Acoustic Instrument Message Specification

v 0.4 Proposal
June 15, 2014

Keith McMillen Instruments
BEAM Foundation

Created by:
Keith McMillen - keith@beamfoundation.org
With contributions from :
Barry Threw - barry@beamfoundation.org &
Chris Shaver - hexxiiiz@gmail.com

Last Modified by Keith McMillen June 15, 2014

1.0 The Problem

1.1 Acoustic MetaData is Useful

The audio signal from an acoustic instrument is rich in information. Pitch, rhythmic and timbral data extracted from such a signal is useful in controlling rich synthesis intimately tied to an acoustic instrument. When designing such synthesizers it is desirable to have such information be high speed, continuous and coherent.

1.2 Limitations of Current Control Protocols

(Taken from ZIPI :Origins and Motivations MIT Press, Computer Music Journal, Winter 1994)

The success of alternate controllers has been less than overwhelming in the history of electronic music. The predominant controller for electronic music synthesizers has been the piano or organ keyboard. Beside the widespread availability of pianos and organs and the people who play them, the very nature of the keyboard makes it an ideal choice from an implementor's point of view.

Keyboard-style instruments decouple the player from the sound generating element. The strike of a finger on a key starts a chain of events that produces a sound. After a key is struck the greatest creative choice left to the musician is when to release it. This series of key closures and releases is the simplest form of information that can be used to control a synthesizer.

The early commercially available synthesizers (Moog's MiniMoog, the ARP Odyssey, or the EMS Putney VCS-3) were monophonic and non-dynamic. As the technology evolved, instruments became polyphonic and capable of wide dynamic response (e.g., the Yamaha DX-7). The control information fed to the synthesis engine grew to include how fast the key was struck (the MIDI key velocity). Joysticks, modulation wheels, after-touch, and foot-pedals added the continuous element to keyboard control.

Historically, this is not unfamiliar to keyboard players. Pipe organs are non-dynamic but volume can be controlled by foot pedal. In many ways the connection of keyboards to synthesizers resulted in very little loss of familiarity of control with a large gain in timbral choice.

For musicians trained on other instruments, the option of synthesis has not been attractive. Woodwinds, bowed strings, and brass instruments all place the player in direct physical contact with the vibrating element---reeds, strings, or columns of air. Instead of limited control of dozens of notes these instruments offer subtle and intimate control over one or a few notes. Whether to "trade in" this control for a wider tonal palette is a difficult decision.

MIDI and Keyboards

MIDI has been serving our interface needs for over a decade. Although many have criticized MIDI (Loy 1985; Moore 1988; Scholz 1991), no one has done much about its obvious problems. Alternate controllers have not been a major factor in the business of electronic music, and therefore have not been well accommodated by the industry. They represent a challenging problem both technically and economically. The persistence of an interface standard that makes the necessary extensions for nuance and control difficult if not impossible has not helped.

The connection of keyboards to sample playback sound modules is well served by MIDI. Even the speed of MIDI (31.25 kBaud) is adequate for transmitting data using the event-based nature of a keyboard. A ten-note chord can be sent in 6.7 msec, a delay which is on the borderline of being imperceptible. The continuous controller information generated from a keyboard usually has no more than three parameters (pitch bend, modulation, and after-touch), keeping the bit count low.

Problems occur, however, when trying to connect alternate controllers to synthesizers (Muir and McMillen 1986). Polyphonic instruments such as guitar and violin can easily "flood" a MIDI channel with data. For example, simply updating 7-bit pitch bend and volume 100 times a second for six guitar strings exceeds MIDI's bandwidth:

6 strings * (3 pitch bytes + 3 volume bytes) * 10 bits / 0.01 sec = 36.0 kBaud

(MIDI takes 10 bits to transmit a 7 bit value).

Independent of bandwidth, MIDI also represents data in a manner that assumes the controller is a keyboard or at least a percussive device. The MIDI "note-on" command is an indivisible integration of timing, pitch, and loudness (velocity) information. This is completely appropriate for a keyboard; every time a key is struck the information for pitch, velocity, and the timing of the "note" is known simultaneously and is sent out over MIDI. Every modification of one of these three values is accompanied by a change, or at least a reassertion, of the other two.

For an instrument of continuous nature, such as a violin, these parameters

are often decoupled. One hand generally determines timing and loudness and the other decides pitch. They can and do change independently of each other. Furthermore, the timing of a note is not as simple as the pressing of a button. Notes can come into audibility gradually as in a crescendo. MIDI requires that an on/off decision be made at some volume threshold. When this threshold is met, the velocity value sent in a MIDI command will usually be the value of this threshold, making the velocity data useless.

MIDI does provide some facility for continuous volume change (controller #7), and for pitch change without articulation. Some synthesizers respond to legato-style commands. Pitch bend can vary pitch up and down up to one octave but with a resolution of only 5.1 divisions per semitone (19.6 cents).

Breaking the Chain

The loudest complaint about alternate controllers that extract information from traditional instruments is the time delay between the performer's gesture and the audible response from the synthesizer. However, the Zeta Mirror 6 guitar, using a combination of switched frets and pitch analysis, restricted latencies to less than 6 msec over most of its range. With the delay issues removed, continuous amplitude control became the next, most obvious, requirement. The technique previously described of controlling the audio in the post-MIDI analog domain met this need.

Amplitude control is essential but not sufficient; many instrumentalists can change the timbre of a note as it evolves over time. The mapping of timbral information extracted from the instrument onto the synthetic voice or voices is the next step for returning control to the performer. This too could be handled in the post-MIDI audio path, but the requirement for greater flexibility and more elaborate processing of control information is best solved in the digital domain. The need for a new music description language, and the means to transport this information in a high speed deterministic network, became clear to us.

The first concepts for what was to become the ZIPI musical data language started in the fall of 1989, coinciding with the start of intense collaboration of Zeta Music and the Center for New Music and Audio Technology (CNMAT) at the University of California, Berkeley. In order to successfully improve the keyboard-MIDI-sampler path, replacements were needed for all three of the elements in the chain. Since that date, research has focused on the Infinity Box (a gesture-guided pitch and timbre to ZIPI converter), the ZIPI network and its specification, and Frisco (an additive resynthesis engine with a control structure that will respond to ZIPI MPDL commands).

2.0 The Acoustic Instrument Message

The Acoustic Instrument Message is a simple data structure and specification for sending spectral information derived from an acoustic signal to analysis or synthesis applications. It is structured as a single 128-bit “frame” of information consisting of a voice number, articulation information, MIDI 1.0 compatibility fields, and seven spectral descriptors representing the state of the acoustic signal.

While this data frame is transport and protocol agnostic as long as the requirements for high transmission speed and coherence are met, it is recommended that the data be packaged with the Open Sound Control content format and sent over UDP.

The frame format and a description of its contents are described in the remainder of section 2, and the transport recommendations in section 3.

2.1 Structure Format

The AIM Packet is a 128-bit payload, and is generally divided into full bytes, with some exceptions. While many parameters are only a single byte some require greater resolution and are split between two bytes.

2.2 Voice

Currently the AIM frame specifies the four least significant bits of the first byte to be used for voice number, allowing for a total of 16 voices. With AIM’s origins in representing the signals from stringed instruments, this number was felt sufficient to address most common acoustic instruments.

The four most significant bits of the frame are left reserved for another level of addressing hierarchy, most likely an “instrument group” number.

Parameter Group	Bytes	Bits	Description	Range
Key Frame / Voice	1	1 Bit	Key Frame	0 - 1
		7 Bits	Voice Number	0 - 15
Articulation	1	6 Bits	Reserved	
		1 Bit	Trigger Flag	0 - 1
		1 Bit	Amplitude Gate Flag	0 - 1
		1 Bit	Null	0
Triggered Pitch	1	7 Bits	MIDI 1.0 Compatible Note Number	0 - 127
		1 Bit	Null	0
Triggered Amplitude	1	7 Bits	MIDI 1.0 Velocity	0 - 127
		1 Bit	Null	0
		7 Bits	MIDI 1.0 Pitch Bend MSB	0 - 127

Parameter Group	Bytes	Bits	Description	Range
Pitch Bend	2	1 Bit	Null	0
		7 Bits	MIDI 1.0 Pitch Bend LSB	0 - 127
		1 Bit	Null	0
Continuous Pitch	2	1 Byte	Semitone	0 - 255
		1 Byte	Fractional	0 - 255
Continuous Amplitude	1	1 Byte	dB in 0.5dB increments	0 - 255 (0 - 127dB)
Centroid	2	1 Byte	Semitone	0 - 255
		1 Byte	Fractional	0 - 255
Even / Odd Harmonic	1	1 Byte	1 = Odd, 255 = Even	1 - 255
Noise Amount	1	1 Byte	dB in 0.5dB increments	0 - 255 (0 - 127dB)
Noise Centroid	2	1 Byte	Semitone	0 - 255
		1 Byte	Fractional	0 - 255
Inharmonicity	1	1 Byte	1 = 126% Squeezed, 127 = None, 255 = 126% Stretched	1 - 255

2.3 Articulation

The first five most significant bits of the second byte of the AIM frame are reserved for future articulation designations.

The next three bits are defined as three flags representing three types of onset events that affect how the packet will be interpreted by the receiver.

2.3.1 Trigger Flag

1 Bit

The Trigger Flag is set high whenever a transient event occurs in the acoustic signal. This could be due to a change in bow direction, pluck, sudden note onset from a wind instrument or any kind of transient event or impulse that should be interpreted as a discrete sonic event.

2.3.2 Gate Flag

1 Bit

The Gate Flag is set high when there is sufficient audio amplitude in the acoustic signal. While what constitutes “sufficient audio amplitude” is not specified, the flag is meant to be a clear demarkation for note on and off for the synthesizer.

2.3.3 Key Frame Flag

1 Bit

The Key Frame Flag distinguishes between two different types of AIM Frames, *continuous* frames and *key* frames. The Key Frame Flag is set high on a key frame, and low on a continuous frame.

AIM *Continuous Frames* are sent on every spectral update of the acoustic signal. Ideally they should be sent for every analysis frame taken of the signal. While the rate may be slowed to accommodate available processor cycles or network bandwidth, they should be ideally be regular and low jitter in order to accurately represent the spectral state of the signal over time.

AIM *Key Frames* are sent upon a change in any of the articulation bits of the frame. They are asynchronous to the continuous frames and thus may occur outside of the continuous frame rate. However, key frames contain all the spectral information available when the key frame was triggered.

2.4 MIDI 1.0 Compatibility

The next four bytes are dedicated toward maintaining simple MIDI 1.0 compatibility with preexisting receivers.

2.4.1 Triggered Pitch

1 Byte

Triggered Pitch is a MIDI 1.0 compatible semitone value, from 0 to 127. The most significant bit is set to 0.

Note that this value should be identical to the value of the first byte of the next non-zero Continuous Pitch spectral descriptor following a frame with the Trigger Flag set high.

2.4.2 Triggered Amplitude

1 Byte

Triggered Amplitude is a MIDI 1.0 compatible velocity value, from 0 to 127. The most significant bit is set to 0.

2.4.3 Pitch Bend

2 Bytes

Pitch Bend is output as a a dual precision, 14-bit, standard MIDI value. The most significant bits of both bytes are set to 0.

2.5 Spectral Descriptors

2.5.1 Continuous Pitch

2 Bytes

Continuous Pitch represents the current fundamental pitch of the acoustic signal. It consists of two bytes. The first byte represents the current semitone value, with the MIDI standard 60 equal to C3.

The second byte represents a fractional part of the semitone, with 256 divisions per semitone. This gives a resolution of down to ~0.39 cents as the smallest representable pitch change.

When a fundamental pitch is undetermined, the continuous pitch value should be set to 0.

2.5.2 Continuous Amplitude

1 Byte

Continuous Amplitude represents the current volume of acoustic signal. It is represented in dB, in half dB increments. Therefore to find the current level of the signal in dB you must multiply this value by 0.5. This gives a total representable dynamic range of 127 dB.

2.5.3 Centroid

2 Bytes

Centroid represents the "center of mass" of the spectrum. Perceptually, it has a robust connection with the impression of "brightness" of a sound. It is calculated as the weighted mean of the frequencies present in the signal, with their magnitudes as the weights.

It consists of two bytes. The first byte represents the current semitone value, with the MIDI standard 60 equal to C3 or ~261.62 Hz.

The second byte represents a fractional part of the semitone, with 256 divisions per semitone. This gives a resolution of down to ~0.39 cents as the smallest representable pitch change.

2.5.4 Even/Odd Harmonic Balance

1 Byte

The Even/Odd Harmonic Balance is the ratio of odd harmonic energy to even harmonic energy in a signal. This allows the disambiguation between sounds with predominantly odd harmonics, such as clarinets, and sounds with predominantly even harmonics, such as trumpets.

A signal with completely odd harmonics is represented as 1, with 50% even/odd balance as 127, and completely even harmonics as 255.

2.5.5 Noise Amount

1 Byte

Noise Amount is the ratio of the energy of the non-harmonic part of a signal to the total energy. It is close to 0 for a purely harmonic signal and close to 255 for a purely noisy signal.

2.5.6 Noise Centroid

2 Bytes

Noise Centroid represents the "center of mass" of the spectrum of the non-harmonic part of the signal. Perceptually, it has a robust connection with the impression of "brightness" of a sound. It is calculated as the weighted mean of the frequencies present in the non-harmonic part of the signal, with their magnitudes as the weights.

It consists of two bytes. The first byte represents the current semitone value, with the MIDI standard 60 equal to C3 or ~261.62 Hz.

The second byte represents a fractional part of the semitone, with 256 divisions per semitone. This gives a resolution of down to ~0.39 cents as the smallest representable pitch change.

2.5.7 Inharmonicity

1 Byte

Inharmonicity represents the divergence of the signal’s spectral components from a purely harmonic signal. It is computed as an energy weighted divergence of the spectral components from the multiple of the fundamental frequency.

A signal with a 126% squeezed spectra is represented with a 1, a balanced spectra as a 127, and a 126% stretched spectra with a 255.

2.6 Parameter Timing

While each AIM frame (whether it is a continuous or a key frame) contains the full set of articulation and spectral information, the differing latency of various analysis methods means that data will not be available concurrently. The AIM frame should not be delayed to wait for a complete data set, but rather should be sent when any data is available.

In general, when a note starts the first parameter to be recognized will be the amplitude gate, followed by a trigger, followed by volume information, followed by pitch and spectral information. As an AIM key frame is triggered by both the amplitude gate and a trigger, those frames will most likely contain indeterminate or best guess spectral information. To start a note on a synthesizer with the least apparent latency a non-pitched or indeterminately pitched sound should be played upon the rising of an amplitude gate and then be updated with correct pitch, loudness and spectral information when it becomes available.

3.0 Recommendations

3.1 Open Sound Control

3.1.1 OSC Messages

3.1.2 OSC Bundles

3.1.2 OSC Time Tags

OSC Messages

An OSC message consists of an *OSC Address Pattern* followed by an *OSC Type Tag String* followed by zero or more *OSC Arguments*.

Note: some older implementations of OSC may omit the OSC Type Tag string. Until all such implementations are updated, OSC implementations should be robust in the case of a missing OSC Type Tag String.

OSC Address Patterns

An OSC Address Pattern is an OSC-string beginning with the character '/' (forward slash).

OSC Type Tag String

An OSC Type Tag String is an OSC-string beginning with the character ',' (comma) followed by a sequence of characters corresponding exactly to the sequence of OSC Arguments in the given message. Each character after the comma is called an *OSC Type Tag* and represents the type of the corresponding OSC Argument. (The requirement for OSC Type Tag Strings to start with a comma makes it easier for the recipient of an OSC Message to determine whether that OSC Message is lacking an OSC Type Tag String.)

This table lists the correspondance between each OSC Type Tag and the type of its corresponding OSC Argument:

OSC Bundles

An OSC Bundle consists of the OSC-string "#bundle" followed by an *OSC Time Tag*, followed by zero or more *OSC Bundle Elements*. The OSC-timetag is a 64-bit fixed point time tag whose semantics are described below.

An OSC Bundle Element consists of its *size* and its *contents*. The size is an int32 representing the number of 8-bit bytes in the contents, and will always be a multiple of 4. The contents are either an OSC Message or an OSC Bundle.

Note this recursive definition: bundle may contain bundles.

This table shows the parts of a two-or-more-element OSC Bundle and the size (in 8-bit bytes) of each part.

Parts of an OSC Bundle

Data	**Size**	**Purpose**
OSC-string "#bundle"	8 bytes	How to know that this data is a bundle
OSC-timetag	8 bytes	Time tag that applies to the entire bundle
Size of first bundle element	int32 = 4 bytes	
First bundle element		
First bundle element's contents	As many bytes as given by "size of first bundle element"	
Size of second bundle element	int32 = 4 bytes	

Second bundle element
Second bundle element's contents
As many bytes as given by "size of second bundle element"
etc.

Addtional bundle elements

Temporal Semantics and OSC Time Tags

An OSC server must have access to a representation of the correct current absolute time. OSC does not provide any mechanism for clock synchronization.

When a received OSC Packet contains only a single OSC Message, the OSC Server should invoke the correponding OSC Methods immediately, i.e., as soon as possible after receipt of the packet. Otherwise a received OSC Packet contains an OSC Bundle, in which case the OSC Bundle's OSC Time Tag determines when the OSC Bundle's OSC Messages' corresponding OSC Methods should be invoked. If the time represented by the OSC Time Tag is before or equal to the current time, the OSC Server should invoke the methods immediately (unless the user has configured the OSC Server to discard messages that arrive too late). Otherwise the OSC Time Tag represents a time in the future, and the OSC server must store the OSC Bundle until the specified time and then invoke the appropriate OSC Methods.

Time tags are represented by a 64 bit fixed point number. The first 32 bits specify the number of seconds since midnight on January 1, 1900, and the last 32 bits specify fractional parts of a second to a precision of about 200 picoseconds. This is the representation used by Internet NTP timestamps.The time tag value consisting of 63 zero bits followed by a one in the least signifigant bit is a special case meaning "immediately."

OSC Messages in the same OSC Bundle are *atomic*; their corresponding OSC Methods should be invoked in immediate succession as if no other processing took place between the OSC Method invocations.

When an OSC Address Pattern is dispatched to multiple OSC Methods, the order in which the matching OSC Methods are invoked is unspecified. When an OSC Bundle contains multiple OSC Messages, the sets of OSC Methods corresponding to the OSC Messages must be invoked in the same order as the OSC Messages appear in the packet. (example)

When bundles contain other bundles, the OSC Time Tag of the enclosed bundle must be greater than or equal to the OSC Time Tag of the enclosing bundle. The atomicity requirement for OSC Messages in the same OSC Bundle does not apply to OSC Bundles within an OSC Bundle.

3.2 Transport

3.3 Zeroconf Networking