**LRC Module Description Document**

**1. Introduction**

The AP80 series chip's lyrics parsing is a software-based implementation method. Considering the AP80 system's resource consumption and module portability, memory is allocated by the application layer, and character conversion and character libraries are also implemented by the application layer. Due to memory limitations, there are restrictions on the maximum length of lyrics files and the maximum length of lyrics that can be retrieved. Currently, only one LRC lyrics parser can be supported at a time.

**1.1. LRC Format**

LRC lyrics text contains two types of tags:

First, identifier tags, formatted as “[identifier name:value],” primarily include the following predefined tags: [ar:artist name], [ti:song title], [al:album title], [by:editor (referring to the creator of the LRC lyrics)], [offset:time compensation value] (the unit is milliseconds, with positive values indicating an overall advance and negative values the opposite. This is used for overall adjustment of display speed, but most MP3 players may not support this tag).

Second, time tags, in the format “[mm:ss]” or “[mm:ss.fff]” (minutes:seconds.milliseconds). Time tags must be located at the beginning of a line of lyrics, and a single line of lyrics can contain multiple time tags (e.g., in the repeated sections of lyrics). When the song reaches a specific time point, the MP3 player will search for the corresponding time tag and display the lyrics text following the tag, thereby achieving the “lyrics synchronization” function.

Standard format: [minutes:seconds.milliseconds] lyrics

Note: Parentheses, colons, and periods must all be entered in English input mode.

2. Other format ①: [minutes:seconds] lyrics

3. Other format ②: [minutes:seconds:milliseconds] lyrics. Compared to the standard format, the period after the seconds has been changed to a colon.

**1.2. Character encoding format**

1. Common character set classifications

**ASCII and its extended character sets**

Purpose: Represents English and Western European languages.

Bit count: ASCII uses 7 bits to represent 128 characters; its extension uses 8 bits to represent 256 characters.

Range: ASCII ranges from 00 to 7F, and the extension ranges from 00 to FF.

**ISO-8859-1 Character Set**

Purpose: Extends ASCII to represent Western European languages, Greek, etc.

Bit Count: 8 bits,

Range: From 00 to FF, compatible with the ASCII character set.

**GB2312 Character Set**

Purpose: National simplified Chinese character set, compatible with ASCII.

Bit Count: Uses 2 bytes to represent 7,445 symbols, including 6,763 Chinese characters, covering almost all high-frequency Chinese characters.

Range: High byte from A1 to F7, low byte from A1 to FE. Adding 0XA0 to the high byte and low byte separately yields the encoding.

**BIG5 Character Set**

Purpose: Unified encoding for traditional Chinese characters.

Bit Count: Uses 2 bytes to represent 13,053 Chinese characters.

Range: High byte from A1 to F9, low byte from 40 to 7E, A1 to FE.

**GBK Character Set**

Purpose: An extension of GB2312, adding support for traditional Chinese characters while remaining compatible with GB2312.

Bit Count: Uses 2 bytes to represent 21,886 characters.

Range: High byte from 81 to FE, low byte from 40 to FE.

**GB18030 Character Set**

Purpose: It addresses the encoding of Chinese, Japanese, Korean, etc., and is compatible with GBK.

Bit Count: It uses variable-byte representation (1 ASCII, 2, or 4 bytes). It can represent 27,484 characters.

Range: 1 byte from 00 to 7F; 2-byte high byte from 81 to FE, low byte from 40 to 7E and 80 to FE; 4-byte first three bytes from 81 to FE, second four bytes from 30 to 39.

**UCS Character Set**

Purpose: The international standard ISO 10646 defines the Universal Character Set (UCS). It is an organization similar to Unicode, and UCS-2 is compatible with Unicode.

Bit Count: It has two formats: UCS-2 and UCS-4, which are 2 bytes and 4 bytes, respectively.

Range: Currently, UCS-4 is simply UCS-2 with 0×0000 prepended.

**UNICODE Character Set**

Purpose: To provide a unified encoding for 650 languages worldwide, compatible with ISO-8859-1.

Bit Count: The UNICODE character set has multiple encoding methods, including UTF-8, UTF-16, and UTF-32.

2. Classification by the text represented

Language Character Set Official Name

English, Western European Languages ASCII, ISO-8859-1 MBCS Multi-byte

Simplified Chinese GB2312 MBCS Multi-byte

Traditional Chinese BIG5 MBCS Multi-byte

Simplified and Traditional Chinese GBK MBCS Multi-byte

Chinese, Japanese, and Korean GB18030 MBCS Multibyte

Languages of various countries Unicode, UCS DBCS Wide-byte

**1.3. Basic Principles of Lyric Parsing**

1. Identifying Tags

This parser only identifies time compensation values for marker tags, formatted as:

[offset:ms]

Three types of time tags are recognized:

[mm:ss], [mm:ss.ms], and [mm:ss.ms]

Multiple time tags within a single lyric line are supported, e.g.:

[02:23.44][01:37.63][00:36.63] Running with the wind, freedom is the direction

2. Lyric Line Identification

The end position of a lyric line is identified by checking for “\r”, “\n”, and “\r\n”, and the length of the lyric line is determined accordingly.

3. Lyric Parsing Order

Based on analysis of mainstream lyric editing software and lyrics downloaded from websites, the vast majority of lyric files arrange lyric lines in chronological order from top to bottom. For lyric lines with multiple time tags, most use a time-based order from right to left, while a small portion of lyrics arrange time tags from left to right. For such lyrics, due to their strong regularity, they can be parsed quickly using a top-to-bottom, right-to-left (or left-to-right) order with partial loading, ensuring accurate results.

Of course, since lyric files are text-based, they can be directly opened and edited using Notepad. Therefore, lyrics with random time order also exist. In addition to random order, there are many other abnormal formats. such lyrics files are quite special, and due to the diversity of differences, most audio players will not fully support the parsing of such lyrics files. For such lyrics files, where only the tags conform to the rules but other aspects lack regularity, compatibility can only be achieved to the greatest extent possible. For smaller lyrics files, a full-file scan is performed to support random time tag order and situations where a single line of lyrics spans multiple lines.

4. Lyrics Search

All time tags in the lyrics file are treated as the distribution of multiple time points on a timeline. as the distribution of multiple time points on a timeline. The parser loads a specific time segment each time. If the time point of the lyrics being queried falls within the currently loaded time segment, the parser reads the lyrics content from the file based on the lyrics' position in the file. Otherwise, it reloads the time segment corresponding to the current lyrics' time point (the time segment range can be adjusted via parameters) and returns the lyrics content along with other information (start time point, duration, lyrics length, etc.) upon finding it.

The duration value of the lyrics is calculated based on the difference between the start time point of the next line of lyrics and the current time point. For the last line of lyrics, the default duration is 5 seconds.

5. Parameter Settings

To balance resource usage and performance, the parser should support compatibility across multiple platforms. If ported to a high-end platform, parameters can be set to adjust the parser to prioritize performance. If ported to a low-end platform, parameters can be set to prioritize resource efficiency.

**1.4. Basic Principles of Encoding Recognition**

This parser primarily recognizes the following encoding formats: GBK, UTF-8, UNICODE, UNICODE-BIG (big-endian), and others are defaulted to ANSI.

The recognition methods for the above encoding formats are as follows:

1. GBK

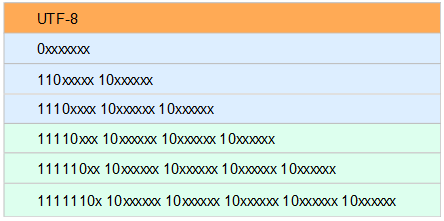
High bytes range from 81 to FE, and low bytes range from 40 to FE.

2. UTF-8 with BOM

The file header begins with three specific bytes: EFBBBF

3. UTF-8 without BOM

UTF-8 characters are composed of 2 or more bytes, up to a maximum of 6 bytes. The first byte is preceded by multiple consecutive bits with a value of 1, followed by a bit with a value of 0.



4. UNICODE with BOM

The file header begins with two specific bytes: FFFE

5. UNICODE-BIG with BOM

The file header begins with two specific bytes: FEFF

6. ANSI

If none of the above encodings match, the default is ANSI format;

**2. Typical applications**

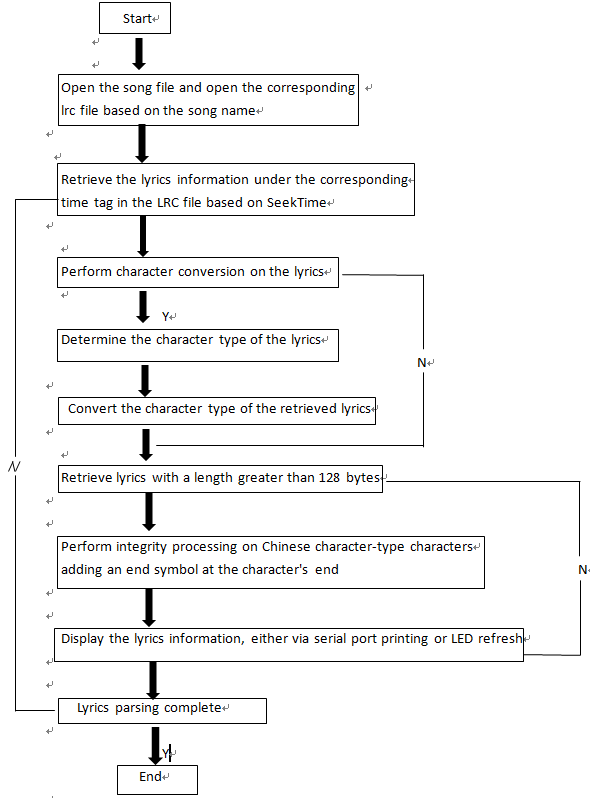


Figure 1 Typical application process for LRC lyrics analysis