CDP HOUSEKEEP (SF Utils) Functions

(with Command Line Usage)

Functions to HOUSEKEEP (SF Utils) soundfiles

(Names in brackets mean that these are separate programs. The others are sub-modules of HOUSEKEEP.)

BAKUP
Concatenate soundfiles in one backup file, with silences inbetween

BATCHEXPAND
Expand an existing batch file

BUNDLE
List files into a textfile, for sorting, backup or creating a dummy mixfile

[CHANPHASE]
Invert the phase of a specified channel of an input sound

CHANS
Extract channels or change channel format of a soundfile

COPY
Make and delete exact copies of a sound

DEGLITCH
Attempt to deglitch a sound

DISK
Show available space on disk

ENDCLICKS
Remove clicks from the start or end of a soundfile

EXTRACT
Extract significant sound from a recording, top & tail, remove glitches, etc.

GATE
Chop at zeros

[GATE]
Remove low-level sound from signal

REMOVE
Remove existing copies of a file

RESPEC
Change sample rate, format or properties of a soundfile (CARE!!)

SORT
Sort files listed in a textfile

ALSO SEE:

CHANNELX
: Extract all or selected channels from a (multi-channel) soundfile

COPYSFX
: Copy/convert soundfiles

TOPANTAIL2
: Fade beginning & end of a sound

UTILS Group
: Sytem utilities

HOUSEKEEP BAKUP – Concatenate soundfiles in one backup file, with silences inbetween

Usage

housekeep bakup *infile1 [infile2...] outfile*

Parameters

infile1 – first sound to back up
infile2 ... – second and additional sounds to back up
outfile – concatenated soundfile

Understanding the HOUSEKEEP BAKUP Function

The input sounds are assembled into a single file in the same order that they are listed, with a few seconds of silence inbetween each file.

It is possible to have only one input soundfile to back up. This wouldn't normally happen, but you could be backing up groups of files to named backup files, and one of these groups might have only one member.

Musical Applications

HOUSEKEEP BAKUP is similar to SFEDIT JOIN except that silences are placed between the sounds. This makes it easy to transfer a list of soundfiles to tape or CD, to be listened to as separate soundfiles.

End of HOUSEKEEP BAKUP

HOUSEKEEP BATCHEXPAND – Expand an existing batch file

Usage

housekeep batchexpand mode *batchfile sndfile1 [infile2...] datafile ic oc pc*

Modes

A new batchfile is created, modelled on the original...
1 – In Mode 1:

1. The **existing *batchfile*** must operate on only one soundfile as input to the process(es).
2. The same sequence of operations is applied to each *in-soundfile* using each new parameter in turn.
3. Thus a 3-line *batchfile* with 3 sound inputs becomes a 9-line file.

2 – In Mode 2:

1. The **existing batchfile** can have only a single line
2. The soundfile in the original *batchfile* is replaced by the chosen input soundfile(s), using new parameters.
3. Thus the 1-line *batchfile*, when given 3 sound inputs, applies its process to each soundfile in turn, producing 3 outputs. This implies the presence of 3 different values in the *datafile*.

Parameters

batchfile – an existing batchfile (**.bat**) that uses the same kind of parameter in one of its columns. Whether the programs called on different lines of the batchfile are the same or different, the same parameter must be in the same column for BATCHEXPAND to work.
sndfile1 – name of a soundfile in the batchfile, different to or identical with the names in the **existing batchfile**
[*sndfile2 ...*] – any number of additional soundfile names
datafile – text file of parameter values, where the 1st parameter applies to the 1st soundfile, the 2nd parameter applies to the 2nd soundfile, etc.
ic – the number of the column in the *batchfile* containing the input filename (usually '2')
oc – the number of the column in the *batchfile* containing the output filename (usually '3')
pc – the number of the column in the *batchfile* containing the parameter to be replaced (usually '4' ...)

> Column numbering will usually begin at 4: '1' is the program name & mode, '2' is the *infile*, '3' is the *outfile*, and '4' is, therefore, the first parameter.

Understanding the HOUSEKEEP BATCHEXPAND Function

Mode **1** produces variants of the one input soundfile. Here is a typical sequence:

```
Existing Batchfile:
modify speed 2 infile outfile -3  [one input soundfile]

Soundfile Inputs to HOUSEKEEP BATCHEXPAND:
filea fileb  [two or more infiles are OK]

Datafile:  [two different values for the same parameter]
 7
10
```

The resulting output batchfile will look like this:

```
modify speed 2 filea outfile1  7  [each parameter value is applied to filea]
modify speed 2 filea outfile2 10
modify speed 2 fileb outfile3  7  [each parameter value is applied to fileb]
modify speed 2 fileb outfile4 10
```

You could also use Mode **1** to produce variants of a soundfile. There could be one process and one soundfile in the **existing batchfile**, and one *infile* given to BATCHEXPAND, but e.g., 5 different parameter values in the *datafile*. 5 different variants of *infile* would be produced.

Mode **2** deals with existing *batchfiles* which have only a single line, but several *infiles* may be given to it to process. If, for example, you have 3 soundfiles and 3 parameters in the **existing batchfile**, you create a *datafile* with 3 values and these replace those in the batchfile, producing 3 output soundfiles. A typical sequence should look something like this:

```
Existing Batchfile:
modify speed 2 sound1 outfile1 2  [only a single line]

Input soundfiles to HOUSEKEEP BATCHEXPAND:
infilea infileb infilec  [3 infiles, one could be the same
                          as that in the batchfile]

Datafile:  [3 different parameter values are supplied]
 7
10
-5
```

The resulting output batchfile will look like this:

```
modify speed 2 infilea outfile1  7 [each parameter value is
modify speed 2 infileb outfile2 10 applied once to each infile
modify speed 2 infilec outfile3 -5 in the same order]
```

Musical Applications

Given the restrictions on matching formats, HOUSEKEEP BATCHEXPAND can help to speed up ringing the changes on exploring variant soundfile transformations.

- In Mode **1** you just have to alter the *datafile* and supply the names of the input soundfiles you want to process. It then processes each input soundfile with each of the parameter values in the *datafile*.

- In Mode **2**, the first parameter is applied to the first process and soundfile, the second to the second process and soundfile (could be the same process), etc. Your soundfile inputs supply different soundfiles to the existing *batchfile*.

End of HOUSEKEEP BATCHEXPAND

HOUSEKEEP BUNDLE – List files into a textfile, for sorting, backup or creating a dummy mixfile

Usage

housekeep bundle mode *infile* [*infile2 ...*] *outtextfile*

Modes

1 Bundle all files entered
2 Bundle all non-text files entered
3 Bundle all non-text files of same type as first non-text file, e.g., all soundfiles or all analysis files, etc.
4 As **3** but only files with the same properties
5 As **4** but if file1 is a soundfile, files with the same channel count only

Parameters

infile – any type of file with any extension, e.g., asound.wav, atext.txt, amix.mix
infile2 ... – continued series of file inputs
outtextfile – textfile containing a grouped list of all the *infiles*

Understanding the HOUSEKEEP BUNDLE Function

You enter your all your filenames on the command line, and the program groups them according to the Mode instructions and writes an appropriate text file.

Musical Applications

This facility could be used to document a project or to create the first stage of a mixfile. SUBMIX DUMMY can then be used to take a list of sounds a step further by creating a dummy mixfile – 'dummy' because it is presumed that some further editing will be required. When entering soundfile etc. names, you don't need to include the extension if the environment variable CDP_SOUND_EXT has been set. **HOUSEKEEP BUNDLE and SUBMIX DUMMY make a great time-saving combination when creating mixfiles.**

SUBMIX DUMMY no longer takes a text file as an input: the files are listed on the command line (the graphic user interfaces have their own ways of doing this). The text file created in HOUSEKEEP BUNDLE can therefore only be used as a reference guide when entering the filenames.

End of HOUSEKEEP BUNDLE

CHANPHASE – Invert the phase of a specified channel of an input sound

Usage

chanphase chanphase *infile outfile channel*

Example command line to invert a channel :

```
chanphase chanphase insndfile outsndfile 2
```

Parameters

infile – input soundfile
outfile – output soundfile with one channel phase-inverted
channel – channel to invert

Understanding the CHANPHASE Process

This program seeks to rectify a problem encountered by a couple of students at Durham University. They had bought what seemed to be stereo digital recorders, but the output display on *Sound Loom's* SOUND VIEW showed no display.

The SOUND VIEW display is a mono display. To display stereo signals it mixes the 2 channels to mono. The display was empty because the recorders were generating 2 identical mono, phase-inverted, signals. This may have been a manufacturing fault or a misrepresentation on the part of the manufacturer. Who knows?!

Musical Applications

CHANPHASE allows you to phase-invert one of the channels so that is no longer identical with the other one and will therefore display in SOUND VIEW. It is therefore a utility program with a specific purpose. At the time of writing, it is not available in *Sound Loom* on the MAC.

End of CHANPHASE CHANPHASE

HOUSEKEEP CHANS – List channels or change channel format of a soundfile

Usage

housekeep chans 1 *infile channo*
housekeep chans 2 *infile*
housekeep chans 3 *infile outfile channo*
housekeep chans 4 *infile outfile* **-p**
housekeep chans 5 *infile outfile*

> **Release 6**: HOUSEKEEP CHANS Mode **4** now accepts a multi-channel *infile* and mixes it to a MONO *outfile*. The **-p** (invert phasing) flag is ignored.

Modes & Parameters

1 Extract a channel

> *channo* is channel to extract
> *outfile* is named *inname_c1* (for channel 1) etc.

2 Extract all channels

> *outfiles* are named *inname_c1* (for channel 1) etc.

3 Zero one channel

> *channo* is channel to zero
> a mono file goes to just one side of a stereo *outfile*
> a stereo file has one channel zeroed out

4 Stereo to Mono

> The **-p** option inverts the phase of the 2nd channel before mixing

5 Mono to Stereo

> Creates a 2-channel equivalent of a mono *infile*

Understanding the HOUSEKEEP CHANS Process

Soundfiles may have one channel (mono), two channels (stereo), four channels (quad) and possible 6 or 8 channels. Each channel is a different stream of data. The streams may duplicate each other or have different contents. The data for the various channels is interleaved in a multichannel soundfile; for example, the samples of a quad file will be ordered: ch1-ch2-ch3-ch4 ch1-ch2-ch3-ch4 etc. HOUSEKEEP CHANS can take apart this data, or merge a stereo file to form a mono file. You can also piece together single channels of data to make real multi-channel files using SUBMIX INTERLEAVE.

Mode **1** will take out, for example, all the ch2's and stream them together into a single channel (mono) soundfile – whichever channel is specified by *channo*.

Mode **2** will take apart all the channels present and create as many separate single channel (mono) soundfiles as there are channels.

Mode **3** will create a stereo soundfile with one empty channel. With a mono input, *channo* will be empty in the stereo output and the other channel will contain the original mono input. With a stereo input, the *channo* side will be zeroed.

Mode **4** will change a stereo file into a mono file – often invoked when it is found that a certain function will only accept mono inputs.

> **Release 6**: HOUSEKEEP CHANS Mode **4** now accepts a multi-channel *infile* and mixes it to a MONO *outfile*. The **-p** (invert phasing) flag is ignored.
>
> In *SoundLoom* this process is used when mono **thumbnails** of multichannel files are generated. This can be done from the **Workspace** menus, or when **Sound View** is called. (See the *SoundLoom* update information for more details). It is not directly accessible as a normal CDP process on the Loom.

Mode **5** will change a mono file into a stereo file, with both channels containing identical data.

NB: HOUSEKEEP CHANS does **not** extract channels from analysis files.

Musical Applications

Besides the obvious 'housekeeping' chores of Modes **4** & **5**, the other Modes make it possible to dissect and rearrange channel data from different soundfiles. Rejoining separated channels is done with **SUBMIX INTERLEAVE** or **INTERLX**.

The primary application involves processing the extracted channels separately. The logic of using HOUSEKEEP CHANS and SUBMIX INTERLEAVE rather than a mixing routine would seem to be that the intended output is conceived of as a single, more complex sound. For example, it is possible to create pulsations of different frequencies with DISTORT ENVEL. An effective way to do this is to split a stereo file into separate channels (HOUSEKEEP CHANS Mode **2**), process each with different rates of pulsation (DISTORT ENVEL), and then reunite them as a single stereo file with phased pulsations (SUBMIX INTERLEAVE).

HOUSEKEEP CHANS 2 also provides a means of processing stereo or multi-channel files where the CDP function accepts only mono input. Split the file into individual channels, process these separately and re-combine the results by interleaving.

ALSO SEE: **CHANNELX** – extracts all or selected channels from a (multi-channel) soundfile.

End of HOUSEKEEP CHANS

HOUSEKEEP COPY – Make and delete exact copies of a sound

Usage

housekeep copy 1 *infile outfile*
OR: **housekeep copy 2** *infile count* [**-i**]

Modes

1 Copy once: make one copy of *infile* named *outfile*
2 Copy many: produce *count* copies of *infile*

If *infile* is named *X*, the *outfiles* will be named *X_001*, *X_002*, etc.

Parameters

infile – input soundfile to be copied
outfile – name in Mode **1** of the single copy to be made
count – number of copies to make in Mode **2**
-i – option to ignore existing duplicates: i.e., don't overwrite them. The Default is to stop the copying process upon finding a pre-existing soundfile, i.e., one already named like the copies being made.
-a – check all names in numbered sequence: i.e., search for all possible duplicate filenames – this may take some time. In the standard case, once a numbered file is missing, the program checks for 10 more named files before stopping.

Understanding the HOUSEKEEP COPY Process

The system copy program, COPYSFX, focuses on making a single copy of a soundfile. Although HOUSEKEEP COPY can also make a single copy, here the focus is on making multiple copies – and deleting them.

Mode **1** uses COPYSFX code, and both HOUSEKEEP COPY Mode **1** and COPYSFX are able to convert between **.wav** and **.aif** soundfile formats. Be careful to specify the extension of the soundfile type different from the one you have made your default when you set the environment variable CDP_SOUND_EXT.

Musical Applications

This facility is designed as a time-saver when working on mixes. When creating a multi-event texture, repeated use of the same sound is frequently required. On systems which can open the same soundfile more than once, using actual copies of the original soundfile is not necessary, though it could still be handy to make *count* copies, use them (don't forget about HOUSEKEEP BUNDLE) and then delete them.

A previous third mode to delete soundfiles named with this numbering convention has been made into the separate function HOUSEKEEP REMOVE.

End of HOUSEKEEP COPY

HOUSEKEEP DEGLITCH – Attempt to deglitch a soundfile

Usage

housekeep deglitch *infile outfile glitch sil thresh splice window* [**-s**]

Parameters

infile – input soundfile to deglitch
outfile – output, hopefully deglitched, soundfile
glitch – maximum duration in milliseconds of an glitch to find
sil – minimum duration in milliseconds of 'silence' on either side of the glitch
thresh – maximum level of 'silence' on either side of the glitch
splice – splice length in milliseconds to cut out the glitch. It must be at least half the length of *sil*.

With *thresh* = 0, use *splice* = 0. Otherwise *splice* = 0 will make clicks.

window – window length in milliseconds in which to search for glitches and 'silence'.

Note that very short windows may mistake parts of the waveform for 'silence'. Use larger windows for see larger features.

[**-s**] – option to see details about the (possible) glitches found

Understanding the HOUSEKEEP DEGLITCH Process

A **glitch** is heard as a brief click in the soundfile. This is caused by a very rapid and large change in amplitude, usually from one sample to the next. If they also overmodulate, there is also a nasty noise component. These glitches appear in the Time Domain *time amplitude* sample display of a soundfile as vertical lines, and can comprise only a very small number of samples. HOUSEKEEP DEGLITCH provides a way to search for these anomalies and remove them.

The length of a glitch might be able to be determined aurally, with PLAY FROM - TO used to pin it down further, though it may be too short for this to work. In a soundfile display, you could block one out and see its time in seconds (convert this to milliseconds by multiplying by 1000). Alternatively, the display might show sample positions. Then you would need to subtract the start from the end sample number and divide the result by the sample rate to get the time in seconds, and then multiply by 1000 to get the time in milliseconds. E.g., 10573 - 9455 = 1118 samples. 1118 ÷ 44100 (the sample rate) = 0.02535 seconds * 1000 = 25.35 milliseconds, rounding to 25.

Note that extensive overmodulation after processing a sound is not a glitch problem. Rather the amplitude level of the input to sound was too high for that particular process. The solution in this case is to reduce the level of the input sound before processing (MODIFY LOUDNESS, Mode **1**, Gain).

HOUSEKEEP DEGLITCH works by identifying and removing the glitch, closing up the gap. The *splice* parameter smooths the join – but it doesn't want to be too long either, lest it cause an audible dip in amplitude.

Musical Applications

Certain processes involving extensive editing of small fragments of sound may lead to sharp amplitude changes audible as clicks. It is extremely useful to have a tool to search for and eliminate them amongst the many thousands of samples per second in a soundfile.

Another situation where it might be useful is after a process that leaves only a handful of glitches, perhaps as overmodulations, but it is important to keep the overall amplitude level as high as possible.

Also see: HOUSEKEEP EXTRACT for similar, but more comprehensive facilities.

End of HOUSEKEEP DEGLITCH

HOUSEKEEP DISK – Show available space on disk

Usage

housekeep disk *anyinfile*

Parameters

anyinfile – any of the various types of files used with the CPP System can be used as an input, including text files

Understanding the HOUSEKEEP DISK Function

The purpose of HOUSEKEEP DISK is to show you how much free space you have available on your hard disk.

The display show available space in bytes, in samples (2 bytes per sample), and available time in hours, minutes and seconds for Mono and Stereo. If a soundfile is used as the *anyinfile*, the Mono and Stereo will relate to the sample rate of that soundfile (its header is interrogated). If your input is a mixfile, it will list time available for all valid sample rates, and will also tell you if a soundfile listed in the mixfile is no longer present on your disk (its name could be mis-spelled or it could be deleted.)

Musical Applications

HOUSEKEEP DISK is very handy when about to make a very long recording, or when you know that you are starting to get low on disk space.

'Disk' in this context means a partition of a hard disk (such as the c: or d: 'drive') if there is more than one on the physical hard disk unit.

Hard disk software normally protects against problems occuring when the disk reaches full capacity. Computer systems have been known in the past to 'wrap' round to the beginning of the disk and overwrite the FAT table (File Allocation Table: system index of all the files on the disk!). To be on the safe side, it's best to stop writing to the disk when it's nearly full. Stop and do some housework: clean out redundant files to make more space on the disk. Disks are happiest when they have some headroom in which to store their little bits and pieces of system information.

Recently, my Internet Explorer refused to load. It would start loading, and then I would get a system error message about an illegal operation. Richard Dobson acutely worked out that I didn't have enough space left on my disk for it to run, meaning places to which to copy temporary information.

End of HOUSEKEEP DISK

HOUSEKEEP ENDCLICKS – Remove clicks from the start or end of a soundfile

Usage

housekeep endclicks *infile outfile gate splicelen* [**-b**] [**-e**]

Parameters

infile – input soundfile, presumably with clicks
outfile – output, declicked, soundfile
gate – level **above** which the signal is retained. (Range: 0 to 1)
splicelen – length of splice taper in milliseconds
[**-b**] – trim start of sound
[**-e**] – trim end of sound

Understanding the HOUSEKEEP ENDCLICKS Process

Some CDP processes can produce output where the signal stops (or starts) abruptly (rather than falling, or rising, gradually from zero). This can happen with the output of transformed and resynthesized PVOC sounds, or at the end of mixes where the endtime of the mix process is not set at the end duration of the mixfile. Such sharp cutoffs (or cut-ins) produce clicks, which can be removed using this process.

Note that if a sharp cutoff occurs before the very end of the outfile data, a simple TOPNTAIL may not shave off the endclick, whereas this process searches for (start or) endclicks within the file in order to shave them off.

Dale Perkins has noticed that this program can fail if there are no edge clicks in the input soundfile. In *Sound Loom*, a red error message is displayed. In **bulk process**, however, Trevor Wishart explains, the bulk processor can lose the pipe-markers of the many processes it is running and display a message such as: *cannot find channel named 'file 103'*. These messages can be safely ignored – you just have to click on the 'OK' button at the bottom of the TK/Tcl error-message box(es) that come up. The bulk processor just carries on processing the files that do not 'fail'.

Musical Applications

This function provides a quick way to deal with clicks or even just overly sharp edges, without going as far as the, usually longer, slopes of a DOVETAIL. The *gate* parameter provides a way to retain a touch of 'edge' by raising the level, while a longer *splicelen* will add some extra smoothness to the attack and decay portions of the sound.

The **-b** and **-e** parameter provide the option to trim only the start or the end. Both would be handled by default, i.e., by not using either.

End of HOUSEKEEP ENDCLICKS

HOUSEKEEP EXTRACT – Extract significant sound from a recording, top & tail, remove glitches, etc.

Usage

housekeep extract 1 *infile* [**-g***gate*] [**-s***splice*] [**-e***endcutoff*] [**-t***threshold*] [**-h***hold*] [**-b***baktrak*] [**-i***Initial*] [**-l***minlength*] [**-w***gate_window*] [**-n**]
housekeep extract 2 *infile outfileview*
housekeep extract 3 *infile outfile* [**-g***gate*] [**-s***splice*] [**-b**] [**-e**]
housekeep extract 4 *infile outfile shift*
housekeep extract 5 *infile outfile valsfile* (No longer available)
housekeep extract 6 *insndfile outtextfile gate endcutoff threshold baktrak initlevel minlength gate_window*

Gustav Ciamaga provided an example command line for Mode **6**, in which *gate* is 0.035 and *minlength* is 0.05, and all other parameters are 0:
`housekeep extract 6 count countonsets.txt 0.035 0 0 0 0 0.05 0`
There is more information about this in the reference for SFEDIT SPHINX.

Note that in Mode **1**, the above example command line would look like this:
`housekeep extract 1 count -g0.035 -l0.05` and would produce a soundfile for each cut point.

Modes

1 Cut out and keep significant events from *infile*, creating a series of outfiles. With an *infile* named *nn*, the soundfiles produced from the segments of significant data will be *n0*, *n1*, *n2*, etc., the last character being replaced by a number.

2 Extraction preview, showing envelope of *infile* as a pseudo-soundfile, sector by sector. It will often be useful to run this first and view the result in order to help choose parameter values for Mode **1**

3 Top and tail: remove low level signal from start and end of sound. This creates *outfile*, a version of the *infile* without unwanted silence or low level signal at the beginning and/or end.
ALSO SEE TOPANTAIL2 for gated extraction of a sound with 'top and tailing' and a backtracking facility.

4 Remove: shift (offset) the entire signal to eliminate unwanted DC component (in reference to a 0 to 1 amplitude range)

(**5** Modify 'by hand' – this Mode is no longer available)

6 Find onset times: find the places where significant signal begins and write to a text file

Parameters

infile – original sound from which to extract significant data
outfileview – (Mode **2**): a pseudo-soundfile which shows amplitude levels for each sector (rather than for each sample). It provides an overview which shows at what levels significant data occurs. View with a soundfile display program, but remember that the time is compressed.
outfile – top and tailed or shifted versions of *infile*. In Mode **1**, the output soundfile name is derived from the input soundfile name, replacing the last character with '0', '1', etc.
-g*gate* – amplitude level **above which** signal is to be accepted. Range: 0 to 1 (Default: 0)
-s*splice* – length of splice slope in milliseconds (Default: 15ms)
-e*endcutoff* – end cut-off level below which the **end** of the sound is cut. If zero, defaults to *gate*.
-t*threshold* – amplitude level to be reached within the retained segments. If the level never exceeds *threshold*, the segment is not kept. (Default: 0)
-h*hold* – hold the sound for *hold* sectors before the start of the next segment. (Default: 0)
-b*baktrak* – go backwards in the soundfile and keep *baktrak* sectors prior to the current gate-on, but only if the level there is above the *Initial* level. (Range: 0 to 64 sectors)
-i*Initial* – amplitude level at start of backtracked segment. Range: 0 to 1; Default = 0. Use with **-b***baktrak*.
-l*minlength* – minimum length in seconds of segment to keep (Range: 0 to length of input soundfile – but I have seen it fail at 1.0 (with an 8 second soundfile) with the message "Cannot achieve task: 1 segments are shorter than min dur can get or than twice splicelen." Gustav Ciamaga recommends '0.05'.[AE])
-w*gate_window* – length of the gate window in sectors (see Mode **2**). Range: 0 to 64. The gate closes only if the level goes below*gate* for *gate_window* + 1 sectors. The larger the *gate_window*, the fewer the 'significant events' are likely to be. I have noticed the number of 'significant events' starting to drop off with values of 10 and more.
-n – the process stops if one of the soundfiles generated duplicates the name of an existing soundfile. The Default is to not generate that particular soundfile and continue processing.
-b – Mode **3**, don't trim the beginning of *infile*
-e – Mode **3**, don't trim the end of *infile*

The Default is to trim both ends.

shift – amplitude value applied to the whole file (subtracted from amplitudes): in order to remove any DC (direct current) component which may be present. (Range: lowest part of a 0 to 1 range, as required by the level of the interference)

NB: Mode **6** does not use 'flags' (e.g., **-b**), only the parameter values.

Understanding the HOUSEKEEP EXTRACT Process

Lots of parameters, but really a very simple process. When recording, there are inevitably going to be pauses, low-level sections before one 'really starts' to play or sing or speak, and often a number of events recorded at once. The HOUSEKEEP EXTRACT process simply goes through the soundfile looks for signal above a certain amplitude level. When it finds acceptably loud signal, it opens a 'gate', as it were, through which that signal passes, where it is stored as a separate soundfile. When the signal drops in level again, the gate closes and the soundfile for that segment is terminated. Thus the original recording is split up into a number of separate soundfiles, hopefully with the signal starting at the beginning and finishing nicely at the end..

The various parameters are designed to help get good signal levels at or near the beginning and end of each of the soundfiles produced.

- *gate* defines where 'significant signal' starts and ends: it sets the level which must be reached before starting to create one of the soundfiles
- *endcutoff* deals with the end of the *infile* which may often have a level lower than those which occur during the soundfile: i.e, the last segment may need a lower amplitude cut-off point.
- *threshold* handles matters within a segment to be retained: you may want to set a level which is lower or higher than *gate*
- *hold* enables you to extend the length of a a segment, even if it contains separate elements: it prevents saving every tiny chunk as a separate file.
- *baktrak* enables you to be a little more sensitive about the start of each segment: e.g., go back a bit and start at an *initial* level a little lower than *gate*
- *minlength* helps avoid keeping undesirably short segments
- *gate_window* enables you to give more or less time in which to find an acceptable signal level

In practice, running Mode **2** will most often be done first, so as to get your bearings with the amplitude levels of that particular *infile*. This will become less necessary as experience increases.

Some signals have a DC constant component which you can see in your soundfile viewer because the whole signal is raised (or lowered) away from the zero line. You also hear this as a click at the start and end of the sound, because of the sudden jumps in amplitude. Mode **4** enables you to introduce a shift which will restore the signal to the zero line: add a negative shift to lower to the zero line (e.g., 0.5 + -0.5 = 0), or a positive shift to raise it to the zero line (e.g., -0.5 + 0.5 = 0). You can see in your viewer what value to enter by observing how far the signal is away from the zero line. CDP's VIEWSF, which can zoom to the single sample, will provide a precise value.

Musical Applications

Move quickly from raw recordings to useable soundfiles.
Top and tail: quickly remove unwanted silence or low level signal from the beginning and/or end of a soundfile.
Eliminate DC components.

End of HOUSEKEEP EXTRACT

HOUSEKEEP GATE – Cut file at zero amplitude points

Usage

housekeep gate *infile outfile* [-**z***zerocount*]

Parameters

infile – input soundfile
outfile – output soundfile
[-**z***zerocount*] – the number of consecutive zero samples (per channel) to indicate a silent gap in the sound, where it can be cut

Understanding the HOUSEKEEP GATE Process

HOUSEKEEP GATE finds the points of zero amplitude in a sound, and cuts it at those points.

It is useful for extracting distinct features from a sound (or separate sounds from a sequence of recordings) provided the segments have true silence (zero amplitude) in between them.

Musical Applications

HOUSEKEEP GATE helps to isolate distinct areas of a sound, or distinct sounds in a sequence of recordings, or to get rid of unwanted silence.

End of HOUSEKEEP GATE

GATE – remove low-level sound from signal

Usage

gate gate *mode infile outfile gatelevel*

Example command line to gate a sound :

```
gate gate 1 count countgate -9
```

Modes

1 low level sound is replaced by silence (output will be the same length as the source)
2 low level sound is edited out (output is therefore shorer than the source

Parameters

infile – input soundfile
outfile – output soundfile
gatelevel – level below which sound is to be removed (in dB - put value, but not 'dB') – Range: 1.0 (full amplitude) to -96.0 (silence)

Understanding the GATE Process

Everything *below* the *gatelevel* is removed. Therefore, the closer *gatelevel* is to 1.0, the greater amount will be removed. Note that it does not reduce the amplitude, but cuts out signal below the threshold. It reminds me of the sign in *The Hitchhikers Guide to the Galaxy* that had sunk into the ground, such that only the top part was visible, reading "Go stick your head in a pig." With GATE GATE, only the top part of the sound, above *gatelevel*, remains. It is not quieter, but there may be bits missing.

Musical Applications

The applications can be twofold. On the one hand, low level noise can be removed. On the other hand, a sound can be reduced to a staccato image by using a *gatelevel* close to 1.0.

ALSO SEE HOUSEKEEP GATE.

End of GATE GATE

HOUSEKEEP REMOVE – Remove existing copies of a soundfile

Usage

housekeep remove *filename* [**-a**]

Parameters

filename – Deletes any copies of *filename*, having names *filename_001*, *filename_002* ...

No checks are made that these are actually copies of *filename*!

-a – The program checks all names in numbered sequence.

In the standard case, once a numbered file is missing, the program checks for 10 more named files before halting. Setting the **-a** flag forces the program to search for all possible duplicate filenames. This may take some time.

Understanding the HOUSEKEEP REMOVE Function

A series of soundfiles whose names end with '_001', '_002' etc. can be created with HOUSEKEEP COPY and HOUSEKEEP EXTRACT. HOUSEKEEP REMOVE is a handy way to remove these files after you have decided (and renamed) the ones you want to keep.

Musical Applications

This is just a housekeeping utility.

End of HOUSEKEEP REMOVE

HOUSEKEEP RESPEC – Change sample rate, format or properties of a soundfile (CARE!!)

Usage

housekeep respec 1 *infile outfile new_samplerate*
housekeep respec 2 *infile outfile*
housekeep respec 3 *infile outfile* [**-s***srate*] [**-c***channels*]

Modes

1 resample at some different sample rate (rewrites sample data)
2 convert from integer to float samples, or *vice versa*
3 change properties of sound (affects header only – use with caution!)

Parameters

infile – soundfile to adjust
outfile – new soundfile with alterations made
new_samplerate – Mode **1**, new sample rate at which to rewrite the soundfile

> New sample rate must be one of: 96000, 88200, 64000, 48000, 24000, 44100, 22050, 32000 or 16000

-s*srate* – Mode **3**, new sample rate to impose

> New sample rate must be one of: 96000, 88200, 64000, 48000, 24000, 44100, 22050, 32000 or 16000
> **NB:** This process **does not** resample the data. It simply causes the original data to be read at a different sample rate. The sound will have the same number of samples, a different duration and be transposed in pitch (up or down).

-c*channels* – Mode **3**, the new channel count to impose

> **NB:** This **does not** rechannel the data. A stereo file, for example, will appear twice as long.

Understanding the HOUSEKEEP RESPEC Process

The key issue here is the difference between 'resampling' (Mode **1**) and simply changing the information in the header of the soundfile (Mode **2**).

If the sample rate is 22050, resampling with a *new_samplerate* of 44100 will rewrite the file, doubling up every sample so that there will then be 44100 samples per second in the file. (The information in the header is also changed accordingly). This is what is meant by the phrase 'rewrites the sample data'. This is what happens in (Mode **1**), and this is why the length of the file in bytes will be longer or shorter, but the duration and pitch of the sound will remain **unchanged**. When the PLAY program checks the header and finds e.g., 44100, there are indeed 44100 samples to play, etc.

The resampling operation is relatively simple in nature, which is why it works only with the named sample rates. A fully flexible resampling algorithm is much more complex, requires extensive filtering procedures and has not been implemented in CDP. Resampling should be used sparingly for this reason, as some loss of quality is likely, especially in downsampling, as no anti-aliasing is applied to the sound file.

In Mode **3** the information in the header is changed but the actual sample data is left untouched. Thus, when the PLAY program checks the header and finds e.g., 44100 when the file actually contains only 22050 samples, it will play it at a 44100 rate – which gets through the 22050 samples at double speed, takes half as long and raises the pitch by an octave (twice as fast).

Mode **2** converts between 16-bit and floating-point soundfile data. At the moment, all soundfiles are normally 16-bit. The CDP System automatically converts to floating point, does the processing and then at the output stage the data is automatically converted from float back to short, thus losing the extra precision (= audio quality) which the floating point provides. It has to do this as there are at present no soundcards which can handle floating point soundfiles.

If there is a floating point soundfile that you want to use, Mode **2** will convert it to shorts so you can use it with the CDP System. However, we are gradually extending the capacity of the System to handle floating point files, both as inputs, and to output a floating point result. HOUSEKEEP RESPEC will itself be updated in this way.

Mode **2** will come into its own in the future as soundcards appear which can play floating point soundfiles, but will also be less needed as more CDP functions become capable of creating floating point soundfiles.

Floating point output is also useful for DSP testing.

Musical Applications

Sometimes when mixing, one finds that the sample rates of the component soundfiles are not the same (they all must be the same for a mix). HOUSEKEEP RESPEC in Mode **1** provides a reasonably effective way to get the sample rates to conform. However, it shouldn't be regarded as a standard tool. **You should really work at a consistent sample rate.** However, if at some inconvenient moment you find you have a stray soundfile at an incompatible rate for mixing, you can quickly alter it – with some loss of quality.

Some versions of *Cooledit Pro* will incorrectly write a floating point header on a 16-bit soundfile. The *sampletype* can be changed to SHORT with HOUSEKEEP RESPEC Mode **3**.

End of HOUSEKEEP RESPEC

HOUSEKEEP SORT – Sort files listed in a textfile

Usage

housekeep sort 1 *listfile*
housekeep sort 2-3 *listfile small large step* [**-l**]
housekeep sort 4 *listfile* [**-l**]
housekeep sort 5 *listfile*
housekeep sort 6 *listfile*

Modes

1 Sort by filetype, listing in textfile(s) with 3-letter file extension(s) to identify the groups of files (see table below)
2 Sort by sample rate, listing in textfile(s) with extension(s) indicating SR
3 Sort by duration, listing in textfile named 'your_listfile.len' (in *step* duration groups if *step* was specified)
4 Sort by log duration, relating file lengths by a duration ratio, and listing in textfile named 'your_listfile.len'
5 Sort into duration order, listing in 'your_listfile.len'
6 Find rogues (check for invalid files and report potential anomalies)

Parameters

listfile – a text file containing a list of the files to be sorted. The normal extension of the file must be included: e.g., .wav or .aif for a soundfile.
small – maximum size of the smallest soundfiles, in seconds
large – minimum size of the largest soundfiles, in seconds
step – groups the files according to duration increment
-l – causes file durations **not** to be written to the output text file

Understanding the HOUSEKEEP SORT Function

Mode **1** reads a textfile containing a list of files (with their extensions) and groups them into one or more text files containing the names of all the files of a given type. These output textfiles are given extensions by which you can easily identify their contents. The extensions of the actual source files named in *listfile* can vary, but the conventions now recognised by the CDP System are placed in the 3rd column of the table below.

Extensions to identify filetypes in SORT output textfiles

SORT textfile extension	File type	Source's actual extension
.mot	mono soundfiles	**.wav** or **.aif**
.stt	stereo soundfiles	**.wav** or **.aif**
.ant	analysis files	**.ana**
.qtt	4-channel soundfiles	**.wav** or **.aif**
.pct	binary pitch data files	**.frq**
.ent	binary envelope files	**.evl**
.fot	binary formant files	**.for**
.trt	binary transposition files	**.trn**
.ott	any other files	**.txt** or whatever

In Mode **2** you get a list of all the soundfiles in your current directory grouped by sample rate. The sample rate of the sounds in a given textfile generated by SORT Mode **2** is indicated by a numerical extension: **.24** for 24000, **.44** for 44100 etc.

Musical Applications

As a directory for a given project starts to fill up, confusion about what is what may begin to set in, especially if the naming conventions vary for one reason or another. HOUSEKEEP SORT Mode **1** looks at the headers of all the files and automatically groups them correctly, writing a series of files for future reference. This is also a great way to document a project.

The other modes tend to be useful when preparing mixes. You may need a quick check on whether all the files you intend to mix have the same sample rate (required by the mixing operation), or you might want a listing of all the soundfiles with durations within a certain range, or you might want to get an ordered list of all your soundfiles from the shortest to the longest. HOUSEKEEP SORT handles all these various operations. The resulting files can, if appropriate, be used as source textfiles for your actual mixfile, with SUBMIX DUMMY automatically turning them into a prototype mixfile.

All of these operations are simple and quick, and the larger your mixes, the more useful these facilities become. SUBMIX MIX can handle more than 50 soundfiles at once, so it may be worth exploring the potential speed and power of a non-graphic approach.

End of HOUSEKEEP SORT