AS5 Subtitle Format

By Rodrigo Braz Monteiro, Niels Martin Hansen and David Lamparter

July 10, 2007

Contents

1 Abstract

This document specifies the *AS5 Subtitle Format*, developed jointly by the Aegisub[1] and asa[2] teams in order to replace the old *Sub Station Alpha*[3] subtitle format and its extensions:

- Advanced Sub Station Alpha (ASS) implemented by Gabest in VSFilter[5]
- Advanced Sub Station Alpha 2 (ASS2), also implemented by Gabest in VSFilter
- Advanced Sub Station Alpha 3 (ASS3) implemented by equinox in asa.

The goal is to create a flexible, easy to understand and powerful subtitle format that can be used in hardsubs or multiplexed into Matroska Video[7] files as softsubs.

AS5 has no official meaning. The "'A'" can stand for Aegisub, asa, ASS or Advanced, the "'S'" for Subtitles, and the 5 is a reference to the fact that it's a major improvement over SSA4 format (from which ASS, ASS2 and ASS3 derive). The full name of the format is "'AS5 Subtitle Format'".

2 AS5 Files

2.1 File Format

All AS5 files are *REQUIRED* to comply with the three requirements below:

- Be encoded with one of *UTF-8*[8], *UTF-16 Big Endian* [9] or *UTF-16 Little Endian* Unicode Transformation Formats. UTF-8 is preffered.
- Not to have any character below Unicode code point U+20, except for U+09, U+0A, U+0D. That is, it must be a plain-text file.
- All lines must end with Windows line endings, that is, U+0D followed by U+0A.

The character set of a subtitle file can be autodetermined by its Byte-Order Mark or by the value of the first two bytes. See below.

2.2 File Structure

The file is divided in *sections*, which are uniquely identified by a string inside square brackets, in a line of its own. From that point on, every next line is considered to be part of the last found section until another section is found. There is no end-of-section termination mark; they always end at the start of the next one or at the end of the file. *Section names are case sensitive.*

Each section is divided in lines, each line representing one command or definition. Empty lines MUST be ignored. It is recommended that programs generating AS5 files insert a blank line at the end of each section to increase readability. There MUST always be a blank line at the end of the file (as every line is required to end in a line break).

Each line in a section takes the general form of *Type: data1,data2,...,dataN*. An unknown *Type* MUST be ignored by a parser. It is recommended that subtitle editing programs keep such ignored lines in the file after re-saving it. Note that the space after the colon is *mandatory*.

There are two sections which are required, *[AS5]* and *[Events]*, the former being the equivalent of *[Script Info]* in previous formats. If either of those sections is missing, the file is deemed invalid and *(*MUST) be refused by the parser. Any other section can be ommitted from the file, and need not be implemented by all parsers. However, any unknown section MUST be preserved in the file by a subtitle editing program when it re-saves a file with sections that it does not recognize. It can, however, be removed at the user's discretion.

Finally, there is a special type of undefined group, *[Private:PROGNAME]*, which MUST be *ENTIRELY* preserved by other programs when re-saving it. This is used to store program-specific data, for example, Aegisub would create a group called *[Private:Aegisub]* to store its data inside. This type of group should be identified by the fact that it starts with *"[Private:"*.

2.2.1 [AS5]

This must be the first section in every AS5 file. If the very first line of the file is not [AS5], the file MUST be rejected by the parser as invalid. Note, however, that the first line is allowed to contain a Byte-Order Mark (BOM), which is the character U+FEFF encoded in the encoding used for the rest of the script[10]. The first four bytes will therefore be:

- 0xEF 0xBB 0xBF 0x5B - UTF-8 (with BOM)
- 0x5B 0x41 0x53 0x53 - UTF-8 (without BOM)
- 0xFF 0xFE 0x5B 0x00 - UTF-16 LE (with BOM)
- 0x5B 0x00 0x41 0x00 - UTF-16 LE (without BOM)
- 0xFE 0xFF 0x00 0x5B - UTF-16 BE (with BOM)
- 0x00 0x5B 0x00 0x41 - UTF-16 BE (without BOM)

It is possible, therefore, to determine the encoding of the file by checking its first two bytes.

This section is used to declare several script properties that affect its parsing and rendering. All properties are stored in the format *Name: data*, with one property per line. This section MUST always declare the following properties:

- ScriptType: Should always be set to *AS5*, for this particular version of the specification. If this contains a value that the parser does not understand, it MUST abort parsing.
- Resolution: Should contain the script resolution in *WxH* format. For example, for a 640x480 script, this should say *"Resolution: 640x480"*. Note that this does not need to correspond to the video resolution, however, subtitles MUST be rendered on such a coordinate space. That is, in a 640x480 script, \pos(320,240) always represents the center of the script, no matter the resolution of the video it's being drawn on. Also, in a 100x100 script, a radius 50 circle centered on the center will always take half of the height and half of the width of the video, even if that means being distorted if drawn on a 640x480 video.

Also, the following items are not required, but are recommended. They all have default values:

- Generator: The name of the program that generated this script, e.g. *"“Generator: Aegisub”"*. Default value is empty. This should be ignored by the renderer, but might be useful for inter-editing-program interaction.
- Wrapping: The line wrapping style. This can be "“Manual”", in which case only \n can break lines or "“Automatic”", in which the renderer chooses how to break them. The default is "“Automatic”". Note that if this is set to manual, the line can NEVER be broken at anywhere other than forced line breaks, even if it means that the line will become unreadable because it goes outside the display area.
- Extensions: A comma-separated list of all extensions being used in this file. At the moment, there are no extensions available. Renderers should read this to enable any extensions that they might support. Editing programs MUST keep this field intact, unless the user chooses otherwise. Scripts WILL break if the list of extensions is suddenly lost.
- Credits: Credits for the people who worked on this subtitle file. Should be ignored by the renderer.
- Title: The title of this script. Should be ignored by the renderer. Subtitling programs may opt to display this title to the user.

Although any other lines are allowed in this group, this is not encouraged, as they might conflict with future revisions of the format. Instead, they should be stored in *[Private:PROGNAME]* groups, as mentioned above.

2.2.2 [Events]

The most important section, [Events], lists all the actual subtitle lines in the file. Each line is declared as *"“Line: start,end,style,user,content”"* - the syntax has been radically simplified from previous incarnations of the format, and now consist of only five fields:

- Start: The start time of the line. See below for the timestamp format. A line is only displayed if the timestamp of the current frame is *greater than or equal* to the start time. That is, start time is *inclusive*.
- End: The end time of the line. It follows the same format as the start time. The line is only displayed if the timestamp of the current frame is *lesser than* the end time. That is, end time is *exclusive*. In particular, it means that a line whose start time is equal to its end time will never be displayed. If the end time is lesser than the start time, the renderer may issue a warning, but should render the remaining lines regardless of the issue.
- Style: The name of the default style used for this line. See the [Style] section below. Should be left blank if you want to use the the script's global default style. If an unknown style is specified, the renderer MUST fallback to default, and might issue a warning.
- User: This field is used by the program to store program-specific data in each line. Renderers should ignore this. This should be left blank if it's not used.
- Content: The actual text of the line. This contains actual text and override tags. See the section on override tags for more information.

The timestamp format is h...h:mm:ss[.s...], that is, it begins with an integer of arbitrary length (up to a maximum of 4 digits) representing the number of hours, followed by a one-digit or two-digit integer representing minutes, and a floating point number representing seconds. Localization is irrelevant: a period (”.”) is always used to separate the decimal point. This way, 0:21:42.5 and 0000:21:42.5000 are equivalent, and both represent 0 hours, 21 minutes, 42 seconds and 500 miliseconds.

Spaces between each field MUST be ignored by all parsers. Any spaces at the beginning of the content line should be stripped. A hard space or empty override block should be used if space at the start of a line is truly desirable. That is, the two following lines are identical:

```
Line:  0:2:31.57  ,  0:02:34.22  ,  ,  ,  Hello world of {\b1}AS5{\b0}!
Line: 0:02:31.570,00:02:34.22,,,Hello world of {\b1}AS5{\b0}!
```

2.2.3 [Styles]

This is equivalent to the *[V4 Styles]* (and subsequent variations) from the Sub Station Alpha format. Each entry is declared as ”“Style: name,parent,overrides””. Like *[Events]*, it has been greatly simplified when compared to the previous formats, and now contains only three fields. They are:

- Name: The name of this style. Style names are not case-sensitive, but MUST be unique. A script with conflicting style names must be refused by the parser. If the style name is ”“Default””, it will be used for all lines that ommit the style name. If there is no ”“Default”” line, the renderer default is used.
- Parent: The style from which the current style derives from. See below for more information. Leaving this field blank means that the style derives from the renderer’s default style.
- Overrides: A list of override tags to define this style. See below.

Styles work in a very different way from the way they did on previous formats (with the notable exception of ASS3, which actually implements this very same style based on this format, as ”“StyleEx””). Instead of setting multiple parameters across many commas, you simply specify override tags. When a line uses a style, it’s as if the overrides of the style were inserted right before the start of the line contents.

Also, a style can inherit from another style, and define new overrides which are then appended to those of the parent style. The parent style MUST have been declared *BEFORE* the style trying to use it as a parent. If the parent doesn’t exist or wasn’t declared yet, the parser must refuse to parse the script. This is important because otherwise you could get a ”“inheritance loop””, where styles derive from each other in a cycle.

For example, see the following *[Styles]* group:

```
[Styles]
Style: Default,,\fn(Arial)\fs20
Style: Speech,,\fn(Respublica)\fs24\bord2\shad2\4a#80\2c#000000
Style: Actor1,Speech,\1c#B9C5E3
Style: Actor2,Speech,\1c#FFB3CF
Style: UglinessItself,Default,\fn(Comic Sans MS)
```

In the above fragment, the first style defines the Default style that will be used on all lines that don't set any style and the second style defines a base speech style that will be used for all actors (note that it doesn't inherit from Default, even though Default overrode the renderer's default, that one is still used for style definitions).

The third and fourth styles are based on the second, and simply assign different colours to it. They will both have all properties of Speech, and only differ in primary colour. Finally, the last example shows how to derive from the overriden default. In this case, font size would be 20 points, regardless of renderer's default.

The two Actor styles could have been defined without a parent style as follows:

```
[Styles]
Style: Actor1,,\fn(Respublica)\fs24\bord2\shad2\4a#80\2c#000000\1c#B9C5E3
Style: Actor2,,\fn(Respublica)\fs24\bord2\shad2\4a#80\2c#000000\1c#FFB3CF
```

Since all that deriving a style from another does is append the new tags to the end of the previous, this way of declaring styles is identical to the one above, but is more verbose.

References

[1] Rodrigo Braz Monteiro, Niels Martin Hansen, David Lamparter et al., Aegisub. Application, 2005-2007.
http://www.aegisub.net/

[2] David Lamparter, asa. Application, 2004-2007.
http://asa.diac24.net/

[3] Kotus, Sub Station Alpha. Website, 1997-2003.
http://web.archive.org/web/*/http://www.eswat.demon.co.uk/substation.html

[4] #Anime-Fansubs, Advanced Sub Station Alpha.
http://www.anime-fansubs.org
http://moodub.free.fr/video/ass-specs.doc

[5] Gabest, VSFilter. Application, 2003-2007.
http://sourceforge.net/projects/guliverkli/

[6] David Lamparter, Advanced Sub Station Alpha 3. Website, 2007.
http://asa.diac24.net/ass3.pdf

[7] The Matroska project.
http://www.matroska.org/

[8] The Internet Society, RFC 3629, "'UTF-8, a transformation format of ISO 10646'". Website, 2003.
http://tools.ietf.org/html/rfc3629

[9] The Internet Society, RFC 2781, "'UTF-16, an encoding of ISO 10646'". Website, 2000.
http://tools.ietf.org/html/rfc2781

[10] Unicode, Inc, The Unicode Standard, Chapter 13. PDF, 1991-2000.
http://www.unicode.org/unicode/uni2book/ch13.pdf