works best if the sender manages the size of the checkpoint history to ensure that uncovered losses occur infrequently, by taking into account the delay and loss characteristics of the network. Also, as each checkpoint packet change incurs the risk of an uncovered loss, senders should only move the checkpoint if it reduces the size of the journal.

C.2.3. Recovery Journal Chapter Inclusion Parameters

The recovery journal chapter definitions (Appendices A-B) specify under what conditions a chapter MUST appear in the recovery journal. In most cases, the definition states that if a certain command appears in the checkpoint history, a certain chapter type MUST appear in the recovery journal to protect the command.

In this section, we describe the chapter inclusion parameters. These parameters modify the conditions under which a chapter appears the journal. These parameters are essential to the use of the open-loop policy (Appendix C.2.2.3) and may also be used to simplify implementations of the closed-loop (Appendix C.2.2.2) and anchor (Appendix C.2.2.1) policies.

Each parameter represents a type of chapter inclusion semantics. An assignment to a parameter declares which chapters (or chapter subsets) obey the inclusion semantics. We describe the assignment syntax for these parameters later in this section.

A party MUST NOT accept chapter inclusion parameter values that violate the recovery journal mandate (Section 4). All assignments of the subsetting parameters (cm_used and cm_unused) MUST precede the first assignment of a chapter inclusion parameter in the parameter list.

Below, we normatively define the semantics of the chapter inclusion parameters. For clarity, we define the action of parameters on complete chapters. If a parameter is assigned a subset of a chapter, the definition applies only to the chapter subset.

- ch_never. A chapter assigned to the ch_never parameter MUST NOT appear in the recovery journal (Appendix A.4.1-2 defines exceptions to this rule for Chapter M). To signal the exclusion of a chapter from the journal, an assignment to ch_never MUST be made, even if the commands coded by the chapter are assigned to cm_unused. This rule simplifies the handling of commands types that may be coded in several chapters.

- ch_default. A chapter assigned to the ch_default parameter MUST follow the default semantics for the chapter, as defined in Appendices A-B.

- ch_anchor. A chapter assigned to the ch_anchor MUST obey a modified version of the default chapter semantics. In the modified semantics, all references to the checkpoint history are replaced with references to the session history, and all references to the checkpoint packet are replaced with references to the first packet sent in the stream.

Parameter assignments obey the following syntax (see Appendix D for ABNF):

```
<parameter> = [channel list]<chapter list>[field list]
```

The chapter list is mandatory; the channel and field lists are optional. Multiple assignments to parameters have a cumulative effect and are applied in the order of parameter appearance in a media description.

To determine the semantics of a list of chapter inclusion parameter assignments, we begin by assuming an implicit assignment of all channel and system chapters to the ch_default parameter, with the default values for the channel list and field list for each chapter that are defined below.

We then interpret the semantics of the actual parameter assignments, using the rules below.

A later assignment of a chapter to the same parameter expands the scope of the earlier assignment. In most cases, a later assignment of a chapter to a different parameter cancels (partially or completely) the effect of an earlier assignment.

The chapter list specifies the channel or system chapters for which the parameter applies. The chapter list is a concatenated sequence of one or more of the letters corresponding to the chapter types (ACDEFMNPQTVWX). In addition, the list may contain one or more of the letters for the sub-chapter types (BGHJKYZ) of System Chapter D.

The letters in a chapter list MUST be uppercase and MUST appear in alphabetical order. Letters other than (ABCDEFGHJKMNPQTVWXYZ) that appear in the chapter list MUST be ignored.

The channel list specifies the channel journals for which this parameter applies; if no channel list is provided, the parameter applies to all channel journals. The channel list takes the form of a list of channel numbers (0 through 15) and dash-separated channel number ranges (i.e., 0-5, 8-12, etc.). Dots (i.e., "." characters) separate elements in the channel list.

Several of the systems chapters may be configured to have special semantics. Configuration occurs by specifying a channel list for the systems channel, using the coding described below (note that MIDI Systems commands do not have a "channel", and thus the original purpose of the channel list does not apply to systems chapters). The expression "the digit N" in the text below refers to the inclusion of N as a "channel" in the channel list for a systems chapter.

For the J and K Chapter D sub-chapters (undefined System Common), the digit 0 codes that the parameter applies to the LEGAL field of the associated command log (Figure B.1.4 of Appendix B.1), the digit 1 codes that the parameter applies to the VALUE field of the command log, and the digit 2 codes that the parameter applies to the COUNT field of the command log.

For the Y and Z Chapter D sub-chapters (undefined System Real-time), the digit 0 codes that the parameter applies to the LEGAL field of the associated command log (Figure B.1.5 of Appendix B.1) and the digit 1 codes that the parameter applies to the COUNT field of the command log.

For Chapter Q (Sequencer State Commands), the digit 0 codes that the parameter applies to the default Chapter Q definition, which forbids the TIME field. The digit 1 codes that the parameter applies to the optional Chapter Q definition, which supports the TIME field.

The syntax for field lists follows the syntax for channel lists. If no field list is provided, the parameter applies to all controller or note numbers. For Chapter C, if no field list is provided, the controller numbers do not use enhanced Chapter C encoding (Appendix A.3.3).

For Chapter C, the field list may take on values in the range 0 to 255. A field value X in the range 0-127 refers to a controller number X, and indicates that the controller number does not use enhanced Chapter C encoding. A field value X in the range 128-255 refers to a controller number "X minus 128" and indicates the controller number does use the enhanced Chapter C encoding.

Assignments made to configure the Chapter C encoding method for a controller number MUST be made to the ch_default or ch_anchor parameters, as assignments to ch_never act to exclude the number from the recovery journal (and thus the indicated encoding method is irrelevant).

A Chapter C field list MUST NOT encode conflicting information about the enhanced encoding status of a particular controller number. For example, values 0 and 128 MUST NOT both be coded by a field list.

For Chapter M, the field list codes the Registered Parameter Numbers (RPNs) and Non-Registered Parameter Numbers (NRPNs) for which the parameter applies. The number range 0-16383 specifies RPNs, the number range 16384-32767 specifies NRPNs (16384 corresponds to NRPN 0, 32767 corresponds to NRPN 16383).

For Chapters N and A, the field list codes the note numbers for which the parameter applies. The note number range specified for Chapter N also applies to Chapter E.

For Chapter E, the digit 0 codes that the parameter applies to Chapter E note logs whose V bit is set to 0, and the digit 1 codes that the parameter applies to note logs whose V bit is set to 1.

For Chapter X, the field list codes the number of data octets that may appear in a SysEx command that is coded in the chapter. Thus, the field list 0-255 specifies SysEx commands with 255 or fewer data octets, the field list 256-4294967295 specifies SysEx commands with more than 255 data octets but excludes commands with 255 or fewer data octets, and the field list 0 excludes all commands.

A secondary parameter assignment syntax customizes Chapter X (see Appendix D for complete ABNF):

```
<parameter> = "__" <h-list> ["_" <h-list>] "__"
```

The assignment defines a class of SysEx commands whose Chapter X coding obeys the semantics of the assigned parameter. The command class is specified by listing the permitted values of the first N data octets that follow the SysEx 0xF0 command octet. Any SysEx command whose first N data octets match the list is a member of the class.

Each <h-list> defines a data octet of the command, as a dot-separated (".") list of one or more hexadecimal constants (such as "7F") or dash-separated hexadecimal ranges (such as "01-1F"). Underscores ("_") separate each <h-list>. Double-underscores ("__") delineate the data octet list.

Using this syntax, each assignment specifies a single SysEx command class. Session descriptions may use several assignments to the same (or different) parameters to specify complex Chapter X behaviors. The ordering behavior of multiple assignments follows the guidelines for chapter parameter assignments described earlier in this section.

The example session description below illustrates the use of the chapter inclusion parameters:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP6 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP6 2001:DB80::7F2E:172A:1E24
a=rtpmap:96 rtp-midi/44100
a=fmtp:96 j_update=open-loop; cm_unused=ABCFGHJKMQTVWXYZ;
cm_used=__7E_00-7F_09_01.02.03__;
cm_used=__7F_00-7F_04_01.02__; cm_used=C7.64;
ch_never=ABCDEFGHJKMQTVWXYZ; ch_never=4.11-13N;
ch_anchor=P; ch_anchor=C7.64;
ch_anchor=__7E_00-7F_09_01.02.03__;
ch_anchor=__7F_00-7F_04_01.02__
```

(The a=fmtp line has been wrapped to fit the page to accommodate memo formatting restrictions; it comprises a single line in SDP.)

The j_update parameter codes that the stream uses the open-loop policy. Most MIDI command-types are assigned to cm_unused and thus do not appear in the stream. As a consequence, the assignments to the first ch_never parameter reflect that most chapters are not in use.

Chapter N for several MIDI channels is assigned to ch_never. Chapter N for MIDI channels other than 4, 11, 12, and 13 may appear in the recovery journal, using the (default) ch_default semantics. In practice, this assignment pattern would reflect knowledge about a resilient rendering method in use for the excluded channels.

The MIDI Program Change command and several MIDI Control Change controller numbers are assigned to ch_anchor. Note that the ordering of the ch_anchor chapter C assignment after the ch_never command acts to override the ch_never assignment for the listed controller numbers (7 and 64).

The assignment of command-type X to cm_unused excludes most SysEx commands from the stream. Exceptions are made for General MIDI System On/Off commands and for the Master Volume and Balance commands, via the use of the secondary assignment syntax. The cm_used assignment codes the exception, and the ch_anchor assignment codes how these commands are protected in Chapter X.

C.3. Configuration Tools: Timestamp Semantics

The MIDI command section of the payload format consists of a list of commands, each with an associated timestamp. The semantics of command timestamps may be set during session configuration, using the parameters we describe in this section

The parameter "tsmode" specifies the timestamp semantics for a stream. The parameter takes on one of three token values: "comex", "async", or "buffer".

The default "comex" value specifies that timestamps code the execution time for a command (Appendix C.3.1) and supports the accurate transcoding Standard MIDI Files (SMFs, [MIDI]). The "comex" value is also RECOMMENDED for new MIDI user-interface controller designs. The "async" value specifies an asynchronous timestamp sampling algorithm for time-of-arrival sources (Appendix C.3.2). The "buffer" value specifies a synchronous timestamp sampling algorithm (Appendix C.3.3) for time-of-arrival sources.

Ancillary parameters MAY follow tsmode in a media description. We define these parameters in Appendices C.3.2-3 below.

C.3.1. The comex Algorithm

The default "comex" (COMmand EXecution) tsmode value specifies the execution time for the command. With comex, the difference between two timestamps indicates the time delay between the execution of the commands. This difference may be zero, coding simultaneous execution.

The comex interpretation of timestamps works well for transcoding a Standard MIDI File (SMF, [MIDI]) into an RTP MIDI stream, as SMFs code a timestamp for each MIDI command stored in the file. To transcode an SMF that uses metric time markers, use the SMF tempo map (encoded in the SMF as meta-events) to convert metric SMF timestamp units into seconds-based RTP timestamp units.

New MIDI controller designs (piano keyboard, drum pads, etc.) that support RTP MIDI and that have direct access to sensor data SHOULD use comex interpretation for timestamps, so that simultaneous gestural events may be accurately coded by RTP MIDI.

Comex is a poor choice for transcoding MIDI 1.0 DIN cables [MIDI], for a reason that we will now explain. A MIDI DIN cable is an asynchronous serial protocol (320 microseconds per MIDI byte). MIDI commands on a DIN cable are not tagged with timestamps. Instead, MIDI DIN receivers infer command timing from the time of arrival of the bytes. Thus, two two-byte MIDI commands that occur at a source simultaneously are encoded on a MIDI 1.0 DIN cable with a 640 microsecond time offset. A MIDI DIN receiver is unable to tell if this time offset existed in the source performance or is an artifact of the serial speed of the cable. However, the RTP MIDI comex interpretation of timestamps declares that a timestamp offset between two commands reflects the timing of the source performance.

This semantic mismatch is the reason that comex is a poor choice for transcoding MIDI DIN cables. Note that the choice of the RTP timestamp rate (Section 6.1-2 in the main text) cannot fix this inaccuracy issue. In the sections that follow, we describe two alternative timestamp interpretations ("async" and "buffer") that are a better match to MIDI 1.0 DIN cable timing, and to other MIDI time-of-arrival sources.

The "octpos", "linerate", and "mperiod" ancillary parameters (defined below) SHOULD NOT be used with comex.

C.3.2. The async Algorithm

The "async" tsmode value specifies the asynchronous sampling of a MIDI time-of-arrival source. In asynchronous sampling, the moment an octet is received from a source, it is labelled with a wall-clock time value. The time value has RTP timestamp units.

The "octpos" ancillary parameter defines how RTP command timestamps are derived from octet time values. If octpos has the token value "first", a timestamp codes the time value of the first octet of the command. If octpos has the token value "last", a timestamp codes the time value of the last octet of the command. If the octpos parameter does not appear in the media description, the sender does not know which octet of the command the timestamp references (for example, the sender may be relying on an operating system service that does not specify this information).

The octpos semantics refer to the first or last octet of a command as it appears on a time-of-arrival MIDI source, not as it appears in an RTP MIDI packet. This distinction is significant because the RTP coding may contain octets that are not present in the source. For example, the status octet of the first MIDI command in a packet may have been added to the MIDI stream during transcoding, to comply with the RTP MIDI running status requirements (Section 3.2).

The "linerate" ancillary parameter defines the timespan of one MIDI octet on the transmission medium of the MIDI source to be sampled (such as a MIDI 1.0 DIN cable). The parameter has units of nanoseconds, and takes on integral values. For MIDI 1.0 DIN cables, the correct linerate value is 320000 (this value is also the default value for the parameter).

We now show a session description example for the async algorithm. Consider a sender that is transcoding a MIDI 1.0 DIN cable source into RTP. The sender runs on a computing platform that assigns time values to every incoming octet of the source, and the sender uses the time values to label the first octet of each command in the RTP packet. This session description describes the transcoding:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP4 192.0.2.94
a=rtpmap:96 rtp-midi/44100
a=sendonly
a=fmtp:96 tsmode=async; linerate=320000; octpos=first
```

C.3.3. The buffer Algorithm

The "buffer" tsmode value specifies the synchronous sampling of a MIDI time-of-arrival source.

In synchronous sampling, octets received from a source are placed in a holding buffer upon arrival. At periodic intervals, the RTP sender examines the buffer. The sender removes complete commands from the buffer and codes those commands in an RTP packet. The command timestamp codes the moment of buffer examination, expressed in RTP timestamp units. Note that several commands may have the same timestamp value.

The "mperiod" ancillary parameter defines the nominal periodic sampling interval. The parameter takes on positive integral values and has RTP timestamp units.

The "octpos" ancillary parameter, defined in Appendix C.3.1 for asynchronous sampling, plays a different role in synchronous sampling. In synchronous sampling, the parameter specifies the timestamp semantics of a command whose octets span several sampling periods.

If octpos has the token value "first", the timestamp reflects the arrival period of the first octet of the command. If octpos has the token value "last", the timestamp reflects the arrival period of the last octet of the command. The octpos semantics refer to the first or last octet of the command as it appears on a time-of-arrival source, not as it appears in the RTP packet.

If the octpos parameter does not appear in the media description, the timestamp MAY reflect the arrival period of any octet of the command; senders use this option to signal a lack of knowledge about the timing details of the buffering process at sub-command granularity.

We now show a session description example for the buffer algorithm. Consider a sender that is transcoding a MIDI 1.0 DIN cable source into RTP. The sender runs on a computing platform that places source data into a buffer upon receipt. The sender polls the buffer 1000 times a second, extracts all complete commands from the buffer, and places the commands in an RTP packet. This session description describes the transcoding:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP6 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP6 2001:DB80::7F2E:172A:1E24
a=rtpmap:96 rtp-midi/44100
a=sendonly
a=fmtp:96 tsmode=buffer; linerate=320000; octpos=last; mperiod=44
```

The mperiod value of 44 is derived by dividing the clock rate specified by the rtpmap attribute (44100 Hz) by the 1000 Hz buffer sampling rate and rounding to the nearest integer. Command timestamps might not increment by exact multiples of 44, as the actual sampling period might not precisely match the nominal mperiod value.

C.4. Configuration Tools: Packet Timing Tools

In this appendix, we describe session configuration tools for customizing the temporal behavior of MIDI stream packets.

C.4.1. Packet Duration Tools

Senders control the granularity of a stream by setting the temporal duration ("media time") of the packets in the stream. Short media times (20 ms or less) often imply an interactive session. Longer media times (100 ms or more) usually indicate a content streaming session. The RTP AVP profile [RFC3551] recommends audio packet media times in a range from 0 to 200 ms.

By default, an RTP receiver dynamically senses the media time of packets in a stream and chooses the length of its playout buffer to match the stream. A receiver typically sizes its playout buffer to fit several audio packets and adjusts the buffer length to reflect the network jitter and the sender timing fidelity.

Alternatively, the packet media time may be statically set during session configuration. Session descriptions MAY use the RTP MIDI parameter "rtp_ptime" to set the recommended media time for a packet. Session descriptions MAY also use the RTP MIDI parameter "rtp_maxptime" to set the maximum media time for a packet permitted in a stream. Both parameters MAY be used together to configure a stream.

The values assigned to the rtp_ptime and rtp_maxptime parameters have the units of the RTP timestamp for the stream, as set by the rtpmap attribute (see Section 6.1). Thus, if rtpmap sets the clock rate of a stream to 44100 Hz, a maximum packet media time of 10 ms is coded by setting rtp_maxptime=441. As stated in the Appendix C preamble, the senders and receivers of a stream MUST agree on common values for rtp_ptime and rtp_maxptime if the parameters appear in the media description for the stream.

0 ms is a reasonable media time value for MIDI packets and is often used in low-latency interactive applications. In a packet with a 0 ms media time, all commands execute at the instant they are coded by the packet timestamp. The session description below configures all packets in the stream to have 0 ms media time:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP4 192.0.2.94
a=rtpmap:96 rtp-midi/44100
a=fmtp:96 rtp_ptime=0; rtp_maxptime=0
```

The session attributes ptime and maxptime [RFC4566] MUST NOT be used to configure an RTP MIDI stream. Sessions MUST use rtp_ptime in lieu of ptime and MUST use rtp_maxptime in lieu of maxptime. RTP MIDI defines its own parameters for media time configuration because 0 ms values for ptime and maxptime are forbidden by [RFC3264] but are essential for certain applications of RTP MIDI.

See the Appendix C.7 examples for additional discussion about using rtp_ptime and rtp_maxptime for session configuration.

C.4.2. The guardtime Parameter

RTP permits a sender to stop sending audio packets for an arbitrary period of time during a session. When sending resumes, the RTP sequence number series continues unbroken, and the RTP timestamp value reflects the media time silence gap.

This RTP feature has its roots in telephony, but it is also well matched to interactive MIDI sessions, as players may fall silent for several seconds during (or between) songs.

Certain MIDI applications benefit from a slight enhancement to this RTP feature. In interactive applications, receivers may use on-line network models to guide heuristics for handling lost and late RTP packets. These models may work poorly if a sender ceases packet transmission for long periods of time.

Session descriptions may use the parameter "guardtime" to set a minimum sending rate for a media session. The value assigned to guardtime codes the maximum separation time between two sequential packets, as expressed in RTP timestamp units.

Typical guardtime values are 500-2000 ms. This value range is not a normative bound, and parties SHOULD be prepared to process values outside this range.

The congestion control requirements for sender implementations (described in Section 8 and [RFC3550]) take precedence over the guardtime parameter. Thus, if the guardtime parameter requests a minimum sending rate, but sending at this rate would violate the congestion control requirements, senders MUST ignore the guardtime parameter value. In this case, senders SHOULD use the lowest minimum sending rate that satisfies the congestion control requirements.

Below, we show a session description that uses the guardtime parameter.

```
v=0
o=lazzaro 2520644554 2838152170 IN IP6 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP6 2001:DB80::7F2E:172A:1E24
a=rtpmap:96 rtp-midi/44100
a=fmtp:96 guardtime=44100; rtp_ptime=0; rtp_maxptime=0
```

C.5. Configuration Tools: Stream Description

As we discussed in Section 2.1, a party may send several RTP MIDI streams in the same RTP session, and several RTP sessions that carry MIDI may appear in a multimedia session.

By default, the MIDI name space (16 channels + systems) of each RTP stream sent by a party in a multimedia session is independent. By independent, we mean three distinct things:

- If a party sends two RTP MIDI streams (A and B), MIDI voice channel 0 in stream A is a different "channel 0" than MIDI voice channel 0 in stream B.

- MIDI voice channel 0 in stream B is not considered to be "channel 16" of a 32-channel MIDI voice channel space whose "channel 0" is channel 0 of stream A.

- Streams sent by different parties over different RTP sessions, or over the same RTP session but with different payload type numbers, do not share the association that is shared by a MIDI cable pair that cross-connects two devices in a MIDI 1.0 DIN network. By default, this association is only held by streams sent by different parties in the same RTP session that use the same payload type number.

In this appendix, we show how to express that specific RTP MIDI streams in a multimedia session are not independent but instead are related in one of the three ways defined above. We use two tools to express these relations:

- The musicport parameter. This parameter is assigned a non-negative integer value between 0 and 4294967295. It appears in the fmtp lines of payload types.

- The FID grouping attribute [RFC3388] signals that several RTP sessions in a multimedia session are using the musicport parameter to express an inter-session relationship.

If a multimedia session has several payload types whose musicport parameters are assigned the same integer value, streams using these payload types share an "identity relationship" (including streams that use the same payload type). Streams in an identity relationship share two properties:

- Identity relationship streams sent by the same party target the same MIDI name space. Thus, if streams A and B share an identity relationship, voice channel 0 in stream A is the same "channel 0" as voice channel 0 in stream B.
- Pairs of identity relationship streams that are sent by different parties share the association that is shared by a MIDI cable pair that cross-connects two devices in a MIDI 1.0 DIN network.

A party MUST NOT send two RTP MIDI streams that share an identity relationship in the same RTP session. Instead, each stream MUST be in a separate RTP session. As explained in Section 2.1, this restriction is necessary to support the RTP MIDI method for the synchronization of streams that share a MIDI name space.

If a multimedia session has several payload types whose musicport parameters are assigned sequential values (i.e., i, i+1, ... i+k), the streams using the payload types share an "ordered relationship". For example, if payload type A assigns 2 to musicport and payload type B assigns 3 to musicport, A and B are in an ordered relationship.

Streams in an ordered relationship that are sent by the same party are considered by renderers to form a single larger MIDI space. For example, if stream A has a musicport value of 2 and stream B has a musicport value of 3, MIDI voice channel 0 in stream B is considered to be voice channel 16 in the larger MIDI space formed by the relationship. Note that it is possible for streams to participate in both an identity relationship and an ordered relationship.

We now state several rules for using musicport:

- If streams from several RTP sessions in a multimedia session use the musicport parameter, the RTP sessions MUST be grouped using the FID grouping attribute defined in [RFC3388].

- An ordered or identity relationship MUST NOT contain both native RTP MIDI streams and mpeg4-generic RTP MIDI streams. An exception applies if a relationship consists of sendonly and recvonly (but not sendrecv) streams. In this case, the sendonly streams MUST NOT contain both types of streams, and the recvonly streams MUST NOT contain both types of streams.

- It is possible to construct identity relationships that violate the recovery journal mandate (for example, sending NoteOns for a voice channel on stream A and NoteOffs for the same voice channel on stream B). Parties MUST NOT generate (or accept) session descriptions that exhibit this flaw.

- Other payload formats MAY define musicport media type parameters. Formats would define these parameters so that their sessions could be bundled into RTP MIDI name spaces. The parameter definitions MUST be compatible with the musicport semantics defined in this appendix.

As a rule, at most one payload type in a relationship may specify a MIDI renderer. An exception to the rule applies to relationships that contain sendonly and recvonly streams but no sendrecv streams. In this case, one sendonly session and one recvonly session may each define a renderer.

Renderer specification in a relationship may be done using the tools described in Appendix C.6. These tools work for both native streams and mpeg4-generic streams. An mpeg4-generic stream that uses the Appendix C.6 tools MUST set all "config" parameters to the empty string ("").

Alternatively, for mpeg4-generic streams, renderer specification may be done by setting one "config" parameter in the relationship to the renderer configuration string, and all other config parameters to the empty string ("").

We now define sender and receiver rules that apply when a party sends several streams that target the same MIDI name space.

Senders MAY use the subsetting parameters (Appendix C.1) to predefine the partitioning of commands between streams, or they MAY use a dynamic partitioning strategy.

Receivers that merge identity relationship streams into a single MIDI command stream MUST maintain the structural integrity of the MIDI commands coded in each stream during the merging process, in the same way that software that merges traditional MIDI 1.0 DIN cable flows is responsible for creating a merged command flow compatible with [MIDI].

Senders MUST partition the name space so that the rendered MIDI performance does not contain indefinite artifacts (as defined in Section 4). This responsibility holds even if all streams are sent over reliable transport, as different stream latencies may yield indefinite artifacts. For example, stuck notes may occur in a performance split over two TCP streams, if NoteOn commands are sent on one stream and NoteOff commands are sent on the other.

Senders MUST NOT split a Registered Parameter Name (RPN) or Non-Registered Parameter Name (NRPN) transaction appearing on a MIDI channel across multiple identity relationship sessions. Receivers MUST assume that the RPN/NRPN transactions that appear on different identity relationship sessions are independent and MUST preserve transactional integrity during the MIDI merge.

A simple way to safely partition voice channel commands is to place all MIDI commands for a particular voice channel into the same session. Safe partitioning of MIDI Systems commands may be more complicated for sessions that extensively use System Exclusive.

We now show several session description examples that use the musicport parameter.

Our first session description example shows two RTP MIDI streams that drive the same General MIDI decoder. The sender partitions MIDI commands between the streams dynamically. The musicport values indicate that the streams share an identity relationship.

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
a=group:FID 1 2
c=IN IP4 192.0.2.94
m=audio 5004 RTP/AVP 96
a=rtpmap:96 mpeg4-generic/44100
a=mid:1
a=fmtp:96 streamtype=5; mode=rtp-midi; profile-level-id=12;
config=7A0A0000001A4D546864000000060000000100604D54726B0
000000600FF2F000; musicport=12
m=audio 5006 RTP/AVP 96
a=rtpmap:96 mpeg4-generic/44100
a=mid:2
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; musicport=12
```

(The a=fmtp lines have been wrapped to fit the page to accommodate memo formatting restrictions; they comprise single lines in SDP.)

Recall that Section 2.1 defines rules for streams that target the same MIDI name space. Those rules, implemented in the example above, require that each stream resides in a separate RTP session, and that the grouping mechanisms defined in [RFC3388] signal an inter-session relationship. The "group" and "mid" attribute lines implement this grouping mechanism.

A variant on this example, whose session description is not shown, would use two streams in an identity relationship driving the same MIDI renderer, each with a different transport type. One stream would use UDP and would be dedicated to real-time messages. A second stream would use TCP [RFC4571] and would be used for SysEx bulk data messages.

In the next example, two mpeg4-generic streams form an ordered relationship to drive a Structured Audio decoder with 32 MIDI voice channels. Both streams reside in the same RTP session.

```
v=0
o=lazzaro 2520644554 2838152170 IN IP6 first.example.net
s=Example
t=0 0
m=audio 5006 RTP/AVP 96 97
c=IN IP6 2001:DB80::7F2E:172A:1E24
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=13; musicport=5
a=rtpmap:97 mpeg4-generic/44100
a=fmtp:97 streamtype=5; mode=rtp-midi; config="";
profile-level-id=13; musicport=6; render=synthetic;
rinit="audio/asc";
url="http://example.com/cardinal.asc";
cid="azsldkaslkdjqpwojdkmsldkfpe"
```

(The a=fmtp lines have been wrapped to fit the page to accommodate memo formatting restrictions; they comprise single lines in SDP.)

The sequential musicport values for the two sessions establish the ordered relationship. The musicport=5 session maps to Structured Audio extended channels range 0-15, the musicport=6 session maps to Structured Audio extended channels range 16-31.

Both config strings are empty. The configuration data is specified by parameters that appear in the fmtp line of the second media description. We define this configuration method in Appendix C.6.

The next example shows two RTP MIDI streams (one recvonly, one sendonly) that form a "virtual sendrecv" session. Each stream resides in a different RTP session (a requirement because sendonly and recvonly are RTP session attributes).

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
a=group:FID 1 2
c=IN IP4 192.0.2.94
m=audio 5004 RTP/AVP 96
a=sendonly
a=rtpmap:96 mpeg4-generic/44100
a=mid:1
a=fmtp:96 streamtype=5; mode=rtp-midi; profile-level-id=12;
config=7A0A0000001A4D546864000000060000000100604D54726B0
000000600FF2F000; musicport=12
m=audio 5006 RTP/AVP 96
a=recvonly
a=rtpmap:96 mpeg4-generic/44100
a=mid:2
a=fmtp:96 streamtype=5; mode=rtp-midi; profile-level-id=12;
config=7A0A0000001A4D546864000000060000000100604D54726B0
000000600FF2F000; musicport=12
```

(The a=fmtp lines have been wrapped to fit the page to accommodate memo formatting restrictions; they comprise single lines in SDP.)

To signal the "virtual sendrecv" semantics, the two streams assign musicport to the same value (12). As defined earlier in this section, pairs of identity relationship streams that are sent by different parties share the association that is shared by a MIDI cable pair that cross-connects two devices in a MIDI 1.0 network. We use the term "virtual sendrecv" because streams sent by different parties in a true sendrecv session also have this property.

As discussed in the preamble to Appendix C, the primary advantage of the virtual sendrecv configuration is that each party can customize the property of the stream it receives. In the example above, each stream defines its own "config" string that could customize the rendering algorithm for each party (in fact, the particular strings shown in this example are identical, because General MIDI is not a configurable MPEG 4 renderer).

C.6. Configuration Tools: MIDI Rendering

This appendix defines the session configuration tools for rendering.

The "render" parameter specifies a rendering method for a stream. The parameter is assigned a token value that signals the top-level rendering class. This memo defines four token values for render: "unknown", "synthetic", "api", and "null":

- An "unknown" renderer is a renderer whose nature is unspecified. It is the default renderer for native RTP MIDI streams.
- A "synthetic" renderer transforms the MIDI stream into audio output (or sometimes into stage lighting changes or other actions). It is the default renderer for mpeg4-generic RTP MIDI streams.
- An "api" renderer presents the command stream to applications via an Application Programmer Interface (API).
- The "null" renderer discards the MIDI stream.

The "null" render value plays special roles during Offer/Answer negotiations [RFC3264]. A party uses the "null" value in an answer to reject an offered renderer. Note that rejecting a renderer is independent from rejecting a payload type (coded by removing the payload type from a media line) and rejecting a media stream (coded by zeroing the port of a media line that uses the renderer).

Other render token values MAY be registered with IANA. The token value MUST adhere to the ABNF for render tokens defined in Appendix D. Registrations MUST include a complete specification of parameter value usage, similar in depth to the specifications that appear throughout Appendix C.6 for "synthetic" and "api" render values. If a party is offered a session description that uses a render token value that is not known to the party, the party MUST NOT accept the renderer. Options include rejecting the renderer (using the "null" value), the payload type, the media stream, or the session description.

Other parameters MAY follow a render parameter in a parameter list. The additional parameters act to define the exact nature of the renderer. For example, the "subrender" parameter (defined in Appendix C.6.2) specifies the exact nature of the renderer.

Special rules apply to using the render parameter in an mpeg4-generic stream. We define these rules in Appendix C.6.5.

C.6.1. The multimode Parameter

A media description MAY contain several render parameters. By default, if a parameter list includes several render parameters, a receiver MUST choose exactly one renderer from the list to render the stream. The "multimode" parameter may be used to override this default. We define two token values for multimode: "one" and "all":

- The default "one" value requests rendering by exactly one of the listed renderers.
- The "all" value requests the synchronized rendering of the RTP MIDI stream by all listed renderers, if possible.

If the multimode parameter appears in a parameter list, it MUST appear before the first render parameter assignment.

Render parameters appear in the parameter list in order of decreasing priority. A receiver MAY use the priority ordering to decide which renderer(s) to retain in a session.

If the "offer" in an Offer/Answer-style negotiation [RFC3264] contains a parameter list with one or more render parameters, the "answer" MUST set the render parameters of all unchosen renderers to "null".

C.6.2. Renderer Specification

The render parameter (Appendix C.6 preamble) specifies, in a broad sense, what a renderer does with a MIDI stream. In this appendix, we describe the "subrender" parameter. The token value assigned to subrender defines the exact nature of the renderer. Thus, "render" and "subrender" combine to define a renderer, in the same way as MIME types and MIME subtypes combine to define a type of media [RFC2045].

If the subrender parameter is used for a renderer definition, it MUST appear immediately after the render parameter in the parameter list. At most one subrender parameter may appear in a renderer definition.

This document defines one value for subrender: the value "default". The "default" token specifies the use of the default renderer for the stream type (native or mpeg4-generic). The default renderer for native RTP MIDI streams is a renderer whose nature is unspecified (see point 6 in Section 6.1 for details). The default renderer for mpeg4-generic RTP MIDI streams is an MPEG 4 Audio Object Type whose ID number is 13, 14, or 15 (see Section 6.2 for details).

If a renderer definition does not use the subrender parameter, the value "default" is assumed for subrender.

Other subrender token values may be registered with IANA. We now discuss guidelines for registering subrender values.

A subrender value is registered for a specific stream type (native or mpeg4-generic) and a specific render value (excluding "null" and "unknown"). Registrations for mpeg4-generic subrender values are restricted to new MPEG 4 Audio Object Types that accept MIDI input. The syntax of the token MUST adhere to the token definition in Appendix D.

For "render=synthetic" renderers, a subrender value registration specifies an exact method for transforming the MIDI stream into audio (or sometimes into video or control actions, such as stage lighting). For standardized renderers, this specification is usually a pointer to a standards document, perhaps supplemented by RTP-MIDI-specific information. For commercial products and open-source projects, this specification usually takes the form of instructions for interfacing the RTP MIDI stream with the product or project software. A "render=synthetic" registration MAY specify additional Reset State commands for the renderer (Appendix A.1).

A "render=api" subrender value registration specifies how an RTP MIDI stream interfaces with an API (Application Programmers Interface). This specification is usually a pointer to programmer’s documentation for the API, perhaps supplemented by RTP-MIDI-specific information.

A subrender registration MAY specify an initialization file (referred to in this document as an initialization data object) for the stream. The initialization data object MAY be encoded in the parameter list (verbatim or by reference) using the coding tools defined in Appendix C.6.3. An initialization data object MUST have a registered [RFC4288] media type and subtype [RFC2045].

For "render=synthetic" renderers, the data object usually encodes initialization data for the renderer (sample files, synthesis patch parameters, reverberation room impulse responses, etc.).

For "render=api" renderers, the data object usually encodes data about the stream used by the API (for example, for an RTP MIDI stream generated by a piano keyboard controller, the manufacturer and model number of the keyboard, for use in GUI presentation).

Usually, only one initialization object is encoded for a renderer. If a renderer uses multiple data objects, the correct receiver interpretation of multiple data objects MUST be defined in the subrender registration.

A subrender value registration may also specify additional parameters, to appear in the parameter list immediately after subrender. These parameter names MUST begin with the subrender value, followed by an underscore ("_"), to avoid name space collisions with future RTP MIDI parameter names (for example, a parameter "foo_bar" defined for subrender value "foo").

We now specify guidelines for interpreting the subrender parameter during session configuration.

If a party is offered a session description that uses a renderer whose subrender value is not known to the party, the party MUST NOT accept the renderer. Options include rejecting the renderer (using the "null" value), the payload type, the media stream, or the session description.

Receivers MUST be aware of the Reset State commands (Appendix A.1) for the renderer specified by the subrender parameter and MUST insure that the renderer does not experience indefinite artifacts due to the presence (or the loss) of a Reset State command.

C.6.3. Renderer Initialization

If the renderer for a stream uses an initialization data object, an "rinit" parameter MUST appear in the parameter list immediately after the "subrender" parameter. If the renderer parameter list does not include a subrender parameter (recall the semantics for "default" in Appendix C.6.2), the "rinit" parameter MUST appear immediately after the "render" parameter.

The value assigned to the rinit parameter MUST be the media type/subtype [RFC2045] for the initialization data object. If an initialization object type is registered with several media types, including audio, the assignment to rinit MUST use the audio media type.

RTP MIDI supports several parameters for encoding initialization data objects for renderers in the parameter list: "inline", "url", and "cid".

If the "inline", "url", and/or "cid" parameters are used by a renderer, these parameters MUST immediately follow the "rinit" parameter.

If a "url" parameter appears for a renderer, an "inline" parameter MUST NOT appear. If an "inline" parameter appears for a renderer, a "url" parameter MUST NOT appear. However, neither "url" or "inline" is required to appear. If neither "url" or "inline" parameters follow "rinit", the "cid" parameter MUST follow "rinit".

The "inline" parameter supports the inline encoding of the data object. The parameter is assigned a double-quoted Base64 [RFC2045] encoding of the binary data object, with no line breaks. Appendix E.4 shows an example that constructs an inline parameter value.

The "url" parameter is assigned a double-quoted string representation of a Uniform Resource Locator (URL) for the data object. The string MUST specify a HyperText Transport Protocol URL (HTTP, [RFC2616]). HTTP MAY be used over TCP or MAY be used over a secure network transport, such as the method described in [RFC2818]. The media type/subtype for the data object SHOULD be specified in the appropriate HTTP transport header.

The "cid" parameter supports data object caching. The parameter is assigned a double-quoted string value that encodes a globally unique identifier for the data object.

A cid parameter MAY immediately follow an inline parameter, in which case the cid identifier value MUST be associated with the inline data object.

If a url parameter is present, and if the data object for the URL is expected to be unchanged for the life of the URL, a cid parameter MAY immediately follow the url parameter. The cid identifier value MUST be associated with the data object for the URL. A cid parameter assigned to the same identifier value SHOULD be specified following the data object type/subtype in the appropriate HTTP transport header.

If a url parameter is present, and if the data object for the URL is expected to change during the life of the URL, a cid parameter MUST NOT follow the url parameter. A receiver interprets the presence of a cid parameter as an indication that it is safe to use a cached copy of the url data object; the absence of a cid parameter is an indication that it is not safe to use a cached copy, as it may change.

Finally, the cid parameter MAY be used without the inline and url parameters. In this case, the identifier references a local or distributed catalog of data objects.

In most cases, only one data object is coded in the parameter list for each renderer. For example, the default renderer for mpeg4-generic streams uses a single data object (see Appendix C.6.5 for example usage).

However, a subrender registration MAY permit the use of multiple data objects for a renderer. If multiple data objects are encoded for a renderer, each object encoding begins with an "rinit" parameter, followed by "inline", "url", and/or "cid" parameters.

Initialization data objects MAY encapsulate a Standard MIDI File (SMF). By default, the SMFs that are encapsulated in a data object MUST be ignored by an RTP MIDI receiver. We define parameters to override this default in Appendix C.6.4.

To end this section, we offer guidelines for registering media types for initialization data objects. These guidelines are in addition to the information in [RFC4288] [RFC4289].

Some initialization data objects are also capable of encoding MIDI note information and thus complete audio performances. These objects SHOULD be registered using the "audio" media type, so that the objects may also be used for store-and-forward rendering, and "application" media type, to support editing tools. Initialization objects without note storage, or initialization objects for non-audio renderers, SHOULD be registered only for an "application" media type.

C.6.4. MIDI Channel Mapping

In this appendix, we specify how to map MIDI name spaces (16 voice channels + systems) onto a renderer.

In the general case:

- A session may define an ordered relationship (Appendix C.5) that presents more than one MIDI name space to a renderer.

- A renderer may accept an arbitrary number of MIDI name spaces, or it may expect a specific number of MIDI name spaces.

A session description SHOULD provide a compatible MIDI name space to each renderer in the session. If a receiver detects that a session description has too many or too few MIDI name spaces for a renderer, MIDI data from extra stream name spaces MUST be discarded, and extra renderer name spaces MUST NOT be driven with MIDI data (except as described in Appendix C.6.4.1, below).

If a parameter list defines several renderers and assigns the "all" token value to the multimode parameter, the same name space is presented to each renderer. However, the "chanmask" parameter may be used to mask out selected voice channels to each renderer. We define "chanmask" and other MIDI management parameters in the sub-sections below.

C.6.4.1. The smf_info Parameter

The smf_info parameter defines the use of the SMFs encapsulated in renderer data objects (if any). The smf_info parameter also defines the use of SMFs coded in the smf_inline, smf_url, and smf_cid parameters (defined in Appendix C.6.4.2).

The smf_info parameter describes the "render" parameter that most recently precedes it in the parameter list. The smf_info parameter MUST NOT appear in parameter lists that do not use the "render" parameter, and MUST NOT appear before the first use of "render" in the parameter list.

We define three token values for smf_info: "ignore", "sdp_start", and "identity":

- The "ignore" value indicates that the SMFs MUST be discarded. This behavior is the default SMF rendering behavior.

- The "sdp_start" value codes that SMFs MUST be rendered, and that the rendering MUST begin upon the acceptance of the session description. If a receiver is offered a session description with a renderer that uses an smf_info parameter set to sdp_start, and if the receiver does not support rendering SMFs, the receiver MUST NOT accept the renderer associated with the smf_info parameter. Options include rejecting the renderer (by setting the "render" parameter to "null"), the payload type, the media stream, or the entire session description.

- The "identity" value indicates that the SMFs code the identity of the renderer. The value is meant for use with the "unknown" renderer (see Appendix C.6 preamble). The MIDI commands coded in the SMF are informational in nature and MUST NOT be presented to a renderer for audio presentation. In typical use, the SMF would use SysEx Identity Reply commands (F0 7E nn 06 02, as defined in [MIDI]) to identify devices, and use device-specific SysEx commands to describe current state of the devices (patch memory contents, etc.).

Other smf_info token values MAY be registered with IANA. The token value MUST adhere to the ABNF for render tokens defined in Appendix

D. Registrations MUST include a complete specification of parameter usage, similar in depth to the specifications that appear in this appendix for "sdp_start" and "identity".

If a party is offered a session description that uses an smf_info parameter value that is not known to the party, the party MUST NOT accept the renderer associated with the smf_info parameter. Options include rejecting the renderer, the payload type, the media stream, or the entire session description.

We now define the rendering semantics for the "sdp_start" token value in detail.

The SMFs and RTP MIDI streams in a session description share the same MIDI name space(s). In the simple case of a single RTP MIDI stream and a single SMF, the SMF MIDI commands and RTP MIDI commands are merged into a single name space and presented to the renderer. The indefinite artifact responsibilities for merged MIDI streams defined in Appendix C.5 also apply to merging RTP and SMF MIDI data.

If a payload type codes multiple SMFs, the SMF name spaces are presented as an ordered entity to the renderer. To determine the ordering of SMFs for a renderer (which SMF is "first", which is "second", etc.), use the following rules:

- If the renderer uses a single data object, the order of appearance of the SMFs in the object's internal structure defines the order of the SMFs (the earliest SMF in the object is "first", the next SMF in the object is "second", etc.).
- If multiple data objects are encoded for a renderer, the appearance of each data object in the parameter list sets the relative order of the SMFs encoded in each data object (SMFs encoded in parameters that appear earlier in the list are ordered before SMFs encoded in parameters that appear later in the list).
- If SMFs are encoded in data objects parameters and in the parameters defined in C.6.4.2, the relative order of the data object parameters and C.6.4.2 parameters in the parameter list sets the relative order of SMFs (SMFs encoded in parameters that appear earlier in the list are ordered before SMFs in parameters that appear later in the list).

Given this ordering of SMFs, we now define the mapping of SMFs to renderer name spaces. The SMF that appears first for a renderer maps to the first renderer name space. The SMF that appears second for a renderer maps to the second renderer name space, etc. If the associated RTP MIDI streams also form an ordered relationship, the first SMF is merged with the first name space of the relationship, the second SMF is merged to the second name space of the relationship, etc.

Unless the streams and the SMFs both use MIDI Time Code, the time offset between SMF and stream data is unspecified. This restriction limits the use of SMFs to applications where synchronization is not critical, such as the transport of System Exclusive commands for renderer initialization, or human-SMF interactivity.

Finally, we note that each SMF in the sdp_start discussion above encodes exactly one MIDI name space (16 voice channels + systems). Thus, the use of the Device Name SMF meta event to specify several MIDI name spaces in an SMF is not supported for sdp_start.

C.6.4.2. The smf_inline, smf_url, and smf_cid Parameters

In some applications, the renderer data object may not encapsulate SMFs, but an application may wish to use SMFs in the manner defined in Appendix C.6.4.1.

The "smf_inline", "smf_url", and "smf_cid" parameters address this situation. These parameters use the syntax and semantics of the inline, url, and cid parameters defined in Appendix C.6.3, except that the encoded data object is an SMF.

The "smf_inline", "smf_url", and "smf_cid" parameters belong to the "render" parameter that most recently precedes it in the session description. The "smf_inline", "smf_url", and "smf_cid" parameters MUST NOT appear in parameter lists that do not use the "render" parameter and MUST NOT appear before the first use of "render" in the parameter list. If several "smf_inline", "smf_url", or "smf_cid" parameters appear for a renderer, the order of the parameters defines the SMF name space ordering.

C.6.4.3. The chanmask Parameter

The chanmask parameter instructs the renderer to ignore all MIDI voice commands for certain channel numbers. The parameter value is a concatenated string of "1" and "0" digits. Each string position maps to a MIDI voice channel number (system channels may not be masked). A "1" instructs the renderer to process the voice channel; a "0" instructs the renderer to ignore the voice channel.

The string length of the chanmask parameter value MUST be 16 (for a single stream or an identity relationship) or a multiple of 16 (for an ordered relationship).

The chanmask parameter describes the "render" parameter that most recently precedes it in the session description; chanmask MUST NOT appear in parameter lists that do not use the "render" parameter and MUST NOT appear before the first use of "render" in the parameter list.

The chanmask parameter describes the final MIDI name spaces presented to the renderer. The SMF and stream components of the MIDI name spaces may not be independently masked.

If a receiver is offered a session description with a renderer that uses the chanmask parameter, and if the receiver does not implement the semantics of the chanmask parameter, the receiver MUST NOT accept the renderer unless the chanmask parameter value contains only "1"s.

C.6.5. The audio/asc Media Type

In Appendix 11.3, we register the audio/asc media type. The data object for audio/asc is a binary encoding of the AudioSpecificConfig data block used to initialize mpeg4-generic streams (Section 6.2 and [MPEGAUDIO]).

An mpeg4-generic parameter list MAY use the render, subrender, and rinit parameters with the audio/asc media type for renderer configuration. Several restrictions apply to the use of these parameters in mpeg4-generic parameter lists:

- An mpeg4-generic media description that uses the render parameter MUST assign the empty string ("") to the mpeg4-generic "config" parameter. The use of the streamtype, mode, and profile-level-id parameters MUST follow the normative text in Section 6.2.

- Sessions that use identity or ordered relationships MUST follow the mpeg4-generic configuration restrictions in Appendix C.5.

- The render parameter MUST be assigned the value "synthetic", "unknown", "null", or a render value that has been added to the IANA repository for use with mpeg4-generic RTP MIDI streams. The "api" token value for render MUST NOT be used.

- If a subrender parameter is present, it MUST immediately follow the render parameter, and it MUST be assigned the token value "default" or assigned a subrender value added to the IANA repository for use with mpeg4-generic RTP MIDI streams. A subrender parameter assignment may be left out of the renderer configuration, in which case the implied value of subrender is the default value of "default".

- If the render parameter is assigned the value "synthetic" and the subrender parameter has the value "default" (assigned or implied), the rinit parameter MUST be assigned the value "audio/asc", and an AudioSpecificConfig data object MUST be encoded using the mechanisms defined in C.6.2-3. The AudioSpecificConfig data MUST encode one of the MPEG 4 Audio Object Types defined for use with mpeg4-generic in Section 6.2. If the subrender value is other than "default", refer to the subrender registration for information on the use of "audio/asc" with the renderer.

- If the render parameter is assigned the value "null" or "unknown", the data object MAY be omitted.

Several general restrictions apply to the use of the audio/asc media type in RTP MIDI:

- A native stream MUST NOT assign "audio/asc" to rinit. The audio/asc media type is not intended to be a general-purpose container for rendering systems outside of MPEG usage.

- The audio/asc media type defines a stored object type; it does not define semantics for RTP streams. Thus, audio/asc MUST NOT appear on an rtpmap line of a session description.

Below, we show session description examples for audio/asc. The session description below uses the inline parameter to code the AudioSpecificConfig block for a mpeg4-generic General MIDI stream. We derive the value assigned to the inline parameter in Appendix E.4. The subrender token value of "default" is implied by the absence of the subrender parameter in the parameter list.

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP4 192.0.2.94
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; render=synthetic; rinit="audio/asc";
inline="egoAAAAaTVRoZAAAAAYAAAABAGBNVHJrAAAABgD/LwAA"
```

(The a=fmtp line has been wrapped to fit the page to accommodate memo formatting restrictions; it comprises a single line in SDP.)

The session description below uses the url parameter to code the AudioSpecificConfig block for the same General MIDI stream:

```
v=0
o=lazzaro 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
m=audio 5004 RTP/AVP 96
c=IN IP4 192.0.2.94
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; render=synthetic; rinit="audio/asc";
url="http://example.net/oski.asc";
cid="xjflsoeiurvpa09itnvlduihgnvet98pa3w9utnuighbuk"
```

(The a=fmtp line has been wrapped to fit the page to accommodate memo formatting restrictions; it comprises a single line in SDP.)

C.7. Interoperability

In this appendix, we define interoperability guidelines for two application areas:

- MIDI content-streaming applications. RTP MIDI is added to RTSP-based content-streaming servers, so that viewers may experience MIDI performances (produced by a specified client-side renderer) in synchronization with other streams (video, audio).

- Long-distance network musical performance applications. RTP MIDI is added to SIP-based voice chat or videoconferencing programs, as an alternative, or as an addition, to audio and/or video RTP streams.

For each application, we define a core set of functionality that all implementations MUST implement.

The applications we address in this section are not an exhaustive list of potential RTP MIDI uses. We expect framework documents for other applications to be developed, within the IETF or within other organizations. We discuss other potential application areas for RTP MIDI in Section 1 of the main text of this memo.

C.7.1. MIDI Content Streaming Applications

In content-streaming applications, a user invokes an RTSP client to initiate a request to an RTSP server to view a multimedia session. For example, clicking on a web page link for an Internet Radio channel launches an RTSP client that uses the link’s RTSP URL to contact the RTSP server hosting the radio channel.

The content may be pre-recorded (for example, on-demand replay of yesterday’s football game) or "live" (for example, football game coverage as it occurs), but in either case the user is usually an "audience member" as opposed to a "participant" (as the user would be in telephony).

Note that these examples describe the distribution of audio content to an audience member. The interoperability guidelines in this appendix address RTP MIDI applications of this nature, not applications such as the transmission of raw MIDI command streams for use in a professional environment (recording studio, performance stage, etc.).

In an RTSP session, a client accesses a session description that is "declared" by the server, either via the RTSP DESCRIBE method, or via other means, such as HTTP or email. The session description defines the session from the perspective of the client. For example, if a media line in the session description contains a non-zero port number, it encodes the server’s preference for the client’s port numbers for RTP and RTCP reception. Once media flow begins, the server sends an RTP MIDI stream to the client, which renders it for presentation, perhaps in synchrony with video or other audio streams.

We now define the interoperability text for content-streaming RTSP applications.

In most cases, server interoperability responsibilities are described in terms of limits on the "reference" session description a server provides for a performance if it has no information about the capabilities of the client. The reference session is a "lowest common denominator" session that maximizes the odds that a client will be able to view the session. If a server is aware of the capabilities of the client, the server is free to provide a session description customized for the client in the DESCRIBE reply.

Clients MUST support unicast UDP RTP MIDI streams that use the recovery journal with the closed-loop or the anchor sending policies. Clients MUST be able to interpret stream subsetting and chapter inclusion parameters in the session description that qualify the sending policies. Client support of enhanced Chapter C encoding is OPTIONAL.

The reference session description offered by a server MUST send all RTP MIDI UDP streams as unicast streams that use the recovery journal and the closed-loop or anchor sending policies. Servers SHOULD use the stream subsetting and chapter inclusion parameters in the reference session description, to simplify the rendering task of the client. Server support of enhanced Chapter C encoding is OPTIONAL.

Clients and servers MUST support the use of RTSP interleaved mode (a method for interleaving RTP onto the RTSP TCP transport).

Clients MUST be able to interpret the timestamp semantics signalled by the "comex" value of the tsmode parameter (i.e., the timestamp semantics of Standard MIDI Files [MIDI]). Servers MUST use the "comex" value for the "tsmode" parameter in the reference session description.

Clients MUST be able to process an RTP MIDI stream whose packets encode an arbitrary temporal duration ("media time"). Thus, in practice, clients MUST implement a MIDI playout buffer. Clients MUST NOT depend on the presence of rtp_ptime, rtp_maxtime, and guardtime parameters in the session description in order to process packets, but they SHOULD be able to use these parameters to improve packet processing.

Servers SHOULD strive to send RTP MIDI streams in the same way media servers send conventional audio streams: a sequence of packets that either all code the same temporal duration (non-normative example: 50 ms packets) or that code one of an integral number of temporal durations (non-normative example: 50 ms, 100 ms, 250 ms, or 500 ms packets). Servers SHOULD encode information about the packetization method in the rtp_ptime and rtp_maxtime parameters in the session description.

Clients MUST be able to examine the render and subrender parameter, to determine if a multimedia session uses a renderer it supports. Clients MUST be able to interpret the default "one" value of the "multimode" parameter, to identify supported renderers from a list of renderer descriptions. Clients MUST be able to interpret the musicport parameter, to the degree that it is relevant to the renderers it supports. Clients MUST be able to interpret the chanmask parameter.

Clients supporting renderers whose data object (as encoded by a parameter value for "inline") could exceed 300 octets in size MUST support the url and cid parameters and thus must implement the HTTP protocol in addition to RTSP.

Servers MUST specify complete rendering systems for RTP MIDI streams. Note that a minimal RTP MIDI native stream does not meet this requirement (Section 6.1), as the rendering method for such streams is "not specified".

At the time of this memo, the only way for servers to specify a complete rendering system is to specify an mpeg4-generic RTP MIDI stream in mode rtp-midi (Section 6.2 and C.6.5). As a consequence, the only rendering systems that may be presently used are General MIDI [MIDI], DLS 2 [DLS2], or Structured Audio [MPEGSA]. Note that the maximum inline value for General MIDI is well under 300 octets (and thus clients need not support the "url" parameter), and that the maximum inline values for DLS 2 and Structured Audio may be much larger than 300 octets (and thus clients MUST support the url parameter).

We anticipate that the owners of rendering systems (both standardized and proprietary) will register subrender parameters for their renderers. Once registration occurs, native RTP MIDI sessions may use render and subrender (Appendix C.6.2) to specify complete rendering systems for RTSP content-streaming multimedia sessions.

Servers MUST NOT use the sdp_start value for the smf_info parameter in the reference session description, as this use would require that clients be able to parse and render Standard MIDI Files.

Clients MUST support mpeg4-generic mode rtp-midi General MIDI (GM) sessions, at a polyphony limited by the hardware capabilities of the client. This requirement provides a "lowest common denominator" rendering system for content providers to target. Note that this requirement does not force implementors of a non-GM renderer (such as DLS 2 or Structured Audio) to add a second rendering engine. Instead, a client may satisfy the requirement by including a set of voice patches that implement the GM instrument set, and using this emulation for mpeg4-generic GM sessions.

It is RECOMMENDED that servers use General MIDI as the renderer for the reference session description, because clients are REQUIRED to support it. We do not require General MIDI as the reference renderer, because for normative applications it is an inappropriate choice. Servers using General MIDI as a "lowest common denominator" renderer SHOULD use Universal Real-Time SysEx MIP message [SPMIDI] to communicate the priority of voices to polyphony-limited clients.

C.7.2. MIDI Network Musical Performance Applications

In Internet telephony and videoconferencing applications, parties interact over an IP network as they would face-to-face. Good user experiences require low end-to-end audio latency and tight audiovisual synchronization (for "lip-sync"). The Session Initiation Protocol (SIP, [RFC3261]) is used for session management.

In this appendix section, we define interoperability guidelines for using RTP MIDI streams in interactive SIP applications. Our primary interest is supporting Network Musical Performances (NMP), where musicians in different locations interact over the network as if they were in the same room. See [NMP] for background information on NMP, and see [RFC4696] for a discussion of low-latency RTP MIDI implementation techniques for NMP.

Note that the goal of NMP applications is telepresence: the parties should hear audio that is close to what they would hear if they were in the same room. The interoperability guidelines in this appendix address RTP MIDI applications of this nature, not applications such as the transmission of raw MIDI command streams for use in a professional environment (recording studio, performance stage, etc.).

We focus on session management for two-party unicast sessions that specify a renderer for RTP MIDI streams. Within this limited scope, the guidelines defined here are sufficient to let applications interoperate. We define the REQUIRED capabilities of RTP MIDI senders and receivers in NMP sessions and define how session descriptions exchanged are used to set up network musical performance sessions.

SIP lets parties negotiate details of the session, using the Offer/Answer protocol [RFC3264]. However, RTP MIDI has so many parameters that "blind" negotiations between two parties using different applications might not yield a common session configuration.

Thus, we now define a set of capabilities that NMP parties MUST support. Session description offers whose options lie outside the envelope of REQUIRED party behavior risk negotiation failure. We also define session description idioms that the RTP MIDI part of an offer MUST follow, in order to structure the offer for simpler analysis.

We use the term "offerer" for the party making a SIP offer, and "answerer" for the party answering the offer. Finally, we note that unless it is qualified by the adjective "sender" or "receiver", a statement that a party MUST support X implies that it MUST support X for both sending and receiving.

If an offerer wishes to define a "sendrecv" RTP MIDI stream, it may use a true sendrecv session or the "virtual sendrecv" construction described in the preamble to Appendix C and in Appendix C.5. A true sendrecv session indicates that the offerer wishes to participate in a session where both parties use identically configured renderers. A virtual sendrecv session indicates that the offerer is willing to participate in a session where the two parties may be using different renderer configurations. Thus, parties MUST be prepared to see both real and virtual sendrecv sessions in an offer.

Parties MUST support unicast UDP transport of RTP MIDI streams. These streams MUST use the recovery journal with the closed-loop or anchor sending policies. These streams MUST use the stream subsetting and chapter inclusion parameters to declare the types of MIDI commands that will be sent on the stream (for sendonly streams) or will be processed (for recvonly streams), including the size limits on System Exclusive commands. Support of enhanced Chapter C encoding is OPTIONAL.

Note that both TCP and multicast UDP support are OPTIONAL. We make TCP OPTIONAL because we expect NMP renderers to rely on data objects (signalled by "rinit" and associated parameters) for initialization at the start of the session, and only to use System Exclusive commands for interactive control during the session. These interactive commands are small enough to be protected via the recovery journal mechanism of RTP MIDI UDP streams.

We now discuss timestamps, packet timing, and packet sending algorithms.

Recall that the tsmode parameter controls the semantics of command timestamps in the MIDI list of RTP packets.

Parties MUST support clock rates of 44.1 kHz, 48 kHz, 88.2 kHz, and 96 kHz. Parties MUST support streams using the "comex", "async", and "buffer" tsmode values. Recvonly offers MUST offer the default "comex".

Parties MUST support a wide range of packet temporal durations: from rtp_ptime and rtp_maxptime values of 0, to rtp_ptime and rtp_maxptime values that code 100 ms. Thus, receivers MUST be able to implement a playout buffer.

Offers and answers MUST present rtp_ptime, rtp_maxptime, and guardtime values that support the latency that users would expect in the application, subject to bandwidth constraints. As senders MUST abide by values set for these parameters in a session description, a receiver SHOULD use these values to size its playout buffer to produce the lowest reliable latency for a session. Implementers should refer to [RFC4696] for information on packet sending algorithms for latency-sensitive applications. Parties MUST be able to implement the semantics of the guardtime parameter, for times from 5 ms to 5000 ms.

We now discuss the use of the render parameter.

Sessions MUST specify complete rendering systems for all RTP MIDI streams. Note that a minimal RTP MIDI native stream does not meet this requirement (Section 6.1), as the rendering method for such streams is "not specified".

At the time this writing, the only way for parties to specify a complete rendering system is to specify an mpeg4-generic RTP MIDI stream in mode rtp-midi (Section 6.2 and C.6.5). We anticipate that the owners of rendering systems (both standardized and proprietary) will register subrender values for their renderers. Once IANA registration occurs, native RTP MIDI sessions may use render and subrender (Appendix C.6.2) to specify complete rendering systems for SIP network musical performance multimedia sessions.

All parties MUST support General MIDI (GM) sessions, at a polyphony limited by the hardware capabilities of the party. This requirement provides a "lowest common denominator" rendering system, without which practical interoperability will be quite difficult. When using GM, parties SHOULD use Universal Real-Time SysEx MIP message [SPMIDI] to communicate the priority of voices to polyphony-limited clients.

Note that this requirement does not force implementors of a non-GM renderer (for mpeg4-generic sessions, DLS 2, or Structured Audio) to add a second rendering engine. Instead, a client may satisfy the requirement by including a set of voice patches that implement the GM instrument set, and using this emulation for mpeg4-generic GM sessions. We require GM support so that an offerer that wishes to maximize interoperability may do so by offering GM if its preferred renderer is not accepted by the answerer.

Offerers MUST NOT present several renderers as options in a session description by listing several payload types on a media line, as Section 2.1 uses this construct to let a party send several RTP MIDI streams in the same RTP session.

Instead, an offerer wishing to present rendering options SHOULD offer a single payload type that offers several renderers. In this construct, the parameter list codes a list of render parameters (each followed by its support parameters). As discussed in Appendix C.6.1, the order of renderers in the list declares the offerer’s preference. The "unknown" and "null" values MUST NOT appear in the offer. The answer MUST set all render values except the desired renderer to "null". Thus, "unknown" MUST NOT appear in the answer.

We use SHOULD instead of MUST in the first sentence in the paragraph above, because this technique does not work in all situations (example: an offerer wishes to offer both mpeg4-generic renderers and native RTP MIDI renderers as options). In this case, the offerer MUST present a series of session descriptions, each offering a single renderer, until the answerer accepts a session description.

Parties MUST support the musicport, chanmask, subrender, rinit, and inline parameters. Parties supporting renderers whose data object (as encoded by a parameter value for "inline") could exceed 300 octets in size MUST support the url and cid parameters and thus must implement HTTP protocol. Note that in mpeg4-generic, General MIDI data objects cannot exceed 300 octets, but DLS 2 and Structured Audio data objects may. Support for the other rendering parameters (smf_cif, smf_info, smf_inline, smf_url) is OPTIONAL.

Thus far in this document, our discussion has assumed that the only MIDI flows that drive a renderer are the network flows described in the session description. In NMP applications, this assumption would require two rendering engines: one for local use by a party, a second for the remote party.

In practice, applications may wish to have both parties share a single rendering engine. In this case, the session description MUST use a virtual sendrecv session and MUST use the stream subsetting and chapter inclusion parameters to allocate which MIDI channels are intended for use by a party. If two parties are sharing a MIDI channels, the application MUST ensure that appropriate MIDI merging occurs at the input to the renderer.

We now discuss the use of (non-MIDI) audio streams in the session.

Audio streams may be used for two purposes: as a "talkback" channel for parties to converse, or as a way to conduct a performance that includes MIDI and audio channels. In the latter case, offers MUST use sample rates and the packet temporal durations for the audio and MIDI streams that support low-latency synchronized rendering.

We now show an example of an offer/answer exchange in a network musical performance application (next page). Below, we show an offer that complies with the interoperability text in this appendix section.

```
v=0
o=first 2520644554 2838152170 IN IP4 first.example.net
s=Example
t=0 0
a=group:FID 1 2
c=IN IP4 192.0.2.94
m=audio 16112 RTP/AVP 96
a=recvonly
a=mid:1
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; cm_unused=ABCFGHJKMNPQTVWXYZ; cm_used=2NPTW;
cm_used=2C0.1.7.10.11.64.121.123; cm_used=2M0.1.2
cm_used=X0-16; ch_never=ABCDEFGHJKMNPQTVWXYZ;
ch_default=2NPTW; ch_default=2C0.1.7.10.11.64.121.123;
ch_default=2M0.1.2; cm_default=X0-16;
rtp_ptime=0; rtp_maxptime=0; guardtime=44100;
musicport=1; render=synthetic; rinit="audio/asc";
inline="egoAAAAaTVRoZAAAAAYAAAABAGBNVHJrAAAABgD/LwAA"
m=audio 16114 RTP/AVP 96
a=sendonly
a=mid:2
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; cm_unused=ABCFGHJKMNPQTVWXYZ; cm_used=1NPTW;
cm_used=1C0.1.7.10.11.64.121.123; cm_used=1M0.1.2
cm_used=X0-16; ch_never=ABCDEFGHJKMNPQTVWXYZ;
ch_default=1NPTW; ch_default=1C0.1.7.10.11.64.121.123;
ch_default=1M0.1.2; cm_default=X0-16;
rtp_ptime=0; rtp_maxptime=0; guardtime=44100;
musicport=1; render=synthetic; rinit="audio/asc";
inline="egoAAAAaTVRoZAAAAAYAAAABAGBNVHJrAAAABgD/LwAA"
```

(The a=fmtp lines have been wrapped to fit the page to accommodate memo formatting restrictions; it comprises a single line in SDP.)

The owner line (o=) identifies the session owner as "first".

The session description defines two MIDI streams: a recvonly stream on which "first" receives a performance, and a sendonly stream that "first" uses to send a performance. The recvonly port number encodes the ports on which "first" wishes to receive RTP (16112) and RTCP (16113) media at IP4 address 192.0.2.94. The sendonly port number encodes the port on which "first" wishes to receive RTCP for the stream (16115).

The musicport parameters code that the two streams share and identity relationship and thus form a virtual sendrecv stream.

Both streams are mpeg4-generic RTP MIDI streams that specify a General MIDI renderer. The stream subsetting parameters code that the recvonly stream uses MIDI channel 1 exclusively for voice commands, and that the sendonly stream uses MIDI channel 2 exclusively for voice commands. This mapping permits the application software to share a single renderer for local and remote performers.

We now show the answer to the offer.

```
v=0
o=second 2520644554 2838152170 IN IP4 second.example.net
s=Example
t=0 0
a=group:FID 1 2
c=IN IP4 192.0.2.105
m=audio 5004 RTP/AVP 96
a=sendonly
a=mid:1
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; cm_unused=ABCFGHJKMNPQTVWXYZ; cm_used=2NPTW;
cm_used=2C0.1.7.10.11.64.121.123; cm_used=2M0.1.2
cm_used=X0-16; ch_never=ABCDEFGHJKMNPQTVWXYZ;
ch_default=2NPTW; ch_default=2C0.1.7.10.11.64.121.123;
ch_default=2M0.1.2; cm_default=X0-16;
rtp_ptime=0; rtp_maxptime=882; guardtime=44100;
musicport=1; render=synthetic; rinit="audio/asc";
inline="egoAAAAaTVRoZAAAAAYAAAABAGBNVHJrAAAABgD/LwAA"
m=audio 5006 RTP/AVP 96
a=recvonly
a=mid:2
a=rtpmap:96 mpeg4-generic/44100
a=fmtp:96 streamtype=5; mode=rtp-midi; config="";
profile-level-id=12; cm_unused=ABCFGHJKMNPQTVWXYZ; cm_used=1NPTW;
cm_used=1C0.1.7.10.11.64.121.123; cm_used=1M0.1.2
cm_used=X0-16; ch_never=ABCDEFGHJKMNPQTVWXYZ;
ch_default=1NPTW; ch_default=1C0.1.7.10.11.64.121.123;
ch_default=1M0.1.2; cm_default=X0-16;
rtp_ptime=0; rtp_maxptime=0; guardtime=88200;
musicport=1; render=synthetic; rinit="audio/asc";
inline="egoAAAAaTVRoZAAAAAYAAAABAGBNVHJrAAAABgD/LwAA"
```

(The a=fmtp lines have been wrapped to fit the page to accommodate memo formatting restrictions; they comprise single lines in SDP.)

The owner line (o=) identifies the session owner as "second".

The port numbers for both media streams are non-zero; thus, "second" has accepted the session description. The stream marked "sendonly" in the offer is marked "recvonly" in the answer, and vice versa, coding the different view of the session held by "session". The IP4 number (192.0.2.105) and the RTP (5004 and 5006) and RTCP (5005 and 5007) have been changed by "second" to match its transport wishes.

In addition, "second" has made several parameter changes: rtp_maxptime for the sendonly stream has been changed to code 2 ms (441 in clock units), and the guardtime for the recvonly stream has been doubled. As these parameter modifications request capabilities that are REQUIRED to be implemented by interoperable parties, "second" can make these changes with confidence that "first" can abide by them.

D. Parameter Syntax Definitions

In this appendix, we define the syntax for the RTP MIDI media type parameters in Augmented Backus-Naur Form (ABNF, [RFC4234]). When using these parameters with SDP, all parameters MUST appear on a single fmtp attribute line of an RTP MIDI media description. For mpeg4-generic RTP MIDI streams, this line MUST also include any mpeg4-generic parameters (usage described in Section 6.2). An fmtp attribute line may be defined (after [RFC3640]) as:

```
;
; SDP fmtp line definition
;

fmtp = "a=fmtp:" token SP param-assign 0*(";" SP param-assign) CRLF
```

where <token> codes the RTP payload type. Note that white space MUST NOT appear between the "a=fmtp:" and the RTP payload type.

We now define the syntax of the parameters defined in Appendix C. The definition takes the form of the incremental assembly of the <param-assign> token. See [RFC3640] for the syntax of the mpeg4-generic parameters discussed in Section 6.2.

```
;
;
; top-level definition for all parameters
;
;

;
; Parameters defined in Appendix C.1

param-assign =   ("cm_unused="  (([channel-list] command-type
                                  [f-list]) / sysex-data))

param-assign =/  ("cm_used="    (([channel-list] command-type
                                  [f-list]) / sysex-data))
```

```
   ;
   ; Parameters defined in Appendix C.2

   param-assign =/  ("j_sec="      ("none" / "recj" / *ietf-extension))

   param-assign =/  ("j_update="   ("anchor" / "closed-loop" /
                                    "open-loop" / *ietf-extension))

   param-assign =/  ("ch_default=" (([channel-list] chapter-list
                                     [f-list]) / sysex-data))

   param-assign =/  ("ch_never="   (([channel-list] chapter-list
                                     [f-list]) / sysex-data))

   param-assign =/  ("ch_anchor="  (([channel-list] chapter-list
                                     [f-list]) / sysex-data))

   ;
   ; Parameters defined in Appendix C.3

   param-assign =/  ("tsmode="     ("comex" / "async" / "buffer"))

   param-assign =/  ("linerate="    nonzero-four-octet)

   param-assign =/  ("octpos="      ("first" / "last"))

   param-assign =/  ("mperiod="     nonzero-four-octet)

   ;
   ; Parameter defined in Appendix C.4

   param-assign =/  ("guardtime="     nonzero-four-octet)

   param-assign =/  ("rtp_ptime="     four-octet)

   param-assign =/  ("rtp_maxptime="  four-octet)

   ;
   ; Parameters defined in Appendix C.5

   param-assign =/  ("musicport="     four-octet)
   ;
   ; Parameters defined in Appendix C.6

   param-assign =/  ("chanmask="     ( 1*( 16( "0" / "1" ) )))

   param-assign =/  ("cid="          double-quote cid-block
                                     double-quote)

   param-assign =/  ("inline="       double-quote base-64-block
                                     double-quote)

   param-assign =/  ("multimode="    ("all" / "one"))

   param-assign =/  ("render="       ("synthetic" / "api" / "null" /
                                      "unknown" / *extension))

   param-assign =/  ("rinit="        mime-type "/" mime-subtype)

   param-assign =/  ("smf_cid="      double-quote cid-block
                                     double-quote)

   param-assign =/  ("smf_info="     ("ignore" / "identity" /
                                     "sdp_start" / *extension))

   param-assign =/  ("smf_inline="   double-quote base-64-block
                                     double-quote)

   param-assign =/  ("smf_url="      double-quote uri-element
                                     double-quote)

   param-assign =/  ("subrender="    ("default" / *extension))

   param-assign =/  ("url="          double-quote uri-element
                                     double-quote)

   ;
   ; list definitions for the cm_ command-type
   ;

   command-type    = command-part1 command-part2 command-part3

   command-part1   = (*1"A") (*1"B") (*1"C") (*1"F") (*1"G") (*1"H")

   command-part2   = (*1"J") (*1"K") (*1"M") (*1"N") (*1"P") (*1"Q")

   command-part3   = (*1"T") (*1"V") (*1"W") (*1"X") (*1"Y") (*1"Z")
   ;
   ; list definitions for the ch_ chapter-list
   ;

   chapter-list  =  ch-part1 ch-part2 ch-part3

   ch-part1  = (*1"A") (*1"B") (*1"C") (*1"D") (*1"E") (*1"F") (*1"G")

   ch-part2  = (*1"H") (*1"J") (*1"K") (*1"M") (*1"N") (*1"P") (*1"Q")

   ch-part3  = (*1"T") (*1"V") (*1"W") (*1"X") (*1"Y") (*1"Z")

   ;
   ; list definitions for the ch_ channel-list
   ;

   channel-list       = midi-chan-element *("." midi-chan-element)

   midi-chan-element  = midi-chan / midi-chan-range

   midi-chan-range    = midi-chan "-" midi-chan

                      ; decimal value of left midi-chan
                      ; MUST be strictly less than decimal
                      ; value of right midi-chan

   midi-chan          = %d0-15

   ;
   ; list definitions for the ch_ field list (f-list)
   ;

   f-list             = midi-field-element *("." midi-field-element)

   midi-field-element = midi-field / midi-field-range

   midi-field-range   = midi-field "-" midi-field
                      ;
                      ; decimal value of left midi-field
                      ; MUST be strictly less than decimal
                      ; value of right midi-field

   midi-field         = four-octet
                      ;
                      ; large range accommodates Chapter M
                      ; RPN (0-16383) and NRPN (16384-32767)
                      ; parameters, and Chapter X octet sizes.
   ;
   ; definitions for ch_ sysex-data
   ;

   sysex-data         = "__"  h-list *("_" h-list) "__"

   h-list             = hex-field-element *("." hex-field-element)

   hex-field-element  = hex-octet / hex-field-range

   hex-field-range    = hex-octet "-" hex-octet
                      ;
                      ; hexadecimal value of left hex-octet
                      ; MUST be strictly less than hexadecimal
                      ; value of right hex-octet

   hex-octet          = 2("0" / "1" / "2"/ "3" / "4" /
                          "5" / "6" / "7" / "8" / "9" /
                          "A" / "B" / "C" / "D" / "E" / "F")
                      ;
                      ; rewritten version of hex-octet in [RFC2045]
                      ; (page 23).
                      ; note that a-f are not permitted, only A-F.
                      ; hex-octet values MUST NOT exceed 7F.

   ;
   ; definitions for rinit parameter
   ;

   mime-type          = "audio" / "application"

   mime-subtype       = token
                      ;
                      ; See Appendix C.6.2 for registration
                      ; requirements for rinit type/subtypes.

   ;
   ; definitions for base64 encoding
   ; copied from [RFC4566]

   base-64-block      = *base64-unit [base64-pad]

   base64-unit        =  4base64-char

   base64-pad         =  2base64-char "==" / 3base64-char "="

   base64-char        =  %x41-5A / %x61-7A / %x30-39 / "+" / "/"
                      ;  A-Z, a-z, 0-9, "+" and "/"
   ;
   ; generic rules
   ;

   ietf-extension     = token
                      ;
                      ; ietf-extension may only be defined in
                      ; standards-track RFCs.

   extension          = token
                      ;
                      ; extension may be defined by filing
                      ; a registration with IANA.

   four-octet         = %d0-4294967295
                      ; unsigned encoding of 32-bits

   nonzero-four-octet = %d1-4294967295
                      ; unsigned encoding of 32-bits, ex-zero

   uri-element        = URI-reference
                      ; as defined in [RFC3986]

   double-quote       = %x22

                      ; the double-quote (") character

   token              =  1*token-char
                      ; copied from [RFC4566]

   token-char         =  %x21 / %x23-27 / %x2A-2B / %x2D-2E /
                         %x30-39 / %x41-5A / %x5E-7E
                      ; copied from [RFC4566]

   cid-block          = 1*cid-char

   cid-char           =  token-char
   cid-char           =/  "@"
   cid-char           =/  ","
   cid-char           =/  ";"
   cid-char           =/  ":"
   cid-char           =/  "\"
   cid-char           =/  "/"
   cid-char           =/  "["
   cid-char           =/  "]"
   cid-char           =/  "?"
   cid-char           =/  "="
```

```
                      ;
                      ; add back in the tspecials [RFC2045], except for
                      ; double-quote and the non-email safe () <>
                      ; note that "cid" defined above ensures that
                      ; cid-block is enclosed with double-quotes

; external references
; URI-reference: from [RFC3986]

;
; End of ABNF
```

The mpeg4-generic RTP payload [RFC3640] defines a "mode" parameter that signals the type of MPEG stream in use. We add a new mode value, "rtp-midi", using the ABNF rule below:

```
;
; mpeg4-generic mode parameter extension
;

mode                 =/ "rtp-midi"
                     ; as described in Section 6.2 of this memo
```

E. A MIDI Overview for Networking Specialists

This appendix presents an overview of the MIDI standard, for the benefit of networking specialists new to musical applications. Implementors should consult [MIDI] for a normative description of MIDI.

Musicians make music by performing a controlled sequence of physical movements. For example, a pianist plays by coordinating a series of key presses, key releases, and pedal actions. MIDI represents a musical performance by encoding these physical gestures as a sequence of MIDI commands. This high-level musical representation is compact but fragile: one lost command may be catastrophic to the performance.

MIDI commands have much in common with the machine instructions of a microprocessor. MIDI commands are defined as binary elements. Bitfields within a MIDI command have a regular structure and a specialized purpose. For example, the upper nibble of the first command octet (the opcode field) codes the command type. MIDI commands may consist of an arbitrary number of complete octets, but most MIDI commands are 1, 2, or 3 octets in length.

Channel Voice Messages	Bitfield Pattern
NoteOff (end a note)	1000cccc 0nnnnnnn 0vvvvvvv
NoteOn (start a note)	1001cccc 0nnnnnnn 0vvvvvvv
PTouch (Polyphonic Aftertouch)	1010cccc 0nnnnnnn 0aaaaaaa
CControl (Controller Change)	1011cccc 0xxxxxxx 0yyyyyyy
PChange (Program Change)	1100cccc 0ppppppp
CTouch (Channel Aftertouch)	1101cccc 0aaaaaaa
PWheel (Pitch Wheel)	1110cccc 0xxxxxxx 0yyyyyyy

Figure E.1 -- MIDI Channel Messages

System Common Messages	Bitfield Pattern
System Exclusive	11110000, followed by a list of 0xxxxxx octets, followed by 11110111
MIDI Time Code Quarter Frame	11110001 0xxxxxxx
Song Position Pointer	11110010 0xxxxxxx 0yyyyyyy
Song Select	11110011 0xxxxxxx
Undefined	11110100
Undefined	11110101
Tune Request	11110110
System Exclusive End Marker	11110111

System Realtime Messages	Bitfield Pattern
Clock	11111000
Undefined	11111001
Start	11111010
Continue	11111011
Stop	11111100
Undefined	11111101
Active Sense	11111110
System Reset	11111111

Figure E.2 -- MIDI System Messages

Figure E.1 and E.2 show the MIDI command family. There are three major classes of commands: voice commands (opcode field values in the range 0x8 through 0xE), system common commands (opcode field 0xF, commands 0xF0 through 0xF7), and system real-time commands (opcode field 0xF, commands 0xF8 through 0xFF). Voice commands code the musical gestures for each timbre in a composition. Systems commands perform functions that usually affect all voice channels, such as System Reset (0xFF).

E.1. Commands Types

Voice commands execute on one of 16 MIDI channels, as coded by its 4-bit channel field (field cccc in Figure E.1). In most applications, notes for different timbres are assigned to different channels. To support applications that require more than 16 channels, MIDI systems use several MIDI command streams in parallel, to yield 32, 48, or 64 MIDI channels.

As an example of a voice command, consider a NoteOn command (opcode 0x9), with binary encoding 1001cccc 0nnnnnnn 0aaaaaaa. This command signals the start of a musical note on MIDI channel cccc. The note has a pitch coded by the note number nnnnnnn, and an onset amplitude coded by note velocity aaaaaaa.

Other voice commands signal the end of notes (NoteOff, opcode 0x8), map a specific timbre to a MIDI channel (PChange, opcode 0xC), or set the value of parameters that modulate the timbral quality (all other voice commands). The exact meaning of most voice channel commands depends on the rendering algorithms the MIDI receiver uses to generate sound. In most applications, a MIDI sender has a model (in some sense) of the rendering method used by the receiver.

System commands perform a variety of global tasks in the stream, including "sequencer" playback control of pre-recorded MIDI commands (the Song Position Pointer, Song Select, Clock, Start, Continue, and Stop messages), SMPTE time code (the MIDI Time Code Quarter Frame command), and the communication of device-specific data (the System Exclusive messages).

E.2. Running Status

All MIDI command bitfields share a special structure: the leading bit of the first octet is set to 1, and the leading bit of all subsequent octets is set to 0. This structure supports a data compression system, called running status [MIDI], that improves the coding efficiency of MIDI.

In running status coding, the first octet of a MIDI voice command may be dropped if it is identical to the first octet of the previous MIDI voice command. This rule, in combination with a convention to consider NoteOn commands with a null third octet as NoteOff commands, supports the coding of note sequences using two octets per command.

Running status coding is only used for voice commands. The presence of a system common message in the stream cancels running status mode for the next voice command. However, system real-time messages do not cancel running status mode.

E.3. Command Timing

The bitfield formats in Figures E.1 and E.2 do not encode the execution time for a command. Timing information is not a part of the MIDI command syntax itself; different applications of the MIDI command language use different methods to encode timing.

For example, the MIDI command set acts as the transport layer for MIDI 1.0 DIN cables [MIDI]. MIDI cables are short asynchronous serial lines that facilitate the remote operation of musical instruments and audio equipment. Timestamps are not sent over a MIDI 1.0 DIN cable. Instead, the standard uses an implicit "time of arrival" code. Receivers execute MIDI commands at the moment of arrival.

In contrast, Standard MIDI Files (SMFs, [MIDI]), a file format for representing complete musical performances, add an explicit timestamp to each MIDI command, using a delta encoding scheme that is optimized for statistics of musical performance. SMF timestamps usually code timing using the metric notation of a musical score. SMF meta-events are used to add a tempo map to the file, so that score beats may be accurately converted into units of seconds during rendering.

E.4. AudioSpecificConfig Templates for MMA Renderers

In Section 6.2 and Appendix C.6.5, we describe how session descriptions include an AudioSpecificConfig data block to specify a MIDI rendering algorithm for mpeg4-generic RTP MIDI streams.

The bitfield format of AudioSpecificConfig is defined in [MPEGAUDIO]. StructuredAudioSpecificConfig, a key data structure coded in AudioSpecificConfig, is defined in [MPEGSA].

For implementors wishing to specify Structured Audio renderers, a full understanding of [MPEGSA] and [MPEGAUDIO] is essential. However, many implementors will limit their rendering options to the two MIDI Manufacturers Association renderers that may be specified in

AudioSpecificConfig: General MIDI (GM, [MIDI]) and Downloadable Sounds 2 (DLS 2, [DLS2]).

To aid these implementors, we reproduce the AudioSpecificConfig bitfield formats for a GM renderer and a DLS 2 renderer below. We have checked these bitfields carefully and believe they are correct. However, we stress that the material below is informative, and that [MPEGAUDIO] and [MPEGSA] are the normative definitions for AudioSpecificConfig.

As described in Section 6.2, a minimal mpeg4-generic session description encodes the AudioSpecificConfig binary bitfield as a hexadecimal string (whose format is defined in [RFC3640]) that is assigned to the "config" parameter. As described in Appendix C.6.3, a session description that uses the render parameter encodes the AudioSpecificConfig binary bitfield as a Base64-encoded string assigned to the "inline" parameter, or in the body of an HTTP URL assigned to the "url" parameter.

Below, we show a simplified binary AudioSpecificConfig bitfield format, suitable for sending and receiving GM and DLS 2 data:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| AOTYPE  |FREQIDX|CHANNEL|SACNK|  FILE_BLK 1 (required) ...    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|SACNK|              FILE_BLK 2 (optional) ...                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  ...  |1|SACNK| FILE_BLK N (optional) ...                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|        (first "0" bit terminates FILE_BLK list)
+-+-+
```

Figure E.3 -- Simplified AudioSpecificConfig

The 5-bit AOTYPE field specifies the Audio Object Type as an unsigned integer. The legal values for use with mpeg4-generic RTP MIDI streams are "15" (General MIDI), "14" (DLS 2), and "13" (Structured Audio). Thus, receivers that do not support all three mpeg4-generic renderers may parse the first 5 bits of an AudioSpecificConfig coded in a session description and reject sessions that specify unsupported renderers.

The 4-bit FREQIDX field specifies the sampling rate of the renderer. We show the mapping of FREQIDX values to sampling rates in Figure E.4. Senders MUST specify a sampling frequency that matches the RTP clock rate, if possible; if not, senders MUST specify the escape value. Receivers MUST consult the RTP clock parameter for the true sampling rate if the escape value is specified.

FREQIDX	Sampling Frequency
0x0	96000
0x1	88200
0x2	64000
0x3	48000
0x4	44100
0x5	32000
0x6	24000
0x7	22050
0x8	16000
0x9	12000
0xa	11025
0xb	8000
0xc	reserved
0xd	reserved
0xe	reserved
0xf	escape value

Figure E.4 -- FreqIdx encoding

The 4-bit CHANNEL field specifies the number of audio channels for the renderer. The values 0x1 to 0x5 specify 1 to 5 audio channels; the value 0x6 specifies 5+1 surround sound, and the value 0x7 specifies 7+1 surround sound. If the rtpmap line in the session description specifies one of these formats, CHANNEL MUST be set to the corresponding value. Otherwise, CHANNEL MUST be set to 0x0.

The CHANNEL field is followed by a list of one or more binary file data blocks. The 3-bit SACNK field (the chunk_type field in class StructuredAudioSpecificConfig, defined in [MPEGSA]) specifies the type of each data block.

For General MIDI, only Standard MIDI Files may appear in the list (SACNK field value 2). For DLS 2, only Standard MIDI Files and DLS 2 RIFF files (SACNK field value 4) may appear. For both of these file types, the FILE_BLK field has the format shown in Figure E.5: a 32-bit unsigned integer value (FILE_LEN) coding the number of bytes in the SMF or RIFF file, followed by FILE_LEN bytes coding the file data.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     FILE_LEN (32-bit, a byte count SMF file or RIFF file)     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  FILE_DATA (file contents, a list of FILE_LEN bytes) ...      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure E.5 -- The FILE_BLK field format

Note that several files may follow CHANNEL field. The "1" constant fields in Figure E.3 code the presence of another file; the "0" constant field codes the end of the list. The final "0" bit in Figure E.3 codes the absence of special coding tools (see [MPEGAUDIO] for details). Senders not using these tools MUST append this "0" bit; receivers that do not understand these coding tools MUST ignore all data following a "1" in this position.

The StructuredAudioSpecificConfig bitfield structure requires the presence of one FILE_BLK. For mpeg4-generic RTP MIDI use of DLS 2, FILE_BLKs MUST code RIFF files or SMF files. For mpeg4-generic RTP MIDI use of General MIDI, FILE_BLKs MUST code SMF files. By default, this SMF will be ignored (Appendix C.6.4.1). In this default case, a GM StructuredAudioSpecificConfig bitfield SHOULD code a FILE_BLK whose FILE_LEN is 0, and whose FILE_DATA is empty.

To complete this appendix, we derive the StructuredAudioSpecificConfig that we use in the General MIDI session examples in this memo. Referring to Figure E.3, we note that for GM, AOTYPE = 15. Our examples use a 44,100 Hz sample rate (FREQIDX = 4) and are in mono (CHANNEL = 1). For GM, a single SMF is encoded (SACNK = 2), using the SMF shown in Figure E.6 (a 26 byte file).

```
                 --------------------------------------------
                |  MIDI File = <Header Chunk> <Track Chunk>  |
                 --------------------------------------------

   <Header Chunk> = <chunk type> <length>     <format> <ntrks> <divsn>
                     4D 54 68 64  00 00 00 06  00 00    00 01   00 60

   <Track Chunk> = <chunk type>  <length>     <delta-time> <end-event>
                    4D 54 72 6B   00 00 00 04  00           FF 2F 00
```

Figure E.6 -- SMF file encoded in the example

Placing these constants in binary format into the data structure shown in Figure E.3 yields the constant shown in Figure E.7.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 1 1 1 1|0 1 0 0|0 0 0 1|0 1 0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0|0 1 0 0|1 1 0 1|0 1 0 1|0 1 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 1 1 0|1 0 0 0|0 1 1 0|0 1 0 0|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0|0 0 0 0|0 0 0 0|0 1 1 0|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 1|0 0 0 0|0 0 0 0|0 1 1 0|0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 1 0 0|1 1 0 1|0 1 0 1|0 1 0 0|0 1 1 1|0 0 1 0|0 1 1 0|1 0 1 1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 0|0 0 0 0|0 1 1 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0|0 0 0 0|1 1 1 1|1 1 1 1|0 0 1 0|1 1 1 1|0 0 0 0|0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|
+-+-+
```

Figure E.7 -- AudioSpecificConfig used in GM examples

Expressing this bitfield as an ASCII hexadecimal string yields:

```
7A0A0000001A4D546864000000060000000100604D54726B0000000600FF2F000
```

This string is assigned to the "config" parameter in the minimal mpeg4-generic General MIDI examples in this memo (such as the example in Section 6.2). Expressing this string in Base64 [RFC2045] yields:

```
egoAAAAaTVRoZAAAAAYAAAABAGBNVHJrAAAABgD/LwAA
```

This string is assigned to the "inline" parameter in the General MIDI example shown in Appendix C.6.5.

References

Normative References

[MIDI] MIDI Manufacturers Association. "The Complete MIDI 1.0 Detailed Specification", 1996.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3640] van der Meer, J., Mackie, D., Swaminathan, V., Singer, D., and P. Gentric, "RTP Payload Format for Transport of MPEG-4 Elementary Streams", RFC 3640, November 2003.

[MPEGSA] International Standards Organization. "ISO/IEC 14496 MPEG-4", Part 3 (Audio), Subpart 5 (Structured Audio), 2001.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[MPEGAUDIO] International Standards Organization. "ISO 14496 MPEG-4", Part 3 (Audio), 2001.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[DLS2] MIDI Manufacturers Association. "The MIDI Downloadable Sounds Specification", v98.2, 1998.

[RFC4234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3388] Camarillo, G., Eriksson, G., Holler, J., and H. Schulzrinne, "Grouping of Media Lines in the Session Description Protocol (SDP)", RFC 3388, December 2002.

[RP015] MIDI Manufacturers Association. "Recommended Practice 015 (RP-015): Response to Reset All Controllers", 11/98.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC3555] Casner, S. and P. Hoschka, "MIME Type Registration of RTP Payload Formats", RFC 3555, July 2003.

Informative References

[NMP] Lazzaro, J. and J. Wawrzynek. "A Case for Network Musical Performance", 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001) June 25-26, 2001, Port Jefferson, New York.

[GRAME] Fober, D., Orlarey, Y. and S. Letz. "Real Time Musical Events Streaming over Internet", Proceedings of the International Conference on WEB Delivering of Music 2001, pages 147-154.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[ALF]	Clark, D. D. and D. L. Tennenhouse. "Architectural considerations for a new generation of protocols", SIGCOMM Symposium on Communications Architectures and Protocols , (Philadelphia, Pennsylvania), pp. 200--208, IEEE, Sept. 1990.
[RFC4696]	Lazzaro, J. and J. Wawrzynek, "An Implementation Guide for RTP MIDI", RFC 4696, November 2006.
[RFC2205]	Braden, R., Zhang, L., Berson, S., Herzog, S., and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.
[RFC4288]	Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.
[RFC4289]	Freed, N. and J. Klensin, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 4289, December 2005.
[RFC4571]	Lazzaro, J. "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, July 2006.
[RFC2818]	Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
[SPMIDI]	MIDI Manufacturers Association. "Scalable Polyphony MIDI, Specification and Device Profiles", Document Version 1.0a, 2002.
[LCP]	Apple Computer. "Logic 7 Dedicated Control Surface Support", Appendix B. Product manual available from www.apple.com.

Authors' Addresses

John Lazzaro (corresponding author)
UC Berkeley
CS Division
315 Soda Hall
Berkeley CA 94720-1776

EMail: lazzaro@cs.berkeley.edu

John Wawrzynek
UC Berkeley
CS Division
631 Soda Hall
Berkeley CA 94720-1776

EMail: johnw@cs.berkeley.edu

Full Copyright Statement

Copyright (C) The IETF Trust (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST, AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM 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.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.