

 <b>MOTION IMAGERY STANDARDS BOARD</b>	<b>MISB RP 1201</b>
<b>Recommended Practice</b>  <b>Floating Point to Integer Mapping</b>	<b>February 15<sup>th</sup> 2012</b>

**1 Scope**

This recommended practice describes the method for mapping floating point values to integer values and the reverse, mapping integer values back to their original floating point value to within an acceptable precision. There are many ways of optimizing the transmission of floating point values from one system to another; the purpose of this recommended practice is to provide a single method which can be used for all floating point ranges and valid precisions. This recommended practice provides a method for a forward and reverse linear mapping of a specified range of floating point values to a specified integer range of values based on the number of bytes desired to be used for the integer value. Additionally it provides a set of special values which can be used to transmit non-numerical “signals” to a receiving system.

**2 References**

**2.1 Normative References**

[1] IEEE 754-2008, *Standard for Floating-Point Arithmetic [and Floating-Point formats]*

**2.2 Informative References**

None

**3 Modifications and Changes**

- **RP 1201 – Initial Release - February 2012**

**4 Definitions and Acronyms**

- **Ceiling** is defined as rounding any non-integer value up toward  $+\infty$ . The notation used is  $\lceil x \rceil$  and the ceiling function is defined as  $\lceil x \rceil = \min\{n \in \mathbb{Z} | n \geq x\}$ . Example:  $\lceil -1.1 \rceil = -1$  and  $\lceil 1.1 \rceil = 2$ . Note: Microsoft Excel (pre 2010) does not perform this operation correctly.
- **Floor** is defined as rounding any non-integer value down towards  $-\infty$ . The notation used is  $\lfloor x \rfloor$  and the floor function is defined as  $\lfloor x \rfloor = \max\{m \in \mathbb{Z} | m \leq x\}$ . Example:

$\lfloor -1.1 \rfloor = -2$  and  $\lfloor 1.1 \rfloor = 1$ . Note: Microsoft Excel (pre 2010) does not perform this operation correctly.

- The **truncate** function removes the fractional part of a real number (e.g. truncate (100.2) = 100).
- The **round** function has different modes but for this standard round means  $\lfloor x + 0.5 \rfloor$  if  $x \geq 0$  and  $\lfloor x - 0.5 \rfloor$  if  $x < 0$ .

## 5 Introduction - Informative

Many data values which are measured or computed in floating point are inserted into KLV and transmitted from one system to another. Most of the time values do not fully utilize the floating point range or precision and therefore can be transmitted using alternate techniques which will reduce the number of bytes used during the transmission. Additionally, when using floating point values there are special cases for measurements and computations that need to be communicated to receiving systems; examples are having a sensor value that means “beyond measurement range” or during a computation a divide by zero occurs (i.e. +infinity). This standard applies to IEEE 754 floating point values (all precisions; i.e. 16, 32, 64 and 128 bit), including many of the IEEE special values of infinity, and NaN’s.

In section 7 of this document the standard algorithm is succinctly stated for developers to implement; section 8 of this document is informative and shows the derivation of the algorithm and format.

## 6 Algorithm Requirements - Informative

To aid in understanding some of the steps in the standard mapping described below, the requirements for the algorithm are as follows:

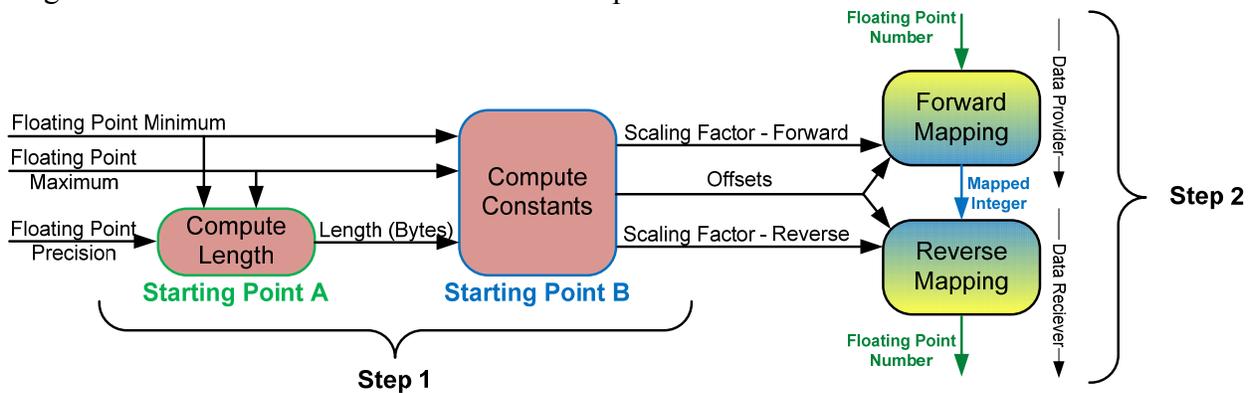
- 1) The binary representation of a floating point value shall be in the format of an IEEE 754-2008 floating point value.
- 2) The algorithm shall perform the mapping based on a [user] specified floating point range from a min value to a max value – inclusive (including the end points).
- 3) The algorithm shall be a linear mapping from a floating point value to a non-negative integer value of [user] defined length in bytes.
- 4) The binary representation of the non-negative integer value shall be in the format of a standard unsigned integer, with the length, in bytes, defined by the [user’s] implementing standard.
- 5) The algorithm shall provide an inverse mapping from a non-negative integer value to the original floating point value within a [user] specified precision. (Note: the mapping algorithm can supply better precision than what the user specifies. The defined length and precision are related – given one the other can be computed but both cannot be specified by the user).
- 6) If the [users] floating point range includes 0.0, then the algorithm shall map 0.0 exactly to a specific integer value, called a “zero offset”. Additionally, all floating point values that are negative shall map to an integer that is less than the zero offset and all values that are positive shall map to values greater than or equal to the zero offset.

- 7) The algorithm shall be designed to be as few operations as possible (during the actual mapping) for computational efficiency.
- 8) The resulting format shall provide a set of special values which map to the IEEE 754-2008 special values, including:  $\pm$  infinity,  $\pm$ QNaN and  $\pm$ SNaN (note: negative zero is not included in this requirement).
- 9) The resulting format shall provide a bit space for custom signals to be sent by the user as defined by the [users] implementing standard (i.e. local data set standard).

## 7 Standard Mapping Algorithm and Integer Format – Normative

This section describes the standard algorithm for mapping the floating point values to integer values and the reverse mapping of integers back to floating point values. Additionally this section defines the notation to use in all MISB standards when “invoking” this standard. Section 8 is an informative section which describes the development of this algorithm in detail.

The algorithm is based on two steps; the first step computes one-time use constants and the second step is either the forward or reverse mappings. There are two “starting points” for the first step. If the precision is known then the desired length can be computed; otherwise if the desired length is known it is used directly. Both “starting points” require the floating point minimum and maximums to be defined. The second step uses the constants computed from the first step to perform as many floating point-to-integer and/or integer-to-floating point mappings as desired – as long as the floating point minimum, maximum and length do not change. The following diagram illustrates the interaction of the two steps:



In Steps 1 and 2 described in the sections below, it is required to perform the computations using the standard order of operations (i.e. operations in parenthesis first, exponents/roots second, multiplication/division third, addition/subtraction fourth).

### 7.1 Step 1: Scaling Factors & Offsets

Step 1 has two starting points based on whether the floating point precision is known or the length (in bytes) of the resulting mapped integer is known.

### 7.1.1 Starting Point A: Specifying Precision

Input
<ul style="list-style-type: none"> <li>• a = Floating Point Minimum, range is from smallest float value to largest float value.</li> <li>• b = Floating Point Maximum, range is from smallest float value to largest float value.</li> <li>• g = Floating Point Precision, range is from zero to largest float value.</li> <li>• Note: <math>a &lt; b</math> and <math>g &lt; b - a</math></li> </ul>
Algorithm
Compute Length L (Bytes)
<ol style="list-style-type: none"> <li>1. <math>Lbits = \text{ceiling}(\log_2((b-a)/g)+1)</math> (note: +1 is for special values, see below)</li> <li>2. <math>L = \text{ceiling}(Lbits/8)</math></li> <li>3. Follow steps in Starting Point B</li> </ol>

### 7.1.2 Starting Point B: Specifying Length

Input
<ul style="list-style-type: none"> <li>• a = Floating Point Minimum, range is from smallest float value to largest float value.</li> <li>• b = Floating Point Maximum, range is from smallest float value to largest float value.</li> <li>• L = Length of resulting mapped integer (note: this includes the extra bit for the special values, see below)</li> <li>• Note: <math>a &lt; b</math></li> </ul>
Algorithm
Compute constants for forward and reverse mapping.
<ol style="list-style-type: none"> <li>1. <math>bPow = \text{ceiling}(\log_2(b-a))</math></li> <li>2. <math>dPow = 8 * L - 1</math></li> <li>3. <math>sF = 2^{(dPow - bPow)}</math></li> <li>4. <math>sR = 2^{(bPow - dPow)}</math></li> <li>5. <math>Zoffset = 0.0</math></li> <li>6. if (<math>a &lt; 0</math> and <math>b &gt; 0</math>) then <math>Zoffset = sF * a - \text{floor}(sF * a)</math></li> </ol>

## 7.2 Step 2: Forward and Reverse Mapping

Step 2 uses the values computed in step 1 to do the forward mapping and reverse mapping. Note that these operations are optimal in that the actual mapping only uses one multiply and two adds. Since the multiplication is by a power of two, it can be implemented as a binary shift if necessary (i.e. in hardware constrained environments).

### 7.2.1 Forward Mapping

Input
<ul style="list-style-type: none"> <li>• sF = Forward scaling factor (computed in step 1)</li> <li>• a = Floating Point Minimum</li> <li>• Zoffset = Zero point offset (computed in step 1)</li> <li>• x = floating point value to map to integer y</li> </ul>
Algorithm
<p>Compute mapped integer value y.</p>
<ol style="list-style-type: none"> <li>1. If x is a special value then:               <ol style="list-style-type: none"> <li>a. y = Table lookup to map to specified bit pattern (see 7.2.3 below).</li> </ol> </li> <li>2. Else if x is a normal floating point number then               <ol style="list-style-type: none"> <li>a. <math>y = \text{truncate}(sF * (x - a) + Zoffset)</math> <ol style="list-style-type: none"> <li>i. <i>Note: Since sF is a power of 2, a floating point shift can be used instead of a multiplication.</i></li> </ol> </li> <li>b. <i>Note: y will need to be converted into L number of bytes values (i.e. the lower L number of bytes is the “value” that needs to be sent.</i></li> </ol> </li> </ol>
Developer Notes
<ol style="list-style-type: none"> <li>1) <i>If doing a shift manually (i.e. the CPU does not support a floating point shift), the most significant 1 bit of a floating point number is never explicitly represented in its mantissa, so it must be OR'ed in before shifting when converting to an integer. As a simple example, the mantissa for 6 is 1000..., it first becomes 11000..., then ...00000110.</i></li> <li>2) <i>An optimization of the <u>forward mapping</u> can be applied if it is known that no special values are going to be used; the “special values check” and table look up can be omitted. The extra bit (special value bit) must be preserved in the data format but the conditional check (step 1 above) can be removed.</i></li> </ol>

### 7.2.2 Reverse Mapping

Input
<ul style="list-style-type: none"> <li>• sR = Reverse scaling factor (computed in step 1)</li> <li>• a = Floating Point Minimum</li> <li>• Zoffset = Zero point offset (computed in step 1)</li> <li>• y = integer value to map to floating point value x.</li> </ul>
Algorithm
Compute mapped floating point value x.
<p>Define: Bit(q,y) = the q<sup>th</sup> bit of y. MSB = Most significant bit</p> <ol style="list-style-type: none"> <li>1. special_value = Bit(MSB,y) &amp; Bit(MSB-1,y)</li> <li>2. If special_value equals 1 then             <ol style="list-style-type: none"> <li>a. x=table lookup to map to specified floating point value (see 7.2.3 Table 2 below).</li> </ol> </li> <li>3. Else y is a normal value so             <ol style="list-style-type: none"> <li>a. <math>x=sR*(y-Zoffset)+a</math> <ol style="list-style-type: none"> <li>i. <i>Note: Since sR is a power of 2, a floating point shift can be used instead of a multiplication.</i></li> </ol> </li> </ol> </li> </ol>
Developer Notes
<ol style="list-style-type: none"> <li>1) <i>If doing a shift manually (i.e. the CPU does not support a floating point shift), the most significant 1 bit of a floating point number is never explicitly represented in its mantissa, so it must be OR'ed in before shifting when converting to an integer. As a simple example, the mantissa for 6 is 1000..., it first becomes 11000..., then ...00000110.</i></li> <li>2) <i>There is an assumption that the length checking (which is typical for KLV parsing) has been performed before applying these methods, see section 7.4 for further information.</i></li> </ol>

### 7.2.3 Special Value Mappings

In addition to normal numeric values this recommended practice defines a list of special values that can be used to indication special circumstances such as a gimbal lock or division by zero, etc. There are a couple of standard values that are supported (based on IEEE-754) and a set of user definable values that can be defined.

The first two bits are used to indicate either a normal or special value as shown in the following table. The MSB is zero for all normal mapped values except for the maximum floating point value, b, (but this only happens if (b-a), as specified by the user, is a power of 2).

Table 1 - Bit Patterns

Bits			Description
$b_n$ (MSB)	$b_{n-1}$	$b_{n-2} - b_0$	
0	x	x	Normal mapped value. (x=1 or 0)
1	0	0	Normal mapped value but this is the max value of the normal mapping values; this is the only normal value that has the MSB=1.
1	0	x	(at least one x bit = 1) – Reserved section of bit space.
1	1	x	Special value indicator (x=1 or 0) – see Table 2

If both the first and second MSB are set to one then the third, fourth and fifth MSBs are used to indicate which special value it is, as shown in Table 2. Since the mapped integer has fewer bytes than the original floating point number not all the source NaN Identifiers can be passed; so at this time there are no defined NaN Identifiers values except for one. NaN Identifiers will be defined as needed by the MISB.

Note that the IEEE 754 value for negative zero is not sent as a special value, instead it is equated to positive zero; all negative zero values shall be mapped to a positive zero and sent as a normal (non-special) value.

In addition to the IEEE bit values a set of user defined bit patterns is available for definition; these can be used on a standard by standard basis. For example, in STD 0601 the sensor depression angle value could include a bit pattern which indicates gimbal lock, this bit pattern would only be valid for that value in the 0601 standard.

Table 2 - Special Value Bit Patterns

Name	Most Significant Bits					Other bits	Description - IEEE-754 values (or user defined)
	$b_n$ (MSB)	$b_{n-1}$ (Special)	$b_{n-2}$ (Sign)	$b_{n-3}$ (NaN)	$b_{n-4}$	$b_{n-5} - b_0$	
+Infinity	1	1	0	0	1	Reserved	Positive Infinity ( $+\infty$ )
-Infinity	1	1	1	0	1	Reserved	Negative Infinity ( $-\infty$ )
+QNaN	1	1	0	1	0	NaN Id*	Positive Quiet NaN (Not a Number)
-QNaN	1	1	1	1	0	NaN Id*	Negative Quiet NaN
+SNaN	1	1	0	1	1	NaN Id*	Positive Signal NaN – Remaining bits are used as the signal value.
-SNaN	1	1	1	1	1	NaN Id*	Negative Signal NaN - Remaining bits are used as the signal value.
Reserved	1	1	1	0	0	Reserved	Reserved
User Defined	1	1	0	0	0	User Defined	Remaining bits can be used for user defined signals
Reserved	1	0	1	X	X	Reserved	Reserved

\*There is only one Nan Id defined at this time and that is to fill the bits  $b_{n-5} - b_0$  with zeros. The MISB will define NaN Id's as necessary when needed; please contact the MISB to define these values. The values must be universally accepted and not special values for a specific processor or application.

### 7.3 MISB and MISB Document Standard Notation

When using or “invoking” this recommended practice in MISB documents it is important to be clear as to which starting point is being used and which values are being used as “inputs”. When using this recommended practice in MISB documents the following notation shall be used:

- Starting Point A - IMAPA (<min float>, <max float>, <float accuracy>)
  - Example: IMAPA(-200.0, 3000.0, 0.5)
  - This means map a value in the range from -200.0 to 3000.0 using, at the minimum, increments of 0.5.
- Starting Point B – IMAPB (<min float>, <max float>, <length bytes>)
  - Example: IMAPB(-200.0, 3.000, 3)
  - This means map a value in the range from -200.0 to 3000.0 using 3 bytes (see 7.4 for comments on length).

When adding user defined bit patterns they shall be listed immediately after the IMAPA or IMAPB notation.

### 7.4 Length Processing

The lengths computed or provided when defining the mapping (IMAPA, IMAPB) are considered the recommended number of bytes to use. Depending on the form that is used to transmit KLV data (Universal Data Sets, Local Data Sets, Individual Items) it is possible to have a different [KLV] length (L) defined than the recommended [IMAP] length. When a different length is used it is important to recompute the constants needed to do the forward and reverse mapping based on the KLV supplied length.

## 8 Algorithm Development – Informative

Because there are multiple techniques for providing this capability the following sections show the development of the algorithm for this recommended practice and the logic behind why it was chosen.

- 1) Goal
- 2) Mapping and Inverse Mapping
- 3) Error Analysis
- 4) Simplification
- 5) Zero Matching
- 6) Computing d
- 7) Power of 2 adjustments for b
- 8) Computing L from Precision
- 9) Summary

### 8.1 Goal

The goal is to map any floating point range ( $F_R$ ) to an integer range ( $I_R$ ) with a chosen maximum value of precision (g), smallest integer range ( $I_R$ ) and least amount of computation. In addition when a  $F_R$  range includes zero the algorithm must map zero to a specific integer value (zero

offset) and all floating point values that are negative must map to values less than the zero offset. Refer to section 6 for the requirements for this algorithm.

Examples:

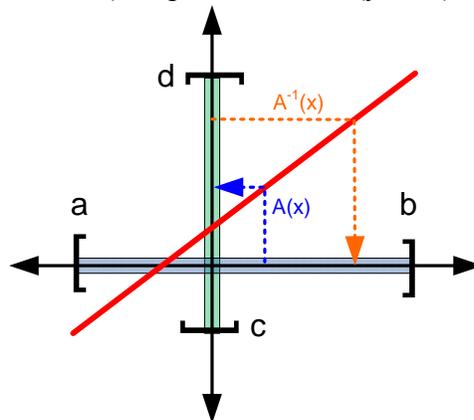
- 1) Map floating point range 0.1 to 3.1 to 2 bytes
- 2) Map -1.0 to +1.0 to 2 bytes
- 3) Map -0.5 to -0.3 to 1 byte
- 4) Map -3.14159265 to 3.14159265 to 4 bytes
- 5) Map floating point range 0.1 to 3.1 with precision of 0.01 or better
- 6) Map -1.0 to +1.0 to 2 bytes with precision of 0.001 or better
- 7) Map -0.5 to -0.3 to 1 byte with precision of 0.01 or better
- 8) Map -3.14159265 to 3.14159265 with precision of 0.00314159265 or better

### 8.2 Mapping and Inverse Mapping Definition

Given a  $F_R = \{x \in \mathbb{R} | x \in [a, b]\}$  and  $I_R = \{y \in \mathbb{Z} | y \in [c, d]\}$

Let  $A(x)$  be a linear mapping from  $F_R$  to  $I_R$  and  $A^{-1}(y)$  be a mapping from  $I_R$  to  $F_R$ .

Graphically this is shown below,  $A(x)$  (blue dotted line) maps from  $F_R$  (x-axis) to  $I_R$  (y-axis) via the red line;  $A^{-1}(y)$  (orange dotted line) maps from the  $I_R$  (y-axis) to  $F_R$  (x-axis) via the red line.



Using the two point method for defining the mapping:

$$(Eq 1) \quad y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1) = y - c = \frac{d - c}{b - a} (x - a)$$

Since  $y$  is an integer the right hand side of the equation must be truncated or rounded before  $y$  is computed.

$$(Eq 2) \quad y = I\left(\frac{d - c}{b - a} (x - a) + c\right)$$

Where  $I(\ )$  is either a truncation or rounding function (discussed in Section 8.3).

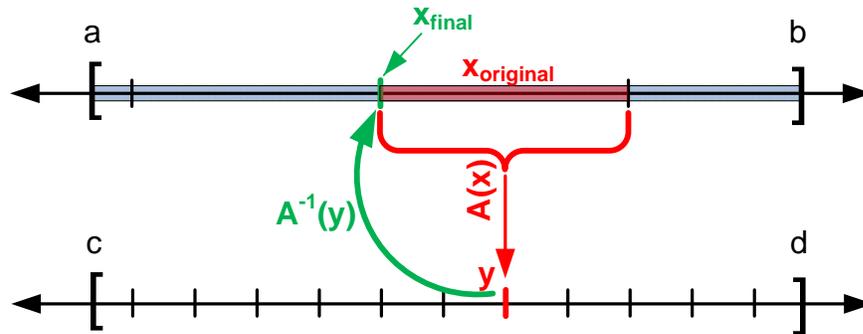
This final equation defines,  $A(x)$ , the forward mapping of any real value (floating point) range to an integer range. The inverse of this equation  $A^{-1}(y)$  is defined as:

$$(Eq 3) \quad x = (y - c) \frac{b - a}{d - c} + a$$

The above equations are for all  $a, b, c$  and  $d$  values, however, additional simplification and improvements can be made which are described in the following sections.

### 8.3 Error Analysis

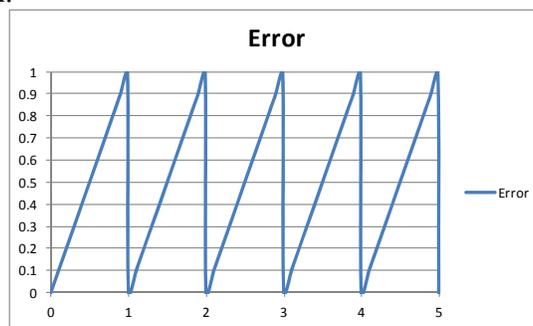
The mapping function is not a one-to-one function which means that the mapping and subsequent inverse mapping introduce errors. For each value  $y$ , in  $I_{\mathbb{R}}$ , a unique range of  $x$  values (in  $F_{\mathbb{R}}$ ) is mapped to  $y$ . When the integer value  $y$  is inverse mapped back to an  $x$  value it will only match one of the values of  $x$ . The tolerance of this error is dependent on the application so it is expected that the implementer of this recommended practice will ensure that large enough  $c$  and  $d$  values are used to create acceptable minimum error – more on this topic in other sections below. The following illustration shows the floating point range along with a subset of values (red) that get mapped to a single integer value  $y$ ; also this shows that the single integer value maps back to only one value in the floating point range,  $x_{\text{final}}$  (green).



Example 1:

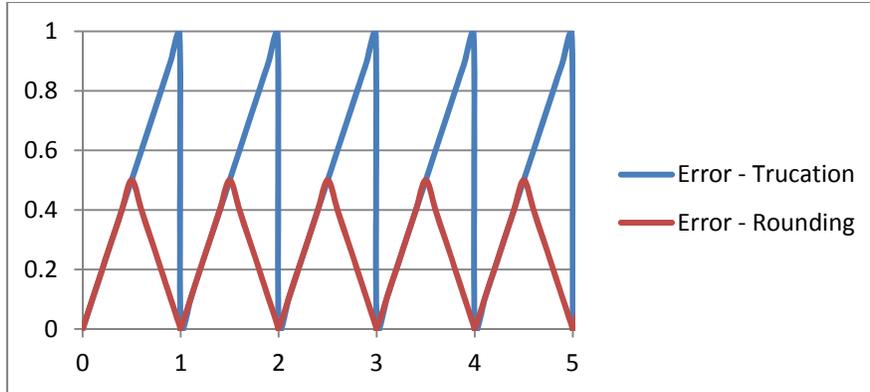
Let  $a=0.0$ ,  $b=5.0$  and  $c=0$  and  $d=5$  then  $y = I\left(\frac{5-0}{5.0-0.0} (x - 0.0) + 0\right) = I\left(\frac{5}{5} x\right) = I(x)$   
 For this example use  $I(x) = \text{truncation}(x)$ , now all values of the  $x_{\text{original}}$  range get mapped to a single  $y$ . The inverse mapping, maps  $y$  to a single  $x$  value,  $x_{\text{final}}$ .  
 If  $x_{\text{original}} = \{\text{all } x \text{ in } [1.0, 2.0)\}$  then the mapped value  $y$  is 1, and the inverse mapped value  $x_{\text{final}} = 1.0$  for all values in the  $x_{\text{original}}$  range.

To compute the amount of error:  $E(x) = |x - A^{-1}(A(x))|$ . Graphing this error for Example 1 produces a saw-tooth graph.



The maximum error in Example 1 is just less than 1.0 and if this level of error is acceptable to the implementer then this mapping function could be used. If less error is desired then it is a simple matter to use a larger integer range; for example if the range is doubled then the error is cut in half.

There are multiple ways of converting a floating point number to an integer, including rounding, truncating, round away from zero, round to zero, etc. When examining the errors the best conversion technique is to do a normal rounding method as shown in the following graph.



However, to support the zero matching requirements a truncation must be used instead (see Section 8.5) so (Eq 2) is changed to include truncation. Since  $x$  ranges from  $[a,b]$ , all values of  $(x-a)$  will be greater than or equal to zero so truncation and floor provide the same function. Instead of truncation notation, the floor notation  $\lfloor \quad \rfloor$ , will be used throughout the remainder of this section.

$$(Eq 4) \quad y = \left\lfloor \frac{d-c}{b-a} (x - a) + c \right\rfloor$$

### 8.4 Simplifications

There are two simplifications to the above equations which will reduce overall (floating point) errors and enable some optimizations (in future steps): shifting the integer range and shifting the floating point range.

The integer range ( $I_R$ ) values,  $c$  and  $d$ , are arbitrary for this algorithm so for all implementations of this recommended practice the mapping equations can be simplified by having the minimum integer value always equal to zero. When  $c$  is zero then  $d$  is the maximum value desired by the implementer for the precision that the implementer desires. An additional benefit of having a minimum value of zero is that the resulting value is not negative which means the value will be an unsigned integer and no sign bit is needed (nor 2's complement formatting). Making  $c=0$  results in the following change to (Eq 4):

$$(Eq 5) \quad y = \left\lfloor \frac{d}{b-a} (x - a) \right\rfloor$$

In a similar fashion the floating point range,  $F_R$ , can also be shifted by using substitution, letting  $x'=x-a$ . The range  $F_R$  is shifted by a constant ( $-a$ ) producing a new range called  $F'_R$ .  $F'_R = \{x' \in \mathbb{R} | x' \in [0, b']\}$  where  $a'=a-a=0$  and  $b' = b-a$ . With this change the minimum value of  $F'_R$  is zero so (Eq 5) and (Eq 3) are simplified to:

$$(Eq 6) \quad y = \left\lfloor \frac{d}{b'} x' \right\rfloor = \left\lfloor \frac{d}{b'} (x - a) \right\rfloor$$

$$(Eq 7) \quad x = \frac{b'}{d} y + a$$

### 8.5 Zero Matching

When the floating point range  $F'_R$  includes zero it is desirable to have that zero value directly map to an integer in  $I_R$ ; this value is called the Zero Point and is noted as  $I_{zero}$ ; furthermore all

negative values in  $F_R$  should map to integers less than  $I_{zero}$  and all positive values should map to integers that are equal to or greater than  $I_{zero}$ .

Example 2: If  $F_R = -0.9$  to  $1.1$  and  $d=11$  then the following values would be mapped:

Note:  $g=2.0/11 = 0.1818$

X	x'	(d/b')x'	ceiling	floor	truncate	round
-0.9	0	0	0	0	0	0
-0.71818	0.181818	1	1	1	1	1
-0.53636	0.363636	2	2	2	2	2
-0.35455	0.545455	3	3	3	3	3
-0.17273	0.727273	4	4	4	4	4
-0.08182	0.818182	4.5	5	4	4	5
0	0.9	4.95	5	4	4	5
0.009091	0.909091	5	5	5	5	5
0.1	1	5.5	6	5	5	6
0.190909	1.090909	6	6	6	6	6
0.372727	1.272727	7	7	7	7	7
0.554545	1.454545	8	8	8	8	8
0.736364	1.636364	9	9	9	9	9
0.918182	1.818182	10	10	10	10	10
1.1	2	11	11	11	11	11

In the table above the red row shows the zero value is computed as 0.9, then, based on the integer conversion technique it gets mapped to the same value as a preceding negative value (in yellow); the Zero Point is undefined since after the linear computation the resulting value was not an integer. This behavior is undesirable; instead, the zero in  $F_R$  should be mapped exactly to an integer (before integer conversion) – this value would be the Zero Point,  $I_{zero}$ . Additionally all negative values in  $F_R$  should be mapped to integers less than  $I_{zero}$ .

To solve this issue the whole mapping equation is shifted upward so that zero in  $F_R$  becomes an integer value ( $I_{zero}$ ); this is just a simple offset that is added before the integer conversion.

Furthermore in order to ensure that all negative values are below  $I_{zero}$ , truncation (or floor) must be performed for the integer conversion; ceiling and rounding will force negative values to be identical to  $I_{zero}$ . Note that this is different than changing the bounds (a,b or c,d) of the equations; the slope of the linear mapping is not affected by this change.

The adjustment to the mapping equations (Eq 5) and (Eq 6) are:

$$(Eq\ 8) \quad Z_{offset} = \frac{d}{b'}a - \left\lfloor \frac{d}{b'}a \right\rfloor, \text{ Note: } 0.0 \leq Z_{offset} < 1.0$$

$$(Eq\ 9) \quad y = \left\lfloor \frac{d}{b'}(x - a) + Z_{offset} \right\rfloor$$

$$(Eq\ 10) \quad x = \frac{b'}{d} (y - Z_{offset}) + a$$

Re-computing Example 2 with these changes results in:

$$Z_{\text{offset}} = \frac{11}{2.0}(-0.9) - \left\lfloor \frac{11}{2.0}(-0.9) \right\rfloor = -4.95 - (-5.0) = 0.05$$

X	x'	(d/b')x' + Zoffset	Ceiling	floor	Truncate	round
-0.9	0	0.05	1	0	0	0
-0.71818	0.181818	1.05	2	1	1	1
-0.53636	0.363636	2.05	3	2	2	2
-0.35455	0.545455	3.05	4	3	3	3
-0.17273	0.727273	4.05	5	4	4	4
-0.08182	0.818182	4.55	5	4	4	5
0	0.9	5	5	5	5	5
0.009091	0.909091	5.05	6	5	5	5
0.1	1	5.55	6	5	5	6
0.190909	1.090909	6.05	7	6	6	6
0.372727	1.272727	7.05	8	7	7	7
0.554545	1.454545	8.05	9	8	8	8
0.736364	1.636364	9.05	10	9	9	9
0.918182	1.818182	10.05	11	10	10	10
1.1	2	11.05	12	11	11	11

This table shows the Zero Point ( $I_{\text{zero}}$ ) is the value 5; all negative x values map to 4 or less and all positive values (and zero) map to 5 or greater.

### 8.6 Computing d

In all the previous equations, the value d, is computed from the number of bytes that the user specified with value L. The value d uses all of the bits in the bytes specified including the most significant bit (MSB); however, the MSB is only used for one value when mapping the floating point values. When mapping the largest floating point value, b, the MSB may be used if b was specified [by the user] as a power of 2. In this case the MSB is set to 1 and all other bits are set to zero. All other uses of the MSB are used to signal special values which are described in Section 7.2.3.

$$(Eq 11) \quad N_{\text{bits}} = 8 * L - 1$$

$$(Eq 12) \quad d = 2^{N_{\text{bits}}}$$

(Eq 12) shows that d is a power of two which, will be used for an optimization step described in Section 8.7.

### 8.7 Power of 2 Adjustment for b

To improve algorithm efficiency the floating point range,  $F_R$ , is adjusted to be a power of 2, however, when this adjustment is made the precision is reduced and the resulting error increases slightly.

Examining the equation for the mapping shows that the mapping is primarily a simple division, multiplication and addition,  $y = \left\lfloor \frac{d}{b'}(x - a) + Z_{offset} \right\rfloor$ . Floating point division and multiplication can easily introduce small numeric floating point rounding errors. To reduce this error and to provide a more efficient algorithm  $b'$  is adjusted to be based on a power of two.

$$(Eq 13) \quad \bar{b} = 2^{\lceil \log_2 b' \rceil}$$

With the adjustments to  $b'$  (Eq 9) and (Eq 10) are updated as follows:

$$(Eq 14) \quad y = \left\lfloor \frac{d}{\bar{b}}(x - a) + Z_{offset} \right\rfloor$$

$$(Eq 15) \quad x = \frac{\bar{b}}{d}(y - Z_{offset}) + a$$

Since both  $d$  and  $\bar{b}$  are both powers of two, a small optimization can be made by pre-computing forward and reverse scaling factors.

$$(Eq 16) \quad \frac{d}{\bar{b}} = S_F = 2^{(N_{bits} - \lceil \log_2 b' \rceil)}$$

$$(Eq 17) \quad \frac{\bar{b}}{d} = S_R = 2^{(\lceil \log_2 b' \rceil - N_{bits})}$$

Now (Eq 14) and (Eq 15) are adjusted and the final mapping equations are complete:

$$(Eq 18) \quad y = \left\lfloor S_F(x - a) + Z_{offset} \right\rfloor$$

$$(Eq 19) \quad x = S_R(y - Z_{offset}) + a$$

### 8.8 Computing L from known precision

When using these equations there are two starting points that can be defined based either on (a) the minimum, maximum and known floating point precision or (b) the minimum, maximum and desired length. The preceding sections describe the steps for computing scaling factors based on (b), knowing the minimum, maximum and number of bytes (L). This section shows the computation of L based on the known minimum, maximum and known floating point precision (i.e. starting point (a)).

Given  $a$ ,  $b$  and  $g$ ; where  $a$  is the minimum value of the range,  $b$  is the maximum value of the range and  $g$  is the floating point precision to use, then the number of bytes,  $L$ , needed to represent this number for the mapping is:

$$(Eq 20) \quad L_{bits} = \left\lceil \log_2 \left( \frac{b-a}{g} \right) \right\rceil + 1 \quad \text{Note: +1 is for special values bit.}$$

$$(Eq 21) \quad L = \left\lceil \frac{L_{bits}}{8} \right\rceil$$

This value of  $L$  is used in (Eq 11).

### 8.9 Summary

In summary the following shows a consolidation of all of the equations needed to map values to and from floating point to integer.

**Starting Point A: User specifies a, b and g (the desired precision to use).**

One time computation

$$L_{\text{bits}} = \left\lceil \log_2 \left( \frac{b-a}{g} \right) \right\rceil + 1$$

$$L = \left\lceil \frac{L_{\text{bits}}}{8} \right\rceil$$

Now a, b and L are defined for starting point B

**Starting Point B: User specifies, a, b and L (number of bytes to use).**

One time computation

$$b_{\text{pow}} = \lceil \log_2(b-a) \rceil$$

$$d_{\text{pow}} = 8 * L - 1 \quad \text{(Number of bits for d)}$$

$$S_F = 2^{(d_{\text{pow}} - b_{\text{pow}})} \quad \text{(Scaling for forward mapping, note same as bitwise shift)}$$

$$S_R = 2^{(b_{\text{pow}} - d_{\text{pow}})} \quad \text{(Scaling for reverse mapping, note same as bitwise shift)}$$

$$Z_{\text{offset}} = 0 \quad \text{(Default offset to zero)}$$

If (a<0 and b>0) then  $Z_{\text{offset}} = S_F a - \lfloor S_F a \rfloor$  (only compute if condition met)

Once the above values,  $S_F$ ,  $S_R$  and  $Z_{\text{offset}}$  are computed they are used for forward mapping and reverse mapping every x and y value, respectively. See Section 7.2.3 for information on handling special values (i.e. infinity, NaN, etc.) of x and y.

Per x value:

$$y = \lfloor S_F(x - a) + Z_{\text{offset}} \rfloor$$

Per y value:

$$x = S_R(y - Z_{\text{offset}}) + a$$

## Appendix A - Usage Examples

The following table shows some examples of ranges, precisions or number of bytes that could be used along with the IMAP notation that shall be used in the standard document.

Name	Min	Max	Precision	Number of Bytes	IMAP Statement
Altitude	-900.0	19000.0	Don't care	2	IMAPB(-900,19000,2)
Covariance	-1.0	1.0	Don't care	2	IMAPB(-1, 1, 2)
Radians	-3.14159265	3.14159265	Don't care	3	IMAPB(-3.14159265, 3.14159265, 3)
Altitude	-900.0	19000.0	0.5	Don't care	IMAPA(-900,19000,0.5)
Covariance	-1.0	1.0	.0001	Don't care	IMAPA(-1, 1, .0001)
Radians	-3.14159265	3.14159265	0.00314159265	Don't care	IMAPA(-3.14159265, 3.14159265, 0.00314159265)
Small Range	0.1	0.9	Don't care	2	IMAPB(0.1, 0.9, 2)

The following illustrates a complete example of the computations for mapping an altitude range using, IMAPA(-900,19000,0.5).

Altitude example (Starting point A):

- a = Floating Point Minimum = -900.0
- b = Floating Point Maximum = 19,000.0
- g = Floating Point Precision = 0.5

Starting Point A - Algorithm: Compute Length L (Bytes)

1.  $Lbits = \text{ceiling}(\log_2((b-a)/g)+1) = \text{ceiling}(\log_2((19,000-(-900))/0.5)+1) = 17$
2.  $L = \text{ceiling}(Lbits/8) = \text{ceiling}(17/8) = 3$
3. Follow steps in Starting Point B

Starting Point B – Algorithm: Compute sF, sR and Zoffset

1.  $bPow = \text{ceiling}(\log_2(b-a)) = \text{ceiling}(\log_2(19,000-(-900))) = 15$
2.  $dPow = 8*L-1 = 8*3-1 = 23$
3.  $sF = 2^{(dPow - bPow)} = 2^{(23-15)} = 2^8$
4.  $sR = 2^{(bPow - dPow)} = 2^{(15-23)} = 2^{(-8)}$
5.  $Zoffset = 0.0$
6. if (a<0 and b>0) then  $Zoffset = sF*a - \text{floor}(sF*a) = (2^8)*(-900) - \text{floor}((2^8)*(-900)) = 0.0$

Forward Mapping: Let x = 10.0 (underlined section is the algorithm that is exercised)

1. If x is a special value then:
  - a. y = Table lookup to map to specified bit pattern.
2. Else if x is a normal floating point number then
  - a.  $y = \text{truncate}(sF*(x-a) + Zoffset) = \text{truncate}(2^8*(10.0-(-900))+0.0) = 232,960$

b.  $y = 0x03\ 8E00$

Reverse Mapping: Let  $y = 232,960 = 0x03\ 8E00$

Define:  $\text{Bit}(q,y)$  = the  $q^{\text{th}}$  bit of  $y$ . MSB = Most significant bit

1.  $\text{special\_value} = \text{Bit}(\text{MSB},y) \ \& \ \text{Bit}(\text{MSB}-1,y) = 0 \ \& \ 0$
2. If  $\text{special\_value}$  equals 1 then
  - a.  $x$ =table lookup to map to specified floating point value.
3. Else if  $y$  is a normal value then
  - a.  $x = sR * (y - \text{Zoffset}) + a = (2^{(-8)}) * (232,960 - 0.0) + (-900) = 10.0$

The following table shows some forward mapping and reverse mapping values for this example.

x	y	y (hex)	x mapped back
-900.0	0	0x00 0000	-900.0
0.0	230,400	0x03 8400	0.0
10.0	232,960	0x03 8E00	10.0
-infinity	See hex	0xE8 0000	-infinity

---

The following illustrates a complete example of the computations for mapping the small range using,  $\text{IMAPA}(0.1, 0.9, 2)$ .

Small range Example (Starting point B):

- $a$  = Floating Point Minimum = 0.1
- $b$  = Floating Point Maximum = 0.9
- $L$  = 2 bytes

Starting Point B – Algorithm: Compute  $sF$ ,  $sR$  and  $Z\text{offset}$

1.  $b\text{Pow} = \text{ceiling}(\log_2(b-a)) = \text{ceiling}(\log_2(0.9-(0.1))) = 0$
2.  $d\text{Pow} = 8 * L - 1 = 8 * 2 - 1 = 15$
3.  $sF = 2^{(d\text{Pow} - b\text{Pow})} = 2^{(15-0)} = 2^{15}$
4.  $sR = 2^{(b\text{Pow} - d\text{Pow})} = 2^{(0-15)} = 2^{(-15)}$
5.  $Z\text{offset} = 0.0$
6. if ( $a < 0$  and  $b > 0$ ) then  $Z\text{offset} = sF * a - \text{floor}(sF * a)$

Forward Mapping: Let  $x = 0.5$  (underlined section is the algorithm that is exercised)

3. If  $x$  is a special value then:
  - a.  $y$  = Table lookup to map to specified bit pattern.
4. Else if  $x$  is a normal floating point number then
  - a.  $y = \text{truncate}(sF * (x - a) + Z\text{offset}) = \text{truncate}(2^{15} * (0.5 - (0.1)) + 0.0) = 13,107$
  - b.  $y = 0x3333$

Reverse Mapping: Let  $y = 13,107 = 0x3333$

Define:  $\text{Bit}(q,y)$  = the  $q^{\text{th}}$  bit of  $y$ . MSB = Most significant bit

4.  $\text{special\_value} = \text{Bit}(\text{MSB},y) \ \& \ \text{Bit}(\text{MSB}-1,y) = 0 \ \& \ 0$
5. If  $\text{special\_value}$  equals 1 then
  - a.  $x = \text{table lookup to map to specified floating point value.}$
6. Else if  $y$  is a normal value then
  - a.  $x = sR*(y-Zoffset)+a = (2^{(-15)}) * (13,107-0.0) + (0.1) = 0.499993896484375$

The following table shows some forward mapping and reverse mapping values for this example.

x	y	y (hex)	x mapped back	Error
0.1	0	0x0000	0.1	0.0
0.5	13,107	0x3333	0.499993896484375	6.10e-6
0.9	26,214	0x6666	0.89998779296875	1.22e-5
-infinity	See hex	0xE800	-infinity	n/a

## **Appendix B - Java Code Example**

**Available upon request.**