This document shortly details how to import and export sound with R using the packages `seewave`, `tuneR` and `audio`.

Contents

1 In
   1.1 Non specific classes .................................... 2
       1.1.1 Vector ............................................ 2
       1.1.2 Matrix ........................................... 2
   1.2 Time series ............................................ 3
   1.3 Specific sound classes ................................... 3
       1.3.1 Wave class (package tuneR) ......................... 4
       1.3.2 audioSample class (package audio) ................. 5

2 Out
   2.1 .txt format ............................................ 5
   2.2 .wav format ............................................ 6
   2.3 .flac format ........................................... 6

3 Mono and stereo ........................................... 6

4 Play sound ................................................ 7
   4.1 Specific functions .................................... 7
       4.1.1 Wave class ....................................... 7
       4.1.2 audioSample class ................................ 7
       4.1.3 Other classes .................................... 7
   4.2 System command ........................................ 8

5 Summary ................................................... 8

---

1 The package sound is no more maintained.
1 Import and export of sound with R

The main functions of *seewave* (>1.5.0) can use different classes of objects to analyse sound:

- usual classes (numeric *vector*, numeric *matrix*),
- time series classes (*ts*, *mts*),
- sound-specific classes (*Wave* and *audioSample*).

1.1 Non specific classes

1.1.1 Vector

Any numeric vector can be treated as a sound if a sampling frequency is provided in the `f` argument of *seewave* functions. For instance, a 440 Hz sine sound (A note) sampled at 8000 Hz during one second can be generated and plot following:

```r
> s1<-sin(2*pi*440*seq(0.1,length.out=8000))
> is.vector(s1)
[1] TRUE
> mode(s1)
[1] "numeric"
> library(seewave)
> oscillo(s1,f=8000)
```

1.1.2 Matrix

Any single column matrix can be read but the sampling frequency has to be specified in the *seewave* functions.

```r
> s2<-as.matrix(s1)
> is.matrix(s2)
[1] TRUE
> dim(s2)
[1] 8000 1
> oscillo(s2,f=8000)
```

If the matrix has more than one column, then the first column only will be considered.

```r
> x<-rnorm(8000)
> s3<-cbind(s2,x)
> is.matrix(s3)
[1] TRUE
> dim(s3)
[1] 8000 2
> oscillo(s3,f=8000)
```
1.2 Time series

The class ts and the related functions ts, as.ts, is.ts can be used also for sound. Here follows the command to similarly generate a time series corresponding to a 440 Hz sine sound sampled at 8000 Hz during one second:

```r
> s4<-ts(data=s1, start=0, frequency=8000)
> str(s4)
```

```
Time-Series [1:8000] from 0 to 1: 0 0.339 0.637 0.861 0.982 ...
```

To generate a 0.5 second random noise:

```r
> s4<-ts(data=runif(4000), start=0, end=0.5, frequency=8000)
> str(s4)
```

```
Time-Series [1:4000] from 0 to 0.5: 0.183 0.871 0.361 0.483 0.574 ...
```

The length of s4 is not 4000 but 4001. Data are actually recycled, s4[4001] being the same as s4[1].

The functions `frequency` and or `deltat` return the sampling frequency (f) and the time resolution (Δt) respectively:

```r
> frequency(s4)
```

```
[1] 8000
```

```r
> deltat(s4)
```

```
[1] 0.000125
```

As the frequency is embedded in ts objects, there is no need to specify it when using `seewave` functions:

```r
> oscillo(s4)
```

In the case of multiple time series, `seewave` functions will consider the first series only:

```r
> s5<-ts(data=s3,f=8000)
> class(s5)
```

```
[1] "mts"  "ts"  "matrix"  "array"
```

```r
> oscillo(s5)
```

1.3 Specific sound classes

There are three object classes corresponding to the binary wav format or to the compressed mp3 format:

- the class Wave of the package `tuneR`,
- the class audioSample of the package `audio`

---

J. Sueur 3 July 16, 2023
1.3.1 Wave class (package tuneR)

The class `Wave` comes with the package `tuneR` managed by Uwe Ligges. This S4 class includes different slots with the data (left or right channel), the sampling frequency (or rate), the number of bits (8 /16 /24 /32) and the type of sound (mono /stereo). High sampled sound (i.e. > 44100 Hz) can be read.

The function to import `.wav` files from the hard-disk is `readWave`:

```r
g6 <- readWave("mysong.wav")
```

The other advantage of using `readWave` is for reading part of long files. It is indeed possible to import only a section of the `.wav` file using the arguments `from` and `to` and by specifying the time units with the arguments `units`. The units can be turned to "samples", "minutes" or "hours". For instance, to read only the section starting at 1s and ending at 5s of the file "mysong.wav":

```r
g7 <- readWave("mysong.wav", from = 1, to = 5, units = "seconds")
g7
```

```
Wave Object
Number of Samples: 32000
Duration (seconds): 4
Sampling rate (Hertz): 8000
Channels (Mono/Stereo): Mono
PCM (integer format): TRUE
Bit (8/16/24/32/64): 16
```

Note that `.mp3` files can be imported as a `Wave` object with the function `readMP3`.

To get information regarding the object (sampling frequency, number of bits, mono /stereo), it is necessary to use the indexing of S4 object classes:

```r
> g7@samp.rate
[1] 8000

> g7@bit
[1] 16

> g7@stereo
[1] FALSE
```

A property not apparent in these call is that `readWave` does not normalise the sound. Values describing the sound will be included between $\pm 2^{bit-1}$:

```r
> range(g7@left)
[1] 0 0
```
1.3.2 audioSample class (package audio)

The package audio, developed by Simon Urbanek, is another option to handle .wav files. Sound can be imported using the function load.wave. The class of the resulting object is audioSample which is essentially a numeric vector (for mono) or numeric matrix with two rows (for stereo). The sampling frequency and the resolution can be called as attributes:

```
library(audio)
s10 <- load.wave("mysong.wav")

head(s10)
```

```
library(audio)
s10 <- load.wave("mysong.wav")

head(s10)
```

```
[1] 0.0000000 0.7070923 0.9999695 0.7070923 0.0000000 -0.7071139
```

```
s10$rate

[1] 8000
```

```
s10$bits

[1] 16
```

The main advantage of the package audio is that sound can be directly acquired within an R session. This is achieved by first preparing a vector of NA and then using the function record. For instance, to get a mono sound of 5 seconds sampled at 16 kHz:

```
> s11 <- rep(NA_real_, 16000*5)
> record(s11, 16000, 1)
```

A recording session can be controlled using three complementary functions: pause, rewind, and resume (see 4.1.2). See the documentation of audio for details regarding the control of audio drivers: http://www.rforge.net/audio/.

2 Out

2.1 .txt format

For a maximal compatibility with other sound softwares, it can be useful to save a sound as a simple .txt file. This can be done using the function export with the argument header=FALSE. By default, the name of the object is used to name the .txt file. The following commands will write a file "tico.txt" on the hard-disk.

```
> data(tico)
> export(tico, f=22050, header=FALSE)
```

For Windows users, the software Goldwave can be helpful when handling long sound files or large number of files. To export a sound as a .txt file that can be directly read by Goldwave, the same function can be used but with the default argument header=TRUE. seewave will automatically add the header needed. Hereafter the name of the exported file is changed using the argument filename:

```
> export(tico, f=22050, filename="tico_Gold.txt")
```

Any header can be specified for a connection with other softwares. For instance, if an external software needs the header "f=sampling frequency; ch=left":

```
> export(tico, f=22050, filename="tico_ext.txt",
+ header="f=22050; ch=left")
```
2.2 .wav format

tuneRand audio have a function to write .wav files: writeWave, and save.wave respectively. Within seeawave, the function savewav, which is based on writeWave, can be used to save data as .wav. By default, the name of the object will be used for the name of the .wav file:

> savewav(tico, f=22050)

As seen before, if the object to be saved is of class ts or Wave, there is no need to specify the argument f. Here we use the argument filename to change the name of the wav file:

> ticofirst<-cutw(tico, f=22050, to=0.5, output="Wave")
> savewav(ticofirst, filename = "tico_firstnote.wav")

2.3 .flac format

Free Lossless Audio Codec (FLAC) is a file format by Josh Coalson for lossless audio data compression. FLAC reduces bandwidth and storage requirements without sacrificing the integrity of the audio source. Audio sources encoded to FLAC are typically reduced in size 40 to 50 percent. See the flac webpage for details.

.flac format cannot be used as such with R. However, the function wav2flac allows to call FLAC software directly from the console. FLAC has therefore to be installed on your OS. If you have a .wav file you wish to compress into .flac, call:

> wav2flac("tico_firstnote.wav", overwrite=TRUE)

To compress a .wav file into .flac, the argument reverse has to be set to TRUE:

> wav2flac("tico_firstnote.flac", reverse=TRUE)

3 Mono and stereo

Wave class can handle stereo files. There are some specific functions regarding mono/stereo type. To generate a stereo sound, two mono sounds are first created using sine, a function that returns a Wave object, and then combined using stereo:

> left<-sine(440)
> right<-sine(2000)
> s12<-stereo(left,right)
> s12

Wave Object
Number of Samples: 44100
Duration (seconds): 1
Samplingrate (Hertz): 44100
Channels (Mono/Stereo): Stereo
PCM (integer format): FALSE
Bit (8/16/32/64): 32

To go back to a mono file taking the left channel only:

> s13<-mono(s12,"left")

The function channel do roughly the same as it extracts one or more channels. To get this time the right channel:

> s14<-channel(s12,"right")
And eventually, the S4 indexing can be used to do it "manually". In this particular case, the returned object will be of class `vector`.

```r
> s13<-s12@left
> is.vector(s13)
[1] TRUE

> s14<-s12@right
> is.vector(s14)
[1] TRUE
```

## 4 Play sound

### 4.1 Specific functions

#### 4.1.1 Wave class

Wave objects can be played with `play` of `tuneR`:

```r
> play(s6)
```

It may happen that the default players of the function `play` are not installed on the OS. Three functions can help in setting the media player: `findWavPlayer` returns the most common system commands on the OS, `WavPlayer` returns the command that is currently used by `play`, `setWavPlayer` is used to define the command to be used by `play`. For instance, if Audacious is the player to use (Linux OS):

```r
> setWavPlayer("audacious")
```

#### 4.1.2 audioSample class

The package `audio` has similarly a function `play` but also have three useful functions to control recording and playback:

- `pause` that stops audio recording or playback,
- `rewind` that rewinds audio recording or playback, *i.e.*, makes the beginning of the source (or target) object the current audio position,
- `resume` that resumes previously paused audio recording or playback.

#### 4.1.3 Other classes

The package `seewave` includes listen a function based on `play` of `tuneR` but accepting all specific and non-specific classes and with two arguments (`from` and `to`) to listen only a section of a sound object:

```r
> listen(s1, f=8000, from=0.3, to=7)
> listen(s13, from=0.3, to=4)
```
4.2 System command

The call of an external sound player can also be achieved using directly `system` that allows invoking directly the system command. For instance, to play a sound with Audacity (Linux OS):

```r
> system("audacity mysong.wav")
```

To run a sound player with Windows is slightly more tricky as the complete path to the .exe file has to be specified and `paste` has to be invoked to combine both program and file names:

```r
> system(paste("C:/Program Files/GoldWave/GoldWave.exe", "mysong.wav"))
```

5 Summary

Here is a temptative of summary of main R functions used for sound input and output:

<table>
<thead>
<tr>
<th></th>
<th>Input</th>
<th>Output</th>
<th>Mono/Stereo</th>
<th>Play</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tuneR</strong></td>
<td>readWave</td>
<td>writeWave</td>
<td>mono, stereo</td>
<td>play</td>
<td>Wave</td>
</tr>
<tr>
<td><strong>audio</strong></td>
<td>load.wave, record</td>
<td>save.wave</td>
<td>mono, stereo</td>
<td>play, pause, resume, rewind</td>
<td>audioSample</td>
</tr>
<tr>
<td><strong>seewave</strong></td>
<td>export, savewav</td>
<td></td>
<td></td>
<td>listen</td>
<td>vector, matrix, ts, mts, Wave, audioSample</td>
</tr>
</tbody>
</table>