LiveMail Format Description version 6

Header

Format header consists of 3 parts: format signature, version info and data stream size field. They are explained below.

Format signature

To determine if it is a valid LiveMail data or not, you should check the first unsigned long value (4 bytes). Correct signature is 1802128716 (decimal), other values are invalid.

Version

The next unsigned short (2 bytes) value is format version. Current format has version 6.

Stream size

Stream size if single unsigned long (4 bytes) value, specifies size of data stream in bytes.

Datastream

Datastream is sequence of *events* and *tags*, with no delimiters or any other special data.

Event is a record, that affects target mail text.
Tag is a record, that affects only decoding process.

Events

Every event has its own data (event-specific) and a timestamp. Timestamp always follows event data and specifies delta time from previous event (or beginning of the recording). Size and precision of timestamp (in bytes) depends on current *time mode*.

There are 3 types of events:

- *Basic*. They are recorded in stream without any prefixes and every event is essential to play for getting a correct message.
- *Extended essential*. They are recorded with prefix 0 (one byte prefix). If the playing engine finds such an event, it will play it anyway or stop playing.
- *Extended skipable*. They are recorded with prefixes 1 and 2 (short and long data events). Can be skipped by playing engine if it is impossible to play such event. To skip such event, player need to know its size, so next stream data specifies length of event data (without timestamp). If event prefix equals 1, then size field is 1-byte (unsigned). If event prefix equals 2, then size field is 4-bytes unsigned long. After size field, there is unsigned byte "number of placeholders" field. If player engine does not support such event, it should replace its output with "number of placeholders" characters (for example, spaces).

(Actually, there are no defined extended event types in version 6 format. Now an older version player can open newer format files using this information).

Tags

Tags are divided into 2 groups: basic and extended. All extended tags have prefix 3. Format v6 has no extended tags, they are for expansibility purposes only.

There are 2 basic types of tags in format v6:

- *Long-wait tag (prefix 4)*. Equals to a NOP event with a double-sized timestamp field and no data at all. All long pauses are implemented using this tag.
- *Set timeline mode tag (prefix 5)*. Has no timestamp, only 1-byte data field. The new timeline mode should be determined using 1-byte data field:
  0 – economic (1 byte per timestamp)
  4 – low-res (1 byte per timestamp)
  1 – high-res (2 bytes per timestamp)

  Note, that first 2 bites of timeline mode specifies ((timestamp size in bytes) – 1) value. This fact can be used to decode unknown timeline modes.

Basic events

ChangeLocale (prefix 6)

Sets new input locale. Has single unsigned long (4 bytes) data field, containing corresponding input locale.

SetCursorPosition (prefix 7)

Sets new cursor caret position to write from. Position is unsigned long (4 bytes) value in chars, measured from beginning of a text (new line characters are simple characters).

Backspace (prefix 8)

No-data event. Just pressure of "Backspace" key.

Tab (prefix 9)

No-data event. Just pressure of "Tab" key.

NewLine (prefix 13)

"Enter" key pressed. No additional data in stream.

Note: NewLine adds indissoluble 2-char sequence into text. This means, that cursor position is incremented by 2, when NewLine event occurs, but it is impossible to delete only one of NewLine chars – they are to be deleted simultaneously.

DeleteBlock (prefix 30)

"Delete" key pressed N times with no delta time between pressures. N is unsigned long (4 bytes) field of this field.

Delete (prefix 31)

"Delete" key pressed, also no additional data.

Remaining (with prefixes >= 32)

Remaining prefixes are characters itself. Character event has no prefix – it contains only data.

Timeline

All events have timestamp values – delta time from previous events or the beginning of the message.
These timestamps depend on current timeline mode:

- Economic: timestamp size is unsigned byte (1 byte) and is measured in 1/10 sec.
- Low-res: timestamp size is unsigned byte (1 byte) and is measured in 1/100 sec.
- High-res: timestamp size is unsigned short (2 bytes) and is measured in 1/1000 sec.

Default mode for player is economic.

Sample decoder in pseudo-code

```
uint32 signature =  uint32read();
uint16 version =    uint16read();
uint32 sizeOfBuffer =     uint32read();

if(signature!= 1802128716) return -1;
if(version==6)      //checking version
{
      while(! endOfBufferReached() )
      {     //analizing
            switch(currentByte)
            {
            case 0: //essental extended command - not implemeted yet
                  return -1;
                  break;
            case 1:       //skipable extended small command - skipping it
                  newEvent = new NOPEvent();
                  skipByte();
                  toSkip = uint8read();
                  dummyCount = uint8read();
                  newEvent = new InsertDummyChars(dummyCount);
                  skipBytes(toSkip);
                  readEventDelta();
                  break;
            case 2:       //skipable extended large command - skipping it
                  newEvent = new NOPEvent();
                  skipByte();
                  toSkip = uint32read();
                  dummyCount = uint8read();
                  newEvent = new InsertDummyChars(dummyCount);
                  skipBytes(toSkip);
                  readEventDelta();
                  break;
            case 4:       //wait long tag - timestamp size is 2x usual size
                  skipByte();
                  newEvent = new NOPEvent();
                  readLongWaitDelta();
                  break;
            case 5:       //new timemode tag
                  skipByte();
                  currentTimeMode = uint8read();
                  break;
            case 6:       //change locale
                  skipByte();
                  newEvent = new SetLocaleEvent(uint32read());
                  readEventDelta();
                  break;
            case 7:       //set new pos
                  skipByte();
                  newEvent = new SetPositionEvent(uint32r());
                  readEventDelta();
                  break;
            case 8:       //backspace
                  skipByte();
                  newEvent = new KeyEmuEvent(VK_BACK);
                  readEventDelta();
                  break;
            case 9:       //tab
                  skipByte();
                  newEvent = new PrintCharEvent(9);
                  readEventDelta();
                  break;
            case 13:      //enter
                  skipByte();
                  newEvent = new KeyEmuEvent(VK_RETURN);
                  readEventDelta();
                  break;
```

```
                case 30:     //deleteblock
                       skipByte();
                       newEvent = new KeyEmuEvent_Repeated(VK_DELETE, uint8read());
                       readEventDelta();
                       break;
                case 31:     //delete
                       skipByte();
                       newEvent = new KeyEmuEvent(VK_DELETE);
                       readEventDelta();
                       break;
                default:     //it is printchar (non-unicode!) event
                       newEvent = new PrintCharEvent(uint8r());
                       readEventDelta();
                }
        }
}
```