22.4. `wave` — Read and write WAV files

Source code: Lib/wave.py

The `wave` module provides a convenient interface to the WAV sound format. It does not support compression/decompression, but it does support mono/stereo.

The `wave` module defines the following function and exception:

`wave.`**`open`**(*file*, *mode=None*)

If *file* is a string, open the file by that name, otherwise treat it as a file-like object. *mode* can be:

`'rb'`

Read only mode.

`'wb'`

Write only mode.

Note that it does not allow read/write WAV files.

A *mode* of `'rb'` returns a `Wave_read` object, while a *mode* of `'wb'` returns a `Wave_write` object. If *mode* is omitted and a file-like object is passed as *file*, `file.mode` is used as the default value for *mode*.

If you pass in a file-like object, the wave object will not close it when its `close()` method is called; it is the caller’s responsibility to close the file object.

The `open()` function may be used in a `with` statement. When the `with` block completes, the `Wave_read.close()` or `Wave_write.close()` method is called.

Changed in version 3.4: Added support for unseekable files.

`wave.`**`openfp`**(*file*, *mode*)

A synonym for `open()`, maintained for backwards compatibility.

exception `wave.`**`Error`**

An error raised when something is impossible because it violates the WAV specification or hits an implementation deficiency.

22.4.1. Wave_read Objects

Wave_read objects, as returned by `open()`, have the following methods:

`Wave_read.`**`close`**()

Close the stream if it was opened by `wave`, and make the instance unusable. This is called automatically on object collection.

`Wave_read.`**`getnchannels`**()

Returns number of audio channels (`1` for mono, `2` for stereo).

`Wave_read.`**`getsampwidth`**()

Returns sample width in bytes.

`Wave_read.`**`getframerate`**()

Returns sampling frequency.

`Wave_read.`**`getnframes`**()

Returns number of audio frames.

`Wave_read.`**`getcomptype`**()

Returns compression type (`'NONE'` is the only supported type).

`Wave_read.`**`getcompname`**()

Human-readable version of `getcomptype()`. Usually `'not compressed'` parallels `'NONE'`.

`Wave_read.`**`getparams`**()

Returns a `namedtuple()` `(nchannels, sampwidth, framerate, nframes, comptype, compname)`, equivalent to output of the `get*()` methods.

`Wave_read.`**`readframes`**(*n*)

Reads and returns at most *n* frames of audio, as a `bytes` object.

`Wave_read.`**`rewind`**()

Rewind the file pointer to the beginning of the audio stream.

The following two methods are defined for compatibility with the `aifc` module, and don't do anything interesting.

`Wave_read.`**`getmarkers`**()

Returns `None`.

`Wave_read.`**`getmark`**(*id*)

Raise an error.

The following two methods define a term "position" which is compatible between them, and is otherwise implementation dependent.

`Wave_read.`**`setpos`**(*pos*)

Set the file pointer to the specified position.

`Wave_read.`**`tell`**()

Return current file pointer position.

22.4.2. Wave_write Objects

For seekable output streams, the `wave` header will automatically be updated to reflect the number of frames actually written. For unseekable streams, the *nframes* value must be accurate when the first frame data is written. An accurate *nframes* value can be achieved either by calling `setnframes()` or `setparams()` with the number of frames that will be written before `close()` is called and then using `writeframesraw()` to write the frame data, or by calling `writeframes()` with all of the frame data to be written. In the latter case `writeframes()` will calculate the number of frames in the data and set *nframes* accordingly before writing the frame data.

Wave_write objects, as returned by `open()`, have the following methods:

Changed in version 3.4: Added support for unseekable files.

`Wave_write.`**`close`**()

Make sure *nframes* is correct, and close the file if it was opened by `wave`. This method is called upon object collection. It will raise an exception if the output stream is not seekable and *nframes* does not match the number of frames actually written.

`Wave_write.`**`setnchannels`**(*n*)

Set the number of channels.

`Wave_write.`**`setsampwidth`**(*n*)

Set the sample width to *n* bytes.

`Wave_write.`**`setframerate`**(*n*)

Set the frame rate to *n*.

Changed in version 3.2: A non-integral input to this method is rounded to the nearest integer.

`Wave_write.`**`setnframes`**(*n*)

Set the number of frames to *n*. This will be changed later if the number of frames actually written is different (this update attempt will raise an error if the output stream is not seekable).

`Wave_write.`**`setcomptype`**(*type*, *name*)

Set the compression type and description. At the moment, only compression type `NONE` is supported, meaning no compression.

`Wave_write.`**`setparams`**(*tuple*)

The *tuple* should be `(nchannels, sampwidth, framerate, nframes, comptype, compname)`, with values valid for the `set*()` methods. Sets all parameters.

`Wave_write.`**`tell`**()

Return current position in the file, with the same disclaimer for the `Wave_read.tell()` and `Wave_read.setpos()` methods.

`Wave_write.`**`writeframesraw`**(*data*)

Write audio frames, without correcting *nframes*.

Changed in version 3.4: Any bytes-like object is now accepted.

`Wave_write.`**`writeframes`**(*data*)

Write audio frames and make sure *nframes* is correct. It will raise an error if the output stream is not seekable and the total number of frames that have been written after *data* has been written does not match the previously set value for *nframes*.

Changed in version 3.4: Any bytes-like object is now accepted.

Note that it is invalid to set any parameters after calling `writeframes()` or `writeframesraw()`, and any attempt to do so will raise `wave.Error`.