
By Ben Hackbarth, Norbert Schnell, Philippe Esling, Diemo Schwarz, Gilbert Nouno.


# Contents

# 1 Installation

The following lists the dependancies of audioguide1.1.0. For the moment audioguide1.1.0 only works on OS X. Note that, regarding numpy, recent versions of OS X (after 10.5) come with python2.7 that automatically has numpy installed. This distribution of audioguide1.1.0 comes with a precompiled version of pysdif for 64-bit python and, on more current machines, should run out of the box. Here is a complete list of the resources that audioguide1.1.0 requires on your computer:

**pysdif** To compile install download the sdif library and configure, make and install. Then download my patched version of the pysdif module. 'cd' into the directory when unzipped and run: 'python2.7 setup.py install'. You're now setup to use pysdif in python2.7.

**numpy** Most python2.7 installations come with numpy, a numerical computation module. Upgrading to the latest python2.7 should get you there. If you don't have it, you can download the source code or a binary installer here.

**csound6** Needed only if you would like audioguide1.1.0 to automatically render concatenations (which you probably do). Download an installer from here.

**matlibplot (optional)** Install this python module to enable graphing of descriptors and soundfile segmentation.

**supervp (optional)** Used to stretch a target before concatenating.

# 2 Quick Start

Using audioguide1.1.0 comes down to interacting with python scripts in the audioguide1.1.0 folder. For soundfile concatenation, one script does segmentation of corpus soundfiles. A second script performs concatenation based on a variety of variables found in a options file. While you do not need to know how to write Python code in order to use audioguide1.1.0, it is not a bad idea to know some of Python's basic syntax.

The reason that segmentation and concatenation are separated into discrete steps that I find is useful to fine-tune the segmentation of corpus sounds *before* using them in a concatenation. Soundfile segmentation is a difficult technical problem and should remain conceptually and aesthetically open-ended. I have yet to find an algorithm that does not require adjustments based on the nature of the sound in question and the intention of the user as to what a segment should be.

## 3 The Concatenation Options File

The options file used by the concatenate script if a python file that defines a bunch of variables. Most variables can be changed with a simple assignments using the '=' symbol. For instance, to change the path of the csound output sound file, write the following in your myOptions.py file:

```
CSOUND_RENDER_FILEPATH = '/path/to/the/file/i/want.aiff' # sets the path of the csound
  output aiff file
DESCRIPTOR_HOP_SIZE_SEC = 0.02049 # change the analysis hop size
```

However, there are five custom objects that are written into the options file as well – 'tsf', 'csf', 'spass', 'd', 'si'. These objects take required parameters and also take keyword arguments. The following sections describe the object-style variables.

## 4 The TARGET Variable and tsf() object

The TARGET variable is written as a tsf() object which requires a path to a soundfile and also takes the following optional keyword arguments:

```
tsf('pathtosoundfile', start=0, end=file-length, thresh=-40, offsetRise=1.5,
 offsetThreshAdd=+12, offsetThreshAbs=-80, scaleDb=0, minSegLen=0.05, maxSegLen=1000,
 midiPitchMethod='composite', stretch=1, segmentationFilepath=None)
```

**start** The time in seconds to start reading the target soundfile.

**end** The time in seconds to stop reading the target soundfile.

**thresh** Segmentation onset threshold: a value from -100 to 0. The lower the value the soft the target's amplitude can be in order to trigger a selection from the corpus. So, -12 yields fewer corpus selections that -24.

**offsetRise** Segmentation offset ratio: a number greater than 1. During segmentation the rise-ratio is the ratio of a frame's amplitude to the next frame's amplitude. In an active segment, if this ratio is greater than user-supplied rise-ratio, it will cause an offset.

**offsetThreshAdd** Segmentation offset relative threshold: a positive value in dB. This value is added to the soundfile's minimum amplitude. During segmentation, if a frame's amplitude is below this threshold, it causes a segment offset. Also see offsetThreshAbs.

**offsetThreshAbs** Segmentation offset absolute threshold: a negative value in dB. When segmenting the target, if a frame's amplitude is below this value it will cause an offset. This variable is an absolute value while offsetThreshAdd is relative to the soundfile's minimum amplitude. Effectively, whichever of these two variable is closer to 0 will be the offset threshold.

**minSegLen** Segmentation: the minimum duration in seconds of a target segment.

**maxSegLen** Segmentation: the maximum duration in seconds of a target segment.

**scaleDb** Applies an amplitude change to the whole target sound. By default, it is 0, yielding no change. -6 = twice as soft. The target's amplitude will usually affect concatenation: the louder the target, the more corpus sounds can be composited to approximate it's energy profile.

**midiPitchMethod** undocumented ! sorry.

**stretch** Uses SuperVp to time stretch/compress the target sound before analysis and concatenation. Stretched sound files are saved and reused in the audioguide/data_stretched_sfs directory. A value of 1, the default, will cause no time stretching to be applied. Note that, in order to use this option, you must set the SUPERVP_BIN variable in the audioguide/defaults.py file.

**segmentationFilepath** by default the Target sound is segmented at runtime. However, if you'd like to specify a user-defined segmentation, you may give a file path to this variable. Note that this file must be a textfile with the same format as corpus segmentation files.

```
TARGET = tsf('cage.aiff') # uses the whole soundfile at its given amplitude
TARGET = tsf('cage.aiff', start=5, end=7, scaleDb=6) # only use seconds 5-7 of cage.
 aiff at double the amplitude.
```

# 5 The CORPUS Variable and csf() object

The CORPUS variable is defined as a list of csf() objects which require a path to a soundfile OR a directory. File paths and/or directory paths may be full paths or relative paths to the location of the options file you're using. A csf() object takes the following optional keyword arguments:

```
csf('pathToFileOrDirectoryOfFiles', allowRepetition=True, concatFileName=None, end=
 None, envelopeSlope=1, excludeStr=None, excludeTimes=[], hasParams=False,
 includeStr=None, includeTimes=[], limit={}, limitDur=None, midiPitchMethod='
 composite', offsetLen='30\%', onsetLen=0.01, recursive=True, restrictInTime=0,
```

```
restrictOverlaps=None,  restrictRepetition=0.1,  scaleDb=0.0,  scaleDistance=1,
postSelectAmpBool=False, postSelectAmpMin=-12, postSelectAmpMax=+12,
postSelectAmpMethod='power-mean-seg', segmentationExtension='.txt',  segmentationFile
=None,  start=None,  superimposeRule=None,  transMethod=None,  transQuantize=0,
wholeFile=False, MWinstrName=None,  MWtext=None, MWnotehead='.')
```

The simplest way to include a soundfile in your corpus is to use its path as the first argument of the 'csf' object:

```
CORPUS = [csf("lachenmann.aiff")] # will search for a segmentation file called
lachenmann.aiff.txt and add all of its segments to the corpus
```

The simplest way to include a directory of soundfiles in your corpus is to use its path as the first argument of the 'csf' object:

```
CORPUS = [csf("lachenmann.aiff"), csf('piano')] # will use segments from lachenmann.
aiff as well as all sounds in the directory called piano
```

However, as you can see above, each 'csf' object has *a lot* of optional arguments to give you better control over what segments are used, how directories are read and how segments are treated during concatenation.

*Note: Each of these keyword arguments only apply to the csf object within which they are written. If you'd like to specify these parameters for the entire corpus, see section 4.2.5: Specifying CORPUS entry attributes globally.*

## 5.1 Manipulating which segments are added to the corpus

**start** Any segments which start before this time will be ignored.

**end** Any segments which start after this time will be ignored.

```
csf('lachenmann.aiff', start=20) # only use segments who start later than 20s.
csf('lachenmann.aiff', start=20, end=50) # only use segments who start between 20-50s.
```

**includeTimes** A list of two-number lists which specify regions of segments to include from this file's list of segment times. See example below.

**excludeTimes** Same as includeTimes but excludes segments in the identified regions.

```
csf('lachenmann.aiff', includeTimes=[(1, 4), (10, 12)]) # only use segments falling
between 1-4 seconds and 10-12 seconds.
csf('lachenmann.aiff', excludeTimes=[(30, 55)]) # use all segments except those
falling between 30-55s.
```

**limit** A list of equation-like strings where segmented descriptor names are used to include/exclude segments from this file / directory.

```
csf('lachenmann.aiff', limit['centroid-seg >= 1000']) # segments whose centroid-seg is equal to or above 1000.

csf('lachenmann.aiff', limit['centroid-seg < 50%']) # only use 50% of segments with the lowest centroid-seg.

csf('lachenmann.aiff', limit['power-seg < 50%', 'power-seg > 10%']) # only use segments whose power-seg falls between 10%-50% of the total range of power-seg's in this file/directory.
```

## 5.2 Manipulating How Directories Are Read

The following keyword arguments are useful when dealing with directories of files.

**wholeFile** (False): if True audioguide1.1.0 will use this soundfile as one single segment. If False, audioguide1.1.0 will search for a segmnetation file made with agSegmentSf.py.

**recursive** (True): if True audioguide1.1.0 will include sounds in all subfolders of a given directory.

```
csf('sliced/my-directory', wholeFile=True) # will not search for a segmentation txt file, but use whole soundfiles as single segments.

csf('/Users/ben/gravillons', recursive=False) # will only use soundfiles in the named folder, ignoring its subdirectories.
```

**includeStr** (None): A string which is matched against the filename (not full path) of each soundfile in a given directory. If part of the soundfile name matches this string, it is included. If not it is excluded. This is case sensitive. See example below.

**excludeStr** (None): Opposite of includeStr.

```
# includeStr/excludeStr have lots of uses. One to highlight here: working with sample databases which are normalized. Rather than having each corpus segment be at 0dbs, we apply a scaleDb value based on the presence of a `dynamic' written into the filename.
csf('Vienna-harpNotes/', includeStr=['_f_', '_ff_'], scaleDb=-6),
csf('Vienna-harpNotes/', includeStr='_mf_', scaleDb=-18),
csf('Vienna-harpNotes/', includeStr='_p_', scaleDb=-30),
# this will use all sounds from this folder which match one of the three dynamics.
```

### 5.3 Manipulating How Segments Will Be Concatenated

**scaleDb** applies an amplitude change to each segment of this collection. by default, it is 0, yielding no change. -6 = twice as soft. Note that amplitude scaling affects both the concatenative algorithm and the csound rendering.

**onsetLen** if onsetLen is a float or integer, it is the fade-in time in seconds. If it is a string formed as '10%', it is interpreted as a percent of each segment's duration. So, onsetLen=0.1 yields a 100 ms. attack envelope while onsetLen='50%' yields a fade in over 50% of the segment's duration.

**offsetLen** Same as onsetLen, but for the envelope fade out.

```
csf('lachenmann.aiff', onsetLen=0.1, offsetLen='50%') # will apply a 100ms fade in
 time and a fade out time lasting 50% of each segments' duration.
```

**transMethod** A string indicating how to transpose segments chosen from this corpus entry.

**transQuantize** Quantization interval for transposition of corpus sounds. 1 will quantize to semitones, 0.5 to quarter tones, 2 to whole tones, etc.

```
csf('piano/', transMethod='f0') # transpose corpus segments to match the target's f0.
csf('piano/', transMethod='f0-chroma', transQuantize=0.5) # transpose corpus segments
 to match the target's f0 mod 12.  Then quantize each resulting pitch to the newest
 quarter of tone.
```

**allowRepetition** If False, any of the segments from this corpus entry may only be picked one time. If True there is no restriction.

**restrictRepetition** A delay time in seconds where, once chosen, a segment from this corpus entry is invalid to be picked again. The default is 0.1, which the same corpus segment from being selected in quick succession.

```
csf('piano/', allowRepetition=False) # each individual segment found in this directory
  of files may only be deleted one time during concatenation.
csf('piano/', restrictRepetition=2.5) # Each segment is invalid to be picked if it has
  already been selected in the last 2.5 seconds.
```

**restrictOverlaps** An integer specifying how many overlapping samples from this collection may be chosen by the concatenative algorithm at any given moment. So, restrictOverlaps=2 only permits 2 overlapping voices at a time.

**restrictInTime** a time in seconds specifying how often a sample from this entry may be selected. – for example restrictInTime=0.5 would permit segments from this collection to be select a maximum of once every 0.5 seconds.

**superimposeRule** This one is a little crazy. Basically, you can specify when this corpus's segments can be chosen based on the number of simultaneously selected samples. You do this by writing a little equation as a 2-item list. superimposeRule=('==', 0) says that this set of corpus segments may only be chosen is this is the first selection for this target segment (sim selection '0'). superimposeRule=('¿', 2) say this corpus's segments are only valid to by picked if there are already more than 2 selections for this target segment. I know, right?

### 5.4 Miscellaneous

**segmentationFile** Manually specify the segmentation text file. By default, audioguide1.1.0 automatically looks for a file with the same name as the soundfile plus the extension '.txt'. You may specify a path file (as a string), or a list of strings to include multiple segmentation files which all use the same soundfile.

**segmentationExtension** Manually specify the segmentation text file extension. See above.

```
csf('lachenmann.aiff', segmentationFile='marmotTent.txt') # will use a segmentation
 file called marmotTent.txt, not the default lachenmann.aiff.txt.

csf('lachenmann.aiff', segmentationExtension='-gran.txt') # will use a segmentation
 file called lachenmann.aiff-gran.txt, not the default lachenmann.aiff.txt.
```

### 5.5 Specifying csf() attributes globally

csf() attributes may be specified globally using the variable CORPUS_GLOBAL_ATTRIBUTES. Note that they are specified in dictionary format rather than keyword format.

```
CORPUS = [csf('lachenmann.aiff', scaleDb=-6), csf('piano/', scaleDb=-6, wholeFile=True
 )]

# is equivalent to

CORPUS = [csf('lachenmann.aiff'), csf('piano/', wholeFile=True)]
CORPUS_GLOBAL_ATTRIBUTES = {'scaleDb': -6}
```

## 6 SEARCH variable and spass() object

The SEARCH variable specifies how audioguide1.1.0 picks corpus segments to match target segments. The idea here is make a *very* flexible searching structure where the user can create multiple search passes on different descriptor criteria.

The SEARCH variable is written as a list of spass() objects. Each spass() has the following parameters:

```
spass(search_type, descriptor1... descriptorN, percent=None, minratio=None, maxratio=None)
```

And the search_type string may be among the following methods:

**closest** Return the best matching segment.

**closest_percent** Return the top *percent* percent of the best matching segments.

**farthest** Return the worst matching segment.

**farthest_percent** Return the worst *percent* percent of segments.

**ratio_limit** Return segments where the ratio of the target descriptor value to the segment's value falls between minratio and maxratio. Only works for averaged descriptors.

Here is the most simple case of a SEARCH variable:

```
SEARCH = [spass('closest', d('centroid'))] # will search all corpus segments and select the one with the `closest' centroid to the target segment.
```

Note that the first argument is the type of search performed – in this case, selecting the closest sample. Following the arguments are a list of descriptor objects which specify which descriptors to use. Finally there are some keyword parameters that we will touch on later.

```
SEARCH = [spass('closest', d('centroid'), d('effDur-seg'))] # will search all corpus segments and select the one with the `closest' centroid and effective duration compared to the target segment.
```

Ok, great. As you can probably imagine, the first argument, 'closest', tells audioguide1.1.0 to pick the closest sound. But, there are also other possibilities:

```
SEARCH = [spass('farthest', d('centroid'))] # return the worst matching segment.

SEARCH = [spass('closest_percent', d('centroid'), percent=20)] # return the top 20 percent best matches.

SEARCH = [spass('farthest_percent', d('centroid'), percent=20)] # return the worst 20 percent of matches.
```

If you use 'closest_percent' or 'farthest_percent' as the one and only spass object in the SEARCH variable, audioguide1.1.0 will select a corpus segment randomly among the final candidates. However, you can also chain spass objects together, essentially constructing a hierarchical search algorithm. So, for example, take the following SEARCH variable with two separate phases:

```
SEARCH = [
spass('closest_percent', d('effDur-seg'), percent=20), # take the best 20% of matches
 from the corpus
spass('closest', d('mfccs')), # now find the best matching segment from the 20 percent
  that remains.
]
```

I use the above example a lot when using audioguide1.1.0. It first matches effDur-seg, the effective duration of the target measured agains't the effective duration of each corpus segment. It retains the 20% closest matches, and throws away the worst 80%. Then, with the remaining 20%, the timbre of the sounds are matched according to mfccs.

```
SEARCH = [spass('ratio_limit', d('centroid-seg'), minratio=0.9, maxratio=1.1)] #
 reduce the number of samples in the corpus such
```

Remember, the order of the spass objects in the SEARCH variable is very important – it is essentially the order of operations.

# 7 The d() object

Use the 'd' object for specifying a descriptor in the SEARCH variable.

```
d('descriptor name', weight=1, norm=2, normmethod='stddev', distance='euclidean',
 energy=False)
```

**weight** How to weight this descriptor in relation to other descriptors.

```
SEARCH= [spass('closest', d('centroid', weight=1), d('noisiness', weight=0.5))]
 # centroid is twice as important as noisiness.
```

**norm** A value of 2 normalizes the target and corpus data separately. A value of 1 normalizes the target and corpus data together. 2 will yield a better rendering of the target's morphological contour. 1 will remain more faithful to concrete descriptor values. I recommend using 2 by default, only using 1 when dealing with very 'descriptive' descriptors like duration or pitch.

```
SEARCH= [spass('closest', d('centroid'), d('effDur-seg', norm=1))]
```

**normmethod** How to normalize data – either 'stddev' or 'minmax'. minmax is more precise, stddev is more forgiving of 'outliers.'

**distance** Only valid for time-varying descriptors. How to arithmetically evaluate distance between continuously valued array. 'euclidean' does a simple least squares distance. Other methods include 'pearson'.

```
SEARCH= [spass('closest', d('centroid', distance='pearson'))] # uses a pearson
 correlation formula for determining distance between the continuously valued
 centroid of target and corpus segments.
```

**energy** Only valid for time-varying descriptors. Weight distance calculations with the corpus segments energy values. So, softer frames will not penalize distance.

## 8 The SUPERIMPOSE variable and si() object

Use the 'si' object for specifying how corpus segments may be superimposed during concatenation.

```
SUPERIMPOSE = si(minSegment=None, maxSegment=None, minOnset=None, maxOnset=8,
 minOverlap=None, maxOverlap=None, searchOrder='power', calcMethod='mixture',
 peakAlign=False)
```

**minSegment** The minimum number of corpus segments that must be chosen to match a target segment.

**maxSegment** The maximum number of corpus segments that must be chosen to match a target segment.

**minOnset** The minimum number of corpus segments that must be chosen to begin at any single moment in time.

**maxOnset** The maximum number of corpus segments that must be chosen to begin at any single moment in time.

**minOverlap** The minimum number of overlapping corpus segments at any single moment in time. Note that an 'overlap' is determined according to an amplitude threshold – see overlapAmpThresh.

**maxOverlap** The maximum number of overlapping corpus segments at any single moment in time. Note that an 'overlap' is determined according to an amplitude threshold – see overlapAmpThresh.

**searchOrder** ('power' or 'time') The default is 'time', which indicated to match corpus segments to target segments in the temporal order of the target (i.e., first searched segment is the first segment in time). 'power' indicates to first sort the target segments from loudest to softest, then search for corpus matches.

**calcMethod** A None/string which denotes how to calculate overlapping corpus sounds. None does nothing – each corpus selection is unaware of previous selections. ‘subtract’ subtracts the energy of a selected corpus sound from the target’s amplitude so that future selections factor in the amplitude of past selections. ‘mixture’ subtracts the amplitude and then attempts to mix the descriptors of simultaneous sounds together. Note that some descriptors are not algorithmically mixable, such as f0, zeroCross.

# 9 Other Options

## 9.1 Descriptor Computation Parameters

**DESCRIPTOR_FORCE_ANALYSIS** (type=bool, default=False) if True, audioguide1.1.0 is forced to remake all SDIF analysis, even if previously made.

**DESCRIPTOR_WIN_SIZE_SEC** (type=float, default=0.04096) the FFT window size of descriptor analysis in seconds. 0.04096 seconds = 1024 @ 25kHz (the default resample rate).

**DESCRIPTOR_HOP_SIZE_SEC** (type=float, default=0.01024) the FFT window overlaps of descriptor analysis in seconds. Important, as it effectively sets of temporal resolution of audioguide1.1.0.

**IRCAMDESCRIPTOR_RESAMPLE_RATE** (type=int, default=25000) The internal resample rate of the IRCAM analysis binary. Important, as it sets the frequency resolution of spectral sound descriptors.

**IRCAMDESCRIPTOR_WINDOW_TYPE** (type=string, default=blackman) see ircamdescriptor documentation for details.

**IRCAMDESCRIPTOR_NUMB_MFCCS** (type=int, default=13) sets the number of MFCCs to make. *Doesn’t seem to work* in this release, as 13 is all I can ever seem to get out of the library.

**IRCAMDESCRIPTOR_F0_MAX_ANALYSIS_FREQ** (type=float, default=5000) see ircamdescriptor documentation for details.

**IRCAMDESCRIPTOR_F0_MIN_FREQUENCY** (type=float, default=20) minimum possible f0 frequency.

**IRCAMDESCRIPTOR_F0_MAX_FREQUENCY** (type=float, default=5000) maximum possible f0 frequency.

**IRCAMDESCRIPTOR_F0_QUALITY** (type=float, default=0.1) see ircamdescriptor documentation for details.

**SUPERVP_BIN** (type=string/None, default=None) Optionally specify a path to the supervp analysis binary. Used for target pre-concatenation time stretching.

**PM2_BIN** (type=string/None, default=None) Optionally specify a path to the pm2 analysis binary. Currently not implemented.

## 9.2 Concatenation

**ROTATE_VOICES** (type=bool, default=False]) if True, audioguide1.1.0 will rotate through the list of corpus entries during concatenation. This means that, when selecting corpus segment one, audioguide1.1.0 will only search sound segments from the first item of the CORPUS variable. Selection 2 will only search the second, and so on. Corpus rotation is modular around the length of the CORPUS variable. If the corpus only has one item, True will have no effect.

**VOICE_PATTERN** (type=list, default=[]) if an empty list, this does nothing. However, if the user gives a list of strings, audioguide1.1.0 will rotate through this list of each concatenative selection and only use corpus segments who's filepath match this string. Matching can use parts of the filename, not necessarily the whole path and it is not case sensitive.

**OUTPUT_QUANTIZE_TIME_METHOD** (type=string/None, default=None) controls the quantisation of the start times of events selected during concatenation. If OUTPUT_QUANTIZE_TIME_METHOD = None, no quantisation takes place. However, a value of 'snapToGrid' will conform the start time's of events to a grid spaced in OUTPUT_QUANTIZE_TIME_INTERVAL second slices. Additionally, a value of 'medianAggregate' will change each event's start time to the median start time of events in slices of OUTPUT_QUANTIZE_TIME_INTERVAL seconds. Any quantisation takes place after the application of OUTPUT_TIME_STRETCH and OUTPUT_TIME_ADD, as detailed below.

**OUTPUT_QUANTIZE_TIME_INTERVAL** (type=float, default=0.25) defines the temporal interval in seconds for quantisation. If OUTPUT_QUANTIZE_TIME_METHOD = None, this doesn't do anything.

**OUTPUT_GAIN_DB** (type=int/None, default=None) adds a uniform gain in dB to all selected corpus units. Affects the subtractive envelope calculations and descriptor mixtures as well as csound rendering.

**OUTPUT_TIME_STRETCH** (type=float, default=1.) stretch the temporality of selected units. A value of 2 will stretch all events offsets by a factor of 2.

**OUTPUT_TIME_ADD** (type=float, default=0.) offset the start time of selected events by a value in seconds.

**RANDOM_SEED** (type=int/None, default=None) sets the pseudo-random seed for random unit selection. By default a value of None will use the system's timestamp. Setting an integer will create repeatable random results.

## 9.3 Concatenation Output Files

For each of the following *_FILEPATH variables, a value of None tells the agConcatenate.py NOT to create an output file. Otherwise a string tells agConcatenate.py to create this output file and also indicates the path of the file to create. Strings may be absolute paths. If a relative path is given, audioguide1.1.0 will create the file relative to the location o the agConcatenate.py script.

**CSOUND_CSD_FILEPATH** (type=string/None, default=output/output.csd) creates an output csd file for rendering the resulting concatenation with csound.

**CSOUND_RENDER_FILEPATH** (type=string/None, default=output/output.aiff) sets the sound output file in the CSOUND_CSD_FILEPATH file. This is the name of csound's output soundfile and will be created at the end of concatenation.

**MIDI_FILEPATH** (type=string/None, default=output/output.mid) a midi file of the concatenation with pitches chosen according to midiPitchMethod from each corpus entry.

**SUPERIMPOSITION_LABEL_FILEPATH** (type=string/None, default=output/superimpositionlabels.txt) Audacity-style labels showing the selected corpus sounds and how they overlap.

**LISP_OUTPUT_FILEPATH** (type=string/None, default=output/output.lisp.txt) a textfile containing selected corpus events as a lisp-style list.

**DATA_FROM_SEGMENTATION_FILEPATH** (type=string/None, default=None) This file lists all of the extra data of selected events during concatenation. This data is taken from corpus segmentation files, and includes everything after the startTime and endTime of each segment. This is useful if you want to tag each corpus segment with text based information for use later.

**DICT_OUTPUT_FILEPATH** (type=string/None, default=output/output.json) a textfile containing selected corpus events in json format.

**MAXMSP_OUTPUT_FILEPATH** (type=string/None, default=output/output.maxmsp.json) a textfile containing a list of selected corpus events. Data includes starttime in MS, duration in MS, filename, transposition, amplitude, etc.

## 9.4 Other Output Files

For each of the following *_FILEPATH variables, a value of None tells the agConcatenate.py NOT to create an output file. Otherwise a string tells agConcatenate.py to create this output file and also indicates the path of the file to create. Strings may be absolute paths. If a relative path is given, audioguide1.1.0 will create the file relative to the location o the agConcatenate.py script.

**LOG_FILEPATH** (type=string/None, default=output/log.txt) a log file with lots of information from the concatenation algorithm.

**TARGET_SEGMENT_LABELS_FILEPATH** (type=string/None, default=output/targetlabels.txt) Audacity-style labels showing how the target sound was segmented.

**TARGET_SEGMENTATION_GRAPH_FILEPATH** (type=string/None, default=None) like TARGET_SEGMENT_LABELS_FILEPATH, this variable creates a file to show information about target segmentation. Here however, the output is a jpg graph of the onset and offset times and the target's power. This output requires you to install python's module matplotlib.

**TARGET_DESCRIPTORS_FILEPATH** (type=string/None, default=targetdescriptors.json) saves the loaded target descriptors to a json dictionary.

**TARGET_PLOT_DESCRIPTORS_FILEPATH** (type=string/None, default=None) creates a plot of each target descriptor used in concatenation. Doesn't work for averaged descriptors ("-seg"), only time varying descriptors.

## 9.5 Printing/Interaction

**SEARCH_PATHS** (type=list, default=[]) a list of strings, each of which is a path to a directory where soundfile are located. These paths extend the list of search paths that audioguide1.1.0 examines when searching for target and corpus soundfiles. The default is an empty list, which doesn't do anything.

**ALERT_ON_ERROR** (type=bool, default=False) will attempt to play the system alert sound when exiting with an error. Pretty irritating, but then again you probably deserve it.

**VERBOSITY** (type=int, default=2) affects the amount of information audioguide1.1.0 prints to the terminal. A value of 0 yields nothing. A value of 1 prints a minimal amount of information. A value of 2 (the default) prints refreshing progress bars to indicate the progress of the algorithms.

**PRINT_SELECTION_HISTO** (type=bool, default=False) if True will print robust information about corpus selection after concatenation. If false (the default) will add this information to the log file, if used.

**PRINT_SIM_SELECTION_HISTO** (type=bool, default=False) if True will print robust information about corpus overlapping selection after concatenation. If false (the default) will add this information to the log file, if used.

### 9.6 Csound Rendering

**CSOUND_SR** (type=int, default=48000) The sample rate used for csound rendering. Csound will interpolate the sample rates of all corpus files to this rate. It will be the sr of csound's output soundfile.

**CSOUND_KR** (type=int, default=128) The control rate used for csound rendering. See csound's documentation for more information.

**CSOUND_CHANNEL_RENDER_METHOD** (type=string, default="mix") Tells audioguide1.1.0 how deal with corpus segments distribution in the output soundfile By default''mix" creates a 2 channel csound file and puts mono corpus sounds in the middle of the stereo field. The string "oneChannelPerVoice" tells audioguide to put selected sounds from each item of the CORPUS list into a separate channel. The number of output channels will therefore equal the length of the CORPUS list variable.

**CSOUND_STRETCH_CORPUS_TO_TARGET_DUR** (type=string/None, default=None) Affects the durations of concatenated sound events rendered by csound. By default None doesn't do anything – csound plays back each corpus sound according to its duration. "pv" uses a phase vocoder to stretch corpus sounds to match the duration of the corresponding target segment. "transpose" does the same, but using the speed of playback to change duration rather than a phase vocoder. Note that, in this case, any other transposition information generated by the selection algorithm is overwritten.

**CSOUND_PLAY_RENDERED_FILE** (type=bool, default=True) if True, audioguide1.1.0 will play the rendered csound file at the command line at the end of the concatenative algorithm.