# AS5 Subtitle Format

By Rodrigo Braz Monteiro, Niels Martin Hansen and David Lamparter





July 11, 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. The syntax is heavily influenced by the older SSA and ASS formats, which in turn vaguely resemble the TeX typesetting language; but AS5 also has many differences compared to these older formats and you should not expect it to behave exactly like them.

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 preferred.
- 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.

These requirements are important so the AS5 format can be edited in most plain-text editors across most operating systems and languages without problems. 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.

When used as a standalone file, the extension should be .AS5. When multiplexed into a Matroska container, the Codec ID used is S_TEXT/AS5.

TODO: Get clearance from the Matroska team to use that Codec ID.

## 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 (that is, lines only containing a line ending) MUST be ignored by the parser. 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. Subtitle editing programs SHOULD 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 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 is identified by the fact that it starts with *"[Private:"*.

Note that *Format:* lines from the previous formats are not admitted in AS5. If the parser finds any of them, or any other unrecognized lines not specified here (outside the *[Private:]* section), including but not limited to unknown sections, it MUST halt parsing, rejecting the file as invalid, and it SHOULD emit a warning specifying where the problem lies.

The sections MAY be written in any order, with the exception of the *[AS5]* section which MUST always be the first section.

Any line where the first character is a semicolon (*;* - U+3B) is considered a "comment line" and MUST be ignored by the parser; they also MUST be preserved by an editing program when resaving. It is suggested that an editing program SHOULD check whether commented lines are actually valid AS5 lines, and if they are, display them to the user in some way as "disabled" lines. Note that commented out lines MUSTnot influence subtitle rendering in any way.

### 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 video with a non-1:1 aspect ratio (for example, a 640x480 video).

The following items MAY also be used; they 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". Even if it is set to Automatic, \n will still insert a line break. On the other hand, if 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. Purely for informational purposes and SHOULD be ignored by the renderer. Subtitling programs SHOULD be able to display these credits to the user.
- Title: The title of this script. Purely for informational purposes and SHOULD be ignored by the renderer. Subtitling programs SHOULD be able to display this title to the user.

Unlike in the previous incarnations of the format, storing private data here is not allowed, which means that this section MUST NOT contain any properties not listed here. Any application-specific or otherwise private data MUST be stored in *[Private:PROGNAME]* groups instead, as mentioned above.

#### 2.2.2 [Events]

The most important section, [Events], lists all the actual subtitle lines in the file. The syntax has been radically simplified from previous incarnations of the format, and now consist of only five fields. Each line is represented as:

```
Line: start,end,style,user,content
```

Where:

- 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 earlier than the start time, the renderer MAY issue a warning, but it 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. If left blank, the script's global default style MUST be used. If an unknown style name is specified, the renderer MUST fallback to default, and MAY issue a warning.

- User: This field is used by the program to store program-specific data in each line. Renderers SHOULD ignore this (but MAY use it for application-specific extension features). This field SHOULD be left blank if it's not used. Note that whatever data is stored here MUST NOT contain any commas!
- 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. Leading zeroes MAY be ommitted. 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 by any editing program. A hard space (see the overrides section) 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 syntactically 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. Like *[Events]*, it has been greatly simplified when compared to the previous formats, and now each entry contains only three fields. They are declared as:

```
Style: name,parent,overrides
```

Where:

- Name: The name of this style. Style names are not case-sensitive, but MUST be unique. A script with conflicting style names MUST be rejected by the parser. If the style name is "Default", it will be used for all lines that omit 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, with one exception: certain tags without parameters revert to the style default. For example, \c will revert the primary colour to the one specified in style. Such use of tags is invalid in the style definition, and MUST be ignored if found in them.

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.

If no Default style is defined, the renderer MUST choose its own defaults to render the text with. The defaults MUST also be used any for any properties not specified in a given style (in other words, styles with no parent inherit from the renderer defaults). These defaults are entirely arbitrary and can be set to anything, but the renderer SHOULD allow the user change them. A simple Sans-Serif font with white text and black borders is recommended if the user does not specify anything else.

#### 2.2.4 [Resources]

The new *[Resources]* section can be used to store information on external file resources, such as images and fonts. The general syntax is:

```
Resource: type,name,path
```

Where:

- Type: Must be either "font" or "image". Any other types MUST be ignored by the parser.

- Name: An unique name identifying this resource. For fonts, it must correspond to the font name, e.g., “Verdana”. For images, it’s the name that the file will be reffered as in the rest of the script. If there is already a resource with this same name, the parser MUST abort the parsing.
- Path: The location of the file relative to the subtitles. This MUST be a relative path for external .as5 files, or a container-specific string for AS5 multiplexed into a container. The relative path MUST use forward slashes and be case-sensitive, in order to avoid UNIX compatibility issues.

# 3 Style Overrides

## 3.1 General Information on Override Tags

As with previous formats, AS5 uses override tags to set the style for lines. Also, it uses those same tags to set style definitions themselves (see above). Although many tags were imported from *Advanced Sub Station Alpha*, do not assume that they behave exactly the same. Some had their behavior changed or properly defined. Also, AS5 defines many new tags in addition to the old ones.

All tags must be inserted between a pair of curly brackets ({}), except on style definitions. A pair can contain any number of override tags inside it. They should be listed one after the other, with no spaces or any other kind of separator between them. Tags then affect all text that follows it, unless re-overriden or reset by the *r* tag. For example:

```
{\fn(Verdana)\fs26\c#FFA040}Welcome to {\b1}AS5{\b0}!
```

In the above example, the first override block affects the entire text, but only "AS5" is bolded.

Some tags might begin with a $ in their names. This means that there are actually five variations of this specific tag, the tag with $ replaced with a number from *1* to *4* (inclusive) or without it altogether - in that case, the tag is assumed to mean the *1* variation. Those numbers represent the four different colours available on any given line:

- 1 - Primary colour, used for the main face of the text.
- 2 - Secondary colour, used on karaoke. See the karaoke tags for more information.
- 3 - Border colour. This is the colour of the border that outlines the text. See the \bord tag for more information.
- 4 - Shadow colour. This is the colour of the shadow dropped by the text. See the \shad tag for more information.

So, for example, you would use \1c or \c to set the primary colour, or \3c to set the colour of the border. \$c, however, does not exist in itself.

When a tag requires a floating point parameter, the decimal part MUST be specified using a period (.); never a comma. When a tag requires a colour parameter, it is given in HTML hexadecimal code, which is # followed by a 6-digit hexadecimal string, where the first two digits represent the red component, the next two the green component, and the last two the blue component (#RRGGBB). Sub Station Alpha style (Visual Basic hexadecimal) is not supported - if a parser finds any colour in a format it does not recognize (including the SSA &HBBGGRR& format) it MUST issue a warning and ignore the tag.

In the tag specification in this document, optional parameters are denoted by being enclosed by square brackets ("[]"), and may be ommitted. For example, *\baseline(curve1[,curve2])* means that the second parameter is entirely optional. It's also possible that the entire parameter set is enclosed in square brackets, e.g. *\vc[(c1,c2,c3,c4)]*.

The parameters of a tag MUST be enclosed within parantheses, with exception for tags with only one numerical parameter, for which the parantheses MAY be omitted. The parser MUST issue a warning and disregard the tag if it omits mandatory parenthesises.

It is forbidden to write comments inside standard curly brackets. Any unknown tags MUST be ignored, (the parser SHOULD issue warnings about unknown tags) and anything that doesn't begin with a backslash MUST be considered an error. For inline comments, you need to use a special variation, in which the first character inside the overrides block is an asterisk (*). Renderers MUST completely ignore any text inside such blocks. For example:

```
{\fn(Verdana)\fs26\c#FFA040}Welcome to {\b1}AS5{\b0}!{*It's a nifty format, isn't it?}
```

## 3.2 Vector Path Format

TODO: Write me

## 3.3 Special Character Escapes

The following tags are not really overrides, but rather escape codes for special characters. They MUST NOT be inside an override block, but only in the middle of the text (i.e. not between { and }).

TODO: Write these following ones

### 3.3.1 \n

### 3.3.2 \h

### 3.3.3 \{, \}

### 3.3.4 \\

## 3.4 Basic Typography Tags

### 3.4.1 \i

**Usage:**

```
\i(1)
\i(0)
```

**Description:** Italics

### 3.4.2 \b

**Usage:**

```
\b(1)
\b(0)
```

**Description:** Bold

### 3.4.3 \u

**Usage:**

```
\u(1)
\u(0)
```

**Description:** Underline

### 3.4.4 \s

**Usage:**

```
\s(1)
\s(0)
```

**Description:** Strikeout

### 3.4.5 \fn

**Usage:**

```
\fn(fontname)
```

**Description:** Set font name

### 3.4.6 \fe

**Usage:**

```
\fe(fontencoding)
```

**Description:** Set font encoding in some ISO code

### 3.4.7 \fs

**Usage:**

```
\fs(size)
```

**Description:** Set font size in points

#### 3.4.8 \bord

**Usage:**

```
\bord(bordersize)
```

**Description:** Set border size in script resolution pixels

#### 3.4.9 \shad

**Usage:**

```
\shad(shadowsize)
```

**Description:** Set shadow depth in script resolution pixels

#### 3.4.10 \bordstyle

**Usage:**

```
\bordstyle(???)
```

**Description:** Set border style in God or amz knows what

### 3.5 Font Scaling Tags

#### 3.5.1 \fsc, \fscx, \fscy

**Usage:**

```
\fsc(scale)
\fscx(xscale)
\fscy(yscale)
```

**Description:** Set font X/Y scaling in percent

#### 3.5.2 \fsp

**Usage:**

```
\fsp(fontspacing)
```

**Description:** Set font spacing between characters in pixels

## 3.6 Colouring Tags

### 3.6.1 \$c

**Usage:**

```
\$c(colour)
```

**Description:** Set font colouring in hexadecimal RGB

### 3.6.2 \$a

**Usage:**

```
\$a(alpha)
```

**Description:** Set font alpha channel (transparency) in hexadecimal RGB

## 3.7 Positioning and Rotation Tags

### 3.7.1 \left, \right, \top, \bottom

**Usage:**

```
\left(distance)
\right(distance)
\top(distance)
\bottom(distance)
```

**Description:** Margins are the distance between the subtitle text and the edge of the frame. They are used for improved aesthetics, readability, and to avoid issues with overscan. Unless manually overriden by another tag (such as \pos), the text should always be contained inside the box defined by the script area minus the four borders, as long as automatic line breaking mode is set (see the section on [AS5]).

All distance values are specified in script coordinates. The default value for all borders is 12. Margin tags can only be present once per line, and will affect all of it, not just the following block. Margin tags cannot be animated.

**Implementation:** The default positioning of the pivot point of the subtitles box is also determined by the margins. On left-align, the *x* of pivot is set to the left margin; on right-align, to $w - r$, and on middle-align, to $\frac{w+r-l}{2}$, where *w* is the script width, *r* is the value of the right margin and *l* is the value of the left margin, that is, it is put halfway between the edges defined by the margins. The rules are analogous to the *y* coordinate.

See the alignment tags for more information regarding screen alignment.

### 3.7.2 \an, \ax, \ay, \nx, \ny

**Usage:**

```
\an(numpadalignment)
\ax(xalignment)
\ay(yalignment)
\nx(xinneralignment)
\ny(yinneralignment)
```

**Description:** Set alignment in various ways

### 3.7.3 \relative

**Usage:**

```
\relative(???)
```

**Description:** it is a mystery

### 3.7.4 \vertical

**Usage:**

```
\vertical(1)
\vertical(0)
```

**Description:** Makes text vertical

### 3.7.5 \q

**Usage:**

```
\q(0)
\q(1)
```

**Description:** Set wrap style to manual (0) or automatic (1)

### 3.7.6 \pos

**Usage:**

```
\pos(x,y)
```

**Description:** Set line position to x,y in script coordinates

#### 3.7.7 \org

**Usage:**

```
\pos(x,y)
```

**Description:** Set rotation origin to x,y in script coordinates

#### 3.7.8 \frx, \fry, \frz

**Usage:**

```
\frx(xrotation)
\fry(yrotation)
\frz(zrotation)
```

**Description:** Set font rotation around x/y/z axis in degrees

#### 3.7.9 \fax, \fay

**Usage:**

```
\fax(xshearing)
\fay(yshearing)
```

**Description:** Set shearing in x and y axis, 0..1 (I think?)

### 3.8 Animation Tags

#### 3.8.1 \fad

**Usage:**

```
\fad(t1,t2)
```

**Description:** Fading text

#### 3.8.2 \t

**Usage:**

```
\t([t1,t2,]tags)
```

**Description:** Animate tags between t1 and t2

## 3.9 Shape Transformation Tags

These are tags characterized by the fact that they distort the shape of the text itself. They were designed to enhance the flexibility of the format while dealing with unusually-shaped imagery.

### 3.9.1 \distort

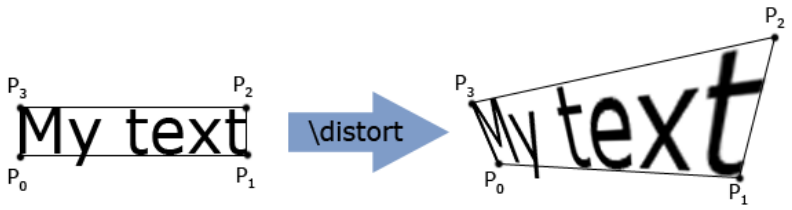
**Usage:**

```
\distort(x1,y1,x2,y2,x3,y3)
```

**Description:** The distort tag allows you to apply an arbitrary distortion to the block that follows it. It takes three coordinate pairs that, along with the origin (at the current baseline position) specify a quadrilateral.

$P_0$ is the origin, $P_1 = (x1, y1)$ is the corner at the end of the baseline for the affected text, $P_2 = (x2, y2)$ is the point above that, and $P_3 = (x3, y3)$ is the point above $P_0$. That is, they are listed clockwise from origin ($P_0$).

The following picture illustrates how this tag works:




If the parameter list is ommitted, the distort reverts to the style's default (none by default). This tag can be animated with \t.

**Implementation:** This tag cannot be reduced to an affine transformation, so it cannot be expressed in Matrix form. In order to transform a given (x,y) coordinate pair to it:

1. Normalize the (x,y) coordinates to a (u,v) system, so that $P_0$ = (0,0) and $P_2$ = (1,1). This can be done by dividing x by the block's baseline length (bl) and y by the block height (h). The affine 3D transformation matrix for this operation is:

$$\begin{bmatrix} \frac{1}{bl} & 0 & 0 & -\frac{P_{0x}}{bl} \\ 0 & \frac{1}{h} & 0 & -\frac{P_{0y}}{h} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   That is, $u = \dfrac{P_x - P_{0x}}{bl}; v = \dfrac{P_y - P_{0y}}{h}$.
2. Apply the following formula: $P = P_0 + (P_1 - P_0)u + (P_3 - P_0)v + (P_0 + P_2 - P_1 - P_3)uv$. This can be interpreted as simple vector operations, that is, apply that once using the x coordinates and another using the y coordinates. Since the four points are constant, the coeficients can

be precalculated, resulting in a very fast transformation.

### 3.9.2 \baseline

**Usage:**

```
\baseline(path1[,path2])
```

**Description:** Similarly to \distort, this tag distorts the text, however, it does so by curving the baseline into a vector path, so you can write curved text. Alternatively, you can specify a second path to work as the "ceiling" of the text. The format of both path parameters is the standard vector path format (see above).

**Implementation:** Implementation of this tag can be summarized by the conversion of a generic $P_n = (x, y)$ point into $P'_n = (x', y')$. Let $c1(t)$ and $c2(t)$ be the parametric equations of the two paths specified. The conversion can then be done in the following manner:

1. Find the parameter *t* along the baseline path that corresponds to the x position of the point being converted. This can be done with a function that calculates the length from the beginning of the path until an arbitrary point $P_t = c1(t)$ along it.
2. Calculate the base point along path1: $P_0 = c1(t)$
3. Calculate *u* so that $u = \frac{y-y_0}{h}$, where $y_0$ is the y coordinate of the original baseline and *h* is the height of the block box.

Now, for the single curve version:

1. Find the tangent vector of path1 at point $c1(t)$ and find the *V* unit vector that is perpendicular to the curve at that point, by rotating the tangent vector by -90 degrees along the Z axis. This should give you a vector pointing "up", towards where the letters go. This can be summarized as:
   $V = (\lim_{h\to 0}(c1_y(t) - c1_y(t+h)), \lim_{h\to 0}(c1_x(t) - c1_x(t+h)))$
   $V = \frac{V}{\|V\|}$
   TODO: Is that correct?
2. Multiply *u* by the vector to find the offset from $P_0$, that is, $P'_n = P_0 + uV$.

And for the two-curve version:

1. Calculate the ceiling point along path2: $P_1 = c2(t)$
2. Get *P* with the parametric equation of the line defined by $(P_0, P_1)$: $P = (1-u)P_0 + uP_1$.

### 3.9.3 \blpos

**Usage:**

```
\blpos#
```

**Description:** This sets the position of the text relative to the baseline start. This tag can be animated. TODO: Write proper specs for this.

### 3.9.4 \bls

**Usage:**

```
\bls[#]
```

**Description:** This sets the baseline shift, that is, the vertical spacing between each character and the baseline in which it is supposed to be sitting on. The default value is 0, and the parameter is given in script coordinates.

This tag can be animated with \t, and can be reverted to style default by ommitting its parameter.

## 3.10 Rastering Tags

These tags affect how the subtitles are rasterized, that is, they affect things such as colour, blurring, etc.

### 3.10.1 \$vc

**Usage:**

```
\$vc(colour1,colour2,colour3,colour4)
```

**Description:** Sets the primary colour to blend with each of the four vertices of draw polygon. The primary use for this is to make smooth gradients easily, which are often required for proper blending with the background. Note that you can also set alpha using this tag.

### 3.10.2 \$blend

**Usage:**

```
\$blend(mode)
```

**Description:** Sets the blending mode for the colour specified. Acceptable values are ”normal”, ”add” and ”multiply”.

### 3.10.3 \clip

**Usage:**

```
\clip(x1,y1,x2,x2)
```

**Description:** Clips so only text inside the rectangle formed by x1,y1,x2,y2 will be drawn

### 3.10.4 \iclip

**Usage:**

```
\iclip(x1,y1,x2,x2)
```

**Description:** The inverse of \clip, i.e. clips so only text outside the rectangle formed by x1,y1,x2,y2 will be drawn.

### 3.10.5 \$blur

**Usage:**

```
\$blur(???)
```

**Description:** Blurs stuff

## 3.11 Advanced Typography Tags

These are more advanced tags, which might prove to be fairly complex to implement. They include things such as ruby text support (also known as furigana, when used with Japanese Kanji).

TODO: Write me

# 4 Renderer Behaviour Specification

TODO: Write this section

# 5 Container Multiplexing Specification

## 5.1 Matroska

Storage of AS5 files in Matroska files is similar to how similar formats are stored.[11] The Codec ID used is S_TEXT/AS5

First, the entire file is converted to UTF-8 (if it isn't already UTF-8). Then, all sections other than *[Events]* and *[Resources]* are stored on the *CodecPrivate* element. For the *[Resources]* section, each line is parsed and files are converted to Matroska file attachments. TODO: Specify this more clearly.

Finally, each line in the *[Events]* section is read and stored each in a block. The *start* and *end* fields are parsed (see the specifications on the section describing [Events]) and set as the *TimeStamp* and *BlockDuration* elements. The line itself is then stored in the following format:

```
Line: readOrder,style,userData,contents
```

Where *readOrder* is the number that the line had on the file. This is necessary so the file can be demultiplexed back in its original order, since lines will be stored in chronological order while inside the Matroska file. The remaining fields should just be copied from the original line.

# 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. Website.
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

[11] The Matroska project, specification for SSA/ASS subtitle formats. Website.
http://www.matroska.org/technical/specs/subtitles/ssa.html