Note:
To change the product logo for your own print manual or PDF, click "Tools > Manual Designer" and modify the print manual template.
This is “Title Page 1” - you may use this page to introduce your product, show title, author, copyright, company logos, etc.

This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view. This is the reason why the previous page was blank (the previous page is the back side of the cover)
Aegisub

© 2005 Rodrigo Braz Monteiro. All rights reserved.

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: oktober 2006 in (wherever you are located)

Publisher
...enter name...

Managing Editor
...enter name...

Technical Editors
...enter name...
...enter name...

Cover Designer
...enter name...

Team Coordinator
...enter name...

Production
...enter name...

Special thanks to:

All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...

Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.
Table of Contents

Foreword 6

Part I  Introduction  8
  1 What is Aegisub?  8
  2 Highlights  8
  3 Credits  9
  4 Support Aegisub  11
  5 Mini-FAQ  11

Part II  Basics  15
  1 Working with subtitles  15
  2 Working with video  22
  3 Working with audio  27
  4 Keyboard Shortcuts  32
  5 Options  34
  6 Attaching Subtitles to Video  46

Part III  Typesetting  51
  1 Introduction  51
  2 Editing styles  52
  3 Tutorial  57
  4 ASS Override Tags  62
  5 Karaoke  73

Part IV  Tools  75
  1 Attachment Manager  75
  2 Color Picker  76
  3 FexTracker  77
  4 Fonts Collector  78
  5 Styling Assistant  80
  6 Timing Postprocessor  83
  7 Translation Assistant  84
  8 Other Tools  86
  9 Export  89

Part V  Automation  92
  1 Introduction  92
  2 The Automation Manager  93
3 Lua Quickstart................................................................................................................................. 94

4 Reference ........................................................................................................................................ 96
   The scripting environment .............................................................................................................. 97
   process_lines ................................................................................................................................. 97
   Special variables ......................................................................................................................... 98
   Data structures ............................................................................................................................ 99
      Meta data..................................................................................................................................... 99
      Style data................................................................................................................................. 99
      Subtitle data............................................................................................................................. 100
      Karaoke data............................................................................................................................ 101
   Configuration system .................................................................................................................. 102
   Library functions ....................................................................................................................... 105

5 Included scripts............................................................................................................................. 108
   Demo scripts .............................................................................................................................. 108
   utils.lua ....................................................................................................................................... 110
   Karaskel ....................................................................................................................................... 112
      Basic karaskel.......................................................................................................................... 115
      Advanced karaskel.................................................................................................................. 116

Index ............................................................................................................................................... 118
Foreword

This is just another title page placed between table of contents and topics
Top Level Intro

This page is printed before a new top-level chapter starts
1 Introduction

Welcome to Aegisub. You can check the following topics to get an introduction to the program:

What is Aegisub?
Highlights
Credits
Support Aegisub

1.1 What is Aegisub?

Aegisub is a program whose goal is to make typesetting a much better experience. Most programs that support Advanced Substation Alpha (ASS) lack many vital functions, which makes typesetting with them much more difficult than it should be. Aegisub has many tools as well as advanced support for typesetting and audio timing. Moreover, there is a built-in powerful automator for karaoke effects, amongst other things.

1.2 Highlights

Aegisub supports:

- Unicode in UTF-7, UTF-8, UTF-16LE and UTF-16BE encodings
- Importing non-Unicode subtitles from over 30 different character encodings, including Shift_JIS, which allows you to resume work on any subtitles, regardless of system locale settings
- Loading of Advanced Substation Alpha (ASS), Substation Alpha (SSA), Subrip (SRT) and Plain-Text (TXT) (with actor data) subtitle formats
- Full tag conversion from SRT to ASS
- Multiple Undo/Redo levels
- Powerful Automation module using the Lua scripting language to create advanced karaoke effects (including per-syllable effects, similar to AssCalc in this regard) or any other kind of subtitle manipulation.
- Syntax highlighting
- Fonts collector to place all fonts used in a script into a folder
- Opening of videos via Avisynth for a true preview of subtitles using VSFilter in any format supported by DirectShow
- Aspect ratio overriding for anamorphic videos
- Variable Frame Rate (VFR) videos via a Matroska timecodes file (v1 and v2), including exporting processed times for hardsubbing
- Display of video coordinates under mouse and simple auto positioning support
- Motion tracker to make subtitles automatically follow a certain video element, including resizing support
- Audio mode which allows you to open files in any format that your DirectShow can decode, including Wave, MP3, Ogg Vorbis, AAC, AC3 and others
- Powerful and highly customizable audio timing and playback mode, including full support for karaoke manipulation
- Functions and hotkeys easily synchronizing audio/video and subtitles
- Time shifting and manipulation by timestamps or frame numbers (once video or timecodes are loaded)
- Advanced find and replace with support for Regular Expressions
- Cut/Copy/Paste subtitles via plain-text format so they can be pasted in any plain-text editor
- Translation assistant to easily translate subtitles to another language
• Styling assistant to quickly pick styles for each line
• Styles manager that allows you to categorize subtitles styles to make working on several projects painless
• Keyframe navigation for fast seeking to scene boundaries
• Several subtitle manipulation macros, such as making timing continuous or recombining double lines after Optical Character Recognition (OCR)
• Highlighting of subtitles that are visible in the current video frame and a button to select those
• Preview subtitle changes immediately by pressing Ctrl+Enter
• Shortcut buttons to set text as bold, italic, underline, strikeout, change font or any of the 4 colors
• Easy color matching with built-in color picker
• Powerful semi-automatic motion tracking through the FexTracker tool
• Full support for attaching fonts and pictures to the subtitle files (as per the SSA/ASS specifications)
• Easy to use, intuitive interface
• Extensive manual with detailed instructions and tutorials for typesetting and program usage
• Available in 8 different languages

1.3 Credits

Programming

• Rodrigo Braz Monteiro (ArchMage ZeratuL) - Lead developer, all-around programming, PRS library programming
• Niels Martin Hansen (jfs) - Automation programming, all-around programming
• Fredrik Mellbin (Myrsloik) - Avisynth-related programming
• David Lamparter (equinox) - Unix port, FFMPEG-related programming, ASA developer
• Hajo Krabbenhoeft (Tentacle) - Motion tracker programming
• Ghassan Nassar (nmap) - Spell checker programming
• Pomyk - Donating a few patches
• Yuvi - Some Mac-specific patches

Installer

• movax
• Myrsloik

Manual

• ArchMage ZeratuL
• jfs
• Kobi
• movax
• TheFluff
• Jcubed
• Motoko-chan

Localization files

• French: TechNiko & Cryrsal
• Portuguese: ArchMage ZeratuL
• Russian: thrash-sensei
- German: equinox
- Spanish: Nesukun
- Italian: EmBolo & Mazinga
- Korean: oblisk

**Alpha testers**

- b0nk
- Bot1
- deathbygirl
- DoGfOoD
- grimlock
- Jagobah
- Jcubed
- Kintaro
- Kobi
- Kodachrome
- maxx-
- mASSIVe
- Maya
- Mentar
- movax
- msb
- neo2sonic
- nich
- nwa
- omgifos
- SOzuken
- TheFluff
- ]thrash_sensei[
- Vincent
- zegnat

**Additional thanks**

- Mentar for hosting the SVN repository up to version 1.09
- BerliOS for hosting the SVN repository for versions 1.10+
- Bot1 for hosting the forums and Bug Tracker
- Motoko-chan for compiling and hosting pre-release builds
- Firebird for some of the toolbar icons
- Kayle for help with Win32 VfW interface

**Honorable mentions**

- kryptolus for developing Sabbu, the competing program :)
- Gabest for developing VSFilter
- The Avisynth developers: Ben Rudiak-Gould, Edwin van Eggelen, Klaus Post, Richard Berg and others
About

Aegisub was developed using Microsoft Visual Studio 2003 and wxWidgets 2.6.0. The Automation module also uses the Lua Programming Language. The Regular Expressions code used in this program was written by Henry Spence and included in wxWidgets. Aegisub also uses the PortAudio, libpng, zlib and ffmpeg libraries.

1.4 Support Aegisub

Do you want to support Aegisub? Well, it's easy!

Feedback
You can give us feedback - comments, critiques, suggestions, etc. Bug reports and feature requests are always welcome. Check out our forums and the bug tracker, or stop by for a chat in the IRC channel.

Spreading the word
Like Aegisub? Tell your friends about it! Spreading the word is a good way of helping Aegisub to be the best subtitle editor around.

Donating
Feeling generous? Consider giving us a donation! We do this in our spare time, you know.

Programming
"Given enough eyes, all bugs are shallow."
--Linus Torvalds
Feel like really helping out, or do you just have some code you want to donate? Some advice from readme.txt in the source tree:

First, some of the code is pretty readable, some is decent, and some is patched up crap. Good luck. ;)
Second, if you want to code anything for Aegisub, you will need to agree to these terms:
1. You will release the patch to the public domain or give its copyright to one of the developers. This is to stop a source file from being owned by too many people. (Exception: MAJOR changes might be accepted under BSD license under your name. Consult the developers)
2. Make SURE it compiles and works fine before submitting to developers.
3. Stick to the coding standards. That is, no GNU-style indenting and crap.
Third, this is all available under the BSD license. According to GNU itself, BSD is GPL-compatible, meaning that you can link GPL code to BSD code. Keep in mind, though, that if a source file has mixed BSD and GPL content, it becomes ruled by GPL.

1.5 Mini-FAQ

A small collection of Frequently Asked Questions about Aegisub - mostly stuff that didn’t fit anywhere else.

Karaoke effects?
Can I create DVD subtitles with Aegisub?
Not directly, but there's a nifty program called MaestroSBT (Google for it) that can convert SSA to VOBSubs. It does have quite a bunch of restrictions on what tags and other things may be used, so reading its manual first is advised. Also note that it does not accept ASS - only SSA. You can use Aegisub's File -> Export... dialog to save real SSA files.

Does Aegisub allow saving to SRT?
Yes, but only if it means that no information will be lost. In other words, if you have any override tags that aren't \lc, \b, or \i, Aegisub won't allow saving directly to SRT. However, you can still export to SRT by using the File -> Export... dialog box. Just uncheck all the checkboxes (clean script info, VFR transform etc.).

What's this PRS stuff?
Pre-Rendered Subtitles. Basically a way to "encode" ASS (text) subtitles into pictures, which can then be encoded into the video using an Avisynth filter. There's also a DirectShow renderer, but it's in early development and not ready for public use yet. PRS has a few advantages over ASS (and a few disadvantages too). One advantage is that it doesn't require much CPU power to render onto the video - all the effort is spent when "encoding" to PRS. Another is that the subtitles will look exactly as the one who rendered them intended, which is not guaranteed with ASS (not even when rendering with VSFFilter, as it has had, and still has, quite a few rendering bugs). The main disadvantage is that PRS takes more space than ASS and other text-based subtitle formats do.

I've found a bug!?
Report it on the bug tracker. Please include as many details as possible in your report!

Why doesn't Aegisub have <feature X>? <Program Y> has it!
Quite possibly because we didn't know you wanted it. Request it on the bug tracker and see what happens.

Where can I find more information and/or get help?
For Aegisub-related stuff, the forums and the IRC channel are good places to ask questions. The Aegisub wiki also contains some more obscure information not included in the manual for various reasons, and so does the forums.
For general video-related questions, Doom9.org and its forums is generally the place to go.

I click the "Play video" button, and the video and audio aren't synchronized at all!
That's not a question. This is because of the way Aegisub displays video, it does not guarantee sync with the audio at all. Also, sometimes AviSynth can be quite slow at delivering frames, which would make real sync playback impossible. The "play video" feature is intended to display frames in rapid succession to check that some typesetting looks right, not as a replacement for a media player application. You can be sure that, if it sounds right when you click the "play line" button for the audio, and the subtitle looks like it's at the right frames, it is timed correctly to the audio and it is timed correctly to the video. If you don't believe this, just test it in a real media player (or hardsub it) and see for yourself. If you absolutely must have this, feel free to implement it yourself.
Are there any VSFilter bugs I should know about?
In one word: yes.
Top Level Intro

This page is printed before a new top-level chapter starts
2 Basics

The following topics will introduce you to the basic operations of Aegisub:

- Working with subtitles
- Working with video
- Working with audio
- Keyboard Shortcuts
- Options

2.1 Working with subtitles

Layout of the subtitle buttons and fields

![Aegisub's main window, with the ever-present subtitles grid.](image)

© 2005 Rodrigo Braz Monteiro. All rights reserved.
Subtitle Editbox

Items in panel:

- **Comment**: When checked, sets the line to a Comment. Comment lines are not displayed during playback.
- **Style (Dropdown)**: Select a style to apply to the line.
- **Actor (Combo box)**: Select or type the actor for a line (this field is only for the script creator's reference).
- **Effect (Text field)**: Type in the "effect" for the line. It was originally used to tell the renderer to apply line-wide effects, but that use is now deprecated. Hence, this field can be used to store information for karaoke effects or automation scripts, for example.
- **Margins (Left, Right, Vertical)**: These three fields are for the left, right and vertical margins, respectively.
- **Layer**: Starting off the second row of the panel, this field allows you to specify the layer number for current line. Subtitles on different layers do not collide with each other. Numerically higher layers are rendered above lower layers.
- **Start Time**: Start time (in h:mm:ss.cs) when the "Time" radio button is selected, or starting frame of the subtitle when the "Frame" radio button is selected.
- **End Time**: End time (in h:mm:ss.cs) when the "Time" radio button is selected, or ending frame of the subtitle when the "Frame" radio button is selected.
- **Duration**: Duration (in h:mm:ss.cs) when the "Time" radio button is selected or frames when the "Frame" radio button is selected.

### Toolbar Buttons:

- **B**: Toggles "Bold" style on selection.
- **I**: Toggles "Italics" style on selection.
- **U**: Toggles "Underline" style on selection.
- **<del>**: Toggles "Strikeout" style on selection.
- **<>**: Inserts a font override tag at the cursor position.
- **#**: Selects the primary color on cursor position.
- **&&**: Selects the secondary color on cursor position.
- **&&**: Selects the outline color on cursor position.
- **&&**: Selects the shadow color on cursor position.

### Example:

<table>
<thead>
<tr>
<th>{[0]}</th>
<th>but we didn't end up like this in the way that you're thinking.</th>
</tr>
</thead>
<tbody>
<tr>
<td>But we didn't end up like {[1]} this in the way that you're thinking.</td>
<td></td>
</tr>
</tbody>
</table>

As seen from the example, the cursor was at the start of the text so the tag is inserted at the start of the text.

In this example, the cursor was in the middle of the text, so the tag was inserted at that point.
Note: In addition to setting the parameters at the cursor position, the style/color can be set for selected text only.

```
but we didn't end up {\textbf{like this in the way\textit{}} that you're thinking.}
```

Here, the tag is applied only to the selected area and not the entire line. This only works for styling parameters that can be enabled or disabled. The parameters are:
- Bold
- Italics
- Underline
- Strikeout

### Grid columns

<table>
<thead>
<tr>
<th>#</th>
<th>L</th>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Left</th>
<th>Right</th>
<th>Vert</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>00:00:00.00</td>
<td>00:03:55.00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Some text with tags</td>
</tr>
</tbody>
</table>

**Subtitle Display Field**

All in all, there are 11 possible columns to work with. However, some of them are hidden when empty. You can right-click on the column headers to set which ones you want to display.

**Columns (from left to right):**

- **First Field (#):** Shows the line number, relative to the entire subtitle field.
- **L:** Layer number of the line.
- **Start:** Shows the starting time of the line.
- **End:** Shows the ending time of the line.
- **Style:** Displays the name of the style used for the line.
- **Act:** Displays the name of the actor for the speech of the line. Hidden if empty.
- **Eff:** Displays the effect(s) used on the line. Hidden if empty.
- **Left:** Displays the value of the left margin (in pixels).
- **Right:** Displays value of the right margin (in pixels).
- **Vert:** Displays value of the vertical margin (in pixels).
- **Text:** Displays the text and tags of the line.

### Line colors

<table>
<thead>
<tr>
<th>Time</th>
<th>Color</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:35.56</td>
<td>Standard(Air)</td>
<td>0000 0000 0000 That's all you've been saying.</td>
</tr>
<tr>
<td>0:00:37.02</td>
<td>Alternative(Air)</td>
<td>0000 0000 0000 You're so slow.</td>
</tr>
</tbody>
</table>

There are two different colors for the lines. The aquamarine one is the current/selected line. The yellow one is the line on the current frame of the video.

**Subtitle Context Menu**
The subtitle context menu makes subtitle operations much easier. The following functions are present:

- **Insert (before)**: Insert an empty line before the selected one with the start and end time set to the end time of the previous line.
- **Insert (after)**: Insert an empty line after the selected one with the start and end time set to the end time of the selected line.
- **Insert at video time (before)**: Inserts an empty line before the selected one with the start time set to the time of the video and the end time set to 5 seconds after the start time.
- **Insert at video time (after)**: Inserts an empty line after the selected line with the start time set to the time of the video and the end time set to 5 sec after the start time.
- **Duplicate**: Creates a copy of the selected line(s) and puts them before the original(s).
- **Duplicate and shift by 1 frame**: Creates a copy of the selected line(s), shifts the start and end times one frame forward, and puts the resulting line(s) after the original(s). Useful for frame-by-frame typesetting.
- **Split (by karaoke)**: Splits the selected line(s) into one new line per karaoke syllable (timed according to the syllable times), and removes the \k tags. Does nothing if the line doesn't have any karaoke syllables.
- **Swap**: Swaps two selected lines.
- **Join (concatenate)**: With this, the start time of first line gets the start time of the joined line. The end time of the joined line is the end time of the last line from the join. The text from the different lines is separated with a \N tag (hard break).
- **Join (keep first)**: With this, only the text of first line stays. The end time of the last selected line is used for the end time.
- **Join (as karaoke)**: Creates a new line with the text of the selected lines. Between the text from the different lines, the \k<duration> tag is added. The duration is set so that it matches the length of the line before it was joined.
- **Make times continuous (change start)**: Sets the start time of the second line to match the end time of the previous selected line. Works on two or more selected lines.
- **Make times continuous (change end)**: Sets the end time of the first line to match the start time of the second selected line. Works on two or more selected lines.
- **Recombine(1,1+2) into (1,2)**: Used to recombine text in two lines where the text from the first line is in the second line separated with the \N tag (further explained in the example).
- **Recombine(1+2,2) into (1,2)**: Used to recombine text in two lines where the text from the second line is in the first one separated with the \N tag (further explained in the example).
- **Recombine(1,1+2,2) into (1,2)**: Used to recombine text that's separated into three lines into two lines without the \N tag. (further explained in the example).
- **Copy**: Used to copy one or more lines from the list field.
- **Cut**: Used to cut one or more lines from the list field.
- **Paste**: Used to paste one or more lines in the list field.
- **Delete**: Used to delete one or more lines in the list field.

**Examples:**
- **Swap**:

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00.00</td>
<td>0:00:00.00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- Before Swap ---</td>
<td></td>
</tr>
<tr>
<td>0:03:53.72</td>
<td>0:03:54.18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to swap</td>
<td></td>
</tr>
<tr>
<td>0:03:55.10</td>
<td>0:03:56.06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2 to swap</td>
<td></td>
</tr>
<tr>
<td>0:00:00.00</td>
<td>0:00:00.00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- After Swap ---</td>
<td></td>
</tr>
<tr>
<td>0:03:55.10</td>
<td>0:03:56.06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2 to swap</td>
<td></td>
</tr>
<tr>
<td>0:03:53.72</td>
<td>0:03:54.18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to swap</td>
<td></td>
</tr>
</tbody>
</table>

With this function, the two lines get switched. The only thing that is changed is the position of the lines; the time stamps stay as they were. So, both lines keep the start and end time they had before. Same goes for the other parameters. This can only be used on two or more lines.

- **Join (concatenate)**:

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00.00</td>
<td>0:00:00.00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- Before Join(concatenate) ---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:03:53.72</td>
<td>0:03:54.18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to be joined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:03:55.10</td>
<td>0:03:56.06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2 to be joined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:00:00.00</td>
<td>0:00:00.00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- After Join(concatenate) ---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0:03:53.72</td>
<td>0:03:56.06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to be joined</td>
<td>Line 2 to be joined</td>
<td></td>
</tr>
</tbody>
</table>

This type of join merges the two lines into one, so the start time of the joined line is the same as the start time of first line, and the end time is the same as the end time of the last time. The text of the joined lines is separated with a \N tag. On the example, lines 1 and 2 were joined. This can only be used on two or more lines.
• **Join (keep first):**

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- Before Join(keep first) ---</td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:54:18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to be joined</td>
</tr>
<tr>
<td>0:03:55:10</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2 to be joined</td>
</tr>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- After Join(keep first) ---</td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to be joined</td>
</tr>
</tbody>
</table>

This type of join makes one line using the text of the first line with the end time set to match the end time of the last line. All other lines get removed. This is shown in the example. This can only be used on two or more lines.

• **Join (as Karaoke):**

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
<th>Dialogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- Before Join(as Karaoke) ---</td>
<td></td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:54:18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1 to be joined</td>
<td></td>
</tr>
<tr>
<td>0:03:55:10</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2 to be joined</td>
<td></td>
</tr>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- After Join(as Karaoke) ---</td>
<td></td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>{\textbackslash}k{Line 1 to be joined}{\textbackslash}k{Line 2 to be joined}</td>
<td></td>
</tr>
</tbody>
</table>

With this type of join the lines are joined into one line where the start time is set to match the start time of the first line and the end time matches the end time of the last line. The text is separated with \k as seen in the example. The value of \k is set to match the duration of the line that contained the text before. The \k value is set for text from every one of the lines, so it's set for the text from line 1 and line 2. Same goes if there are more lines, \k gets set for all of them. This can only be used on two or more lines.

• **Make times continuous:**

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- Before Make times continuous ---</td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:54:18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1</td>
</tr>
<tr>
<td>0:03:55:10</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2</td>
</tr>
<tr>
<td>0:03:55:27</td>
<td>0:03:58:05</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 3</td>
</tr>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- After Make times continuous ---</td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:54:18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1</td>
</tr>
<tr>
<td>0:03:54:18</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2</td>
</tr>
<tr>
<td>0:03:55:06</td>
<td>0:03:58:05</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 3</td>
</tr>
</tbody>
</table>

As seen in the image, this function sets the start and end times of the selected lines so that there is no empty time. So the start time of the second line is set to match the end time of the first line. The start time of the third line is set so that it matches the end time of the second line. It's the same if used on more lines; the start times match the end times of the previous lines.

• **Recombine (1,1+2) into (1,2):**

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
<th>Dialogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- Before Recombine(1,1+2) into (1,2) ---</td>
<td></td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:54:18</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1</td>
<td></td>
</tr>
<tr>
<td>0:03:55:10</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1/Line 2</td>
<td></td>
</tr>
<tr>
<td>0:00:00:00</td>
<td>0:00:00:00</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>--- After Recombine(1,1+2) into (1,2) ---</td>
<td></td>
</tr>
<tr>
<td>0:03:53:72</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 1</td>
<td></td>
</tr>
<tr>
<td>0:03:55:10</td>
<td>0:03:56:06</td>
<td>Default</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>Line 2</td>
<td></td>
</tr>
</tbody>
</table>

As seen from the image, this recombines two lines where the second line contains text.
from the first line as well as additional text separated with the \N tag. The end time of the first line is set so that it matches the end time of the second line. The text in the second line that is in front of the \N tag gets removed together with the tag. The result are two lines that contain only the text from one actor unlike before where the second line contained the text from two actors.

- Recombine (1+2,2) into (1,2):

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
<th>DiaIo</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00</td>
<td>0:00:03</td>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:03:53</td>
<td>0:03:54</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:03:55</td>
<td>0:03:56</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:03:57</td>
<td>0:03:58</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:04:00</td>
<td>0:04:00</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
</tbody>
</table>

This type of recombine is similar to the previous one. The difference is that this one is used when the combined text is in the first line and not the second one. Here the start time of the second line is set so that it matches the start time of the first line.

- Recombine (1,1+2,2) into (1,2):

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Style</th>
<th>Act</th>
<th>Eff</th>
<th>Lef</th>
<th>Rig</th>
<th>Ver</th>
<th>DiaIo</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:00:00</td>
<td>0:00:03</td>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:03:53</td>
<td>0:03:54</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:03:55</td>
<td>0:03:56</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:03:57</td>
<td>0:03:58</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
<tr>
<td>0:04:00</td>
<td>0:04:00</td>
<td>Defa ult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----</td>
</tr>
</tbody>
</table>

As seen in the image, there are three lines before the use of the option. The second line is a combination of the first and third lines and as such is not needed. When the function is applied to the lines, the end time of the first line is set to match the end time of the second line. The start time of the third line is set so that it matches the start time of the original second line. In the end, the second line is removed and as a result there are two lines that represent the previous three lines.

**Hiding tags**

In Aegisub the tags can be hidden in the subtitle field. By default they are represented with a sun like symbols as seen in the image below.

*Presented by* **\N** *Infidel*

This option can be disabled or replaced with any other character in the config.dat file. The image below is an example of when the tags are enabled and shown in the subtitle field.

*Presented by* **\N** *Infidel*

To enable or disable the hiding of the tags, the "grid hide overrides" option in the config.dat file has to be changed. To change the symbol for the hidden tags, the "grid hide overrides char" option has to be changed.

Check the [Options](#) page for more information on editing config.dat.
Importing plain text files

It is possible to import "scripts" from plain text files. Aegisub assumes that your text file is formatted something like this:

Captain: What happen?
        Did someone set up us the bomb?
# TL note: lolz, I'm gonna make this a meme
Cats: All your base are belong to us.

In this case, "What happen" and "Did someone set up us the bomb?" will turn out as two different lines, with the "Actor" field set to "Captain". "All your base are belong to us" will be a third line, with the "Actor" field set to "Cats". The line starting with a "#" will be imported as a comment line, with the # stripped.
If you like writing in some other way - like this, for example:

Captain -- What happen?
        Did someone set up us the bomb?
// TL note: lolz, I'm gonna make this a meme
Cats -- All your base are belong to us.

You may want change the text actor separator and text comment starter settings in config.dat.

2.2 Working with video

Opening Video

To open a video, simply go to Video -> Open Video... and choose a file. Aegisub accepts AVI, D2V, VOB and AVS files by default. Loading of the D2V and VOB formats requires you to have a plugin that provides the mpeg2source() (such as dgdecode or mpeg2dec3) function in your Avisynth plugin autoloading folder.
You can also open any other format that DirectShow recognizes (e.g. MKV and OGM) by selecting All Files, but beware that DirectShow is not required to give you the exact frame, so seeking might not be exact! If you do this anyway, Aegisub will warn you about it.

Once you open the video, Aegisub will switch to Video Mode, which looks something like this:
The video toolbar controls

The second bar under the video screen is the Seek Bar. You can use this bar to navigate through the video using the mouse, by dragging it, or using the keyboard Left Arrow and Right Arrow keys to step frame by frame. The black lines at the top indicate keyframes.

Holding down the Shift key while pressing Left Arrow or Right Arrow will cause the program to jump to the next keyframe. This functionality is only available for AVI and MKV files at the moment.

Holding down the Control key while pressing Left Arrow and Right Arrow will make program jump between start and end timecodes of each line. This is especially useful to look for subtitles “bleeding” into adjacent scenes.

Hint: You can right click the Seek Bar to focus it without changing its position. This is useful if you want to adjust position with keyboard, as left clicking can accidentally move the bar. Also, you can use the Ctrl+Space shortcut to toggle focus between the seek bar and whatever was focused before.
Playback

You can play the video from current frame to end by pressing the Play button. Also, you can play only current active line by pressing the Play Line button. Press the Pause button to, well, pause. Aegisub does playback via software rendering, which means that it is quite CPU intensive.

Aegisub also supports audio playback along with video, but you must first load the audio via the Audio menu.

*Note and mantra:* Aegisub is not a media player. Do not expect it to play video/audio perfectly. There are ZERO guarantees that the video and audio will be in sync, or even play at the right speed at all! If you want a media player, use a media player.

Zooming

The drop-down menu above the video window (the one with a percentage in it) is the zooming menu. It defaults to 75% of original size - a reasonable value for a NTSC-resolution video source and a 1024x768 display. It is possible to change this default value in the options file (config.dat). The setting is called video default zoom, logically enough.

You can also change zoom levels with the "magnifying glass" buttons in the toolbar.

Motion tracking

The two rightmost buttons below the video belong to FexTracker, a highly useful motion tracker tool. It's complicated enough to have its own manual page, and is thus documented in the tools section and not here.

Position displays

To the right of the zoom bar are two boxes. The first is the current frame box; it will tell you the current frame time and frame number. This value is always accurate, provided that you are working with a seeking-accurate file (e.g. a XviD in AVI without null frames). Also, the background of this box becomes green if the current frame is a keyframe.

The second box indicates the time, in milliseconds, of current frame relative to current line start and end. This is the value you must enter in some ASS tags, such as `\move()`, if you want something to happen at this frame.

The Video menu

Adjusting Aspect Ratio
Sometimes, you might deal with anamorphic video, that is, video whose true aspect ratio does not correspond to the display aspect ratio. For example, some widescreen videos are stored as 704x480 and resized to something that is 16:9 (such as 853x480) during playback.

Under the Video menu, you will find five override options:

- **Default Aspect Ratio** - leaves video intact
- **Fullscreen Aspect Ratio** - forces 4:3
- **Widescreen Aspect Ratio** - forces 16:9
- **2:35 Aspect Ratio** - forces 2.35:1
- **Custom Aspect Ratio** - Forces any fractional aspect ratio of your choice, or a specific resolution. The latter way is the only way to change the video height (see the note below). If you enter an aspect ratio (as opposed to a resolution), you can use either a fraction (such as 4:3) or a decimal number (such as 2.35) - the latter example is equivalent to writing "2.35:1". To force a specific resolution, enter it as \( WIDTH \times HEIGHT \) (example: 852x480).

**Note:** All the modes, except Custom (in specific-resolution mode) and Default, modify the video width, not the height.

**Note 2:** The video is resized *after* the subs has been applied (the same thing happens when playing softsubs in DirectShow).

### Jumping to a position

To jump to a specific position in video, go to the Video->Jump to... menu, or hit the shortcut **Ctrl+J**. This allows you to jump to either a specific time or specific frame.

### Jumping to a subtitle line/snapping a line to a video position

There are six "subtitle interaction" controls. They all have a **Ctrl+<number>** shortcut by...
Variable Frame Rate

Some videos are Variable Frame Rate (VFR) - meaning they don't follow a constant frame rate. Aegisub supports these in two ways: by loading specific timecode files to override the frame rate from video; and by loading a Matroska video via DirectShow. Due to aforementioned issues, it's recommended that you open an AVI and apply timecodes to it.

In order to load a timecodes file, go to Video -> Open Timecodes... and choose the file. Aegisub supports Matroska v1 and v2 timecodes files.

If you don't do this, subtitles will most likely appear at the wrong time. Also, if you export the subtitles, you can enable VFR compensation (useful when hardsubbing).

Positioning
Putting mouse over video (when it's not playing) will show a crosshair with coordinates of current position. This can be used for any ASS override tags which require coordinates as a parameter, such as \pos(), \move() and the drawing commands.

Double click the video to add a \pos() tag at the current video location.

Additionally, holding down Shift will make the coordinates relative to the bottom right corner of the video (instead of the default upper left), which can be useful in certain cases. Note that you have to move the mouse while holding shift to get the display to update.

2.3 Working with audio

Opening Audio

To load an audio file into Aegisub, just go to the Audio menu and press Open audio file. If you have a video file already loaded, you can use Open audio from video instead, which obviously will load the audio track from the video file you currently have loaded. You can open any type of audio file that your DirectShow environment knows how to decode. For example, if you want to load an AC3 file, you will need an AC3 DirectShow decoder (e.g. AC3filter or fdshow). Note, however, that some formats seem buggy at the moment. Ones more or less guaranteed to work are WAV, MP3 and Vorbis, so if your audio doesn't work, try transcoding to one of them, at least temporarily.

Warning: If you have opened a video file with more than one audio track (most commonly an MKV or OGM file), and try to open audio from it, Aegisub is completely at the mercy of the splitter when it comes to what audio stream is delivered. Some splitters may deliver both audio streams at once (this will happen for dual audio AVI's, when using the default Windows splitter), and since Aegisub very much doesn't expect that, you will get weird results (and probably crashes). Just remux the file to single audio, or better yet, decompress the desired audio stream to WAV.

When loaded, the audio is downmixed to mono (see the audio downmixer option in the
options documentation if you want to grab one channel only), decompressed to PCM (a.k.a. WAV), and (by default) loaded into a RAM cache. This means that you will need a large amount of RAM to open a long audio file. If your computer doesn't have a lot of RAM, or if you're working with a full-length movie, refer to the audio cache option in the options documentation for instructions on how to make Aegisub use its (slower) hard drive cache instead.

The exact amount of memory used can be calculated with the following formula:

$$ s = \frac{b \times r \times l}{8} $$

where $s$ is the amount of memory (in bytes - divide by 1024 to get kB), $b$ is bits per sample (always 16 in the current implementation), $r$ is the sample rate in Hz (usually 48000, or 44100 in some cases), and $l$ is the length of the audio (in seconds).

Loading and decompressing the audio into the cache takes a little while. When it is done, Aegisub will transform into something like the picture below:

Green and red buttons are toggle buttons. A green background indicates that the option is turned on. A red background indicates that the option is turned off.

The buttons are as follows (many of these have keyboard shortcuts associated with them):
1. Previous line (previous syllable when in karaoke mode)
2. Next line (next syllable when in karaoke mode)
3. Play selected area of the audio waveform
4. Play currently selected line
5. Pause playback
6. Play 500ms before selection start
7. Play 500ms after selection end
8. Play first 500ms of selection
9. Play last 500ms of selection
10. Play from selection start to end of file (or pause is pressed)
11. Add lead-in (how much is determined by a setting in config.dat - see the Options section, audio lead in setting)
12. Add lead-out (exactly like the above, but the option is called audio lead out, logically enough)
13. Commit (save) changes
14. Go to selection
15. Toggle auto-commit (all timing changes will be committed immediately, without the user pressing commit if this is on)
16. Toggle auto-scrolling (will center waveform on the currently selected line automatically)
17. Toggle SSA mode (see below)
18. Toggle spectrum analyzer mode (see below)
19. Audio display zoom (horizontal)
20. Audio display zoom (vertical)
21. Audio volume
22. Toggle linking of vertical audio zoom slider with volume slider
23. Toggle karaoke mode
24. Join selected syllables (karaoke mode only)
25. Split selected syllables (karaoke mode only)

Normal audio timing

When you click on a line, Aegisub will highlight it, and, if you have the Auto checkbox
checked, center the audio display on it (the writer of this manual section prefers to have it unchecked, as it can be quite annoying when timing). Now, you can drag around the boundaries of the selection (red lines) to change the start and end times of the line. The dark blue lines are second boundaries, and the pink ones are indicate keyframes (probably scene changes) in the video, if you have it loaded. The selection will become red and show the word Modified in red on the audio display until you either press the Commit button (or press Enter, which also will take you to the next line) to save the changes (if you have autocommit on, it will never turn red - all changes are saved immediately). Press the Play selection button to listen to the selected area, or the other play buttons to listen to other areas around the selected area. When you are satisfied with the timing, press Commit. Normal audio timing is as easy as that - it's just a lot of work, going through all of the lines and fitting them to the audio. All serious timers use keyboard shortcuts extensively - they make the timing orders of magnitude faster.

Note: Aegisub takes up huge amounts of screen area when having both video and audio loaded. If you don't need to look at the video, but want it loaded for keyframe display, use the View menu to hide it (by choosing Audio + Subs view).

The SSA timing mode

The SSA timing mode makes Aegisub behave more like the old Substation Alpha when timing. Many timers prefer this mode, since it can be incredibly fast. When you are in SSA mode, left-clicking on the waveform sets the line start, and right-click sets the line end. When you press Commit, Aegisub automatically goes to the next line and marks out a new selection, starting at the previous line end for you. With a liberal use of keyboard shortcuts, this timing mode can get incredibly fast.

Note that by default, autocommitting is forced to disabled when you are in SSA mode (for good reason, since with it on, Aegisub will go to the next line immediately after you click anywhere on the graph). This can be changed in config.dat, together with a few other SSA mode quirks (such as going to the next line on committing).

The spectrum analyzer

Pressing the spectrum analyzer button (number 18 in the screenshot above) will make the waveform look drastically different:

When you press the button, the waveform does no longer show amplitude (signal strength) on the vertical axis - instead it shows frequency. The higher up, the higher the
frequency. The colors indicate amplitude, with dark blue being silence and red being the strongest sound (note that by default, the colors of the selected area will be inverted - see the screenshot). This may seem confusing, but since the frequency window is set to fit human voices rather well, it can make it easy to tell where a line (or a word in karaoke mode) starts and ends when there's a lot of background noise (or music) that makes it hard to tell from the normal waveform. It can be especially useful when timing karaoke. Play around with it for a little while, and you'll understand how it works. Unfortunately, the spectrum analyzer mode is quite slow.

**Karaoke timing**

Enter the harder part of audio timing: karaoke. Aegisub's karaoke mode is quite similar to, but not the same as, Medusa's equivalent, so if you are familiar with that, Aegisub should give you little trouble with karaoke.

When you have timed each line to match the lines of the song, enter the karaoke mode by pressing the *Karaoke* button. This will reward you with Aegisub changing the audio view to something like the picture below:

![Karaoke View](image)

The first part of karaoke timing is timing the words to match the words of the song. Here, the karaoke mode works just like the normal timing mode with one exception: you can click on a word (or later on, a syllable) to highlight it and then use the numpad + and - buttons to make fine-grained adjustments (just make sure numlock is on). Also note that the play buttons have changed slightly - next/previous line is now next/previous syllable, and most of the other buttons now work on syllables instead of on lines. Also note that the next/previous syllable buttons can take you from one line to the next if you go past/before the last/first syllable in a line - Aegisub will, however, warn you that you're about to lose all changes to the current line if you do.

Remember that if you want to shift the beginning or end of the line, you have to exit the karaoke mode to do so. Remember to press *Commit* before you exit the karaoke mode, or you will lose all the changes you just worked on! If you frequently forget, it might be a good idea to turn autocommiting on.

When you have finished timing the words, it's time to split the words into syllables. Hit the *Split* button, and watch the word display change:
Move the cursor (red) with the mouse, and click to place a split marker. Hit the Commit button (or press enter) to commit the splitting and write out the `{\k}` tags. To insert an empty (zero-length, no-text) syllable (useful for non-vocal parts between words or syllables) between two others, place the split cursor between the syllable end and the first/last character. Sometimes, it can be a good idea to add a split marker between the last character in the word and the following space (as the example above shows) - this depends on what kind of karaoke effect you are planning to do, and if there's a pause between one word and the next. If you are not the person who is going to do the effect, ask the responsible person about how they would like it.

If you want some other tag than \k (like \kf or \ko), right-click the syllable to toggle.

When you have the syllables split, time them in the now (hopefully) familiar way. If you want to insert an extra empty syllable, highlight a syllable before or after the empty one, press Split and type a pipe before or after the syllable, depending on where you want the empty syllable. When you feel satisfied with the line, remember to press Commit! The karaoke override tags won’t be written until you do.

### 2.4 Keyboard Shortcuts

This is a list of all keyboard shortcuts available in Aegisub. The section header indicates where the focus needs to be for shortcut to take effect. Note that some of these functions cannot be accessed in any way other than by shortcut.

Aegisub now also features a Hotkeys dialog (Tools menu -> Hotkeys...) where you can change these to suit your own preferences. This list shows the default keys, and what they do (the text in the dialog is rather terse).

**Video display**
- Double left click - Set line position to mouse coordinates

**Video seek bar**
- Right click - Select bar without scrolling (not a keyboard shortcut, but...)
- Left arrow - Previous frame
- Right arrow - Next frame
- Shift+Left arrow - Previous keyframe (keyframes are fast to decode and usually at
  scene transitions, so this is very useful)
- Shift+Right arrow - Next keyframe
- Ctrl+Left arrow - Previous subtitle start/end (use this to transverse through all lines'
  start/end times)
- Ctrl+Right arrow - Next subtitle start/end
- Alt+Left arrow - Jump to 10 frames before
- Alt+Right arrow - Jump to 10 frames after
- Down arrow - Forwards the event to the subtitles grid (see below)
- Up arrow - Forwards the event to the subtitles grid (see below)

**Audio display**
- Space - Play selection
- Left arrow - Previous line (or previous syllable in karaoke mode)
- Z - Same as above
- Right arrow - Next line (or next syllable in karaoke mode)
- X - Same as above
- Numpad '+' - Increase syllable length (karaoke mode only)
- Numpad '-' - Decrease syllable length (karaoke mode only)
- Shift+Numpad '+' - Increase syllable length, shifting other syllables (karaoke mode
  only)
- Shift+Numpad '-' - Decrease syllable length, shifting other syllables (karaoke mode
  only)
- A - Moves back
- S - Play selection
- D - Play 500ms before end of selection
- F - Move forward
- G - Commit
- Ctrl+Enter - Commit
- R - Play original line
- Q - Play 500ms before selection
- W - Play 500ms after selection
- E - Play 500ms after start of selection
- C - Add lead in
- V - Add lead out

**Subtitles edit box**
- Enter (on contents box) - Accepts changes to contents and moves to next line. Creates
  a new line if this is the last.
- Ctrl+Enter - Accepts changes to contents but stays on line

**Subtitles grid**
- Down arrow - Moves to the next line
- Up arrow - Moves to the previous line
- Ctrl+Down Arrow - Activates the next line, without selecting it
- Ctrl+Up Arrow - Activates the previous line, without selecting it
• Alt+Down arrow - Moves line down (effectively swaps it with next)
• Alt+Up arrow - Moves line up (effectively swaps it with previous)
• Left arrow - Forwards event to video seek bar (see above)
• Right arrow - Forwards event to video seek bar (see above)

**Global**
- F1 - Help
- Ctrl+S - Save subtitles
- Ctrl+O - Open subtitles
- Ctrl+N - New subtitles
- Ctrl+Z - Undo
- Ctrl+I - Shift times
- Ctrl+F - Find
- F3 - Find next
- Ctrl+H - Replace
- Ctrl+X - Cut selected lines
- Ctrl+C - Copy selected lines
- Ctrl+V - Paste selected lines
- Ctrl+Numpad 8 - Previous row
- Ctrl+Numpad 2 - Next row
- Ctrl+Space - Toggle focus between control and seek bar
- Alt+F4 - Exit

**Global (with video loaded only)**
- Ctrl+J - Jump to frame/time
- Ctrl+1 - Jump video to start of current line
- Ctrl+2 - Jump video to end of current line
- Ctrl+3 - Set start of current line to video frame
- Ctrl+4 - Set end of current line to video frame
- Ctrl+5 - Snap to scene (sets start and end of subtitles to the keyframes around current video frame)
- Ctrl+Numpad 4 - Previous frame
- Ctrl+Numpad 6 - Next frame

### 2.5 Options

**What is the Aegisub config.dat file?**

The config.dat file in the Aegisub directory stores all the Aegisub settings and some history over recently opened files, recently used search strings etc. It's a plain text file, so you can open it in any normal text editor (e.g. notepad). Since Aegisub doesn't yet have an options dialog, this is the only way to change a lot of options (for now, at least).

The settings are in the form of: "**some setting=value**", with one setting per line. The first line is always "[Config]" - don't remove it. If you delete config.dat, all settings will be restored to their default values and the file will be recreated when you restart Aegisub. If the file doesn't exist after installation, it's because you never ran Aegisub (it's created with default values the first time the program runs). Note that renamed or deprecated settings are not automatically removed from config.dat, so you may have old settings in it that doesn't do anything anymore. It is generally safe to assume that anything not documented on this page is obsolete and unneeded.
Important note: some of these settings are completely harmless and can be very useful for tweaking Aegisub to your personal preference. However, some of them are not intended to be changed by the casual user, and there are some that can completely break your Aegisub installation and/or all scripts you save if you change them. This means: read the documentation carefully before changing any settings - the dangerous ones are marked as such.

Options Conventions used

foo=BOOL means that "foo" takes a boolean value as its argument - i.e., it can be either true or false, with 1 meaning true, and 0 meaning false, as per standard. Making "foo=1" will cause setting "foo" to be on or enabled.

foo=NUM means that "foo" takes a decimal number as its argument. If any limits apply, those are mentioned in the explanation about the setting.

foo=STRING means that "foo" takes an (almost) arbitrary text string as its argument. This is often used for paths or directory names, in which case all the operating system's limits apply. In Windows, this means that the string can't contain any of the following characters (slash and backslash are allowed, but both work as directory separators):
   \ / : * ? " < > |

foo=HEX means that "foo" takes a six-digit hexadecimal number, preceded by a # sign, as its argument. This is used for colors, and the syntax is the familiar RRGGBB one (note: it's backwards compared to ASS, which has BBGRR). That is, "foo=#FF0000" will set "foo" to full intensity red.

Settings list

allow ancient avisynth=BOOL
Aegisub normally requires Avisynth 2.5.6a or later. Enabling this setting prevents Aegisub from complaining when running with an older version of Avisynth. This might lead to weird issues and/or complete audio/video breakage, so it's not recommended. The only ones who should use this options are people who want to test Aegisub under Wine (Avisynth versions from 2.5.3 on doesn't work in Wine, you have to use 2.5.2) or people who are trying to find out how much stuff breaks. Obviously defaults to disabled (0).

audio autocommit=BOOL
If this is true (1), dragging the markers in the audio view will commit changes immediately, thus making the Commit button unnecessary. Identical to the Autocommit button in the GUI - use that instead of editing config.dat. Defaults to 0.

audio autofocus=BOOL
When true, the audio display area will automatically gain focus upon mouseover. Defaults to disabled (0), since it can be very annoying.

audio autoscroll=BOOL
Has the same effect as clicking the "Auto" button in the audio view. When it is enabled,
it scrolls the audio view to center on the line you click. Just use the button instead of editing config.dat. Defaults to 1.

**audio background=HEX**
The color of the background in the audio view, as a hexadecimal RRGGBB value. Defaults to #000000 (black).

**audio cache=NUM**
Determines the way Aegisub caches audio. Valid values are:
- 0 - Cache disabled, which means streaming (reading from the file on demand). This is almost always a Bad Thing, since it results in bad performance, inaccurate seeking and inaccurate pointers (for Vorbis, the audio can be off the indicated value by a second or more).
- 1 - RAM cache. This is the default. RAM caching means that Aegisub decompresses the entire audio stream to the RAM, which is fast, but might be inconvenient for a full-length movie or if you are short on RAM.
- 2 - Hard drive cache. This resembles RAM caching, but the stream is decompressed to disk instead, which is slower, but better if you are low on RAM or have a full-length movie to work with.

**audio display height=NUM**
The height of the audio display, in pixels. This can be changed from inside Aegisub, so generally there's no need to touch this setting. Defaults to 100.

**audio downmixer=STRING**
This is the AviSynth function that will be used to give you the one channel audio that Aegisub uses. Valid values are:
- GetLeftChannel - Grabs the left channel only
- GetRightChannel - Grabs the right channel only
- ConvertToMono - Merges all channels into one (this is the default)
GetLeft/GetRightChannel are slightly faster than ConvertToMono, but when using only one channel, the waveform might not be precise.

**audio draw secondary lines=BOOL**
Controls whether additional boundary lines are drawn on the audio display. This includes video keyframe markers and minute-markers. Defaults to true (1). Note: This setting was called audio draw boundary lines prior to beta 1.08.

**audio draw selection background=BOOL**
Controls whether the background for the current audio selection is shown in different colors or not. Defaults to true (1).

**audio inactive lines display mode=NUM**
Controls how Aegisub displays the lines surrounding the currently selected one in audio mode. Valid modes are:
- 0 - Disabled (only show the current line)
- 1 - Show the previous and next lines in grey (this is the default behaviour)
- 2 - Show all unselected lines in grey

**audio lead in=NUM**
**audio lead out=NUM**
The amount of lead-in/lead-out that is added when the corresponding buttons are pressed in Aegisub. Values are given in milliseconds and defaults to 100 (lead-in) and 300 (lead-out).
audio line boundaries thickness=NUM
Thickness in pixels of the boundary lines for the current audio selection. Defaults to 2.

audio line boundary start=HEX
audio line boundary end=HEX
The color of respectively the start and end line boundaries in the audio view, as hexadecimal RRGGBB values. Both default to #FF0000 (red).

audio line boundary inactive line=HEX
The color of the lines surrounding the currently selected one in the audio waveform (see the audio inactive lines display mode setting). Defaults to #808080.

audio lock scroll on cursor=BOOL
If set to true, the audio display will automatically scroll to follow the playback cursor when playing audio. Many people find this feature very annoying, and hence it defaults to disabled (0).

audio play cursor=HEX
The color of the playing cursor/pointer in the audio view, as a hexadecimal RRGGBB value. Defaults to #FFFFFF.

audio sample rate=NUM
If set to a nonzero value, Aegisub will convert the audio to the given sample rate (in Hertz) on loading. 0 (the default value) means keep the original sample rate. Note: Resampling the audio is ridiculously slow and can easily double the audio loading times. It is doubtful if there's anything to gain from resampling, so just leave this option at 0.

audio seconds boundaries=HEX
The color of the lines that marks the seconds boundaries in the audio view, as a hexadecimal RRGGBB value. Defaults to #0064FF.

audio selection background=HEX
The color of the background of the selected area in the audio view, as a hexadecimal RRGGBB value. Defaults to #404040.

audio selection background modified=HEX
The color of the background of the selected area, when modified but not yet committed, in the audio view, as a hexadecimal RRGGBB value. Defaults to #5C0000.

audio spectrum=BOOL
Has the same effect as clicking the spectrum analyzer button in Aegisub. Use that button instead of editing config.dat. Defaults to off (0).

audio spectrum cutoff=NUM
The range of the spectrum analyzer (it goes from 0 to the given value). Unfortunately, noone knows what unit it is given in (due to the FFT code being such a mess), so don't ask about it. Defaults to 32 (presumably a good value, so leave it there).

audio spectrum invert selection=BOOL
If true, the selected audio area will have inverted colors when in spectrum analyzer mode. Defaults to true (1).

audio spectrum window=NUM
The size of the spectrum analyzer's FFT (fast fourier transform). The actual value used is 2 to the power of the given value. This indicates the number of samples from the audio signal used in the analysis and also determines the number of discrete frequency bands. When a high number of frequency bands are used, the bands have a smaller bandwidth, which allows for more accurate frequency readings. This means that setting this value lower will make Aegisub's spectrum analyzer mode faster but more inaccurate. Defaults to 11, which means the actual size becomes $2^{11} = 2048$ (a good value - changing it is not recommended, but if you want to experiment, feel free to do so at your own risk).

```
audio ssa allow autocommit=BOOL
    Allows or disallows having autocommit (see the "audio autocommit" option) on when timing audio in SSA mode. Defaults to false (0). Note: this setting was previously known as audio allow ssa autocommit.
```

```
audio ssa mode=BOOL
    Has the same effect as clicking the SSA button in Aegisub. Use that instead of editing config.dat. Defaults to false (0).
```

```
audio ssa next line on commit=BOOL
    In SSA mode, turning this on will make Aegisub automatically go to the next line when you press Commit. Defaults to off (0).
```

```
audio syllable boundaries=HEX
    The color of the syllable boundaries in audio view's karaoke mode, as a hexadecimal RRGGBB value. Defaults to #FFFF00.
```

```
audio syllable text=HEX
    The color of the syllable text in audio view's karaoke mode, as a hexadecimal RRGGBB value. Defaults to #FF0000.
```

```
audio waveform=HEX
    The color of the audio waveform, as a hexadecimal RRGGBB value. Defaults to #00C800.
```

```
audio waveform inactive=HEX
    The color of the audio waveform of the lines surrounding the current one (see audio inactive lines display mode). Defaults to #404040.
```

```
audio waveform modified=HEX
    The color of the selected area of audio waveform, when modified but not yet committed, as a hexadecimal RRGGBB value. Defaults to #FFC8C8.
```

```
audio waveform selected=HEX
    The color of the selected area of audio waveform, as a hexadecimal RRGGBB value. Defaults to #FFFFFF.
```

```
audio wheel default to zoom=BOOL
    Controls the behaviour of the mouse wheel. If false, the mousewheel scrolls the audio display, and zooms it if shift is pressed. If true, the functions are inversed - i.e., it zooms by default and scrolls when shift is pressed. Defaults to false (0).
```

```
auto backup=BOOL
    Turns the auto backup system on or off. When on, it will save a backup of the script
```
when opening it, to the folder specified in the auto backup path setting. Defaults to 1 (on).

auto backup path=STRING
The path where backups of the scripts are stored, if auto backup is on. It can be both an absolute path (like C:\backup\scripts) or a relative one. In the latter case, it is relative to wherever aegisub.exe is. Defaults to autoback (which means that the backups will be stored in a folder called "autoback" in your Aegisub folder).

auto recovery path=STRING
The directory where recovery scripts are saved, if Aegisub should crash. It can be both an absolute path (like C:\backup\scripts) or a relative one. In the latter case, it is relative to wherever aegisub.exe is. Defaults to recovered (which means that the recovered scripts will be stored in a folder called "recovered" in your Aegisub folder).

auto save every seconds=NUM
Makes Aegisub autosave the script you are working with to the folder specified in the auto save path setting every X seconds, where X is the number you choose. Defaults to 60. Setting it to 0 or less than 0 disables autosave.

auto save path=STRING
The path where autosaves of the scripts are stored, if autosave is on. It can be either an absolute path (like C:\autosave\scripts) or a relative one. In the latter case, it is relative to wherever aegisub.exe is. Defaults to autosave (which means that the autosaves will be stored in a folder called "autosave" in your Aegisub folder).

autoload linked files=NUM
Controls how Aegisub loads video/audio/timecodes files associated with the script, when opening it. It has three modes:
• 0 - Never load any associated files
• 1 - Always load associated files
• 2 - Ask to load associated files when script is opened (this is the default behaviour)

automation include path=STRING
The include path for automation scripts. Defaults to Aegisub directory/automation/include.

automation script editor=STRING
The path to the .exe of the text editor used to edit automation scripts. This line does not exist by default, it gets created when you set it for the first time.

avisynth memorymax=NUM
The maximum allowed memory for Avisynth (Aegisub's audio/video handler) to use, in megabytes (the same as Avisynth's SetMemoryMax() function). 0 means unlimited. Defaults to 64. Setting this lower than 16 or so is probably a bad idea.

color picker recent=STRING
The 24 most recently selected colors in the color picker dialog. This is a space-separated list of color values. Any values in excess of 24 are ignored; if less than 24 are specified, the rest is filled in with black. There is generally no point in editing these values. **NOTE:** these colors are specified in ASS hex, not HTML hex! This means the byte order is different from the rest of the color values used in the config file (BBGRR as opposed to the rest of the config file's RRGGBB).
color picker mode=NUM
The last used color spectrum mode in the color picker dialog. Setting it in Aegisub instead of changing config.dat is a good idea. Can be one of the following values (the default is 4):
- 0, 1, 2 - The RGB modes, respectively with red, green and blue for the slider bar channel.
- 3 - YUV mode
- 4 - HSL mode
- 5 - HSV mode

edit box need enter background=HEX
The color used in the time boxes when their content is modified but not yet saved. Defaults to #C0C0FF.

find affect=NUM
find field=NUM
find match case=BOOL
find regexp=BOOL
find update video=BOOL
These are all saved settings from the Find dialog. Set them using that dialog instead of editing config.dat.

font face=STRING
The font for the grid and the edit box. Defaults to an empty string (nothing), which means that it ends up as whatever wxWidgets feels like using.

font size=NUM
The default font size of all normal text. Normal sanity limits apply. Defaults to 9 when running under Windows, but 11 under anything else (*nix, in other words).

fonts collector destination=STRING
The destination path of the fonts collector feature. Set it using that dialog instead of editing config.dat. Defaults to ?script, which means "wherever the currently open script is located".

grid active border=HEX
The border color of active lines in the subtitle lines grid. Defaults to #FF5BEF.

grid allow focus=BOOL
If set to false, the subtitles grid will not be allowed to gain focus. This means that either the audio or the video will be in focus at all times, which may be considered useful by people who get annoyed by the audio display losing focus all the time. However, it might also lead to problems with keyboard shortcuts (among other things). Defaults to 1 (enabled). Keeping it enabled is recommended - the concept of focus is central in Aegisub.

grid background=HEX
The background color of the grid. Defaults to #FFFFFF (white).

grid collision foreground=HEX
The foreground color of lines in the grid detected as colliding with other lines. Defaults to #FF0000 (red).

grid comment background=HEX
The background color in the grid of a line marked as a ASS/SSA comment. Defaults to #D8DEF5.

grid font size=NUM
The font size used in the grid. Normal sanity limits apply here as well. Defaults to 8 under Windows, 10 under anything else.

grid header=HEX
The background color of the grid header line (the one at the top). Defaults to #A5CFE7.

grid hide overrides=NUM
Sets the way override tags (everything inside curly braces) are displayed in the grid. It has three modes:
- 0 - Leave the tags as they are
- 1 - Replace the tags with string given in the "grid hide overrides char" setting (this is the default behaviour)
- 2 - Hide the tags completely

grid hide overrides char=STRING
The character that will replace the override tags if "grid hide overrides" is on. Despite the setting name, this can actually be a string. Defaults to a "sun", for lack of a better term. The Unicode definition of the character is BLACK SUN WITH RAYS (sun - 2609).

grid inframe background=HEX
The color of a line in the grid that is currently visible in the video display. Defaults to #FFFDEA.

grid left column=HEX
The color of the leftmost column in the grid (the one with the line numbers). Defaults to #C4ECC9.

grid lines=HEX
The color of field outlines in the grid. Defaults to #808080 (dark grey).

grid selected comment background=HEX
The background color of a comment line when selected. Defaults to #D3EEEE.

grid selection background=HEX
The background color of a selected line in the grid. Defaults to #CEFFE7.

grid selection foreground=HEX
The foreground (text) color of a selected line in the grid. Defaults to #000000 (black).

grid show column 0=BOOL ... grid show column 10=BOOL
What columns (numbered from left to right) that will be shown in the grid. You can change this in the UI by right-clicking the top row of the grid, so there's no need to change this setting manually in config.dat.

grid standard foreground=HEX
The standard foreground (text) color of the grid. Defaults to #000000 (black).

highlight subs in frame=BOOL
When true, all lines that are currently visible in the video will be highlighted in the grid
with the color specified in the "grid inframe background" setting. Defaults to true (1).

keep raw dialogue data=BOOL
Aegisub 1.09 and earlier used huge amounts of RAM, mostly due to keeping a lot of line data in memory all the time. In 1.10 the behavior was changed so that the data is generated on demand instead, lowering RAM usage by a lot. Setting this option to true will make Aegisub go back to the old behaviour. It's only intended for testing and should NOT be enabled under normal conditions. Defaults to disabled (0).

last open audio path=STRING
last open automation path=STRING
last open subtitles path=STRING
last open timecodes path=STRING
last open video path=STRING
These lines store the last opened file of the respective type. These are subtly different from the "recent ..." lines mentioned below. Editing them is generally not necessary.

link time boxes commit=BOOL
When true, the time editing boxes will be linked so that pressing enter in one of them will commit changes in both. Defaults to true (1).

locale code=NUM
Sets the language to use. The default is -1, which means "Ask user when program starts", and after you have answered that, it is whatever you chose there. The language can be changed later through View -> Languages, and the setting will be saved here. If some tragedy happens and you can't get to View -> Languages, you can set it back to -1 to get Aegisub to ask you on start again. Setting it to a bogus value will make Aegisub fallback to English. **Note:** the locale is not stored as a language/country code, it's just an internal number.

recent aud #1=STRING ... recent aud #N=STRING
recent find #1=STRING ... recent find #N=STRING
recent replace #1=STRING ... recent find #N=STRING
recent sub #1=STRING ... recent sub #N=STRING
recent timecodes #1=STRING ... recent timecodes #N=STRING
recent vid #1=STRING ... recent vid #N=STRING
These are the audio, find, replace, script, timecodes and video histories, respectively. There's no reason to edit them, except if you want to clear Aegisub's history, in which case you can delete them.

recent aud max=NUM
recent find max=NUM
recent replace max=NUM
recent sub max=NUM
recent timecodes max=NUM
recent vid max=NUM
The maximum size of the respective recently opened files histories. Each defaults to 16. It's probably unwise to set any of these higher than 64.

save charset=STRING
Selects what character set ("charset") Aegisub saves the subtitles as when you press Save. **You should NOT touch this setting unless you really know why you'd want to change it!** A much better (and safer) way to save to other charsets, should you feel the need to do so, is by using the File -> Export... dialog, which lets you specify the charset you want to use. Note: The only two settings that can be
considered "not broken" is UTF-8 and UTF-16, since they're the only ones that supports saving to all languages.

Some examples of possible values (the default is UTF-8):
- **UTF-8** - Works with about every imagineable language on Earth, and some extraterrestrial too. Ever feel like subbing something in Klingon?
- **UTF-16** - Similar to, but not the same as UTF-8. Takes more space and is byte-order dependent, and doesn't really have any advantages over UTF-8, except it works in some broken programs (i.e. Microsoft Wordpad and ancient Vobsub versions) that can't handle UTF-8 properly.
- **local** - Saves to whatever your system default charset is. **Not recommended**, since your system default charset is never Unicode and will not handle languages other than your system default properly.
- **Shift_JIS** - Japanese charset. **Not recommended**, since it only works correctly with Japanese.
- **ISO-8859-1** - Standard western European charset, also known as Latin-1. Supports most western European languages, including the Scandinavian ones. **Not recommended**, for the same reasons as above.
- **US-ASCII** - American Standards Code for Information Interchange. **NEVER USE IT** unless you really, really have to, since it's only 7-bit ASCII! I.e., about the only language it supports is American English, and that with a very limited set of accents etc..

```
select action=NUM
select condition=NUM
select field=NUM
select match case=BOOL
select mode=NUM
select text=STRING
```

These are all saved settings from the "Select lines" dialog. Use that to change them instead of editing config.dat.

```
shift times all rows=BOOL
shift times bytime=BOOL
shift times direction=BOOL
shift times length=NUM
shift times type=NUM
```

These are all saved settings from the "Shift times" dialog. Use that dialogue to change them instead of editing config.dat.

```
show associations=BOOL
```

If true, Aegisub will popup a file association dialog with options to associate Aegisub with .ass, .ssa and .srt on startup, and then set this setting to false. Useful if something else stole the file associations and you want them back.

```
show keyframes on video slider=BOOL
```

When true, and a video file from which Aegisub knows how to get keyframe information from (i.e. an AVI) is loaded, keyframes will get displayed as grey lines on the video slider. Defaults to enabled (1).

```
show splash=BOOL
```

When true, Aegisub will show its splash screen on startup. Defaults to true (1).

```
sync video with subs=BOOL
```

When true, Aegisub will seek the video to the line start when you select a line in the
grid. Defaults to true (1).

**syntax highlight brackets=HEX**
The color the curly brackets will take if syntax highlighting is on. Defaults to #1432FF.

**syntax highlight enabled=BOOL**
Enables or disables syntax highlighting. The colors are defined by the "syntax highlight X" settings. Defaults to enabled (true).

**syntax highlight error=HEX**
The color a malformed tag detected by the syntax highlighter will get. Defaults to #C80000.

**syntax highlight normal=HEX**
The color all normal text (outside tags) will get when the syntax highlighter is on. Defaults to #000000.

**syntax highlight slashes=HEX**
The color all backslashes (ASS/SSA overrides) will get when the syntax highlighter is on. Defaults to #FF00C8.

**syntax highlight tags=HEX**
The color all text inside curly brackets (except backslashes) will get when the syntax highlighter is on. Defaults to #5A5A5A.

**text actor separator=STRING**
Helps determine Aegisub's behavior when importing a plain text file to subtitles. It defines what string separates the actor from the text he speaks. Defaults to : (a colon). A more detailed explanation of this setting as well as the text comment starter setting can be found under the heading "Importing plain text files" in the Working with subtitles section.

**text comment starter=STRING**
Helps determine Aegisub's behavior when importing a plain text file to subtitles. All lines prefixed by this string will be imported into the script as comment lines. Defaults to # (a "hash" or "square" character).

**threaded video=BOOL**
Enables/disables an EXPERIMENTAL threaded video mode, which places the video in its own thread, making the interface more responsive while seeking in the video. So far a value of 0 (disabled) does not end the world, whereas a value of 1 usually causes Aegisub to die a fiery death. Will be more useful in the future.

**timing default duration=NUM**
The default duration a new line will have, in milliseconds. Also affects the way SSA mode behaves when going to the next line. Defaults to 2000.

**timing processor adjacent bias=NUM**
**timing processor adjacent thres=NUM**
**timing processor enable adjacent=BOOL**
**timing processor enable keyframe=BOOL**
**timing processor enable lead-in=BOOL**
**timing processor enable lead-out=BOOL**
**timing processor key end after thres=NUM**
**timing processor key end before thres=NUM**
timing processor key start after thres=NUM
timing processor key start before thres=NUM
These are all saved settings from the timing postprocessor dialog. Change them there instead of editing config.dat.

tips current=NUM
The internal number of the startup tip that was last displayed. Used to avoid displaying the same tip two times in a row. Editing it manually should not be necessary.

tips enabled=BOOL
Enables/disables startup tips. Since they can be turned off from the tips dialog, there's no real reason to change this value. Defaults to true (1).

undo levels=NUM
The maximum amount of undo levels. While a higher value might be useful in some cases, it will also consume more RAM, especially for complicated files like large karaoke files (which can eat RAM at an amazing rate). The default (and recommended) value is 8. Setting it to 0 disables undoing, and while this normally is not a good idea, it might be useful if you have huge (several MB) karaoke files that make Aegisub eat all your RAM on loading.

use nonstandard millisecond times=BOOL
Setting this to true makes Aegisub handle ASS times with millisecond precision instead of the normal centiseconds. Defaults to false (0). Do NOT change this setting unless you know exactly what you're doing! Normal VSFilter versions (including the one distributed with Aegisub) do not handle millisecond times correctly!

video check script res=NUM
Controls what is done when a new video source is loaded, and the resolution of the video doesn't match the resolution set for the subtitles. It can take one of the following values:
• 0 - Never change the subtitles resolution (the default behavior)
• 1 - Ask if the subtitles resolution should be changed
• 2 - Always change the subtitles resolution without asking

video default zoom=NUM
The default zoom level of the video window. Higher numbers mean bigger video and vice versa. Defaults to 7 (100%). Note that this number isn't directly related to the percentage level of the zoom - it's Aegisub's internal representation of the zoom level.

video fast jump step=NUM
The number of frames that will be seeked when you press Alt+Left or Alt+Right with the video seek bar in focus. Defaults to 10.

video provider=STRING
Determines which method is used to load video. This is a part of the ongoing work to port Aegisub to POSIX systems (such as Linux and BSD) - wxWidgets already work, but Avisynth does not, which means that video and audio does not work either. At the moment, the only valid option is Avisynth, since the alternatives are disabled in the current release versions. In the future you might be able to set this to ffmpeg and expect video to work on both POSIX and Windows, but for now, changing this option will do nothing.

video resizer=STRING
The algorithm used to resize the video display. Any Avisynth resizer that takes arguments in the \texttt{Resizer(x\textunderscore destination, y\textunderscore destination)} form will work - even ones contained in plugins as long as you have them in your autoloading directory (not recommended, though, and very pointless). Valid Avisynth 2.56 built-in resizers are:

- \texttt{PointResize} - very fast, but very ugly (nearest neighbor algorithm)
- \texttt{BilinearResize} - Quite fast, smooths the image a bit, especially when upsizing
- \texttt{BicubicResize} - Slower but sharper than bilinear
- \texttt{LanczosResize} - Very sharp and very precise, but also slower than bicubic
- \texttt{Lanczos4Resize} - The sharpest, but also the slowest algorithm
- \texttt{Spline16Resize} and \texttt{Spline36Resize} - Spline-based resizers. 16 is faster than 36.
- \texttt{GaussResize} - Gaussian resizer. About as fast as \texttt{Lanczos4Resize}. Note that the "p" (blurring/sharpening) parameter can't be changed in Aegisub.

The default (and recommended) resizer is \texttt{BilinearResize}. If you have a very slow computer, you can try \texttt{PointResize}, but the others are probably more or less pointless unless you plan on doing resizer tests with Aegisub.

### 2.6 Attaching Subtitles to Video

In digital encoding, there are two main ways of including subtitles in a video. There is softsubbing, and hardsubbing. Both methods has unique advantages and disadvantages, along with various arguments both for and against each method.

#### Hardsubbing

Hardsubbing is a method that embeds subtitles into the actual video portion of a movie. Digital hardsubs are much like subtitled VHS files, the subtitles cannot be turned off.

#### Advantages of Hardsubbing

Hardsubbing is usually much less demanding on the playback device. Since the text is already part of the video, it will only take as much processing as the unsubtitled video would. You are also often able to make special effects that would be difficult to replicate in a soft subtitle format, because of the large amount of CPU usage required to render them. Even in softsubbed anime fansubs, the opening and closing karaoke are often hardsubbed because of the special effects used.

Some people argue that with hardsubs, scripts are harder to steal, since the text is embedded in the image - thieves cannot simply extract subtitles as in a softsub. However, the presence of very good subtitle extractors designed for the purpose of extracting this embedded text removes much of the argument that hardsubs prevent script stealing.

Many playback devices and computer platforms cannot display the special fonts and formatings that softsubs contain, but this problem is removed with hardsubs, where the style is preserved. Also, these stylings will show back exactly the same on any device, unlike softsubs which depend on the playback device to properly intrepret and display the stylings.

#### Disadvantages of Hardsubbing
Despite what some may call numerous advantages for hardsubbing, there are several distinct disadvantages that should be evaluated before making a decision.

The method of hardsubbing requires that the source video is re-encoded so the subtitles can be written on the image. This, by the nature of lossy video encoding, causes a reduction in video quality.

Subtitles add a sharp contrast in a video image due to their nature. This will cause compression artifacts along the edges of the encoded subtitle, and blurring of the subtitle. This effect is especially evident at lower bitrates.

In some circumstances, the inclusion of the subtitles will cause an increase in the bitrate needed for the video to keep the same quality. This, of course, means an increased filesize, or lower quality at the same size.

Changing the subtitles requires a re-encode of the video source, which can add a lot of time and extra work to the release process.

**Softsubbing**

Softsubbing is a method that keeps subtitles separate from the video and relies on the playback device to combine the two when the video is being played. This method can be best compared to subtitles on most DVDs. The subtitling can be turned on or off as needed, and multiple languages can be supported with just one combined media file. Unlike with a DVD though, digital softsubs are actually text (DVD subtitles are pictures) which adds many nice features at the cost of complexity.

**Advantages of Softsubbing**

Softsubs are much clearer on display. Since they are not part of the video image, video compression does not affect them, and with a good subtitle renderer, they are sharp and crisp - a huge benefit to readability.

Softsubs can be smaller. Since the subtitle is just a text file, it can take up less room because it isn’t hogging video bitrate. This allows for an encoder to either make a smaller file with the same video quality, or a same-sized file with higher video quality.

People with vision problems have an opportunity to adjust how the subtitles look on-screen.

Without a huge impact on size, multiple languages can be supported in one video file.

If you find a subtitling mistake in a file, you can fix it without having to re-encode the video - saving a lot of time.

**Disadvantages of Softsubbing**

Softsubs add processing complexity to the video. The playback device has to render and overlay the text before displaying the video, as a result, this means that low-powered
devices will not be able to play the video.

Since the subtitles are bundled as straight text they are very easy to extract and use. This makes things easier on bootleggers or other script stealers. Note that grabbing subtitles from a hardsub is very easy currently, so this argument doesn't hold much weight.

The playback device is responsible for rendering the subtitles on screen. As a result, they might not look the same as the subtitler intended. In some cases, the playback device might not support the subtitle format, or might have bugs with it.

The AVI file format is not reliable for supporting softsubs, if you plan to use it. Please note that Matroska (MKV) is very well supported by computers, so this isn't a major negative.

Subtitles with effects added (usually for karaoke) take up a lot of processing time, and may cause playback issues if the device cannot handle the processing requirements. A solution for this is to hardsub the complex parts such as opening and ending karaoke, and softsub the normal dialog.

What method do I choose?

The method you should choose depends greatly on your audience. Will they have relatively new and powerful playback devices? Will they possibly be able to install something to play back softsubs if they don't have it? Is your destination a digital format (Matroska, DVD, etc.) or will you be printing to tape?

While every situation will be different, you can use some of the following suggestions to guide you. These are based on making a digital format for playback on a computer system.

- If you want your file playable on the largest range of computers and operating systems, you will want to hardsub.
- If your audience will have relatively new machines (500 MHz or greater) and will be running on a platform where your subtitle format is well-supported, softsubs are a good idea.
- If you want to speed up your release process, use softsubs. They are faster to fix if an error is found.

Hardsubbing with AVISynth

Many people use the AVISynth package to add filters to their video to clean up defects, or otherwise manipulate the video image before encoding it. It is a very flexible tool, and can be also used to add subtitles directly to the video stream, allowing an easy and scriptable method to hardsub a video.

If you are unfamiliar with AVISynth, it is recommended that you look into it, as it has lots of nice features and a large community contributing video filters, allowing easy video fixes for any source. This tutorial assumes you have some basic knowledge of AVISynth.

To allow adding subtitles to the video stream, you need the VSFilte plugin. You can find
this at Gabest's [Sourceforge project page](https://sourceforge.net), or in your Aegisub directory (it's called vsfilter.dll). Once you have gotten hold of vsfilter.dll, place it in your AVISynth plugins folder.

To just add subtitles, you will want to make a simple AVS file containing the script lines you need. Here is an example:

```plaintext
AVISource("c:\projects\project1\video\mycoolvideo.avi")
TextSub("c:\projects\project1\subs\mainsubtitles.ass")
TextSub("c:\projects\project1\subs\endkaraoke.ass")
```

The above script will take an AVI file (mycoolvideo.avi), and then draw the contents of two subtitle files on the video. You can then encode this video in any program that supports AVS, such as VirtualDub.

**Hardsubbing with VirtualDub**

If you're already familiar with VirtualDub filters, and don't intend to do any other video processing, you should note that it's possible to use VSFilter as a VirtualDub filter as well. Just rename the .dll to .vdf and copy it to the VirtualDub plugins folder.

**Softsubbing**

Softsubbing a video can be done in several ways, all of which require that VSFilter is installed to view the subtitles.

- **Distributing script files.** This method works best when you want to encode the video in an AVI wrapper. You simply send the raw subtitle files along with the video. The viewer then needs to load them in a player that supports external subtitles. When using this method, you either need to make sure you use fonts that everyone can be expected to have installed, or distribute a separate ZIP archive with the fonts.
- **Softsubs inside the video container.** Matroska (MKV) is currently the best container for this method (MP4, OGM and even AVI can technically contain softsubs, but none supports font attachments, and all of them has various other issues). Using a muxer that supports attachments (i.e. mkvmerge GUI), you simply add your subtitle files to the Matroska file as separate tracks (just like you add audio and video tracks), and any fonts as attachments (make sure they have the MIME type `application/x-truetype-font`). The fonts will then be installed temporarily by Haali Media Splitter (on Windows) or MPlayer (on *nix and MacOS X) during playback.
Top Level Intro

This page is printed before a new top-level chapter starts
3 Typesetting

If you're new to typesetting, it's recommended that you start by reading the Introduction, followed by the section on Editing styles and the Tutorial.

3.1 Introduction

What is typesetting?

The proper definition of typesetting does not exactly match how it's often used when it comes to using ASS subtitles, especially amongst the fansub community. From Wikipedia:

"Typesetting involves the presentation of textual material in an aesthetic form on paper or some other media."

The first goal of typesetting is to define styles for the subtitles, so they are easy to read, visually interesting and match the video. See the example below:

In the first image, you see how the subtitles normally look. In the second one, you can notice how the hue was shifted to green, so it better matches that particular scene.
The second goal is to animate special effects (often called "flying objects" or "f.o.s") to translate signs and other text in video by having the subtitles for them "blend" with it. For example:

In the first image, the original video only had the Japanese kanji and furigana in it. The translation ("Suki - Chance") was added by the typesetter, in such a way that it looks like it was always there. This is the holy grail of typesetting: translating so that it looks like the translation was originally there.

The second example illustrates the same - the English translation was added afterwards by the typesetter.

### 3.2 Editing styles

This section will give you the basics about styling and how to do it with Aegisub.

**Something About Styling**

Styles are meant to improve subtitle aesthetics, match the video, and most importantly, be easy to read.
The most common method is to have two styles for the subtitles. One is the standard style and the other is the alternative style, the later is used when two characters talk at the same time.

Sometimes a typesetter uses more than two styles (often called "multicolor subtitles"). In most cases, different colors are used for the more important characters, but there are other criteria on how to style it.

This screenshot represents the use of two styles, and because there are two lines at once (a collision) both styles are used at the same time. This is the most common way to style subtitles. From the screenshot of the lines displayed, it's easy to see the different styles applied to the subtitles.
As seen on these two screenshots, the typesetter used different styles for the characters. This makes the subtitles more interesting and sometimes when there are more characters speaking it can help to make the subtitles easier to read. Looking at the
screenshot of the lines, it's easy to see the different styles used from how they are named (after the character names, which makes it easier to apply them).

Aegisub and Styles

In Aegisub there is a Style Manager for editing styles. It's meant to make editing, creating and managing the styles easier. To open it, simply go to Tools -> Styles Manager.

The styles manager window is simple. It's meant for selecting styles for a subtitle script, as well as creating new styles and storing them. For storing the styles, Aegisub uses the Catalog. Styles aren't stored directly in the catalog, but inside of "Storage". That makes storing easier and more accessible for the typesetter.

Next is the Storage window. It displays the styles available in the selected Storage from the Catalog. The Current Script Window displays the styles used in the currently loaded script.

The typesetter can add, delete and edit the styles from both the Storage and the Current style window, as well as move them from window to window.
This window shows the style creator and editor. Here the typesetter sets the parameters as he wants them.

Text displayed in [] displays the code for the parameters as it's used in line editing. 
Example: Font [\fs].

- **Style Name**: Name of the style. Name it something simple and concise to make life a lot easier.
- **Fontname and Fontsize**: Font [\fn<name>] used and the size [\fs<size>], then the other font parameters (like Italic, Bold, etc.).
- **Colors**: Parameters for the different colors [\<1,2,3,4>c&H<bbggrr>&](primary, secondary, etc). Transparency is for the colors as well with [\<1,2,3,4>a&H<aa>&], which goes from 0 (opaque) to 255 (transparent).
- **Margins**: 3 parameters: left, right and vertical. Values are in pixels, and represent the distance from the edge of video to the subtitle position.
- **Alignment [\an<alignment>]**: Alignment of the subtitle follows the numpad layout (e.g. 1 is lower left, 6 is middle right, 8 is top middle, etc.).
- **Outline [\bord<width>]** sets the width of the border (in pixels) around text and Shadow [\shad<depth>] sets the offset (in pixels as well) that the shadow is drawn from the subtitle.
- **Opaque Box**: Checking this will make a filled, opaque rectangle behind the subtitles instead of the usual border.
• **Scale X/Scale Y**: \[\fscx<\text{percent}>\] and \[\fscy<\text{percent}>\] set the font scale in the X and Y axis. Parameter should be in percent form.

• **Angle**: \[\frz<\text{degrees}>\]: Degree rotation of the subtitle in the Z axis. Value in degrees.

• **Encoding**: \[\fe<\text{charset}>\]: The character encoding can be specified. Example: 0=ASCII, 128=Japanese. This is ignored when Unicode is in use.

**Note:** The code that was inside of [] is further explained in the [ASS Override Tags](#) page.

### 3.3 Tutorial

#### Introduction

Welcome to Aegisub's typesetting tutorial. This will give you step-by-step instructions on how to make a basic typeset effect, and hopefully this knowledge will be enough to get you started. Only a few Advanced Substation Alpha (ASS) tags will be covered in this article, for information on the rest of them, please see the [ASS Override Tags](#) page.

**Note:** This tutorial assumes you've read the [Introduction](#) and [Editing styles](#) pages.

#### Tutorial

For this lesson, we will be typesetting a simple, static sign. It says "Shuu-chan" in Japanese, and we want to have the translation right next to it.

Before you begin, please download the tutorial package from [here](#) (~1 MB). It includes the video for the tutorial, as well as a blank script and a script with the final result.

Start by opening the "aegisub_typeset_tut.ass" script included in the tutorial package. The script that we are using is actually 800x600, which is totally different from the video resolution (704x396) - even Aspect Ratio is off. This is to illustrate the importance of trusting script resolution, not video resolution.
Now, go to the Video menu and open the "aegisub_typeset_tut.avi" file.

Insert a new line, or select the existing blank line, and enter "Shuu-chan" as its text.

Now, time it exactly to the scene boundaries. It's important that the sign starts and ends at the exact proper frames, to prevent annoying visual artifacts ("bleeding").

This is easily done with Aegisub. First, make sure the line is selected. Now, seek the video to the very first frame in scene. You can use the Left Arrow key to go back one frame and make sure that you HAVE picked the first frame. Once you're in the right frame, hit Ctrl+3. This will set start of selected lines so they begin at the video frame.

Hint: Whenever you are using an AVI video with keyframe information, you can use shift+left/right to jump to keyframes, which are usually placed at scene transitions. When you are viewing a keyframe, the editbox with the current time and frame number will turn green.

At this point, you may or may not see the subs on video (depending on where is the end time set to). It doesn't matter. Seek the video to the very last frame in scene and hit Ctrl+4. This will set end of subtitles to current frame, and now you should see the subtitles throughout the entire scene (but not anywhere else).

This is what it should look like:

Now it's time to actually start typesetting. For starters, we will remove the border and shadow from the line, and center it on screen. To do so, we will use the following ASS override tags:

- \an: Alignment to numpad. We will set it to 5 (centered).
- \bord: Border width. We will set it to 0.
- \shad: Shadow distance. We will set it to 0.

Inserting the tags in the line will make it look like this:
\{\texttt{an5\bord0\shad0}\}Shuu-chan

You can now hit $Ctrl+Enter$ to preview changes without changing line. It should look like this:

This is still far from what we want. Next, we will change the color and rotation to match the text. We will use the following tags:

- `\texttt{c}`: Sets primary color. We will use &H383B3C&.
- `\texttt{frz}`: Rotation on Z axis. We will use -10.

To set the color, you can either manually enter the tag, or click the "set primary color" button over the subtitles edit box. At any rate, the line should now look like:

\{\texttt{an5\bord0\shad0\1c&H383B3C&\frz-10}\}Shuu-chan

Hitting $Ctrl+Enter$ for preview should give you:
We're getting closer, but it's still totally mispositioned. We will now use Aegisub's visual typesetting facilities to position the line. Since we are using \an5, the position which we specify will determine where the CENTER of subtitles will be. Move the mouse cursor over video display. It should show a crosshair with coordinates of current position, like in the picture below:

Since 453,164 looks like a good position, double click it to set the position there. The line should now read:
{\an5\bord0\shad0\1c&H383B3C&\frz-10\pos(453,164)}Shuu-chan

And the video should look like this:
Finally, the font is small and doesn't really match sign. Click the "Font Face Name" button to choose a font (Ashley Script MT, in this case). Hit Ctrl+Enter to preview. Seeing it's too small, use the \fs tag to increase its size to 80. Also, the rotation seems a bit off, so I adjusted it to -12. The final line should look like:
{\an5\bord0\shad0\1c&H383B3C&\frz-12\pos(453,164)\fnAshley Script MT\fs80}Shuu-chan

And, in the video:

Though this is a relatively simple sign, with no animation or more advanced features, it's a good starting example. From this, you should be able to figure how to use the remaining ASS Override Tags to create more advanced effects.
Also included in the tutorial pack is a "aegisub_typeset_tut_solved.ass" file, which shows what your file should look like. In particular, check if your start/end times match those in the "aegisub_typeset_tut_solved.ass" file. The line should start on frame 76 (0:00:03.169) and end on frame 152 (0:00:06.339).

3.4 ASS Override Tags

The following is a list of every tag supported by the Advanced Substation Alpha format. This is basically a detailed version of ass-quickref.txt. See the Tutorial for an introduction to typesetting, using some basic tags.

Special characters

The following tags are written in the middle of the text, and not inside override blocks (i.e. not between { and }).

\n - Soft line break.
  Makes the line "break" (i.e. skip to the next line) using mode \q2 (see \q<0-3>). Using any other mode, it's replaced by a space.

\N - Hard line break.
  This is always replaced by a line break.

\h - Hard space.
  This is useful when you need several spaces in a row.

Override tags

These tags must be in an override block, that is, they must be between a { and a }, and will affect all text following the block, unless they get re-overridden.

\i<0/1> - Italic
  Toggles italics in following text. Set to 0 to disable italics or to 1 to enable them.
  e.g.:
  • \i1 (enable italics)
  • \i0 (disable italics)

\b<0/1/weight> - Bold
  Toggles bold in the following text. Set to 0 to disable bold or to 1 to enable. TODO: Does VSFilter properly support "weight"? How does it work?

\u<0/1> - Underline
  Toggles underline in the following text. See description for italics.

\s<0/1> - Strikeout
  Toggles strikeout in the following text. See description for italics. Strikeout means that there will be a line crossing the text.

\bord<width> - Border
  Sets the width of border around text. Value is given in pixels and can have decimal places.
e.g.:
- `\bord1`
- `\bord0.5`
- `\bord2.3`
- `\bord0` (this will disable the border)

`\shad<depth>` - Shadow
Sets the offset that shadow is drawn from the main text. Think of the shadow as the text+outlines in a single color, drawn behind text, and offset by `depth` pixels to the right and bottom. This tag works similarly to `\bord`.

`\be<0/1>` - Blur edges
This tag enables or disables edge blurring. This is a subtle effect, and is not very visible for large text. See description of italics tag for usage.

`\fn<name>` - Font name
This sets the font face name for the following text. Note that there is no space between `\fn` and the parameter name.
e.g.:
- `\fnArial` (uses "Arial" font)
- `\fnTimes New Roman` (uses "Times New Roman" font)

`\fs<size>` - Font size
Specifies the font size, in points. This is the same unit that most text processors use.
e.g.:
- `\fs10` (uses size 10 for font)

`\fscx<percent>` - Font scale in X axis
Sets the font scale for the X (horizontal) axis. Parameter should be in percent form. This is different from simply adjusting size using `\fs` because `\fs` will use font hinting to determine best looking adjustments, especially for small sizes. Also, `\fs` will only accept integer parameters. If you need to, for example, transform scale (with `\t` tag), use this instead of `\fs`, as `\fs` will behave strangely and not smoothly.
e.g.:
- `\fscx150` (makes the text 50% wider)
- `\fscx33` (makes the text have roughly a third of its original width)

`\fscy<percent>` - Font scale in Y axis
Same as `\fscx`, but for the vertical axis.

`\fsp<pixels>` - Font spacing
Sets the spacing between characters, in pixels. This allows you to make the text look something like "t h i s". Some fonts might look bad because the characters are too close to each other. This tag might help in such cases. Also, if used in moderation, it might give the illusion that the font is wider.

`\fr[<x/y/z>]<degrees>` - Font rotation
Rotates the text by the specified number of degrees, in the specified axis. If axis is omitted, then "z" is used as default. The axes are as follows:
- x: this axis runs horizontally on the screen. Rotating on it (with positive values) causes an effect where the top of the text moves farther "into" the screen while the bottom moves "out" of the screen.
- y: this axis runs vertically on the screen. Rotating on it (with positive values) causes
the text to rotate so that the left moves "outside" the screen, when the right moves "into" the screen.

- **z**: this axis runs perpendicular to the screen. Rotating on it (with positive values) causes the text to rotate in 2D, counterclockwise (as standard for degrees).

  In this system, 360 degrees is one full rotation. You can have negative or above 360 values, which get wrapped around. Decimal places are also acceptable.

  The "pivot" point of rotation is defaulted to its alignment. That means that a text with \an5 will have its pivot on the center, so rotation will be relative to it. A text with \an7, on the other hand, will rotate relative to its top-left corner, that is, the whole text will rotate around that point, while that stays still. This can be set with the \org tag, see its description for more details.

  e.g.:

  - \frx45 (rotates text on the X axis by 45 degrees)
  - \fry-45 (rotates text on the Y axis by -45 degrees)
  - \frz180 (rotates text on the Z axis by 180 degrees)
  - \fr90 (same as \frz90, makes the text "vertical")
  - \frz-0.5 (rotates text on Z axis by half a degree, negative)
  - \t(\fry3600) (animates the text to do 10 full revolutions on Y axis. See \t for its description)

Here are three sample screenshots:
Font encoding

For non-unicode subtitles, this sets the character encoding to the specified page number.

- \fe0 (sets to ANSI extended, the standard for the USA)
- \fe128 (sets to SHIFT_JIS extended, the standard for Japan)

Set primary color

Sets the primary color (i.e. the fill color) of the text. Color tags in ASS are in Visual Basic hexadecimal style, which is similar to HTML hexadecimal. It always starts with &H, and ends with &. Between those, are the color values in hexadecimal: two characters for blue, two for green and two for red, in that order (i.e. BBGRR). Any zero color at the start may be omitted. Also note that saying just \c has the same result as saying \1c.

- \c&HFF0000& (Full intensity (FF = 255) blue and zero intensity green and red, making it blue)
- \c&H00FF00& or \c&HFF00& (Full intensity green and zero intensity blue and red, making it green)
- \c&H0000FF& or \c&HFF& (Full intensity red and zero intensity blue and green, making it red)
- \c&HFFFFFFFF& (white)
- \c&HFFFF00& or \c&H2D5F2& (magenta)
- \c&H2DD5F2& (gold)

Aegisub now has a Color Picker that can help you generate these hexadecimal numbers.

Set secondary color

Sets secondary color. This is the color that primary changes from during karaoke effects (i.e. karaoke starts at this color and then changes to primary). See \c for full description.

Set outline color

Sets outline color. See \c for full description.

Set shadow color

Sets shadow color. See \c for full description.

Set primary alpha

© 2005 Rodrigo Braz Monteiro. All rights reserved.
This sets the alpha value (i.e. the transparency) of the primary color. This value is a
two-digit hexadecimal number, ranging from 00 (opaque) to FF (invisible).
e.g.:
- \alpha&H80& (makes primary 50% translucent)
- \alpha&HEF& (makes primary almost invisible)
This is the same as \fa.

\<1/2/3/4>a&H<aa>& - Set Primary/secondary/outline/shadow alpha
Sets alpha value for the specified color. Works the same as the variations of \c. See \alpha for a complete description.

\an<alignment> - Numpad Text Alignment
Sets text alignment relative to numpad layout.
e.g.:
- \an7 (aligns text on top-left. Note that the "7" key on numpad is at the top-left)
- \an5 (aligns text on center)
- \an2 (aligns text on bottom-center)

\a<alignment> - SSA Text Alignment
Sets text alignment based on the old SSA scheme. This is specified by a sum: 1 equals
left, 2 equals middle, 3 equals right. Then add 4 to get top alignment, 8 to get vertical
middle alignment, or leave as is for bottom alignment.
e.g.:
- \a5 (aligns text on top-left, equivalent to \an7)
- \a2 (aligns text on bottom-middle, equivalent to \an2)
- \a11 (aligns text to middle-right, equivalent to \an6)
You should normally use \an instead, as it is more intuitive.

\k<duration> - Karaoke effect
This makes the following text start at the secondary color immediately, and change to
the primary color after the sum of duration (in centiseconds) of all previous karaoke
effects has elapsed.
e.g.:
- \{\k32\}Ko\{\k27\}ko\{\k27\}ro (Shows "Ko" with the primary color immediately and other
two syllables with the secondary, then "ko" changes to primary 32 centiseconds (320
milliseconds or 0.32 seconds) later, then "ro" changes to primary 27 centiseconds
after)

\kf<duration> - Filled karaoke
Similar to \k, but instead of immediately setting color from secondary to primary, it "fills"
it from left to right.

\K<duration> - Filled karaoke
Same as \kf.

\ko<duration> - Filled outline karaoke
Similar to \kf, but applied to the outline.

\q<0-3> - Wrap style
Sets the line wrapping style as follows:
- 0: Smart wrapping, top line is wider
- 1: End-of-line word wrapping, only \N breaks
- 2: No word wrapping, both \n and \N break
• 3: Smart wrapping, bottom line is wider
e.g.:
• \q1 (Following text will only acknowledge \N as line breaks; \n will become a simple space)

\r[<name>] - Reset style
This resets the style to the specified one. If none is given, it resets to this line’s original style.
e.g.:
• -Hey\N{\rAlternate}-Huh?\N{\r}-Who are you? (this has the middle line in the style "Alternate", while first and third are in the originally set style)

\pos(<x>,<y>) - Position
Positions the subtitle at the specified coordinates. Coordinates start at 0,0 for the top-left corner and go to ResX,ResY at the bottom-right corner, where ResX and ResY are defined in the script header (you can set this value in the properties window). Coordinates do not necessarily match video coordinates: if the script resolution is set to 640x480, then doing a \pos(320,240) on a center-aligned text will ALWAYS put it on the center of video, even if it is, for example, 704x396.
You may only have one \pos command in each line. If you have more, only the first one has any effect.
Note that \pos is affected the pivot of subtitles. For example, if the current line has centered alignment (\an5), then the center of line will be placed at that position; if it’s top-left (\an7), then the top-left corner of line will be placed at that position, etc.
e.g.:
• \pos(100,200) (Positions the line at video position x=100 and y=200)
Here follow three examples of \pos(320,240) on a 640x480 resolution script. The only difference between them is the alignment, note how the pivot (represented by a green crosshair) is always at 320,240.
\move(<x1>,<y1>,<x2>,<y2>[,<t1>,<t2>]) - Move
Moves the text from the original position, defined by x1,y1, to the destination, defined by x2,y2. This means that at the very start of animation, the subtitles will look the same as if you had done a \pos(x1,y1), and, at the very end, the same as \pos(x2,y2). If the parameters t1 and t2 are specified, then the subs will start at x1,y1, but won't start moving to x2,y2 until time reaches t1. At that point, they will start moving, and end at t2. t1 and t2 are specified in milliseconds, relative to the start time of current line. If t1
and \t2 are omitted, animation is assumed to last for the whole length of line. You may only have one \move command in each line. If you have more, only the first one has any effect.

\move(100,150,300,350) (Moves from 100,150 to 300,350, lasting whole line duration)
\move(100,150,300,350,500,1500) (Same as before, but it starts 500 ms (that is, half a second) after line appears, and ends one second later (that is, 1.5 seconds after subtitles started))
\org(<x>,<y>) - Origin
Sets the screen coordinates of the origin (or pivot point) for rotation and scaling tags. This can be useful for some relatively advanced effects. Suppose that you have a spinning roulette in the video, and you want some text to follow it. You can set the origin to the center position of roulette, and then animate a \frz with \t, so the text spins around that position. Origin can also be set outside video, which means that you can set it very far away, and make minor adjustments to rotation to give text a "jumping" effect. You may only have one \org command in each line. If you have more, only the first one has any effect.
\org(320,240) (Sets origin at 320,240)
\org(10000,0) (Sets origin at 10000,0, which might be useful for jumping effects)
\fade(<a1>,<a2>,<a3>,<t1>,<t2>,<t3>,<t4>) - Fade
Fades text. This is done in two steps. The text starts with alpha a1 (as if set by \alpha), and at time t1, it will start fading to a2. It ends fading to a2 at time t2. At time t3, it starts fading to value a3, and finally ends on time t4. Times are given in milliseconds relative to start of line, in the same way as \move. Transparency values are given in hexadecimal, in the same way as \alpha. Note that this tag affects all four alphas.
\fade(&HFF&,&H20&,&HE0&,0,500,2000,2200) (Starts invisible, fades to almost totally opaque, then fades to almost totally invisible. First fade starts when the line starts and lasts 500 milliseconds. Second fade starts 1500 milliseconds later, and lasts 200 milliseconds)
\fad(<t1>,<t2>) - Fade (simplified)
A less powerful, but more practical version of \fade. It makes line start invisible, then immediately start fading to full visibility for t1 milliseconds. It will start fading to invisible t2 milliseconds before the END of line, and become invisible just as the line vanishes. It's important to notice that this is the only ASS override tag that has a time reference relative to end of line. It is also equivalent to \fade(&HFF&,&H00&,&HFF&,0,t1,LEN-t2,LEN), where LEN is the length of current line in milliseconds.
\fad(500,500) (Fades text in for 500 milliseconds, then fades out for 500 milliseconds at the end)
\fad(500,0) (Fades text in for 500 milliseconds [does not fade out])
\t([<t1>,<t2>,][<accel>,][<style modifiers>]) - Transform
This is perhaps the most powerful override tag available. It does nothing by itself, instead, it animates (transforms) other tags. The animation will start at t1 milliseconds and end at t2 milliseconds, in much the same way as \move. If those parameters are ommitted, then start and end of the line are assumed, as usual.
The optional acceleration parameter allows the transformation to be alinear. Think of it as the exponent on a $x^n$. Acceleration = 1 is linear. Acceleration between 0 and 1 starts fast and ends slow. Acceleration larger than 1 starts slower and ends fast.

The following tags are supported by $\texttt{\textbackslash t}$:

- $\texttt{\textbackslash c}$
- $\texttt{\textbackslash 1-4c}$
- $\texttt{\textbackslash a}$
- $\texttt{\textbackslash 1-4a}$
- $\texttt{\textbackslash f}$
- $\texttt{\textbackslash fr}$
- $\texttt{\textbackslash frx}$
- $\texttt{\textbackslash fry}$
- $\texttt{\textbackslash fz}$
- $\texttt{\textbackslash fsx}$
- $\texttt{\textbackslash fsy}$
- $\texttt{\textbackslash fps}$
- $\texttt{\textbackslash bord}$
- $\texttt{\textbackslash shad}$
- $\texttt{\textbackslash clip}$ (rectangle version only).

E.g.:

- $\texttt{\{\textbackslash c&H0000FF\textbackslash t\{\textbackslash c&HFF0000\}\}\text{Hello!}$ (Starts text at a red colour and fades it to blue)
- $\texttt{\{\textbackslash an5\textbackslash t\{0,5000,\textbackslash frz3600\}\}\text{Woo!}$ (Makes the text rotate 10 times, counterclockwise, lasting for 5 seconds)
- $\texttt{\{\textbackslash an5\textbackslash t\{0,5000,0.5,\textbackslash frz3600\}\}\text{Woo!}$ (Same as before, but it will start fast and slow down, still doing the 10 rotations in 5 seconds)

$\texttt{\textbackslash clip(}$<$x1>$<$y1>$<$x2>$<$y2>$)$ - Set clipping rectangle

This sets the clipping rectangle of the text, so that it will ONLY be drawn INSIDE this rectangle. For example, if you had a resolution of 640x480 and set $\texttt{\textbackslash clip(0,0,320,240)}$, then text would only be drawn on top-left quadrant - anything outside it would be "clipped" and would be invisible. This tag is useful for when some object moves in front of text: you can use $\texttt{\textbackslash t}$ to animate the "window" where the text is visible.

E.g.:

- $\texttt{\textbackslash clip(0,0,320,240)}$ (Areas of text will only be drawn if they are on top-left quadrant)
- $\texttt{\textbackslash clip(100,300,540,480)}$ (Areas of text will only be drawn if they are on center-bottom)

Sample screenshot (with $\texttt{\textbackslash clip(0,0,704,245)}$ on a 704x480 video):
\[\text{\textbackslash clip([<scale>,<drawings>])}\] - Set clipping vector
Same as before, but it uses drawing vectors instead of a rectangle. See the drawing section below for more information. Note that, unlike the previous tag, this may NOT be animated with \textbackslash t. The optional scale argument is equivalent to the value passed to \textbackslash p on the drawing tag. On an unrelated note, this tag makes me wonder if Gabest was actively trying to prevent other people from writing ASS parsers by creating an overloaded tag.
e.g.:
- \textbackslash clip(1,m 50 0 b 100 0 100 50 100 b 0 100 0 0 50 0) (Limits clip to pseudo-circle defined)

**Note:** VSFilter 2.37 is known to have issues with vector clipping. If you intend to use it, use 2.35 instead (2.36 has another bug, related to border drawing).

**Drawings tags**

Advanced Substation Alpha also supports some advanced drawing tags that allow you to draw with vectorial graphics. Certain familiarity with vectors and splines will make the understanding of this much simpler.

\[\text{\textbackslash p<0/1/...>}\] - Toggles drawing mode
Setting this tag to 1 or above enables drawing mode. Text after this override block will then be interpreted as drawing instructions, and not as actually visible text. Setting this to zero disables drawing mode, restoring normal behavior. When turning on, the value might be any integer larger than zero, and will be interpreted as the scale, in $2^{\text{(value-1)}}$ mode. This is done to allow sub-pixel accuracy.
e.g.:
- \textbackslash p1 (Enables drawing with normal coordinates)
• \p0 (Disables drawing)
• \p2 (Enables drawing, and resolution is doubled. So drawing to 200,200 will actually draw to 100,100)
• \p4 (Enables drawing, and resolution is 8x larger \(2^{(4-1)}\)). So drawing to 400,400 will actually draw to 50,50)

\pbo<y> - Baseline offset
Defines baseline offset for drawing. This is basically an Y offset to all coordinates.
e.g.:
• \pbo-50 (Draws everything 50 pixels above specified)
• \pbo100 (Draws everything 100 pixels below specified)

Drawing commands
These commands should appear either in a \clip tag (vectorial overload) or between \p# and \p0, outside override blocks.
For example (taken straight from the ASS specs):
• Square: {\p1}m 0 0 l 100 0 100 100 0 100{\p0}
• Rounded square: {\p1}m 0 0 s 100 0 100 100 0 100 c{\p0} (c equals to "p 0 0 100 0 100 100" in this case)
• Circle (almost): {\p1}m 50 0 b 100 0 100 100 50 100 0 100 0 50 0{\p0} (note that the 2nd ‘b’ is optional here)

Drawing commands use the primary color for fill and outline color for borders. They also display shadow. The idea of drawing vectors is that there is an invisible "cursor" (think of it as the mouse pointer in a drawing program, or as a pen moving through the image) on the video frame, and you tell it to move to other positions. As it moves, it draws on the area behind it, and when you close the line formed, it fills it with the primary color.

\m <x> <y> - Move
Moves the cursor to \(x,y\). If you have an unclosed shape, it will automatically be closed, as the program assumes that you are now drawing a new, independent shape. All drawing routines must start with this command.

\n <x> <y> - Move (no closing)
Moves the cursor to \(x,y\), without closing the current shape.

\l <x> <y> - Line
Draws a line from the current cursor position to \(x,y\), and moves the cursor there afterwards.

\b <x1> <y1> <x2> <y2> <x3> <y3> - Cubic Bézier curve
Draws a cubic (3rd degree) Bézier curve to point 3, using 1 and 2 as the control points.
Check the article on Wikipedia for more information about Bézier curves. In this picture taken from that article, P0 is the cursor position, P1 is \(x1,y1\), P2 is \(x2,y2\) and P3 is \(x3,y3\):
Note that the curve begins at P0, heads towards P1, then arrives at P3 coming from P2's direction.

\[ s \ <x_1> \ <y_1> \ <x_2> \ <y_2> \ <x_3> \ <y_3> \ .. \ <x_N> \ <y_N> \ - \text{Cubic b-spline} \]

Draws a cubic (3rd degree) uniform b-spline to point N. This must contain at least 3 coordinates (and is, in that case, the same as \( b \)). This basically lets you chain several cubic Bézier curves together. Check this other article on Wikipedia for more information.

\[ p \ <x> \ <y> \ - \text{Extend b-spline} \]

Extends the b-spline to \( x,y \). This is essentially the same as adding another pair of coordinates at the end of \( s \).

\[ c \ - \text{Close b-spline} \]

Closes the b-spline.

### 3.5 Karaoke

Making karaoke effects is a special sub-category of typesetting. It requires extensive knowledge of how to use (and abuse) the different style overrides, a healthy amount of inspiration and/or imagination, and either a LOT of typing or an automated way to generate large to huge amounts of text. Aegisub, or rather the automation module, can help you with the last part, but it requires (at the very least) some thinking and some basic mathematics.

The automation module is immensely powerful, but if you are looking for something like "HOW I MAKE KARAOKE EFFECT FAST!?!", writing entirely new automation scripts in Lua is probably not the path for you. Start with the Automation tutorials on the wiki. Make sure you have some timed karaoke to work with first.

Questions regarding the karaoke scripts and the automation module should be posted on the Aegisub typesetting forum.
Top Level Intro

This page is printed before a new top-level chapter starts
4 Tools

This section outlines the various helpful tools in Aegisub.

- Color Picker
- Timing Postprocessor
- Translation Assistant
- Styling Assistant
- Fonts Collector

4.1 Attachment Manager

The attachment manager allows you to attach fonts and/or pictures to your script (by encoding them as text). This is especially useful for sharing fonts between everyone who is working on a script without having to send the fonts as separate files, for example.

<table>
<thead>
<tr>
<th>Attachment name</th>
<th>Size</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITC Avant Garde Gothic LT Medium Caps 0.ttf</td>
<td>62.9 kB</td>
<td>[Fonts]</td>
</tr>
<tr>
<td>sns_fonttest04.jpg</td>
<td>33.4 kB</td>
<td>[Graphics]</td>
</tr>
<tr>
<td>sns_fonttest05.jpg</td>
<td>32.3 kB</td>
<td>[Graphics]</td>
</tr>
</tbody>
</table>

The attachment manager

The entire dialogue is fairly self-explanatory. The two "Attach ..." buttons add attachments, "Extract" extracts existing attachments into separate files, and "Delete" deletes attachments from the subtitles file.

Limitations and downsides

Supported formats

The SSA format specification only allows certain filetypes to be attached. For fonts, only .ttf is allowed. For pictures, .bmp, .gif, .ico, .jpg and .wmf are allowed (note the absence of .png).

Compatibility issues

Many SSA/ASS editors ignore or strip attachments. The original SubStation Alpha (v4.08)
will work fine, but only for real SSA files. Sabbu will complain about unrecognized fields, and strip the attachments if you save the file. Most other editors either ignore the attachments or crash when encountering them.

A notable exception is mkvmerge, which will convert the attached files to Matroska attachments on muxing. If you demux the script again, the attachments will be stripped from the script, but they’re still in the MKV, as attachments, and can be extracted separately with mkvextract.

**Size**

Unfortunately, storing binary data as text (in this case, a variant of UUEncoding) is not very efficient. The attached files will take considerably more space as script attachments than they do as separate files.

### 4.2 Color Picker

Aegisub introduces a more advanced color picker than the default Windows one. It allows you to enter color values using various color spaces, in both HTML and ASS hex, pick colors from a mini-screenshot and use four different color spectrums for graphically picking colors.

![Color Picker](image)

The color picker has the following components:
- Color spectrum
- Parametric settings for four color-spaces
- HEX input for ASS and HTML formats
- Pick-from-screen control, for taking a mini-screenshot and picking a color from it
- List of last 24 picked colors
The last used mode of the color spectrum is remembered.

**Using the pick-from-screen function**
1. The pick-from-screen control is located to the left of the last-used colors.
2. To start using it, click and drag the X in the corner of the control.
3. As you drag over the screen, the mouse cursor will turn into a cross, and the pick-from-screen control will show a magnification of the area of the screen you're currently pointing to.
4. When you've found the area you want to grab a color from, release the mouse
5. Click on the control outside the X to pick a color.

**Descriptions of the color spaces**
Here's a short round-up of the available color spaces:

- **RGB** - Red, Green and Blue, the three component colors a computer monitor uses to display images. In RGB mode, you specify the intensity of each of those three components.
- **YUV** - This colorspace is commonly used to represent video data. The Y component represents the intensity of the light (luma), while the U and V components represent the color of it (chroma).
- **HSL** - Hue, Saturation and Luminance. *Hue* is the actual "color of the color", that is, red, green, blue or a mix of those. *Saturation* is the "greyness" of the color, the lower the saturation, the closer to a shade of grey, the color becomes. *Luminance* is the lightness, max luminance means pure white, while zero luminance means pure black.
- **HSV** - Hue, Saturation and Value. *Hue* and *Saturation* mean the same as in HSL. *Value*, however, is different from the *luminance* in HSL. *Value* is the "non-blackness" of the color, the smaller the value, the closer the color is to pure black.

4.3 **FexTracker**

As of version 1.10, Aegisub includes a powerful motion tracker developed by Tentacle, called "FexTracker". It allows you to make lines follow points on the video, generating positioning and scaling information for the subtitle line in each frame.

*Note: this section of the manual is incomplete, mostly because FexTracker itself is heavily obfuscated.*

**Tutorial**

First, you will (naturally) need a video loaded. Also, if you are working with a VFR video, make sure that the timecodes are also loaded. Select the line that you want to be affected by FexTracker, and make sure that its start and end times are correct. After that, you are all set to start!

- Seek to the first frame where the subtitles are visible.
- Click the FexTracker button under the video display. A menu will pop-up - select "Track Points".
- A configuration window will pop-up. It is not necessary to change anything, so click "Go!".
- Wait while the video is processed. The time taken depends on video resolution and...
number of frames spanned by the selected line.
- If everything went well, you should now see several red marks on the video, indicating the points that FexTracker considered suitable for tracking.
- Go back to the FexTracker menu and click "Add Points to Movement".
- Click on all points that you want to be included in the tracking. This should include as many points as possible, but ONLY those that are part of the object you're tracking! (Alternatively, just hold down Left mouse button and drag mouse over all points that you want included). Points should turn green when selected.
- If you want to deselect some point, choose "Remove Points from Movement" from the FexTracker menu.
- Move the video display frame by frame until the end of line, to make sure that the points are being tracked correctly. If necessary, you can add new points after a certain time in video (e.g. to keep tracking after all original points scrolled out of image).
- Once you have selected the points, choose "Generate Movement from Points" from the FexTracker menu.
- A red cross should be displayed, indicating the coordinates where the \pos() tag will be generated. A green line is also seen, showing the movement direction vector.
- If you want the subtitles positioned elsewhere, go to the FexMovement menu and choose "Move Subtitle" (or one of the 3 variations) to reposition the red cross. You can then drag and drop it on the video display.
- Once you are done, go to the FexMovement menu and choose "Split line for movement". This will create one copy of the line for each frame, with appropriate transformations.
- You're done!

4.4 Fonts Collector

Introduction

The fonts collector is a very useful tool that finds all fonts used in the current file and copies all of them to a folder. This is useful when you want/need to send the script to someone else.

How to Use
The dialog is very simple. You can choose or type the destination folder. If it doesn't exist, it'll be created. You can also tick the box "As attachments" to add the fonts as attachments to the subtitle file instead of as separate files (see the attachment manager section). Then just press "Start!" to scan the file for the fonts and copy them. Progress will be displayed in the log. It will look something like this in the end:
Please note that the fonts collector won't always succeed. Windows has a braindead way to store font face to filename mapping, and the technique Aegisub uses is not perfect.

4.5 Styling Assistant

Styling assistant is a powerful tool for subtitles that have multiple styles, such as cases when several actors have their own style, or for some other reason you have many different styles for dialog lines.
As seen from the image above, the assistant is composed of the following:

- Current line field
- Styles available field
- Set style window
- Keys field
- Enable preview checkbox
- Play audio button

The Current line field displays the line in the subtitles for which the style will be set. The text in it cannot be edited.

Styles available field displays the styles available in the script. The styles have to be imported through the Styles Manager or created with its help. The styles can then be set for the selected line in the subtitles by clicking on the desired style.

The set style window is the main aspect of the Styling Assistant. Through it, styles can be applied to the line in the subtitles easily and without much work. It supports autocomplete, so it helps to select the desired style after some characters are written in the window.

The Keys fields displays the keyboard shortcuts that are used for work with the assistant.

Enable preview checkbox is to enable a video preview for the subtitles. The video jumps to the first frame of the selected line.

Play audio button helps the typesetter to determine to which actor a line goes. It plays the audio of the line for which the style will be applied.
As seen from this image the styling assistant displays the video for the line if the checkbox is selected. When the style for a line is set via the Styles available or with the Set style window the Styling Assistant moves to the next line as seen in the image.

The autocomplete function.

This window displays how the autocomplete function works. The characters that were typed are on the white background and the rest of the name is displayed as being selected. The selected part is what autocomplete sets as the style that matches with the typed characters.

This image shows the response to a name that doesn't exist. The background of the window is colored red to show that the style doesn't exist on the list of selected styles.

How to use

Usage is simple. Select the first line you want to style and open the assistant. Then you look at text, video and/or audio to determine who is speaking it and type first few letters of style name, after which autocomplete should kick in. Hit enter to go to next line and proceed until you're finished. The other way is to double click on a style in the Styles
available list and the style will be applied to the selected line.

4.6 **Timing Postprocessor**

The timing postprocessor is a highly useful tool for automatically correcting timing in various ways. As shown in the screenshot, there are three functions:

- Add lead-in and/or lead-out
- Make lines that start close to each other continuous by extending their start and/or end times
- Snapping line starts/ends to video keyframes (only available if a video is loaded)

The processing gets applied in the order it is displayed in the dialogue box. That is to say, first lead-in/outs are added, then a check for adjacent lines is performed and lines that are closer than the given threshold are made continuous, and lastly, line start/ends get snapped to keyframes.

**Apply to styles**

This field determines which styles will get processed - check all you want to process. This is useful for only processing dialogue lines while leaving signs and/or karaoke alone.

**Lead-in/Lead-out**

This is fairly obvious. The postprocessor will add the given time (in milliseconds) to each line start and end, respectively. You can add both lead-in and lead-out, only one, or none, by checking and unchecking the boxes as appropriate.

**Make adjacent subtitles continuous**

This function will check if any two lines' starts and ends are closer in time than the given threshold (in milliseconds). If they are, one or both will get their start and/or end time moved so that they are continuous (i.e. one appears directly following the next, without any subtitle-less frames in between).

The Bias slider determines how the lines are extended. Sliding it all the way to the right will extend the end time of the first line all the way to the start time of the second, without touching the second line at all. Sliding it all the way to the left will instead make the start time of the second line extend backwards to the end of the first line, without touching the first line at all. Putting it in the middle will extend the end time of the first line and the start time of the second equally, so that they meet in the middle. Anything in between will makes the lines "meet" where the slider is, so to speak. For example, if
the threshold was 1000, and the slider was 3/4ths of the way to the right (roughly as shown in the screenshot), the end time of the first line would get extended by 750 ms, and the start time of the second extended backwards with 250 ms.

**Keyframe snapping**

The keyframe snapping function is a kind of automatic scene timer. It is probably the most useful of the three, but will only work if there is a video loaded, because of its dependency on keyframes. A brief, highly simplified explanation of what a keyframe is, for the people who missed Video 101:

Modern video codecs compress video by storing as little information as possible. Instead of storing all frames in full (like a large amount of JPG pictures), they take occasionally take a *keyframe*, which is stored fully, and for each frame after that until the next keyframe, they only store what changed in the picture since the last frame. The only reason we need to know this is because keyframes usually appear at scene changes. The first frame in a scene is almost invariably a keyframe, but beware - far from every keyframe is a scene change! Most codecs insert a keyframe at least once every 250-300 frames, even if there hasn't been a scene change.

Back to the timing postprocessor's use of this. The keyframe snapping function will look at how close the start and end of lines are to the nearest keyframe, and if they are closer than the given threshold, it will get extended or shortened to the keyframe. There are four thresholds to consider:

- *Starts before* - if the line starts less than this many frames (inclusive) before a keyframe, its start time will get moved *forward* so that the line starts on the keyframe.
- *Starts after* - if the line starts less than this many frames (inclusive) after a keyframe, its start time will get moved *backward* so that the line starts on the keyframe.
- *Ends before* - if the line ends less than this many frames (inclusive) before a keyframe, its end time will get moved *forward* so that the line ends on the frame before the keyframe.
- *Ends after* - if the line ends less than this many frames (inclusive) after a keyframe, its end time will get moved *backward* so that the line ends on the frame before the keyframe.

When using this feature, remember your lead-in/out times, and the framerate of the video! At 23.976 frames per second, each frame lasts for about 42 ms. This means that if you have a lead-in of 120 ms, you can safely set the "starts before" threshold to at least 3 (3 * 42 = 126) without fear of lines starting showing up after someone has started speaking. Using the same reasoning, a lead-out of 250 ms allows you to safely have at least 6 as your "Ends after" threshold, without fear of lines ending before people stop talking. Figuring out good values for the other thresholds and/or frame rates is left as an exercise for the reader.

Another thing you can do with the keyframe snapping feature is using it to correct one-frame bleeds really, really fast. If your script is full of them, just set all the thresholds to 1 or 2, disable the lead-in/out adding and the adjacent line snapping, choose your dialogue style, and hit Apply. Problem solved.

### 4.7 Translation Assistant

**Introduction**

The translation assistant is a valuable tool for translating subtitles from one language to
the other. It allows you to translate in the most efficient possible manner, by letting you type the translated text without having to worry about selecting the original, or forgetting what was written in it. It also parses ASS override tags, and will skip over them.

Overview

The assistant window is simple: it shows you the original raw line, and a box where you can type the translation. There is also a list of hotkeys specific to this window, and a checkbox to enable previewing, if you have the corresponding video set up. Having this option enabled will automatically jump to the current line when you change it.

How to use

Using the assistant is very easy. First, start it by pressing F3 or going to Tools->Translation Assistant. It will start at the currently selected line. The original line will have the text that you have to translate highlighted in blue. Simply type the translation to it, and hit enter. If there are override tags in the line, you may have to translate more than one block per line.

In the above screenshot, supposing that you are translating to Portuguese, you would first type "Clima" (Portuguese for Weather) and hit enter. It would then highlight "_______". Since I don't want to translate that, I just hit Insert in the keyboard, and it will copy the text over. So I hit enter, and it will ask for the last bit, "Temperature", so I enter "Temperatura" and hit enter again. Then it will jump to the next line, where I will repeat
the process. If I make a mistake, I can use Page Up and Page Down keys to move to the next or previous block/line.

### 4.8 Other Tools

Here's the documentation for some various more or less useful tools and features that are too small to have their own page.

#### Find and replace

This is one of the most underestimated tools in any text editor, and even more so in Aegisub. It does exactly what it says it does - find strings, and optionally replace them with other strings.

The Find dialogue is very straightforward - it finds things. The "Update Video (slow)" checkbox makes the video follow the find cursor - in other words, it makes the video jump to the line selected when the search finds anything. Checking "Match case" turns the search case sensitive (which means that a search for "foo" will NOT find "Foo" or "FOO"). The regular expressions checkbox makes Aegisub interpret the search string as a Perl-style regular expression - more on that below.

Next, the replace dialogue:

The "Find what" field is exactly the same as in the find dialogue, and so is the "Find next" button. It finds strings without replacing them, and hence it's an excellent method of testing that your replace string really is finding what you want to replace. "Replace next"
does exactly what it says - replace the next found instance of the search string with the replacement string. "Replace all" naturally replaces all found strings with the replacement string.

Regular Expressions (aka. regexes or regexps)

Several dialogues in Aegisub can use regular expressions. But what's a regular expression?
First off, regular expressions are an awesomely powerful tool for working with text in various ways. Simply put, regular expressions provide a way to describe text strings in various ways. In many search engines, you can use a simple form of regular expressions - for example, searching for shoe* will find both shoe and shoes. But regular expressions can be much, much more powerful than this. For the full story about regular expressions, see the Wikipedia entry, or the Perl regular expressions tutorial (note that there's quite a bit of things in there that only applies to Perl and not to the wxWindows implementation). For a full list of exactly what Perl 5 extensions are supported in wxWindows (and thus in Aegisub), see the wxWindows manual entry.

Some useful examples

You can use regexes to easily add \{be1\} to the start of every line, for example. Go to the replace dialogue, check "use regular expressions", and enter the following:
Find what: ^
Replace with: {\be1}\1

To remove them again, use this:
Find what: ^\{\be1\}(.+)
Replace with: \1

You can do a simple, but very aggressive, tag stripper as well. The following will kill EVERYTHING inside { }'s:
Find what: \{(.*\}\)
Replace with: (yes, that's an empty string)

Select lines

The select lines dialouge can be accessed from the Edit menu, and it, well... selects lines (based on the conditions given).
The dialog is fairly self-explanatory. All lines that matches or doesn't match the given text string will be selected. You can choose if the string should be searched for in the Text field (the main subtitle text), in the Style field, or in the Actor field. You can also specify if you want to search in dialogue lines, comment lines, or both. The Action setting determines what action should be taken. Set selection sets the selection to all lines that matches the given conditions. Add to selection adds all matching lines to your current selection. Subtract from selection deselects all lines that matches from your current selection. Intersect with selection is a lot simpler than it sounds - it's like Set to selection but only applies to the currently selected lines.

You can also use regular expressions in this dialogue - see Find and replace above.

Resolution resampler

The resolution resampler is an automated tool to make changing the script resolution painless. It changes all tags and styles so that everything will look the same at the new resolution.

Spell checker

Early Aegisub versions (non-public alpha builds) had a spell checker, based on aspell, which was planned to be in all future versions. However, work on it was never finished and the responsible programmer got busy in Real Life. Since having it in its broken state was deemed worse than not having it at all, it was disabled and have stayed disabled on all builds since. The code is still there, though, even if it doesn't get included in the
current builds, and it’s certainly possible that the spell checker will be back in some future version of Aegisub.

4.9 Export

The Export dialogue is used for applying various heavy modifications to your script, such as conversion to other formats, automation scripting, stripping of headers, conversion to other character sets etc. You can find it under File -> Export Subtitles...

The bottom dropdown menu selects text encoding. The default (and recommended) encoding is UTF-8. At the top is the really interesting part: the filters list. There are three built-in filters, as displayed above. Note that the filters get applied in the order they are listed (from top to bottom).

Transform Framerate

The framerate transformation filter manipulates the subtitle timestamps (and \t and \k* tags - everything with times in it, basically) in various ways. You should generally have a video loaded when using it, but it’s not required in some cases. There are two modes of operation; VFR and CFR (“Variable” or “Constant” output).

Variable framerate output mode requires that you have a timecodes file loaded. You should have a video file loaded to make sure you get the correct input framerate, but it’s not required (you can fill it in manually). In this mode, the filter transforms all subtitle timestamps so that the subtitles synch with the video when hardsubbing on a CFR video that is intended for muxing with a timecodes file.

In the constant framerate output mode, you don’t need a timecodes file, since then the filter converts between two constant framerates. This is mostly useful for speeding up/slowing down subtitles, i.e. a Film -> PAL transfer with 4% speedup.
Clean Script Info

This will clean up your script by removing all headers except the required ones. In other words, all the Aegisub-specific headers, the script credits etc. will get removed, which might be desirable for softsub distribution.

Fix Styles

Makes sure the script doesn’t have any "missing styles" (which will cause VSFilter to pop up an alert) by checking so that all used styles have a corresponding style definition. If styles with no definition are found, they’re replaced with the Default style.

Automation scripts

Most automation scripts are applied to your subtitles as export filters.

Some of them has configurations within the Export dialog, as displayed above. The scripts shipped with Aegisub provide some quick documentation in the dialog as an aid.

Note: If you need to do framerate transformation on karaoke effects, be sure the framerate transformation is applied after the karaoke effect, otherwise you might get weird/unexpected results.
Top Level Intro

This page is printed before a new top-level chapter starts

Part V
5 Automation

*Automation* is a scripting system to quickly apply advanced effects to a subtitle file. For an example of what it can be used for, see the introduction.

Automation scripts are managed through the *Automation Manager* window, which can be accessed from the Tools menu.

Note that some of the documentation for Automation is now being maintained on the [Aegisub wiki](https://aegisub.org/wiki).

*If you have programmed before*, just not in the Lua language used for Automation, you might want to just start out with the [Lua Quickstart](https://aegisub.org/wiki/LuaQuickstart) and look at the [reference](https://aegisub.org/wiki/LuaReference) afterwards. The example scripts will probably also be useful to you. You are strongly recommended to use the karaskel scripts if writing karaoke effects. The factorybrews scripts, especially multi-template, might also prove useful.

*If you haven't programmed before*, or just aren't too sure about where to start creating effects, a collection of tutorials are being maintained on the wiki. [Automation tutorials on the wiki](https://aegisub.org/wiki/AutomationTutorials).

**Note:** A new version of Automation (called Automation 4, or Auto4 for short) is being worked on, and is expected to be ready for the next version of Aegisub. While there will be some compatibility scripts included, expect that your scripts will no longer work in Auto4. See [this page](https://aegisub.org/wiki/Compatibility) on the wiki for more information.

5.1 Introduction

*Automation* is the scripting system in Aegisub, used primarily for producing special effects that are complicated enough that you don't want to do them by hand. (Because it would take too long and/or be error prone.)

The Lua scripting language is used for Automation scripts. You can find more information about Lua in [the online manual](https://lua.org/manual).

Below are three examples of what Automation can be used for.

**Creating karaoke effects**

Automation scripts are provided with pre-parsed karaoke data, such that a script can easily produce simple karaoke effects, and with just a little more effort, even some very advanced effects can be produced.

It's possible to create multiple copies of each line, edit every property of a line, and most importantly, calculate the rendered size of any text with any style.

**Automatically styling translation notes**

It's becoming more and more common to use extravagantly styled translation notes, appearing at the top of the video, with an advanced transition effects, transparent boxes and the like.
It can be tiring to manually style each and every note needed in a subtitle file, but with Automation you can just write the notes in as regular, unstyled lines, and write a script to do the actual styling. And the best is, it's fully re-usable!

**Apply per-character styles**
Another thing frequently seen is different subtitle styles for each character participating in a video. With Automation you can make a script to apply a style per line, based on the "character name" field.

Or how about automatically detecting overlapping subs and applying different styles to each one?

### 5.2 The Automation Manager

The Automation Manager is used to load Automation scripts, assigning them to a subtitle file. It can also be used to create new scripts.

You access the Automation Manager through the Tools menu in the main window.

The buttons do the following:

**Create...**
Creates a new Automation script and loads it. This will start the editor you have selected. (See more about that below.)

**Add...**
Load an existing script.

**Remove**
Unloads the currently selected script.

**Apply now**
Run the selected script on the currently loaded subs. If the script supports configuration, you will be asked for settings first. You can undo the effect of running a script after closing the Automation Manager. Note that using this button is generally **discouraged**; often it's more convenient to apply Automation scripts during an export operation. Automation scripts appear as export filters in the Export dialog.
5.3 Lua Quickstart

What is this?
This is a really quick introduction to Lua for long-time programmers who just don't know the language but want to get coding something quickly.

*If you have never programmed before, this is not the right place for you!*

Some of the things here will be specific to Aegisub, and they might even be wrong in other applications using Lua.
For a more complete description, you should see the Lua manual, available at: [http://www.lua.org/manual/5.0/](http://www.lua.org/manual/5.0/).

A quick example
Here's a slightly cut down Automation script, showing most of the syntax of Lua.

Scripts for Automation must be encoded in the UTF-8 character set. The file may start with a byte-order mark (BOM) or not.

```lua
-- Include some useful extra functions
include("utils.lua")
version, kind, configuration = 3, "basic_ass", {}
name = "Sample script"
description = "This is an example script."

-- The main script function starts here
function process_lines(meta, styles, lines, config)
    aegisub.output_debug(string.format("Number of lines to process: %d", lines.n))
    local output = {}
    output.n = lines.n
    for i = 0, lines.n-1 do
        if lines[i].kind ~= "dialog" then
            output[i] = lines[i]
        else
            local newline = copy_line(lines[i])
            newline.text = string.format("{\fad(700,300)\pos(%d,30)\k100}", meta.res_x/2)
            newline.start_time = newline.start_time - 100
            local cursylpos = 1000
            for j = 1, lines[i].karaoke.n-1 do
                local syl = lines[i].karaoke[j]
                newline.text = newline.text .. string.format("{\t(%d,%d,\fscx50\fscy50)}%s", cursylpos, cursylpos + syl.duration*10, syl.text_stripped)
                cursylpos = cursylpos + syl.duration*10
            end
            newline.text = newline.text .. string.format("%s", cursylpos, cursylpos + syl.duration*10)
            output[i] = newline
        end
    end
end
```

© 2005 Rodrigo Braz Monteiro. All rights reserved.
Note: Lua is case sensitive, however, indentation does not matter.

**Data types**
Lua is weakly typed, you can not declare the type of a variable.  
For the most part, there is automatic conversion between strings and numbers.  
All variables hold references to their value. Assignment copies a reference, not the actual value.

nil - The value of "nothing". Any unassigned variables will have the value **nil**.  
boolean - Takes the values true and false. Not compatible with number.  
number - A double-precision floating-point value.  
string - A character string. In Aegisub, strings must be in UTF-8 encoding.  
function - Functions are treated as any other value.  
table - A data structure, mostly similar to a hash table. Can be indexed by any data type but nil.  
userdata - A host-program defined value. Not used in Aegisub.  
thread - A thread handle. Not used in Aegisub either.

Only nil and false are considered as false values. All other values are considered as true.  
(Including the number zero and the empty string.)

**Variables**
All variables hold references. Assignment copies the reference, not the value. (Tables must be copied field by field.)  
Unless declared otherwise, all variables are global.  
Local variables are statically scoped.

**Declarations**
Variables come into existence by assigning something to them.  
Local variables are declared with the **local** keyword.  
Functions can be declared in one of two ways:

```lua
foo = function(bar)
  ...
end

and

function foo(bar)
  ...
end
```

An empty table is written as an empty pair of braces:

```lua
foo = { }
```

**Operators**
Regular arithmetic is done with + - * /
Note that Lua does not have a modular operator (% in C and many other languages.)
You can do exponentiation with the ^ operator.
String concatenation is done with the two-dots operator:
foo .. bar

Assignment is done with the = operator. Assignment is not an expression (it does not return a value.) You can do multiple assignment with lists.

```plaintext
foo = bar
a, b = c, d
x, y = y, x  -- swap two values
```

Comparison is done with the == ~= > < >= <= operators. (Note ~= for "not equal".) Logical conjunction, disjunction and negation is done with the and, or and not keywords.

Accessing tables can be done in two ways. The basic way is using the array-style accessor:

```plaintext
foo[bar]
```

Tables can also be used as structures in C. The following expressions are equivalent:

```plaintext
foo.bar
foo["bar"]
```

**Control structures**

None of the control structures are expressions. (They do not return a value.)

The if statement:

```plaintext
if foo then
...
elsif bar then
...
else
...
end
```

Blocks are automatically generated, independent of indentation. You can have multiple elseif parts. The else part is optional. Remember that only false and nil count as false, 0 and "" among others are true.

The for statement:

```plaintext
for i = a, b do
...
break
...
end
```

i loops through all values from a to b, both inclusive. There is no continue statement. The for loop also has another form, based around iterator functions, but Aegisub does not use those anywhere.

Returning values from functions:

```plaintext
return a, b
```

The return statement can return any number of values, including none.

---

### 5.4 Reference

Want to dive right in or just forgot exactly how a "style" table looks? This is the right place.

- **The process_lines function** - The main function every script has to implement
- **Special variables** - Specially named variables, required in every script
- Data structures - Exactly how does the data passed to a script look?
Library functions - Things you can call to enhance your script's functionality

- include
- output_debug
- csec_to_timestamp
- report_progress
- text_extents

Apart from the functions mentioned here, make sure you also take a look at the functions provided by the `utils.lua` include file.

### 5.4.1 The scripting environment

Every Automation script is run in its own, separate Lua interpreter. This means that two scripts cannot interfere with each other, but also that they cannot (directly) share code/data.

If you need to share code between several scripts, include files is the way to go. See the `include` function for more information.

All strings input to Automation scripts are encoded in UTF-8 Unicode. This means that you should save your script files as UTF-8 files. A byte-order mark (BOM) at the beginning of the file is optional. (Script files are assumed to be UTF-8.)

**Standard libraries**

Most of the standard libraries provided with Lua are loaded into the scripting environment.

The following libraries are available: (links to documentation on the WWW)

- **Base library** (except for the `dofile` function, which is replaced by the `include` function.)
- **Coroutine library**
- **String library**
- **Table manipulation library**
- **Math library**

The following libraries are not available for security reasons:

- File I/O library
- OS facilities library
- Debugging library

A safer replacement for the file I/O library is planned.

### 5.4.2 process_lines

```
function process_lines (meta, styles, lines, config)
```

This is the prototype for the primary script function.

The basic idea is, that the `process_lines` function is called with some input subtitles, and
returns a complete replacement for that input.

**Arguments**

- **meta** (table)
  A metadata structure.

- **styles** (table)
  A list of style tables.
  Index \(n\) contains the number of entries. The entries have index numbers from 0 to \(n-1\).
  Since each style has a name, the styles can also be indexed by their name.

- **lines** (table)
  A list of subtitle lines.
  Index \(n\) contains the number of entries. The entries have index numbers from 0 to \(n-1\).

- **config** (table)
  A table containing the configuration settings the user has selected for this run of the script. See Configuration system for more information.

**Return value**

A table in the same format as the **lines** argument. It should contain an index \(n\), describing the number of lines in the result. The subtitle tables contained must be indexed from 1 to \(n\) without "holes".

This was changed in version 1.07. For compatibility, you can also use indexing from 0 to \(n-1\).

### 5.4.3 Special variables

The following special variables are defined.

- **name** (string)
  The name of the script, displayed in the UI. This must be unique, as it's used for identifying a script in the Export function in Aegisub.
  Required.

- **description** (string)
  A longer description of what the script does, also for display in the UI. This doesn't need to be unique, and it can be empty.
  Required.

- **version** (number)
  The version of the Automation engine, this script is intended to.
  Required. Must have the value 3 for the current version.

- **kind** (string)
  In case Aegisub one day will support more than just the current kind of ASS transformation scripts, this variable will tell what kind of script it is.
  Required. Must have the value "basic_ass".

- **configuration** (table)
  Defines a number of options the user can select for the script before it's run. These will be presented in a graphical interface. See Configuration system for more information.
  Optional. (If not present, the result is as if it was an empty table.)
process_lines (function)
The function called to process the subtitle data. More information here.
Required.

5.4.4 Data structures

5.4.4.1 Meta data
The meta info contains general information about the subtitles. It roughly corresponds to the [Script Info] section in an ASS file. Currently, only two fields are defined:

res_x (number)
res_y (number)
The X and Y pixel resolution of the script. These define the "virtual screen" the subtitles are rendered on. There is currently no way to get the pixel aspect ratio of the actual video.

5.4.4.2 Style data
A style table contains an ASS style definition. A list of style tables is passed to the process_lines function, and it is also used when calling the aegisub.text_extents function. A style table contains the following fields:

name (string)
The name of the style. Can be used to index the style list passed to process_lines.

fontname (string)
fontsize (number)
Name and point size of the font. Remember font names are case sensitive.

color1 (string)
Primary color for the text. In ASS hexadecimal notation.

color2 (string)
Secondary color for the text.

color3 (string)
Outline color.

color4 (string)
Shadow color.

bold (boolean)
italic (boolean)
underline (boolean)
strikeout (boolean)
Whether the text is bold/italic/underlined/striked out.

scale_x (number)
scale_y (number)
X and Y scaling of the text in percent.

spacing (number)
Inter-character spacing in the text in pixels.
angle (number)
    Z axis rotation of the text, in degrees.

borderstyle (number)
    Border style used. (1=outline, 3=solid box.)

outline (number)
    Thickness of the outline, in pixels

shadow (number)
    Distance of the shadow, in pixels.

align (number)
    Numpad style alignment for the lines.

margin_l (number)
margin_r (number)
margin_v (number)
    Left, right and vertical margins, in pixels.

encoding (number)
    Legacy font encoding.

5.4.4.3 Subtitle data

A subtitle table represents a single line in an ASS file. Apart from regular dialog-lines, it
can also be a dialog-style comment line, a "semicolon comment" and a blank line.

A subtitle table always contains the following field:

kind (string)
    One of "dialog", "comment", "scomment" and "blank". Describes what kind of line this
    is in the subtitle file. You should always test against the value of the kind field before
    trying to access any other field in a subtitle table.

If the value of kind is "blank", no further fields are defined. (It's just a blank line.)

If the value is "scomment", the line is a semicolon comment line, that is, a line where the
first character is a semicolon. While comments like this is usually ignored during regular
processing, you might find them useful to store additional parameters needed for an
effect. For "scomment" lines, the following additional field is defined:

text (string)
    The text following the semicolon, until end of line. This includes any leading and
    trailing whitespace but never the semicolon or the line break character(s).

If the value is "dialog" or "comment", the line has the format of a regular dialog line. The
difference is that a "comment" line start with the "Comment:" keyword instead of the
"dialog:" keyword, and that comment lines aren't rendered when the subtitles are played
back. dialog-style lines contain the following additional fields:

layer (number)
    The logical layer the line is rendered on. (Lines with a higher layer number are
    visually rendered on top of lines with a lower layer number.)
5.4.4 Karaoke data

A karaoke table contains information about one karaoke syllable. A karaoke syllable is defined to start with an override tag, where the first character in the tag name is k or K, followed by zero or more additional letters, followed by an integer. The tag name, including the initial k or K, is called the "kind" of the syllable, and the number is the duration.

A karaoke table contains the following fields:

- **duration** (number)
  - Duration of the syllable, in milliseconds. The first syllable on a line always has duration 0.

- **kind** (string)
  - Kind of the syllable. The first syllable on a line always has kind "" (the empty string.)

- **text** (string)
  - Text in the syllable, including any override tags. Care is taken that any override tag
blocks at the start and end of the text are properly closed. Never includes the actual karaoke tag. The first syllable on a line has all text until the first karaoke tag.

text_stripped (string)
Same as the text field, except that all override tags and drawings are stripped from here.

5.4.5 Configuration system

The configuration system in Automation allows you (as a script writer) to define a set of options, the user of the script can set before it's run. This way you can make generally usable scripts (eg. something to make a variation over some general karaoke effect) or just allow yourself to easily tweak some values during script development.

The screenshot above shows the configuration defined by the script simple-k-replacer.lua, which is shipped with the Aegisub distribution. The following code in the script file defines the configuration:

configuration = {
    -- First a label to descrit what special variables can be used
    [1] = {
        name = "label1",
        kind = "label",
        label = "Variables:
%START% = Start-time of syllable (ms)
%END% = End-time of syllable (ms)
%DUR% = Duration of syllable (cs)]
    hint = "",
    -- No "default", since a label doesn't have a value
    },
    -- Then a text field to input the string to replace \k's with
    -- Make the default a "NOP" string
    [2] = {
        name = "k_repstr",
        kind = "text",
        label = "\\k replacement",
        hint = "The string to replace \k tags with. Should start and end with { } characters.";
    }
default = "{\k%DUR%}"
}
-- Allow the user to specify whether to strip tags or not
[3] = {
  name = "striptags";
  kind = "bool";
  label = "Strip all tags";
  hint = "Strip all formatting tags apart from the processed karaoke tags?";
  default = false
}
}

This example only shows three of the kinds of controls supported. A complete list of the supported kinds are:

- **Label** - A static text display, for e.g. showing instructions to the user
- **Text** - A simple text input control
- **Integer** - A text input control with spin buttons, for entering integer numbers only
- **Float** - A text input for entering only floating-point numbers
- **Boolean** - A checkbox for selecting a true/false value
- **color** - A text input for entering color codes (this will be replaced with a better color-input control in the future)
- **Style** - A drop-down listbox for selecting one of the styles currently in the subtitles

### Implementing configuration

Configuration is implemented in an Automation script by defining a number of controls in the global `configuration` variable and accessing the values the user set through the `config` argument passed to the `process_lines` function.

The `configuration` global variable is a table containing a number of other tables, indexed by numbers starting from 1. You do not need to explicitly define the number of options anywhere.

Each of the numbered subtables in the `configuration` table should have the following fields:

- **name** (string)
  The internal name of the configuration variable, this option defines. It must not contain the colon or pipe characters. (ASCII 58 and 124.) Required.

- **kind** (string)
  One of "label", "text", "int", "float", "bool", "color" or "style". Defines what kind of control this option generates. Required.

- **label** (string)
  The label displayed to the left (right for "bool" kind) of the control in the window, or just the text of the label, in case of the "label" kind of control. Required.

- **hint** (string)
  Tooltip displayed when the mouse hovers over the control for some time. This doesn't seem to work for "label" kind controls. You can use this to provide a longer description than fits into the label. Required (but may be an empty string.)

- **min** (number)
  The minimum value the user can enter for "int" and "float" kind options. Optional.

- **max** (number)
  The maximum value the user can enter for "int" and "float" kind options. Optional.
default (type depends on kind)

The default value given to this option when the script is first loaded for a subtitle file. This value is also used if the user enters an invalid value into an "int" or "float" text field. See the table of data-type to option kind mapping below, for what the type of this field should be.

The data-type to option kind mapping is as follows:

<table>
<thead>
<tr>
<th>Option kind</th>
<th>Data-type</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>none</td>
<td>The label kind doesn't contain or generate any data.</td>
</tr>
<tr>
<td>text</td>
<td>string</td>
<td>An unlimited-length UTF-8 string.</td>
</tr>
<tr>
<td>int</td>
<td>number</td>
<td>Might not be exactly the same value as entered, if the number is very large.</td>
</tr>
<tr>
<td>float</td>
<td>number</td>
<td>Currently not guaranteed to be a valid ASS color code.</td>
</tr>
<tr>
<td>bool</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>string</td>
<td>If another export filter before the Automation script changes the styles available in the subtitles, the named style might no longer exist when the script is run.</td>
</tr>
<tr>
<td>style</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

When the script is run, the values the user has selected for the options are stored in the `config` table passed to the `process_lines` function. The option values are accessed using the defined `name` for them as key in the `config` table. For example, this is how it’s done in `simple-k-replacer.lua`:

```lua
if config.striptags then
    newtext = line.karaoke[0].text_stripped
else
    newtext = line.karaoke[0].text
end
```

Storage of configuration settings in ASS files

If you save your original subtitles file after applying an Automation script using the configuration system (either through Export or through the Apply now button in Automation Manager), the settings selected are stored in the ASS file.

**Note:** The rest of the information on this page is in no way required to know in order to use Automation or write scripts for it. It is included only for informational purposes.

The settings for one Automation script is saved on one line in the ASS file. Here is an example of such a line:

```plaintext
Automation Settings simple-k-replacer.lua:
\r\t(%START%#2C%END%#2C\fscy150)%kf%DUR%|striptags:0
```

(The above should be on just one line.)

The format of such a line is as follows:

- The name of the line is "Automation Settings" plus a space plus the filename of the Automation script, with all path elements stripped off.
- After that follows a list of name:value pairs separated by pipe (ASCII 124) characters.
The name and value in such a pair are separated by a single colon (ASCII 58) without spaces around.

The name in a name:value pair is simply the internal name specified in the configuration table, without any mangling at all.
The way the value is stored depends on the kind of the option:

- **label**
  Labels aren't stored in the ASS file at all.
- **text**
  Stored using the inline string encoding described below.
- **int**
  Stored in base 10 ASCII without any kind of group separators.
- **float**
  Stored in exponential base 10 notation using ASCII. (The same way as the C printf() %e argument outputs numbers.)
- **bool**
  True is stored as 1, false as 0.
- **color**
  Stored in ASS hex format without mangling.
- **style**
  Stored using inline string encoding of the style name.

### Inline string encoding

This encoding only works on ASCII characters. Unicode characters can't be encoded. This shouldn't be a problem, though, since this encoding is only used to escape characters that would otherwise be unsafe to use.

This encoding is heavily inspired by the URL-encoding method used in HTTP, and the primary difference is the escape character used. Inline string encoding uses # (hash, sharp, ASCII 35) as escape character. The # is followed by two hexadecimal digits, which are the ASCII code of the encoded character.

The following characters are currently encoded when saved: (All others are passed through as-is.)

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII</th>
<th>Encoded form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control codes</td>
<td>0 to 31</td>
<td>#00 to #1F</td>
</tr>
<tr>
<td># (hash/sharp)</td>
<td>35</td>
<td>#23</td>
</tr>
<tr>
<td>, (comma)</td>
<td>44</td>
<td>#2C</td>
</tr>
<tr>
<td>: (colon)</td>
<td>58</td>
<td>#3A</td>
</tr>
<tr>
<td></td>
<td>124</td>
<td>#7C</td>
</tr>
</tbody>
</table>

### 5.4.6 Library functions

A number of library functions are provided.

**function include (filename)**

Include the named script. The script search-path defined in Aegisub will be used when searching for the script.
If the filename is relative, the regular search path will not be used, but instead the filename will be taken as relative to the directory the current script is located in.

Note that if you use include() inside an included script, relative paths will still be taken relative to the original script, and not relative to the current included script. This is a design limitation.

The included script is loaded as an anonymous function, which is executed in the current environment. This has two implications:
- You can include files based on conditional statements, even in loops (to include the same file multiple times.)
- Included files can return values using the "return" statement.

@filename (string)
The name of the file to include.

Return value
Depends on the file included.
If some error occurred while including the file, the script execution will be terminated.
(This means that the script can fail to load, if the include function is called in during the initialization.)

function aegisub.output_debug (text)
Outputs some text to a debug console.

@text (string)
The text to output.

Return value
Nothing.

function aegisub.set_status (text)
Sets the current status-message. This is shown in the progress dialog during script processing.

@text (string)
The status-message.

Return value
Nothing.

function aegisub.colorstring_to_rgb (colorstring)
Convert an ASS color-string to a set of RGB values.

@colorstring (string)
The color-string to convert.

Return value
Four values, all numbers, being the color-components in the following order: Red, Green, Blue, Alpha-channel.
function **aegisub.report_progress** (progress)

Report the progress of the script. Mostly useful for scripts that take a very long time to calculate the final result.

@**progress** (number)
   The progress of the script, in percent. The value should be between 0 and 100. If the value is out of range, it will be clamped to 0 or 100.

function **aegisub.text_extents** (style, text)

Calculate the rendered size of the given text with the given style. The result can be interpreted as a bounding box around the text, before any rotations, and not including borders and shadows.

If you need the bounding box of the text rotated, you should get the unrotated size first, and then calculate the rotated size from that.

@**style** (table)
   The style used to calculate the size. Must be in valid style table format. (Usually you can just index the styles table passed to the `process_lines` function.) Only the following fields are used in the size calculation: `fontname`, `fontsize`, `bold`, `italic`, `underline`, `strikeout`, `scale_x`, `scale_y` and `spacing`.

@**text** (string)
   The text to be used for size calculation. It should not contain any override tags or blocks, as those will be taken as part of the text and not used with their special meaning. The \n, \h and \h tags are not handled either.

**Return value**
This function returns four values, in this order:
1. Width of the text, in pixels (number)
2. Height of the text, in pixels (number)
3. Descent of the font, in pixels (number)
4. External leading of the font, in pixels (number)

All return values are corrected scaling by rounded using C rounding semantics.

You can find more information about the various typographic measurements in the [FreeType 2 documentation](http://www.freetype.org/).

This function is currently implemented using the wxWidgets wxFont and wxDC classes, meaning it retrieves the sizing information from Windows itself. In future versions, this might be changed to use the FreeType 2 library instead. If/when that happens, several more sizing measurements will be made available.

function **aegisub.frame_from_ms** (ms)

Return the video frame-number for the given time.

@**ms** (number)
   Time in milliseconds to get the frame number for.
**Return value**
A number, the frame number. If there is no framerate data, returns `nil`.

function `aegisub.ms_from_frame` (frame)
Returns the start-time for the given video frame-number.

@frame (number)
Frame-number to get start-time from.

**Return value**
A number, the start-time of the frame. If there is no framerate data, returns `nil`.

### 5.5 Included scripts

A number of Automation scripts and include files are provided with Aegisub, these will be described here.

The first thing you'll want to look at is probably the [demo scripts](#). Some additional functions that can be considered "standard library" can be found in the `utils.lua` include file.

To simplify writing karaoke generation scripts, the [Karaskel](#) family of includes is provided. They simplify the common tasks without taking away any of the power.

#### 5.5.1 Demo scripts

Currently nine demo scripts are provided. They are located in the folder `C:\Program Files\Aegisub\automation\demos\` in a default installation on an English language Windows.

1-minimal.lua
- Shows a very minimal Automation script, that only calls the output_debug function and otherwise passes the data through unchanged.

2-dump.lua
- This script also passes the data through unchanged, but it shows how to loop through the data. Most of the data are dumped with the output_debug function.

3-include.lua
- Includes `utils.lua` and uses the copy_line function to duplicate the first line in the script. It shows how to include a file and add lines to the subs.

4-text_extents.lua
- Uses the text_extents function to get the size of the individual syllables in a karaoke and place them in correct order on screen, centered horizontally and vertically. For highlight effect, the syllables move a bit upwards.

5-configuration.lua
- Shows how to define every kind of configuration option, and uses a few of the defined options to modify the input.
6-simple-effect.lua
   A heavily commented script showing how to generate a relatively simple effect. See below for an example of the effect generated.

7-advanced-effect.lua
   This demo shows how to make a rather advanced effect using multiple lines per syllable and treating syllables differently depending on what kind of tag they're timed with. See below for an example of the effect generated.

8-skeleton.lua
   Shows how to use the karaskel.lua include file to avoid much of the otherwise tedious coding most Automation scripts will contain. Useful as a starting point for simple effects.

9-skeleton-advanced.lua
   Shows how to use karaskel-adv.lua to make more advanced effects than possible with the vanilla karaskel.lua.

10-furigana.lua
   Demonstrates the furigana capabilities of karaskel-adv.lua.

An example of the effect generated by 6-simple-effect.lua:

An example of the effect generated by 7-advanced-effect.lua:
This file contains several useful functions when writing advanced Automation scripts. In default installations on an English language Windows, it is located in `C:\Program Files\Aegisub\automation\include\`, but since the `automation\include\` directory under the Aegisub install directory is automatically searched when using the `include` function, you just need to use the following line to include it:

```lua
include("utils.lua")
```

The following functions are defined in `utils.lua`:

function **copy_line** (input)
Makes a shallow copy of a line table.
You will usually want to use this function when generating new lines based on lines that already exist in the subs. A simple assignment can't be used, since it only copies the reference to the line table. Everything except the karaoke table in the copied line can safely be modified without affecting other copies of the line.

See `9-skeleton-advanced.lua` (among others) for an example of how to use this function.

@**input** (table)
The line to copy. Expected to be a line table.

**Return value**
A copy of the input line.

function **ass_color** (r, g, b)
Makes an ASS hex color code from the given RGB values.

@**r** (number)
@**g** (number)
@**b** (number)
The values of the red, green and blue components of the color to generate the color code for.

**Return value**
A string being the ASS hex color code for the input.

function **HSV_to_RGB** (H, S, V)
Convert a color specified in HSV (hue, saturation, luminance) format into RGB values.

@**H** (number)
Hue of the color.
@**S** (number)
Saturation ("grayness") of the color.
@**V** (number)
Value ("brightness") of the color.

**Return value**
Returns 3 numbers being (in order) the red, green and blue components of the specified HSV color.

function **trim** (s)
Removes whitespace at the beginning and end of the passed string.

@**s** (string)
The string to remove whitespace from.

**Return value**
The string without whitespace at the beginning or end.

function **next_utf_char** (str, off)
Get the index of the next character from @off in the UTF-8 string @str. That is, @off subtracted from the return-value of this function is the number of bytes taken by the UTF-8 character pointed to by @off.

@**str** (string)
The UTF-8 string to walk.
@**off** (number)
The byte-offset into the string for the current character.

**Return value**
An index (number) into the string for the next character.
function `utf_len` (str)

Get the number of characters (not bytes) in the given UTF-8 string.

@str (string)
  The string to get the length of.

**Return value**
The number of characters in the given string.

function `string.headtail` (s)

Get the "head" and "tail" of a string, treating it as a list of words separated by one or more space-characters.

@s (string)
  The string to get head/tail of

**Return values**
Returns two strings. First is the head, second is the tail.
If @s is the empty string, returns two empty strings.
If @s contains no spaces, head is @s and tail is the empty string.

function `xor` (a, b)

Calculate the logical exclusive or of two values.

@a, @b (anything)
  The two values to calculate xor of

**Return values**
If only @a is non-false, returns @a.
If only @b is non-false, returns @b.
Otherwise returns false.

### 5.5.3 Karaskel

Karaskel is the name for a family of include files provided with Automation. The Karaskel scripts provide much of the skeleton code you often need when writing karaoke generation scripts, allowing you to focus on the effect itself, rather than how to structure your code and calculate various positions etc.

The primary idea of Karaskel is to pre-calculate a lot of often-needed positioning and timing information, and structure the output generation by automatically calling certain functions for handling lines and syllables. You can see some basic sample scripts using Karaskel in demos 8, 9 and 10. Consider demo 6 and 7 as samples of how not to do karaoke generation scripts, those both bloat and are hard to read because of not using Karaskel.

There are two Karaskel available, the "basic" and the "advanced". Chances are you'll be using the advanced one most of the time. Here's a quick rundown of the two skeletons:
Simple:
- Works on the *one line in, one line out* principle.
- You write a function to generate a replacement text for each syllable.
- You can optionally add additional text at the start/end of each line.
- Not much different from the *simple-k-replacer* factorybrew script, except possibly more manageable.

Advanced:
- Works with *one line in, any number out* principle.
- You write a function to generate a number of lines for each syllable.
- Support for *furigana* (see demo 10.)
- Most similar to *multi-template*, but much more manageable for complex effects.

New fields added by the pre-calculation

The syllable data pre-calculation function adds a number of new fields to both line tables and syllable tables:

*meta.res_x* (number)
*meta.res_y* (number)
- These two might be zero or undefined in some cases. In these cases, Karaskel mimics VSFilter behaviour, and calculates one from the other, or in the case of both being zero, uses the default of 384x288.

*line.i* (number)
- Index of the line, as counted in the `lines` array. Zero-based.

*line.prev* (table)
*line.next* (table)
- References to the next and previous lines in the file. Allows accessing the lines in a doubly-linked-list manner. Make sure you check these for being non-nil before using them!

*line.styleref* (table)
- Reference to the style table describing the style used by this line. In case the line names an undefined style, the first defined style is picked instead. (This is to avoid seemingly random hard-to-debug crashes. Whenever such a substitution is done, a warning is output.)

*line.text_stripped* (string)
- While this isn't a new field, the meaning of it is changed in the case of furigana being enabled. When furigana is enabled, this is changed to contain only the baseline text, ie. without any furigana text.

*line.width* (number)
*line.height* (number)
*line.ascent* (number)
*line.extlead* (number)
- The simple results of `aegisub.text_extents` being applied to the line, using `line.styleref` for style.
line.centerleft (number)
line.centerright (number)
   Left and right edges of the line, when horizontally centered on screen. Note that these are currently incorrect when the left and right margins are different. Useful in conjunction with syl.left, syl.center and syl.right for per-syllable positioning lines centered on screen.

line.duration (number)
   Duration of the line in milliseconds.

line.ool_fx (table)
   String-indexed table of out-of-line effects. The first word (space-separated) on each ool fx line defines the name of that ool fx, and the name is used as index here. See karaskel.process_lines or the wiki for more information.

syl.i (number)
   Index of the syllable on the line, zero-indexed, with index zero meaning the before-first syllable. (Testing for index zero can be a simple way of doing start-of-line initialisation.)

syl.inline_fx (string)
   Name of the inline fx for this syllable. This is reset to the default per line. If an inline fx isn't defined explicitly for a syllable, it has the same as the previous one. An inline fx is defined with a specially formed override tag, like this: \{-foo\} placed first in the syllable. This defines the inline fx to be "foo". Also see the wiki.

syl.width (number)
syl.height (number)
syl.ascent (number)
syl.extlead (number)
   text_extents results for the syllable.

syl.left (number)
syl.right (number)
syl.center (number)
   Left edge, right edge and center of syllable, relative to the left edge of the line. Note that these are (attempted) corrected for per-syllable positioning, with spaces stripped and other manipulations. The effect of these manipulations aren't extensively tested, but it seems to work correctly.

syl.furigana (table)
   Table with furigana data for this syllable. Only generated when karaskel.
   engage_furigana = true.

syl.highlights (table)
   Table with multiple-highlight data for this syllable. Only generated when karaskel.
   engage_furigana = true.

syl.text_stripped (string)
   Modified the same way as line.text_stripped when furigana is enabled.

syl.start_time (number)
syl.end_time (number)
Start and end times of the syllable, relative to the line start time, in milliseconds.

**Additional options**

The two Karaskel scripts are both based off a base, named `karaskel-base.lua`. Most importantly, this is where the `karaskel.precalc_syllable_data` function is defined, so if you want that function but nothing else of the Karaskel, you only need to include `karaskel-base.lua`.

Furthermore, the Karaskel base defines some variables you can use to customise its behaviour, mostly related to what is pre-calculated, and whether debugging messages should be displayed. The following options are available. These should be set in the script including a Karaskel, as demonstrated in demo 10, furigana.

- **karaskel.engage_positioning** (boolean)
  Enables syllable positioning calculation. This is automatically enabled by `karaskel-adv`.

- **karaskel.precdalc_start_progress** (number)
- **karaskel.precdalc_end_progress** (number)
  These two variables define the range the progress bar covers while syllable data are pre-calculated. They default to 0 and 50.

- **karaskel.engage_furigana** (boolean)
  Enables furigana parsing and calculations. See the wiki page on furigana for more information.

- **karaskel.furigana_scale** (number)
  Defines the scale of furigana text relative to the baseline text size. The default is 0.4, meaning furigana is 40% the size of normal text.

- **karaskel.inline_fx_default** (string)
  Default inline fx name. See the wiki page on inline fx for more information.

- **karaskel.oofx_style** (boolean/string)
  Names the style defining out-of-line effects. Can be either false, to signify that out-of-line effects are not used, or a string naming a style. Defaults to false. Wiki page

- **karaskel.engage_trace** (boolean)
  Whether to show debugging messages. Rarely useful, just leave it at false.

### 5.5.3.1 Basic karaskel

**Implementing in your script**

Include `karaskel.lua` in your own script. This will define the `process_syllables` function, and in turn also include `utils.lua` and `karaskel-base.lua`.

```
include("karaskel.lua")
```
Basic usage

The first and most important thing to do after including karaskel.lua is to define a do_syllable function:

```lua
function do_syllable(meta, styles, config, line, syl)
    return syl.text_stripped
end
```

This is a very minimalistic, boring, tag-stripping do_syllable function. What you want to do in the do_syllable function is to add some tags before the syllable text. See demo 8 for an example of this.

In other words, the basic idea of the do_syllable function is that you provide a replacement text for the syllable in question.

Advanced usage

There are also further functions you can define to modify the behaviour of karaskel.

- **do_line Decide**
  ```lua
do_line Decide(meta, styles, config, line)
    Return a boolean. If you return true, this line is processed, otherwise it's ignored (passed unchanged into output.)
  ```

- **do_line start**
  ```lua
do_line start(meta, styles, config, line)
    Return a string. Adds the returned string to the beginning of the line.
  ```

- **do_line end**
  ```lua
do_line end(meta, styles, config, line)
    Return a string. Adds the returned string to the end of the line.
  ```

- **do_line**
  ```lua
do_line(meta, styles, config, line)
    Return an array of lines. You will usually not need to override this function, only if you need very special behaviour. If you override this function, make sure you read the original function defined in karaskel.lua, or call karaskel.do_line at some point. If you need to override this function, consider using the advanced karaskel instead.
  ```

For an advanced example using karaskel.lua, see simple-k-replacer.lua in the factorybrews directory.

5.5.3.2 Advanced karaskel

Implementing in your script

Include karaskel-adv.lua in your own script. This will define the process_syllables function, and in turn also include utils.lua and karaskel-base.lua.

```lua
include("karaskel-adv.lua")
```

Basic usage

The only thing you need to do after including karaskel-adv.lua is to define a do_syllable function.
function do_syllable(meta, styles, config, line, syl)
  local l = copy_line(line)
  l.text = string.format("{\an\pos(%d,%d)}%s", line.centerleft+syl.center, line.styleref.margin_v syl.text_stripped)
  return { n=1, {1}=l }end

This simple do_syllable function lines of the syllables centered at the top of the screen. See demo 9 for a slightly more advanced example. The line-per-syllable factorybrew might also be useful as an example.

**Advanced usage**

In some cases you might want to do some special per-line processing before syllables are processed. You can override the do_line function for this, just be sure to call karaskel.do_line to return to normal processing afterwards.

function do_line(meta, styles, config, line)
  aegisub.output_debug("Processing line: " .. line.i)
  return karaskel.do_line(meta, styles, config, line)end

If you use *inline fx*, you can consider a trick like this:

function do_syllable(meta, styles, config, line, syl)
  local result = {n=0}
  _G["fx_"..syl.inline_fx](meta, line, syl, result)
  return resultend

function fx_jump(meta, line, syl, result)
  -- do "jump" effect hereend

function fx_slide(meta, line, syl, result)
  -- do "slide" effect hereend
Index

- A -
  Advanced  112

- C -
  C:\Program Files\Aegisub\automation\include\  110

- L -
  line table  110
Endnotes 2... (after index)