SMF with Lyrics Application Guideline

September 4, 1998

1. Objectives

This document contains information that is necessary when SMF with Lyrics related applications (products) are about to be designed. This aim is to keep the meaning of Lyric/Display Meta Event properly and also to establish the compatibility in reading and understanding it when SMFs (song/music data) were designed (exceptionally) without referring SMF with Lyrics Data Creation Guideline (attached reference).

This guideline responds to terms as shown below:

CR: Carriage Return
LF: Line Feed
HT: Horizontal Tab

Please note to see "___" as one event.

2. Scope of Ruby (The area where Ruby will be added.)

2-1. When CR(0D), LF(0A), HT(09), "\r", "\n" or ”\t” exist in a normal lyric or Song Information Tag, Ruby scope will be closed right before them and a new Ruby scope will open right after them.

When you see an event "__\r___[...]" , please understand that it means as same as "__" "\r" "___[...]" .

2-2. CR(0D), LF(0A), HT(09), "\r", "\n" or ”\t” are not supposed to be subjects for Ruby.

2-3. Ruby is not subjected to be adopted beyond CR(0D), LF(0A), HT(09), "\r", "\n" or ”\t” .

By following rule 2-1, "___\r[...]" means as same as "___" "\r" "[...]" . And it will be ignored by adopting rule 2-2. (If there is no character to apply Ruby, the Ruby will be ignored.)

3. Exceptional process in Ruby

3-1. When CR(0D), LF(0A), HT(09), "\r", "\n" or ”\t” exist in Ruby, for example, when it

is "__[..\r..]" , that means "__[..]" "\r" "..]" .

This creates a compatibility with parser — interpretational function that is not compatible with Ruby grammar so far, and also with the number of lines or paragraphs on screen system. It is also effective as a Failsafe (this is to finish Ruby always at CR, LF, HT, "\r", "\n", "\t" and get back to the condition that was before Ruby started) for the data which does not have Ruby End "]".

Here are examples to show its merit.

Please imagine that data creator initially wanted to make a lyric as follows:

"___[...]" "\r" "__[..]"

However, something happened and it lost Ruby End and became like as shown below.

"___[..." "\r" "__[..]"

If this happened, you can fix it as you wish by following the procedure described before.

Also, when there is no Ruby End in "___[...]" "__" "\r" "__[..]" , (just like "___[..." "__" "\r" "__[..]") , at least, the next line will get back to normal.

3-2. If there is a tag in Ruby:

Using tag in Ruby is prohibited. The reason is that there will be a case Ruby scope will not work properly when Character code is changed in the middle of Ruby, for example, at "{@Jp}" "___[.." "{@Unknown}..]" the parser fails to recognize Ruby End.

To switch Character Codes in the middle of Ruby, please refer 3-3.

In the case of "{@Jp}" "__" "[.." "{@Unknown}" "..]" , Ruby will be ended when Tag start "{" is found in Ruby (because Failsafe works.)

3-3. When Ruby shows up repeatedly

The case of "__[..][...]" means "__[.....]" .

If Ruby shows up repeatedly like "__" "[..]"+"[...]" , it is considered as one Ruby.

When you need to switch character codes in the middle of Ruby (as shown at 3-2),
You can end Ruby by using "{@JP}" "___[..]" "{@Unknown}" "[..]" because Ruby End exists right before the unknown character code information. This will prevent missing Ruby and make things safer.