MIDI 2.0 Bit Scaling and Resolution

MIDI Association Document: M2-115-U

Document Version 1.0.2
Draft Date 2023-12-18

Published 2023-12-19

Developed and Published By
The MIDI Association
and
Association of Musical Electronics Industry (AMEI)

PREFACE

MIDI Association Document M2-115-U
MIDI 2.0 Bit Scaling and Resolution

Universal MIDI Packet (UMP) is part of the MIDI 2.0 specifications released in 2018. UMP Provides a way to transmit both MIDI 1.0 Protocol and MIDI 2.0 Protocol and translate the data between them. This document defines recommended practices for scaling values and handling of stepped/enumerated values. For information on how to transmit and receive messages over UMP transports please read the Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification.

http://www.amei.or.jp

MIDI® Association

https://www.midi.org

Version History

Table 1 Version History

Publication Date	Version	Changes
June 15, 2023	1.0.1	Initial release
December 15, 2023	1.0.2	Minor Update to section 3.1, Fixed errors in Table 9

Contents

Figures

Tables

1 Introduction

1.1 Executive Summary

The methods for upscaling values, down scaling values and using stepped values described in this document allow for systems to use both MIDI 1.0 Protocol and MIDI 2.0 Protocol interchangeably.

This document provides a set of rules and guidelines for:

- UMP Resolution vs. Significant Bits, and how to declare the number of significant bits
- Incremental (i.e., “relative”) control change messages
- Scaling values between MIDI 1.0 Channel Voice Messages and MIDI 2.0 Channel Voice Messages
- How to select which scaling method to use
- How to encode stepped values and enumerations in messages such as Control Change, Registered Controller, and Assignable Controller Messages

1.2 Background

MIDI supports different bit resolutions for some properties. Typical values used are expressed in 7 bits, 8 bits, 14 bits, 16 bits, or 32 bits. Some of these properties need to be scaled or translated between these various resolutions. For example:

- Control Change messages can be 7 or 32-bit
- (N)RPNs can be 7-bit (if LSB not sent), 14-bit or 32-bit
- Note velocity can be 7 or 16-bit
- Pitch can be 16 (7.9), 21 (7.14) and 32-bit (7.25)

This document defines specific mechanisms for dealing with the various resolutions available.

Some properties have uniquely defined stepped values. This document also defines some mechanisms for defining and scaling such properties.

1.3 Translation Rules and Special Controllers

The M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification ***[MA02]*** defines rules for translation between MIDI 1.0 and MIDI 2.0 Protocols. That specification lists certain Controllers that have special rules for translation. Those special rules in ***[MA02]*** shall be applied instead of applying the rules contained in this document.

1.4 Bit Depth to use for Values in Future Documents

Newer MIDI specifications should always use 16-bit or 32-bit resolution for value definitions. Older MIDI specifications, especially all MIDI 1.0 specifications, use 7 or 14-bit resolution. These values must be upscaled accordingly for use in MIDI 2.0 Protocol.

1.5 References

1.5.1 Normative References

[MA01] ***Complete MIDI 1.0 Detailed Specification***, Document Version 96.1, Third Edition, Association of Musical Electronics Industry, http://www.amei.or.jp/, and The MIDI Association, https://www.midi.org/

[MA02] ***M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol***, Version 1.1.2, Association of Musical Electronics Industry, http://www.amei.or.jp/, and The MIDI Association, https://www.midi.org/

[MA03] ***M2-117-UM MIDI-CI Property Exchange Controller Resources***, Version 1.0, Association of Musical Electronics Industry, http://www.amei.or.jp/, and The MIDI Association, https://www.midi.org/

1.5.2 Informative References

No informative references.

1.6 Terminology

1.6.1 Definitions

AMEI: Association of Musical Electronics Industry. Authority for MIDI Specifications in Japan.

Device: An entity, whether hardware or software, which can send and/or receive MIDI messages.

MA: MIDI Association.

MIDI 1.0 Protocol: Version 1.0 of the MIDI Protocol as originally specified in ***[MA01]*** and extended by MA and AMEI with numerous additional MIDI message definitions and Recommended Practices. The native format for the MIDI 1.0 Protocol is a byte stream, but it has been adapted for many different transports. MIDI 1.0 messages can be carried in UMP packets. The UMP format for the MIDI 1.0 Protocol is defined in the M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification ***[MA02]***.

MIDI 2.0: The MIDI environment that encompasses all of MIDI 1.0, MIDI-CI, Universal MIDI Packet (UMP), MIDI 2.0 Protocol, MIDI 2.0 messages, and other extensions to MIDI as described in AMEI and MA specifications.

MIDI 2.0 Protocol: Version 2.0 of the MIDI Protocol. The native format for MIDI 2.0 Protocol messages is UMP as defined in M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification ***[MA02]***

MIDI-CI: MIDI Capability Inquiry.

MIDI Association: Authority for MIDI specifications worldwide except Japan. See also MMA.

MIDI Manufacturers Association: a California nonprofit 501(c)6 trade organization, and the legal entity name of the MIDI Association.

MIDI Transport: A hardware or software MIDI connection used by a Device to transmit and/or receive MIDI messages to and/or from another Device.

MMA: MIDI Manufacturers Association.

PE: Property Exchange.

Property Exchange: A set of MIDI-CI Transactions by which one device may access data from another device.

Protocol: There are two defined MIDI Protocols: the MIDI 1.0 Protocol and the MIDI 2.0 Protocol, each with a data structure that defines the semantics for MIDI messages. See ***[MA01]*** and ***[MA02]***

Receiver: A MIDI Device which has a MIDI Transport connected to its MIDI In.

Resource: A defined collection of one or more PE Properties with an associated inquiry to access its Properties.

Transaction: An exchange of MIDI messages between two MIDI Devices with a bidirectional connection. All the MIDI messages in a single Transaction are associated and work together to accomplish one function. The simplest Transaction generally consists of an inquiry sent by one MIDI Device and an associated reply returned by a second MIDI Device. A Transaction may also consist of an inquiry from one MIDI Device and several associated replies from a second MIDI Device. A Transaction may be a more complex set of message exchanges, started by an initial inquiry from one MIDI Device and multiple, associated replies exchanged between the first MIDI Device and a second MIDI Device. Also see MIDI-CI Transaction.

UMP: Universal MIDI Packet, see ***[MA02]***.

Universal MIDI Packet (UMP): The Universal MIDI Packet is a data container which defines the data format for all MIDI 1.0 Protocol messages and all MIDI 2.0 Protocol messages. UMP is intended to be universally applicable, i.e., technically suitable for use in any transport where MA/AMEI elects to officially support UMP. For detailed definition see M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification ***[MA02]***.

1.6.2 Reserved Words and Specification Conformance

In this document, the following words are used solely to distinguish what is required to conform to this specification, what is recommended but not required for conformance, and what is permitted but not required for conformance:

Table 2 Words Relating to Specification Conformance

Word	Reserved For	Relation to Specification Conformance
shall	Statements of requirement	**Mandatory** A conformant implementation conforms to all 'shall' statements.
should	Statements of recommendation	**Recommended but not mandatory** An implementation that does not conform to some or all 'should' statements is still conformant, providing all 'shall' statements are conformed to.
may	Statements of permission	**Optional** An implementation that does not conform to some or all 'may' statements is still conformant, providing that all 'shall' statements are conformed to.

By contrast, in this document, the following words are never used for specification conformance statements; they are used solely for descriptive and explanatory purposes:

Table 3 Words Not Relating to Specification Conformance

Word	Reserved For	Relation to Specification Conformance
must	Statements of unavoidability	Describes an action to be taken that, while not required (or at least not directly required) by this specification, is unavoidable. Not used for statements of conformance requirement (see 'shall' above).
will	Statements of fact	Describes a condition that as a question of fact is necessarily going to be true, or an action that as a question of fact is necessarily going to occur, but not as a requirement (or at least not as a direct requirement) of this specification. Not used for statements of conformance requirements (see 'shall' above).
can	Statements of capability	Describes a condition or action that a system element is capable of possessing or taking. Not used for statements of conformance permission (see 'may' above).
might	Statements of possibility	Describes a condition or action that a system element is capable of electing to possess or take. Not used for statements of conformance permission (see 'may' above).

2 Significant Bits

Devices may capture data with less resolution than available in the Protocol, resulting in fewer significant bits than the Protocol suggests. In particular, the 32-bit values in Control Change messages will rarely reflect the resolution of physical controls.

2.1 Senders: Support Full Resolution or Upscale

Senders often have data of lower resolution from input devices (knobs, faders, etc.) than can be supported by MIDI 2.0 messages (analog-to-digital converters typically provide fewer than 32 bits of valid data). Senders shall always upscale the valid data they have, using the appropriate mechanisms defined by this document, to fit the full resolution of the target value field before sending a MIDI message.

2.2 Receivers: Support Full Resolution or Downscale

When a Receiver gets a MIDI message from a Sender, the Receiver shall assume that the values as presented are expressed in the full resolution of the data field. When a Receiver does not support the full resolution of a value field, it shall downscale the value using the appropriate mechanisms defined by this document.

2.3 Translators

Translators shall always downscale or upscale as necessary to provide the full resolution of the target value field.

Translation is sometimes a lossy process. The level of bit depth for UMP packets can present challenges when scaling down and then scaling up a value. For example, a 32-bit value can be sent to a Device that can only store 8 bits of data. Consequently, that value may not be the same if that data is later returned.

The UMP message does not have a way to declare the number of significant bits. However, MIDI-CI Property Exchange Controller Resources ***[MA03]*** specification describes Resources which declare the significant bits of controllers. Implementing this Resource will be beneficial for receiving/rendering, translation, and compliance tests.

2.4 Incremental/Relative Controllers

Sending incremental control change values with fewer significant bits than the receiver's internal resolution may cause the receiver to not change the respective controller value. To prevent that, receivers can keep track of the value in the sender's resolution, before converting to its internal resolution.

However, the sender cannot expect the receiver to aggregate small increments. It can use, MIDI-CI Property Exchange Controller Resources ***[MA03]*** specification to try to find out the significant bits of the respective receiver and adapt accordingly.

Alternatively, the sender can keep track of the cumulative relative values and send absolute values instead.

3 Min-Center-Max Scaling

The Min-Center-Max Scaling mechanism is best suited to unipolar and bipolar values (ie, continuous controllers whose minimum and maximum value do not vary with resolution). These are the most common value types in MIDI messages.

Min-Center-Max Scaling is the default value scaling. This scaling preserves the minimum, maximum and center value for all resolutions, and is suitable for 0…100% values like Control Change messages. Additional resolution bits are used to increase the possible number of steps available.

3.1 Identifying MIDI Values Which Use Min-Center-Max Scaling

Min-Center-Max is the scaling method for most MIDI Values. Values where this scaling method applies include but are not limited to the following:

- Poly Pressure
- Channel Pressure
- Pitch Bend
- Control Change
- Registered Controller (RPN) Messages where the Index LSB (RPN LSB) is 32 to 127
- Assignable Controller (NRPN) Messages
- Note On/Off Velocity

3.2 Core Rules

These are the Core Rules for Min-Center-Max Scaling:

- Minimum/Lowest value is scaled to Minimum/Lowest
- Maximum/Highest value is scaled to Maximum/Highest
  For example, a 7-bit value of 127 is translated to a 16-bit value of 65535.
- Center Value (rounded up) scales to Center Value:
  `Center = TRUNC ((Highest + 1) / 2)` When upscaling, smoothly distribute low resolution values on the range of the high resolution.
- Scaling down a previously upscaled value yields the original value.

Note: In some cases, scaling in each direction might be performed by independent entities, and in such cases this result is not mandated.

Table 4 Center Value Examples:

Value Size	Center Value	
	Hex	Binary
7 bits	0x40	8’b 01000000
14 bits	0x2000	16’b 00100000 00000000
8 bits	0x80	8’b 10000000
16 bits	0x8000	16’b 10000000 00000000
32 bits	0x80000000	32’b 10000000 00000000 00000000 00000000

3.3 Upscaling Algorithm

For upscaling values of at least 2-bit to higher resolution, use this algorithm:

- For values from minimum to the center, use simple bit shifting. This ensures smooth increments towards the center value. The center value remains the center value.
- Use an expanded bit-repeat scheme for the range from center to maximum. This causes the values to smoothly increase from center to maximum value.

For upscaling 1-bit resolution (on/off), the value is expanded to the resolution boundaries. Zero always scales to zero. A value of one scales to all bits set, i.e., the maximum value.

For example, scaling 1 with 1-bit resolution to a 16-bit resolution, the value is 0xFFFF (65535).

3.3.1 Pseudo Code for the Upscaling Algorithm

```
// power of 2, pow(2, exp)
uint32_t power_of_2(uint8_t exp)
{
    return 1 << exp; // implement integer power of 2 using bit shift
}

// preconditions: srcBits > 1, dstBits<=32, srcBits < dstBits
uint32_t scaleUp(uint32_t srcVal, uint8_t srcBits, uint8_t dstBits)
{
    uint8_t scaleBits = (dstBits - srcBits);    // number of bits to upscale
    uint32_t srcCenter = power_of_2(srcBits-1); // center value for srcBits, e.g.
                                                // 0x40 (64) for 7 bits
                                                // 0x2000 (8192) for 14 bits
    // simple bit shift
    uint32_t bitShiftedValue = srcVal << scaleBits;
    if (srcVal <= srcCenter ) {
        return bitShiftedValue;
    }
    // expanded bit repeat scheme
    uint8_t repeatBits = srcBits - 1; // we must repeat all but the highest bit
    uint32_t repeatMask = power_of_2(repeatBits) - 1;
    uint32_t repeatValue = srcVal & repeatMask; // repeat bit sequence
    if (scaleBits > repeatBits) { // need to repeat multiple times
        repeatValue <<= (scaleBits - repeatBits);
    } else {
        repeatValue >>= (repeatBits - scaleBits);
    }
    while (repeatValue != 0) {
        bitShiftedValue |= repeatValue; // fill lower bits with repeatValue
        repeatValue >>= repeatBits;     // move repeat bit sequence to next position
    }
    return bitShiftedValue;
}
```

Note: This code example is optimized for readability, not efficiency.

First, the scaled value using bit shift is calculated by shifting left by the difference of the different bit sizes. If the original value is the center value or smaller, the bit shifted value is returned.

For values above the center, a `repeatValue` is calculated: it is the original value with the top 2 bits removed. So it has `repeatBits` significant bits. Finally, the `repeatValue` is used according to the Bit-Repeat scheme to fill the low order bits of the bit shifted value.

3.3.2 Value Upscaling Diagram

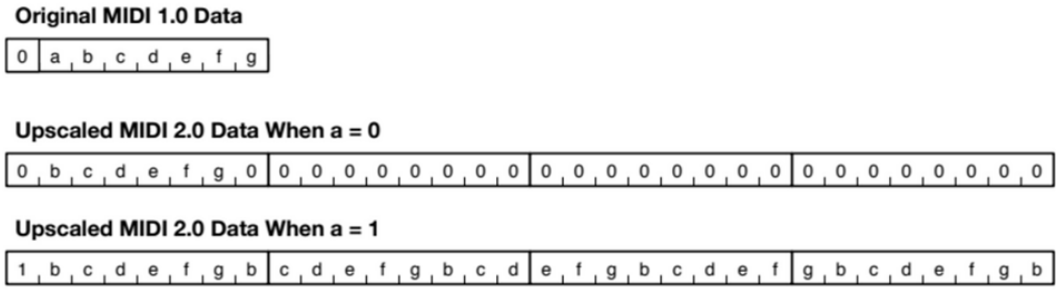

Figure 1 Value Upscaling Diagram

3.3.3 Examples

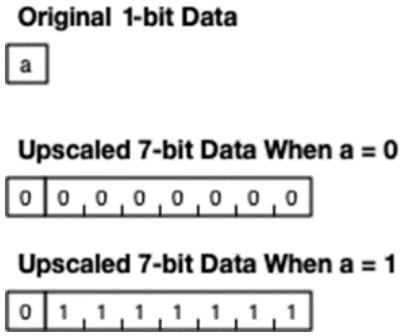

Figure 2 Upscale 1 to 7 bits

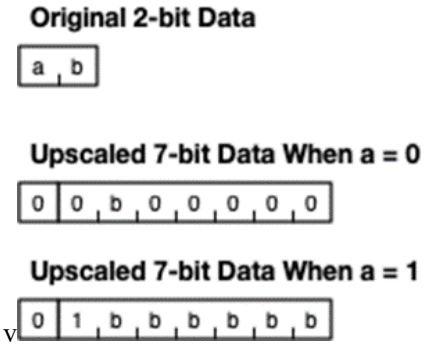

Figure 3 Upscale 2 to 7 bits

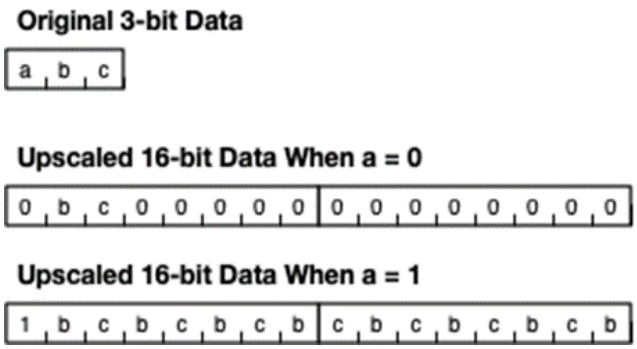

Figure 4 Upscale 3 to 16 bits

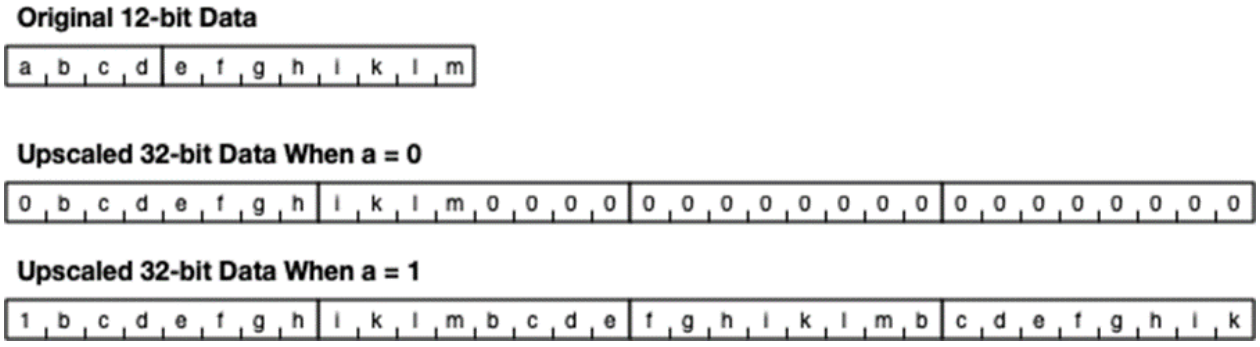

Figure 5 Upscale 12 to 32 bits

Table 5 Numeric Example: Upscale 7 to 16 bits

7-bit decimal	7-bit hex	16-bit decimal	16-bit hex
0	0x00	0	0x0000
5	0x05	2560	0x0A00
30	0x1E	15360	0x3C00
32	0x20	16384	0x4000
64	0x40	32768	0x8000
70	0x46	35888	0x8C30
96	0x60	49412	0xC104
120	0x78	61895	0xF1C7
127	0x7F	65535	0xFFFF

Table 6 Numeric Example: Upscale 7 to 32 bits

7-bit decimal	7-bit hex	32-bit decimal	32-bit hex
0	0x00	0	0x00000000
5	0x05	167772160	0x0A000000
30	0x1E	1006632960	0x3C000000
32	0x20	1073741824	0x40000000
64	0x40	2147483648	0x80000000
70	0x46	2352005900	0x8C30C30C
96	0x60	3238268993	0xC1041041
120	0x78	4056358001	0xF1C71C71
127	0x7F	4294967295	0xFFFFFFFF

Table 7 Numeric Example: Upscale 16 to 32 bits

16-bit decimal	16-bit hex	32-bit decimal	32-bit hex
0	0x0000	0	0x00000000
5	0x0005	327680	0x00050000
30	0x001E	1966080	0x001E0000
16384	0x4000	1073741824	0x40000000
32768	0x8000	2147483648	0x80000000
40000	0x9C40	2621454464	0x9C403880
49152	0xC000	3221258241	0xC0008001
65000	0xFDE8	4259904465	0xFDE8FBD1
65535	0xFFFF	4294967295	0xFFFFFFFF

3.4 Downscaling Method

For scaling a high-resolution value to a value with lower resolution, simple bit shifting (i.e. cutting off the lower bits) is sufficient and accurate enough.

3.4.1 Pseudo Code for Downscaling Algorithm

```
scaleDown(srcVal, srcBits, dstBits) {
    // simple bit shift
    uint scaleBits = (srcBits – dstBits);
    return srcVal >> scaleBits;
}
```

3.4.2 Examples

Table 8 Numeric Example: Downscale 16 to 7 bits

16-bit decimal	16-bit hex	7-bit decimal	7-bit hex
5120	0x1400	10	0x0A
32768	0x8000	64	0x40
44730	0xaeba	87	0x57
65535	0xffff	127	0x7F

4 Zero-Extension Scaling with Rounding

Zero-Extension Scaling with Rounding is useful for Controller values that are fixed point numbers, integers, or are defined in terms of very specific units. Values are padded with zeros when scaling up or rounded when scaling down.

Min-Center-Max Scaling has issues with absolute values like e.g., Pitch. It adds noise to the values in the upper half of the value range.

Example: Pitch 7.9 127.0 (0xFE00, 0b11111110.00000000) becomes
0b11111100.00000001.11111000.00000011 (0xFE01F803)
which is 127.00385, approx. 1/3 of a cent off.

With struct-like RPNs, like MPE Configuration Message, Min-Center-Max Scaling adds noise to the lower bits. Receivers should ignore this noise; however, Senders should not send this noise. The Zero-Extension Scaling mechanism provides a way to scale without sending noise.

Some of these controller values already exist in the Registered Controller / RPN space. This category is distinguished from other controllers by the nature of the maximum value, which differs depending on resolution.

For example, the maximum value of Pitch 7.25 is slightly higher than the maximum value of Pitch 7.9 due to the difference in resolution.

4.1 Identifying Registered Controllers / RPN Which Use Zero-Extension Scaling with Rounding

Zero-Extension Scaling shall be used on all Registered Controller (RPN) Messages where the index (RPN LSB) is 0-31. This provides backward compatibility with existing RPN Messages defined in the Complete MIDI 1.0 Detailed Specification ***[MA01]***

4.2 Core Rules of Zero-Extension Scaling with Rounding

These are the core rules for Zero-Extension Scaling with Rounding:

- Minimum/Lowest value is scaled to Minimum/Lowest
- Maximum/Highest value is upscaled using bitshift only
  For example, a 7-bit value of 127 is translated to a 16-bit value of 65024.
- Center Value (rounded up) scales to Center Value:
  `Center = TRUNC ((Highest + 1) / 2)`
- Scaling down a previously upscaled value yields the original value.

4.3 Zero-Extension Upscaling Method

When upscaling right-open values, the additional bits should be filled with zero; this corresponds to the “left shift” operator in C-like programming languages.

This method should not be used for upscaling 1-bit resolution (on/off). The value expanded for a value of one will scale to the halfway upscaled value and may not provide enough clarity.

4.3.1 Pseudo Code for Zero-Extension Upscaling Algorithm

```
scaleUp(srcVal, srcBits, dstBits) {
    // simple bit shift
    uint scaleBits = (dstBits – srcBits);
    return srcVal << scaleBits;
}
```

4.3.2 Numerical Examples

Table 9 Numeric Example: Upscale 7 to 16 bits

7-bit decimal	7-bit hex	16-bit decimal	16-bit hex
10	0x0A	5120	0x1400
64	0x40	32768	0x8000
87	0x57	44544	0xAE00
127	0x7F	65024	0xFE00

4.4 Zero-Extension Downscaling Method

When downscaling fixed-point values, they should be rounded in the reduced resolution. This involves a rounding step and a "clamping" step, because rounding may result in values being out of bounds.

For example, when downscaling a Pitch 7.25 value to a Pitch 7.14 value, we round in order to produce the UQ7.14 fixed-point value which is closest to the original UQ7.25 value. This minimizes the error introduced by the decrease in resolution.

We can do this by first adding half of the scaled range before shifting down. Then, if the shifted value exceeds the bounds of the destination bit resolution, clamp it to the maximum value.

Note: There are various methods for downscaling. It's up to the manufacturer to choose the best way for their circumstance.

4.4.1 Pseudo Code for Zero-Extension Downscaling Algorithm

```
uint scaleDownRounding(uint srcVal, uint srcBits, uint dstBits) {
  uint scaleBits = (srcBits - dstBits);
  uint halfScaleRange = 1 << (scaleBits - 1);
  uint shifted = (srcVal + halfScaleRange) >> scaleBits;
  uint maxValue = (1 << dstBits) - 1;
  if (shifted > maxValue) {
    shifted = maxValue;
  }
  return shifted;
}
```

4.4.2 Numerical Examples

Table 10 Numeric Example: Downscale 16 to 7 bits

16-bit decimal	16-bit hex	7-bit decimal	7-bit hex
5120	0x1400	10	0x0A
5631	0x15FF	11	0x0B
32768	0x8000	64	0x40
44544	0xAE00	87	0x57
44730	0xAEBA	87	0x57
44800	0xAF00	88	0x58

65024	0xFE00	127	0x7F
65535	0xFFFF	127	0x7F

5 Stepped Values and Enumerations

Controller messages are sometimes used to provide a discrete number of allowed steps, often representing an enumeration. Examples are filter modes or wavetable entries.

An additional transformation step is required for stepped values and enumerations as the number of steps can be any number and do not correspond exactly to a given resolution. Some MIDI 1.0 Devices only use a small subset of the available resolution, e.g. values 0...4 for five filter modes (leaving values 5-127 without function). In UMP, such a mapping results in very small values that do not interact well with physical controllers.

MIDI Devices should use the full resolution for representing stepped values and enumerations. If the Device is expecting additional enumeration values in the future, it is recommended that the Device declare some additional reserved values from the start.

5.1 Sending Stepped Values / Enumerations

To use Enumeration using Stepped Value the full resolution is divided into `stepCount` intervals (`stepValues`), where `stepCount` is the number of steps, or the number of different enumeration values. The `stepCount` must be less than or equal to the maximum value at full resolution.

When sending a given step value `stepValue` (or enumeration value), the following formula can be used to derive the UMP message value at full resolution:

```
messageValue = valueRange * stepValue / stepCount
```

The `valueRange` is the number of discrete maximum values at the given message resolution of the respective UMP message. For example, for a 7-bit resolution, the `valueRange` is 128. The `stepValue` is zero-based, so it is always lower than `stepCount`.

When using integer values, rounding can cause problems. It is a good idea to center the `messageValue` in the respective bin to ensure maximum robustness. The following formula will take care of that, and implement proper rounding:

```
messageValue = TRUNC (((valueRange * stepValue) + stepCount - 1) / stepCount)
               + TRUNC (valueRange / (2 * stepCount))
```

`TRUNC` is a function to discard any decimal places (truncation).

If `stepCount` is 2, use 0 and (`valueRange` - 1) for `messageValue`, similar to how a 1-bit value is scaled up.

Avoid using `stepCount` values of more than half the `valueRange`. Although the formulas presented here will provide proper rounding, it is possible that the receiver will implement a slightly different rounding scheme, so that the step values will be interpreted incorrectly. A `stepCount` of 32 or below provides the most robust transmission, even if downscaled to 7-bit values.

5.2 Receiving Stepped Values / Enumerations

When receiving a stepped value or enumeration, any value within the range of a particular bin will select the respective step or enumerated value.

Use the following division to derive the step value / enumeration value:

```
stepValue = TRUNC (messageValue * stepCount / valueRange)
```

5.3 Examples

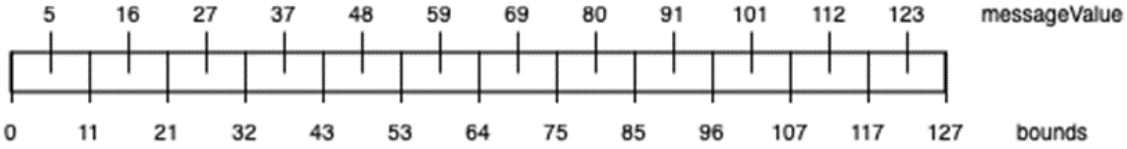

Figure 6 12 steps in 7-bit resolution

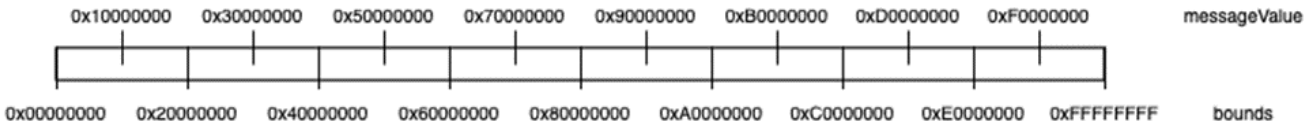

Figure 7 8 steps in 32-bit resolution

http://www.amei.or.jp

MIDI® Association

https://www.midi.org