# Letter of Agreement
for
# Recommended Practice
# MMA/AMEI

Date of issue: 2/28/99
Originated by: MMA

Reference TSBB Item #: 133 TSBB Volume #: 22 (revised)

Title: SMF Language and Display Extensions
RP#:

Related item(s): SMF Lyric Meta Events

**Abstract:**

Proposal for Reserved Characters in Lyric Meta Events to establish Command Codes used for Text Display, allow for Multi-byte Character Code Sets, and include Song Information in Standard MIDI Files. This proposal extends and renames SMF Lyric Meta Events. Lyric Meta Events are to be named Lyric/Display Meta Events.

**Background:**

Lyrics can be described in Standard MIDI Files by using Lyric Meta Events. However, the Standard MIDI Files 1.0 document does not define the usage for Character Codes other than ASCII code. Also, display formatting for Karaoke, as well as styles such as "Ruby", are not well considered. These limitations prevent the wide spread adoption of lyrics in Standard MIDI Files in certain countries.

The purpose of this proposal is to ensure compatibility among Standard MIDI files by defining the rules for (and usage of) alternate character codes for multi-language support (including "Ruby" parts). This proposal also includes rules for adding display formatting and song information to a MIDI file, which is typically needed for Karaoke applications, but might also be useful in any MIDI file player.

**Details:**

Five ASCII characters as defined as reserved characters to allow support for multiple languages in Standard MIDI Files, including pictographic languages such as Japanese and Chinese which require Multi-Byte Character Codes Sets (Character Codes Sets where a single character is defined by more than one byte); and adds some needed functionality for Karaoke applications such as text display instructions, "Ruby" character display capabilities and song information.

**Note:**

The extensions described in this proposal do not apply to, nor do they supplant the use of any other text-based meta events (e.g. FF01 Text Event, FF02 Copyright Notice, FF03 Sequence/Track Name, FF04 Instrument Name, FF06 Marker, FF07 Cue Point, etc.) unless otherwise explicitly stated.

**1. Relationship with previous proposals.**

The SMF ver 1.0 specification already includes a Lyric Meta event which is targeted for the implementation of Lyrics in Standard MIDI File. The MMA adopted at the 1997 NAMM show a proposal for further defining this implementation. This new proposal takes the previously adopted MMA proposal into account and expands it to allow multi-language support.

**2. Renaming Lyric Meta Events**

Lyric Meta Events shall from now on be referred to as Lyric/Display Meta Events.

**3. Reserving 5 characters**

The following five ASCII characters shall be reserved for use in Lyric/Display Meta Events:

Reserved Characters
---
'\' Command Codes
'[' Beginning of Ruby Tag
']' End of Ruby Tag
'{' @ Beginning of Language Tag
'{' # Beginning of Song Information Tag
'}' End of Language @ or # Song Information Tag

**4. Usage of '\' (ASCII 5C), Backslash character to indicate a Command Code**

The backslash in a Lyric/Display Meta Event shall be defined as a Command Code, whose function is determined by the next ASCII character. The backslash followed by the ASCII 1-byte character '\r', '\n', and '\t' are defined respectively as Carriage Return, New Line (Line Feed), and Horizontal Tab. Other Command Codes are not yet defined, and will be assigned by the MMA.

In order to also allow the use of these special reserved characters as ordinary display characters, the sequence of a backslash followed by the special reserved characters (such as '\\', '\{', '\}', '\[', or '\]') shall act as standard display characters and not have the effect of the Command Code.

Backslash Command Codes
---
\r : Carriage Return (0x0D)
\n : New Line (Line Feed) (0x0A)
\t : Horizontal Tab (0x09)
\\ : '\' itself
\{ : '{' itself
\} : '}' itself
\[ : '[' itself
\] : ']' itself

**5. Usage of Brackets ('[' and ']') (ASCII 5B and 5D) to indicate a Ruby Part**

Character strings placed between Brackets ('[' and ']') are specified as Ruby Parts. The Ruby Part is expected to be displayed as smaller characters either above or below pictographic character also contained in the character string as defined below.

The scope of string that is included in the Ruby Part is defined as follows:

a. Character string before '[' and within the same Lyric/Display Meta event:

Example:

Lyric display: tonight
c?nE ( Kanji Character )

t1:lyric/display meta-event: "c?nE[tonight]"

b. Character string in the previous Lyric/Display Meta event, if there are no character string before '['.

Example:

Lyric display: my house
mi casa

t1:lyric/display meta-event: "mi casa"
t1:lyric/display meta-event: "[my house]"

Brackets ('[' and ']') are reserved characters for indicating Ruby Parts, so Brackets should not be used as regular lyrics. Instead, as described in section 3 (above), the following string should be used in lyrics when brackets are desired: '\[' and '\]' (backslash followed by brackets.)

**6. Usage of Tags ('{' and '}') (ASCII 7B and 7D) to indicate Character Code Set and Song Information**

6.1. Character Code Set

A Tag followed by "@" (ASCII 40) defines the Character Code Set and should be placed before any other Lyric/Display Meta Event in the SMF. That Character Code Set shall be in effect until another Character Code Set is encountered in the file. (It is allowable to use multiple Character Code Sets within a MIDI file. If no Character Code Set Tag is provided then the character set by default is the ANSI character set.

usage: {@<code_set>}

<code_set> = LATIN (or Latin, or latin)
ANSI character set for the most common European languages
<code_set> = JP (or Jp, or jp)
MS-Kanji(Shift-JIS) character set for Japanese language

Other <code_sets> for different languages are not defined yet. The MMA will be responsible for defining those <code_sets>.

If an undefined <code_set> appears, lyrics should be ignored until a defined <code_set> appears.

In addition, if a byte order mark which specifies UNICODE such as 'FF FE' or 'FE FF' exists, the character code SET should be treated as UNICODE.

6.2. Song Information

A Tag indicating information about the song is begun by "#" (ASCII 23) and ends with "=" (ASCII 3D). Song Information should always be placed at the beginning of Lyric/Display Meta Events (but after the Character Code Set Tag) so it is available for immediate display.

usage : {#<Item>=<text_string>}

<Item> = {#TITLE=} or {#Title=} or {#title=}
Song Name / Original Song Name / Song Title / Sub Title etc.
Example: {#Title= Beautiful Song}

<Item> = {#COMPOSER=} or {#Composer=} or {#composer=}
Song Writer / Composer etc.
Example: {#Composer= Tom Smith}

<Item> = {#LYRICS=} or {#Lyrics=} or {#lyrics=}
Lyrics / Poem / Words etc.
Example: {#Lyrics= Charles Scott}

<Item> = {#ARTIST=} or {#Artist=} or {#artist=}
Singer's Name / Band Name / Musician's name etc.
Example: {#Artist= Eric Wilson }

<Item> = {#}
Null tag. This tag specifies the boundary or the completion
of tags .
Example: {#Title=Beautiful Song}
{#Composer= Tom Smith}
{#Lyrics= Charles Scott}
{#Artist= Eric Wilson}
{#}

6.3 Null Item Tag

Each <Item> that follows '#' has unique priority and is handled and represented by only one state variable. A tag is defined that it is terminated by a corresponding '}'. But, in case that an unknown Multi-byte character code set is used, this terminator byte '}' may not be scanned correctly. In order to recover this lost state transition, a start of another <Item> has the capability to terminate current Item Tag. This rule works fine to make the parser more robust.

As a typical example, let us take a look at song information paragraph which is written by using several multi-byte character code sets. Even if the user describes every song information items tag events with multi-byte characters that end with '}' carelessly, the parser can recover its parsing state at each item tag event when it finds "{#" in the next item tags. For the last item tag in the same paragraph, placing just one additional null item tag "{#}" is a neat solution.

**7. Definitions**

Character, Word, Syllable, and Ruby notation

In most Western languages, a syllable within a word is represented by one or more alphabetical characters. An alphabetical character by itself (except for vowels) does represent a complete syllable. On the other hand, in Eastern languages which employ Kanji (pictographic) characters such as the Japanese language, there are many Kanji characters that consist of several syllables, because a Kanji character is not representative of a pronounced tone, but is representative of the meaning of the word. So, the structure of characters and syllables in Kanji-based languages are completely different from that of alphabet-based Western languages.

As one of the examples of Kanji-based languages, in Japanese, there are special characters called "Hiragana" and "Katakana" which have closer relation to pronunciation. For the complex Kanji words, these "Kana" notations are added beside the Kanji, in order to assist the reader and

avoid miss-reading. In particular, the "Kana"s in a smaller font beside Kanji word are called "Ruby". "Ruby" is very important for those pictographic languages especially in cases where their pronunciation is difficult or unusual. This is the reason why there is a necessity to support "Ruby" notation, in addition to the Kanji character code set.

The number of Kana characters which are added beside a Kanji character is not always just one. Although a Kanji character has its pronunciation, in some cases, special pronunciations for a word are used when a number of Kanji characters are combined. In another words, the pronunciation which is described by Kana characters is defined for a Kanji word not for each individual Kanji characters which makes up a Kanji word.

Ruby grammar

Ruby is so important for Eastern languages that the notation grammar of Ruby should be carefully defined for simplicity and efficiency. As described above, ruby is a representative of the pronunciation of a Kanji word, but no word delimiter character such as space is used in Eastern Kanji-based language. To satisfy the requirements and to solve the problems, we propose the ruby definition as follows:

Ruby part: character string between '[' and ']'.
Rubied part: character string before '[' within the same lyric/display meta event.

'[' indicates that the former part of character string in that Lyric/Display Meta event has ruby part and is treated as a start mark of ruby part.

Approved by MMA:_____ Date:_____

Approved by AMEI:_____ Date:_____