Package 'ProTrackR2'

May 7, 2026

Title Manipulate and Play 'ProTracker' Modules

Version 0.1.1

Maintainer Pepijn de Vries <pepijn.devries@outlook.com>

Description 'ProTracker' is a popular music tracker to sequence music on a Commodore Amiga machine. This package offers the opportunity to import, export, manipulate and play 'ProTracker' module files. Even though the file format could be considered archaic, it still remains popular to this date. This package intends to contribute to this popularity and therewith keeping the legacy of 'ProTracker' and the Commodore Amiga alive. This package is the successor of 'ProTrackR' providing better performance.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

LinkingTo cpp11

Collate 'ProTrackR2-package.R' 'cell.R' 'command.R' 'cpp11.R' 'data.R' 'demo.R' 'helpers.R' 'instrument.R' 'io.R' 'mod_info.R' 'modplug.R' 'note.R' 'pattern.R' 'render.R' 'play.R' 's3.R' 'samples.R' 'select_ops.R' 'validate.R'

Depends R (>= 4.1.0)

Imports audio, lifecycle

Suggests av, cli, curl, htmltools, kableExtra, knitr, ProTrackR, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://pepijn-devries.github.io/ProTrackR2/, https://github.com/pepijn-devries/ProTrackR2

BugReports https://github.com/pepijn-devries/ProTrackR2/issues

Config/testthat/edition 3

NeedsCompilation yes

Author Pepijn de Vries [aut, cre] (ORCID:
<https://orcid.org/0000-0002-7961-6646>),
Olav Sørensen [aut] (Developer of the ProTracker 2.3d clone)

Repository CRAN

Date/Publication 2025-11-21 20:20:21 UTC

Contents

`as_modplug_pattern` *Format a ProTracker pattern conform OpenMPT specs*

Description

OpenMpt is a popular modern music tracker. This function allows you to format a pattern such that it can be pasted directly into OpenMPT. On Windows you can use `writeClipboard()` for this purpose.

Usage

```
as_modplug_pattern(pattern, ...)
```

Arguments

pattern	An object of class pt2pat to be formatted
...	Ignored

Value

Returns a character object formatted such that it can be copied into OpenMPT

Author(s)

Pepijn de Vries

Examples

```
mod    <- pt2_read_mod(pt2_demo())
as_modplug_pattern(pt2_pattern(mod, 0L))
```

---

as_pt2cell *Convert object into a pt2cell or pt2celllist class object*

Description

Protracker uses pattern tables to annotate music progression. Each cell in the table contains information about the note, sample identifier and sound effect. This function coerces objects to pt2cell or pt2cell class objects, such that they can be inserted into patterns.

Usage

```
as_pt2cell(x, ...)

as_pt2celllist(x, ...)
```

Arguments

x	A (vector of) character string(s), to be coerced to a pt2cell class object. The first two or three characters represent the note (see pt2_note()). this is followed by two numerical characters representing the sample number. And finally three hexadecimal characters representing an effect or trigger. These three parts can optionally be padded with spaces. A valid string would for instance be "C#2 01 C1A.
...	Ignored

Value

An object of class `pt2cell`, `pt2celllist`.

Examples

```
as_pt2cell("A-3 02 121")
as_pt2cell("--- 01 000")
as_pt2celllist(c("A-3 02 121", "--- 01 000"))
```

`effect_commands` *Effect commands (data.frame)*

Description

ProTracker uses codes to achieve certain effects or jump to a specific position (see `vignette("effect_commands")`).

Format

`effect_commands` is a `data.frame` with three columns:

- `Code`: a `character` of pseudo code used to achieve a certain effect in ProTracker
- `Effect`: Name of the effect.
- `Description`: Description of the effect.

It is used for documentation and reference purposes.

Examples

```
data("effect_commands")
```

`format.pt2mod` *Implementation of basic S3 methods*

Description

Implementation of basic S3 methods, such as, `format`, `print`, `as.raw` and `as.character` (see usage section for a complete overview). See `vignette('s3class')` for an overview of ProTrackR2 S3 class objects. See usage section for an overview of available methods.

Usage

```
## S3 method for class 'pt2mod'
format(x, ...)

## S3 method for class 'pt2mod'
print(x, ...)

## S3 method for class 'pt2pat'
format(
  x,
  padding = " ",
  empty_char = "-",
  fmt = getOption("pt2_cell_format"),
  ...
)

## S3 method for class 'pt2command'
as.character(x, ...)

## S3 method for class 'pt2command'
format(x, fmt = getOption("pt2_effect_format"), ...)

## S3 method for class 'pt2command'
print(x, max.print = 10L, ...)

## S3 method for class 'pt2pat'
print(x, sep = "  ", show_header = TRUE, show_row = TRUE, ...)

## S3 method for class 'pt2pat'
as.character(x, ...)

## S3 method for class 'pt2celllist'
as.raw(x, ...)

## S3 method for class 'logical'
as.raw.pt2celllist(x, compact = TRUE, ...)

## S3 method for class 'pt2pat'
as.raw(x, ...)

## S3 method for class 'logical'
as.raw.pt2pat(x, compact = TRUE, ...)

## S3 method for class 'pt2cell'
format(
  x,
  padding = " ",
  empty_char = "-",
```

```
  fmt = getOption("pt2_cell_format"),
  ...
)

## S3 method for class 'pt2cell'
print(x, ...)

## S3 method for class 'pt2cell'
as.character(x, ...)

## S3 method for class 'pt2celllist'
as.character(x, ...)

## S3 method for class 'pt2command'
as.raw(x, ...)

## S3 method for class 'pt2cell'
as.raw(x, ...)

## S3 method for class 'logical'
as.raw.pt2cell(x, compact = TRUE, ...)

## S3 method for class 'pt2samp'
format(x, ...)

## S3 method for class 'pt2samp'
print(x, ...)

## S3 method for class 'pt2patlist'
format(x, ...)

## S3 method for class 'pt2patlist'
print(x, ...)

## S3 method for class 'pt2celllist'
format(x, ...)

## S3 method for class 'pt2celllist'
print(x, ...)

## S3 method for class 'pt2samplist'
format(x, ...)

## S3 method for class 'pt2samplist'
print(x, ...)

## S3 method for class 'pt2mod'
as.raw(x, ...)
```

```
## S3 method for class 'pt2samp'
as.raw(x, ...)

## S3 method for class 'pt2samp'
as.integer(x, ...)

## S3 method for class 'pt2celllist'
length(x, ...)

## S3 method for class 'pt2command'
length(x, ...)
```

Arguments

x	Object to apply S3 method to. See 'usage' section for allowed object types.
...	Passed on to other methods.
padding	A vector of character strings used to pad between note and instrument number (element 1) and between instrument number and effect command (element 2). Values are recycled.
empty_char	A vector of single character values used to represent empty pattern elements. First element is used for notes, second for instrument number, the third for effect commands (see also vignette("effect_commands")). Values are recycled.
fmt	Experimental feature to format a pt2cell. It should be a named list containing formatting strings for elements in the cell. It should contain the elements "note", "padding", "instrument" and "effect". Its implementation may change in future releases.
max.print	Maximum number of elements to be printed.
sep	A separator character string for concatenating pattern table columns (i.e. channels).
show_header	A logical value indicating if a header should be shown for the pattern table.
show_row	A logical value indicating if the row of a pattern table should be labelled with its index.
compact	Should the pattern be formatted using a compact notation (as used for file storage), or a none-compact format as used by the player? This can be set with the compact argument.

Value

The following is returned by the different methods:

- format: a formatted character representation of the object
- print: same as format
- as.character: same as format
- as.raw: a raw representation of the object. In many cases it inherits the same class as x

- `as.integer`: converted `raw` 8 bit sample data to signed pulse code modulation `integer` values between -128 and +127.
- `length` returns number of elements in `x`

Examples

```
## First create some ProTrackR2 objects to which
## S3 methods can be applied:

mod      <- pt2_read_mod(pt2_demo())
patterns <- mod$patterns
pattern  <- patterns[[1]]
cells    <- pattern[1:4,1]
cell     <- cells[[1]]
samples  <- mod$samples
sample   <- samples[[1]]
cmnd     <- pt2_command(cell)

## Let's go wild with S3 methods:
print(mod)
print(patterns)
print(pattern)
print(cells)
print(cell)
print(samples)
print(sample)
print(cmnd)

format(mod)
format(patterns)
format(pattern)
format(cells)
format(cell)
format(samples)
format(sample)
format(cmnd)

as.character(pattern)
as.character(cells)
as.character(cell)
as.character(cmnd)

as.raw(mod)
as.raw(pattern)
as.raw(cells)
as.raw(cell)
as.raw(sample)
as.raw(cmnd)

as.integer(sample)
```

play *Play a ProTrackR2 class objects as audio*

Description

Renders a ProTrackR2 class object as `audio::audioSample()` and plays it.

Usage

```
## S3 method for class 'pt2mod'
play(x, duration = NA, options = pt2_render_options(), position = 0L, ...)

## S3 method for class 'pt2samp'
play(x, duration = 5, options = pt2_render_options(), note = "C-3", ...)

## S3 method for class 'pt2patlist'
play(x, duration = NA, options = pt2_render_options(), samples, ...)

## S3 method for class 'pt2pat'
play(x, duration = NA, options = pt2_render_options(), samples, ...)

## S3 method for class 'pt2celllist'
play(x, duration = 5, options = pt2_render_options(), samples, ...)

## S3 method for class 'pt2cell'
play(x, duration = 5, options = pt2_render_options(), samples, ...)
```

Arguments

`x`	Object to be played.
`duration`	Duration of the rendered output in seconds. When set to `NA` the duration of the module is calculated and used for rendering.
`options`	A list of options used for rendering the audio. Use `pt2_render_options()` to obtain default options, or modify them.
`position`	Starting position in the pattern sequence table (`pt2_pattern_table()`). Should be a non negative value smaller than the mule length (`pt2_length()`).
`...`	Arguments passed to `pt2_render()`.
`note`	Note to be played when `x` is a `pt2samp` class object. Defaults to `"C-3"`.
`samples`	When rendering or playing patterns (or elements of it), samples are needed to interpret the pattern. Pass the samples as a sample list (class `pt2samplist`).

Value

Returns an `[audio::$.audioInstance]` object which allows you to control the playback (pause, resume, rewind).

Author(s)

Pepijn de Vries

Examples

```
if (interactive()) {
  mod <- pt2_read_mod(pt2_demo())

  ## ctrl will contain the audioInstance that will let
  ## you control the audio playback:
  ctrl <- play(mod)

  ## You can also play individual samples
  samp <- mod$samples[[3]]
  play(samp, note = "C-2")
  play(samp, note = "E-2")
  play(samp, note = "G-2")

  ## As well as an individual pattern
  play(mod$patterns[[1]], samples = mod$samples)

  ## Or a subset of a pattern
  play(mod$patterns[[1]][17:32, 1:2], samples = mod$samples)

  ## Or even an individual cell
  play(mod$patterns[[1]][1, 1][[1]], samples = mod$samples)
}
```

---

pt2_cell *Select a cell from a ProTracker pattern table*

Description

A cell is an element at a specific row and column (channel). It holds information aboute the note to be played, the instrument (sample) number and the effect to be applied. For mor information about cells (class `pt2cell`) consult `vignette("s3class")`. For more information about selecting elements from ProTrackR2 class objects check out `vignette("select_opts")`.

Usage

```
pt2_cell(pattern, i, j, ...)
```

Arguments

`pattern`	A `pt2pat` class object to extract a cell (`pt2cell`) from.
`i, j`	Indices for extracting or replacing ProTrackR2 object elements. The indices starts at 0, for consistency with ProTracker!
`...`	Ignored

Value

Returns a cell object from the table as class `pt2cell`.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())

pt2_cell(mod$patterns[[1]], 0L, 0L)
```

`pt2_command` *Extract effect commands from a ProTracker module*

Description

As explained in `vignette("s3class")`, the ProTracker pattern table consists of cells, containing information about the note and instrument to be played. This function can be used to retrieve or replace the effect commands in a module.

Usage

```
pt2_command(x, ...)

pt2_command(x, silent = TRUE, ...) <- value
```

Arguments

`x`	An object of class `pt2cell`, which can be extracted from a pattern table with `pt2_cell()`. A cell list (class `pt2celllist`) is also allowed. See `vignette("sel_assign")` for more details about selecting cells and cell lists.
`...`	Ignored
`silent`	Don't warn about replacement values not being used or recycled.
`value`	A replacement value. It should be an object that can be converted into an effect command. It can be a `character` string as shown in the example below.

Value

Returns a `pt2command` object containing the raw command code. In case of the assign operator (`<-`) an update version of `x` is returned.

Examples

```
mod <- pt2_read_mod(pt2_demo())

## select a specific cell from the module
cell <- pt2_cell(mod$patterns[[1]], 0L, 0L)

## show the command used for this cell
pt2_command(cell)

## convert character strings into ProTracker commands
pt2_command(c("C30", "F06"))

## Set the command for all cells in the first pattern
## to `C20` (volume at 50%):
pt2_command(mod$patterns[[1]][]) <- "C20"
```

---

pt2_demo *Path to demonstration ProTracker module file*

Description

A path to demonstration ProTracker module file. It can be used for demonstration purposes.

Usage

```
pt2_demo()
```

Value

Returns a string with a path of a ProTracker module file

Author(s)

Pepijn de Vries

---

pt2_duration *Calculate the duration of the module*

Description

How long a module will play depends on several aspects such as the length of the pattern sequence table (`pt2_pattern_table()`, `pt2_length()`), the speed and tempo at which the patterns are defined, loops, pattern breaks and delay effects. The duration in seconds of the module is calculated by this function.

Usage

```
pt2_duration(x, options = pt2_render_options(), position = 0L, ...)
```

Arguments

x	Object for which to determine the duration. Should be of class pt2mod.
options	A list of options used for rendering the audio. Use pt2_render_options() to obtain default options, or modify them.
position	Starting position in the pattern sequence table (pt2_pattern_table()). Should be a non negative value smaller than the mule length (pt2_length()).
...	Ignored

Value

The duration in seconds (as a difftime class object)

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())

pt2_duration(mod)
pt2_duration(mod, pt2_render_options(timing_mode = "cia"))
pt2_duration(mod, pt2_render_options(timing_mode = "vblank"))
```

---

pt2_finetune *Get or set ProTracker sample properties*

Description

Get or set properties of a ProTracker sample. See 'details' section for available properties and associated functions.

Usage

```
pt2_finetune(sample, ...)

pt2_finetune(sample, ...) <- value

pt2_volume(sample, ...)

pt2_volume(sample, ...) <- value

pt2_loop_start(sample, ...)
```

```
pt2_loop_start(sample, ...) <- value

pt2_loop_length(sample, ...)

pt2_loop_length(sample, ...) <- value

pt2_is_looped(sample, ...)

pt2_is_looped(sample, ...) <- value
```

Arguments

sample	A module sample of class pt2samp.
...	Ignored
value	Replacement value for the sample property

Details

ProTracker audio samples hold some meta data that affect their playback. The following functions can be used to get or set these properties:

- pt2_finetune(): When a sample is out of tune, you can use the use the 'finetune' value to tune the sample. It is an integer value ranging between -8 and +7.
- pt2_volume(): An integer value to adjust the sample volume. It ranges between 0 (silent) to 64 (maximum volume).
- pt2_loop_start() and pt2_loop_length(): defines if a sample is looped. The loop start is defined as an even integer indicating the sample index (zero based) where the loop should start. The loop length is the number of samples, over which to loop. The loop length should also be an even number greater than 0. The sum of the loop start and the loop length should never be larger than the sample's length.
- pt2_is_looped(): a logical value, indicating if the sample is looped. If you set the loop status to TRUE when the sample is not looped yet, the new loop will start at index 0, and will have a length equal to the sample length.

Value

Returns an integer or logical value. Depending on the called function. See the detail section for specifics.

See Also

pt2_name()

Examples

```
mod <- pt2_read_mod(pt2_demo())

pt2_finetune(mod$samples[[1]])
```

```
pt2_finetune(mod$samples[[1]]) <- -8L

pt2_volume(mod$samples[[2]])
pt2_volume(mod$samples[[2]]) <- 64L

pt2_loop_start(mod$samples[[1]])
pt2_loop_start(mod$samples[[1]]) <- 400L

pt2_loop_length(mod$samples[[1]])
pt2_loop_length(mod$samples[[1]]) <- 274L

pt2_is_looped(mod$samples[[2]])
pt2_is_looped(mod$samples[[2]]) <- TRUE
```

`pt2_instrument` *Extract or replace a sample index from a ProTracker pattern*

Description

As explained in `vignette("s3class")`, the ProTracker pattern table consists of cells containing information about the note and instrument to be played. This function extracts the sample index (instrument) from such a cell.

Usage

```
pt2_instrument(x, ...)

pt2_instrument(x, silent = TRUE, ...) <- value
```

Arguments

`x`	An object of class `pt2cell`, which can be extracted from a pattern table with `pt2_cell()`. A cell list (class `pt2celllist`) is also allowed. See `vignette("sel_assign")` for more details about selecting cells and cell lists.
`...`	Ignored.
`silent`	Don't warn about replacement values not being used or recycled.
`value`	Replacement value for the instrument (sample id). An `integer` value ranging from `0` to `31`.

Value

Returns the `integer` sample index in `x`. The index has a base of `1`. An index of `0` means 'no sample'. In case of the assignment operator (`<-`) an updated version of `x` is returned

Examples

```
mod <- pt2_read_mod(pt2_demo())

## select a specific cell from the first pattern
cell <- pt2_cell(mod$patterns[[1]], 0L, 0L)

## get the sample number used in this cell
pt2_instrument(cell)

## Replace the instrument in all cells of
## pattern 1 with sample number 3:
pt2_instrument(mod$patterns[[1]][]) <- 3
```

---

pt2_length *Obtain ProTracker module information*

Description

Obtain information about a ProTracker module or embedded samples.

Usage

```
pt2_length(mod, ...)

pt2_length(mod, ...) <- value

pt2_n_pattern(mod, ...)

pt2_pattern_table(mod, ...)

pt2_pattern_table(mod, ...) <- value

pt2_name(x, ...)

pt2_name(x, ...) <- value

## S3 method for class 'pt2mod'
pt2_name(x, ...)

## S3 replacement method for class 'pt2mod'
pt2_name(x, ...) <- value

## S3 method for class 'pt2samp'
pt2_name(x, ...)

## S3 replacement method for class 'pt2samp'
pt2_name(x, ...) <- value
```

```
## S3 method for class 'pt2samplist'
pt2_name(x, ...)

## S3 replacement method for class 'pt2samplist'
pt2_name(x, ...) <- value

pt2_n_sample(mod, ...)
```

Arguments

...	Ignored
value	Replacement value. In case of: • pt2_length<-: new length of a module in number of patterns in the pattern • pt2_pattern_table<-: a new patter table. A 128 long vector of integers between 0 and 99. • pt2_name<-: a new name (or names) for x
x, mod	A pt2mod class object for which to obtain information. For x also samples of class pt2samp are allowed as input.

Details

You can use the following functions to get or set information on a ProTracker modules (represented by pt2mod class objects):

- pt2_length(): get or set the length the pattern table.
- pt2_n_pattern(): number of distinct patterns. Same as length(mod$patterns).
- pt2_pattern_table(): get or set table of pattern indexes. Patterns will be played in this order. Normally, only the first pt2_length() number of listed patterns are played. Patterns beyond this length are only played if you explicitly start the module at that position, or the module contains jump commands.
- pt2_name(): get or set the name of a module. Can also be used on samples in a module. Names in a ProTracker module are limited. Module names are automatically truncated to 20 UTF8 characters. For samples, the names are truncated to 22 characters.

Value

Returns information about the specified ProTracker module

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())

pt2_length(mod)
pt2_n_pattern(mod)
pt2_n_sample(mod)
pt2_pattern_table(mod)
pt2_name(mod)
pt2_name(pt2_sample(mod, 4L))

mod2 <- pt2_new_mod("new")
pt2_length(mod2) <- 3L
pt2_pattern_table(mod2)[1L:3L] <- c(0L, 2L, 1L)
pt2_name(mod2) <- "foobar"
```

`pt2_new_mod` *Create a new (empty) ProTracker module*

Description

Creates an empty ProTracker module, it is returned as a `pt2mod` class object.

Usage

```
pt2_new_mod(name, ...)
```

Arguments

`name`	Name for the new module. It will be truncated if longer than 20 characters.
`...`	Ignored

Value

A `pt2mod` class module, with no samples and one empty pattern.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_new_mod("my_song")
```

pt2_new_pattern *Create a new ProTracker pattern*

Description

[Deprecated] Creates a new ProTracker pattern, consisting of four channels and 64 rows.

Usage

```
pt2_new_pattern(..., compact = TRUE)
```

Arguments

`...`	Currently ignored
`compact`	Should the pattern be formatted using a compact notation (as used for file storage), or a none-compact format as used by the player? This can be set with the `compact` argument.

Value

Returns a new clean `pt2pat` object.

Author(s)

Pepijn de Vries

pt2_note *Extract a note from a ProTracker module*

Description

Gets note information from a cell in a pattern table in a ProTracker Module.

Usage

```
pt2_note(x, ...)

pt2_note(x, silent = TRUE, ...) <- value
```

Arguments

`x`	An object of class `pt2cell`, which can be extracted from a pattern table with `pt2_cell()`. A cell list (class `pt2celllist`) is also allowed. See `vignette("sel_assign")` for more details about selecting cells and cell lists.
`...`	Ignored
`silent`	Don't warn about replacement values not being used or recycled.
`value`	A `character` string to replace the selected notes from `x`.

Details

A string representing the note's key is returned by the function. The first letter indicates the position of the note in the diatonic scale. The second character indicates if it is a sharp key (with a hash symbol, and a dash if it is not). The third character indicates the octave of the note. In ProTracker allowed notes range from `"C-1"` to `"B-3"`.

Value

Returns a string representing the note's key.

Examples

```
mod <- pt2_read_mod(pt2_demo())

## select a specific cell from the first pattern
cell <- pt2_cell(mod$patterns[[1]], 0L, 0L)

## get the note played by this particular cell
pt2_note(cell)

## Replace the notes in the first pattern
## with those of the first bar of
## 'Frère Jacques'
pt2_note(mod$patterns[[1]][]) <-
  c("C-2", "---", "---", "---",
    "D-2", "---", "---", "---",
    "E-2", "---", "---", "---",
    "C-2", "---", "---", "---")
```

`pt2_note_to_period` *Get a corresponding period value from a note string*

Description

Back in the days, ProTracker was hardware driven on a Commodore Amiga. It made advantage of a custom chipset where each chip had specific tasks. One of the chips (named Paula) could play 8 bit audio samples stored in memory directly to one of the four audio channels. On that chip you could set the integer 'period' value which is inversely related to the sample rate at which the sample is played. Hence, it defines the pitch of the sample. ProTracker used the period value to play different notes. With this function you can convert a `character` string representing a note to its corresponding period value used by Paula.

Usage

```
pt2_note_to_period(note, empty_char = "-", finetune = 0, ...)
```

Arguments

note	A character string representing notes (see also pt2_note()).
empty_char	A character that is used to represent empty values.
finetune	ProTracker used integer finetune values to tweak the playback rate. it should be in the range of -8, up to +7.
...	Ignored.

Value

Returns a vector of integer period values.

Examples

```
pt2_note_to_period(c("A#2", "C-1"))
```

---

pt2_pattern *Create or retrieve a pattern from a ProTracker module*

Description

Get a pattern table (sequence of notes and effects on each of the 4 channels) at a specific index from a ProTracker module. If mod is missing, a new empty pattern is created.

Usage

```
pt2_pattern(mod, i, ..., compact = TRUE)
```

Arguments

mod	A pt2mod class objects from which to retrieve a pattern table
i	The index (integer) of the pattern. Note that the index starts at 0.
...	Ignored
compact	Should the pattern be formatted using a compact notation (as used for file storage), or a none-compact format as used by the player? This can be set with the compact argument.

Value

A pt2pat object representing the pattern.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())

pt2_pattern(mod, 0L)
pt2_pattern()
```

`pt2_read_mod` *Read and write ProTracker modules*

Description

Functions to read and write ProTracker module. The read function will read a number of mod files that are compatible with ProTracker, this includes files compressed with PowerPacker (PP). The write function will only write modules conform ProTracker specifications.

Usage

```
pt2_read_mod(file, ...)

pt2_write_mod(mod, file, ...)
```

Arguments

`file`	Filename of the file to read from or write to.
`...`	Ignored
`mod`	An object of class `pt2mod`.

Value

`pt2_read_mod()` returns a `pt2mod` class object when successful. `pt_write_mod()` returns `NULL` invisibly.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())
```

`pt2_read_sample` *Read and write ProTracker audio samples*

Description

Functions to read and write ProTracker audio samples. Reading is supported for common types of WAV, IFF and AIFF files. Writing is supported for WAV and IFF files.

Usage

```
pt2_read_sample(file, ...)

pt2_write_sample(sample, file, ...)
```

Arguments

`file`	Filename of the file to read from or write to. For `pt2_write_sample()` the file extension will be used to determine which file format to write.
`...`	Ignored
`sample`	An object of class `pt2samp`.

Value

`pt2_read_sample()` returns a `pt2samp` class object when successful. `pt_write_sample()` returns `NULL` invisibly.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())
my_sample <- pt2_sample(mod, 1L)
my_sample_file <- tempfile(fileext = ".iff")
pt2_write_sample(my_sample, my_sample_file)
```

pt2_render *Render ProTracker modules and other objects to a playable format*

Description

Renders a 16bit pulse-code modulation waveform from a ProTracker module. The rendered format can be played on a modern machine.

Usage

```
pt2_render(x, duration = NA, options = pt2_render_options(), ...)

## S3 method for class 'pt2mod'
pt2_render(
  x,
  duration = NA,
  options = pt2_render_options(),
  position = 0L,
  ...
)

## S3 method for class 'pt2samp'
pt2_render(x, duration = 5, options = pt2_render_options(), note = "C-3", ...)

## S3 method for class 'pt2patlist'
pt2_render(x, duration = NA, options = pt2_render_options(), samples, ...)

## S3 method for class 'pt2pat'
pt2_render(x, duration = NA, options = pt2_render_options(), samples, ...)

## S3 method for class 'pt2celllist'
pt2_render(x, duration = 5, options = pt2_render_options(), samples, ...)

## S3 method for class 'pt2cell'
pt2_render(x, duration = 5, options = pt2_render_options(), samples, ...)
```

Arguments

x	The object to be rendered
duration	Duration of the rendered output in seconds. When set to NA the duration of the module is calculated and used for rendering.
options	A list of options used for rendering the audio. Use pt2_render_options() to obtain default options, or modify them.
...	Ignored
position	Starting position in the pattern sequence table (pt2_pattern_table()). Should be a non negative value smaller than the mule length (pt2_length()).

note	Note to be played when x is a pt2samp class object. Defaults to "C-3".
samples	When rendering or playing patterns (or elements of it), samples are needed to interpret the pattern. Pass the samples as a sample list (class pt2samplist).

Value

Rendered audio inheriting the audio::audioSample() class.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())
aud <- pt2_render(mod)
aud_samp <- pt2_render(mod$samples[[1]])
```

---

pt2_render_options *Retrieve options for rendering*

Description

Retrieve options for rendering ProTracker modules. See also pt2_render().

Usage

```
pt2_render_options(...)
```

Arguments

...	Specify custom options.

Value

Returns a named list of options that can be used for rendering ProTracker modules (see pt2_render() and play()). It contains the following elements:

- sample_rate: an integer value specifying the sample rate of the output in Hz.
- stereo_separation: an integer percentage determining how much the tracker channels will be separated to the left and right stereo output channels.
- amiga_filter: a character string specifying the hardware filter to be emulated. Can be "A500" for emulating Amiga 500 hardware filters, or "A1200" for emulating Amiga 1200 hardware filters.
- speed: An integer value specifying the initial speed of the module measured in 'ticks' per row. Should be in range of 1 and 31.

- `tempo`: An integer value specifying the initial tempo of the module. When speed is set to `6`, it measures the tempo as beats per minute. Should be in the range of `32` and `255`
- `led_filter`: A `logical` value specifying the state of the hardware LED filter to be emultated.
- `timing_mode`: on the original Commodore Amiga timing in tracker modules could be handled using different approaches. The first is the 'vertical blanking' method, were timing was based on each time the monitor blanks (before being redrawn). This method thus depends on the monitor that was used. PAL monitors operated at approximately 50 Hz, wereas NTCS monitors used 60 Hz. Alternatively, the Complex Interface Adapter (CIA) offer hardware-level timing and was system independent. You can set the timing mode by specifying `"cia"` (default) here, or `"vblank"` (currently, only PAL is supported).

Author(s)

Pepijn de Vries

Examples

```
pt2_render_options(stereo_separation = 100)
```

`pt2_sample` *Obtain sample data and info from a ProTracker module*

Description

Obtain sample data and info from a ProTracker module at a specific index. ProTracker modules can hold up to 31 samples. The index should range from 0 to 30.

Usage

```
pt2_sample(mod, i, ...)
```

Arguments

`mod`	An object of class `pt2mod` from which to obtain sample data and information
`i`	The index of the requested sample (between 0 and 30).
`...`	Ignored.

Value

Returns a sample object of class `pt2samp`.

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())

smp <- pt2_sample(mod, 0L)
```

`pt2_sample_to_audio` *Coerce ProTracker sample to audio sample*

Description

Coerce a sample from a ProTracker module to an `audio` package sample. Note that this function differs from `pt2_render()` as it will not mimic Commodore Amiga hardware. It will just pass the pure sample data.

Usage

```
pt2_sample_to_audio(
  sample,
  note = "C-3",
  finetune = 0,
  options = pt2_render_options(),
  loop = 20L,
  ...
)
```

Arguments

`sample`	A ProTracker sample of class `pt2samp`.
`note`	A character representing a note (`"C-3"` by default). It is used to calculate the playback rate for the sample.
`finetune`	An `integer` fine tune value (between -8 and 7). Used to tune the sample's note.
`options`	Options used for calculating play back rate. See `pt2_render_options()` for all available options. But note that not all options affect the play back rate.
`loop`	An `integer` value indicating how often the sample should be looped (if it is looped). It will be ignored when `loop` has a value of zero (or less), or when the sample is not looped (see `pt2_is_looped()`).
`...`	Ignored

Value

Returns an audio sample of class `audio::audioSample()`.

Examples

```
mod <- pt2_read_mod(pt2_demo())
aud <- pt2_sample_to_audio(mod$samples[[1]])
```

pt2_validate *Validate ProTrackR2 S3 class objects*

Description

Check aspects of S3 class objects for validity. For samples for instance it is checked if all parameters (volume, finetune, etc.) are within ProTracker specifications.

Usage

```
pt2_validate(x, ...)

## S3 method for class 'pt2samp'
pt2_validate(x, ...)
```

Arguments

x	object to be validated
...	Ignored

Value

A `logical` value indicating whether the object is valid or not

Author(s)

Pepijn de Vries

Examples

```
mod <- pt2_read_mod(pt2_demo())

pt2_validate(mod$samples[[1]])
```

$.pt2mod *Select and assign operators for ProTrackR2 S3 class objects*

Description

Functions to select and assign elements to ProTracker modules. See `vignette('s3class')` for an overview of ProTrackR2 S3 class objects. See `vignette('sel_assign')` for practical guidance on selecting and assigning elements of ProTrackR2 class objects.

Usage

```
## S3 method for class 'pt2mod'
x$i, ...

## S3 replacement method for class 'pt2mod'
x$i <- value

## S3 replacement method for class 'pt2mod'
x[[i]] <- value

## S3 method for class 'pt2mod'
x[[i, ...]]

## S3 method for class 'pt2patlist'
x[i, ...]

## S3 method for class 'pt2patlist'
x[[i, ...]]

## S3 replacement method for class 'pt2patlist'
x[[i]] <- value

## S3 method for class 'pt2samplist'
x[i, ...]

## S3 method for class 'pt2samplist'
x[[i, ...]]

## S3 replacement method for class 'pt2samplist'
x[[i]] <- value

## S3 method for class 'pt2pat'
x[[i, ...]]

## S3 replacement method for class 'pt2pat'
x[[i]] <- value

## S3 method for class 'pt2pat'
x[i, j, ...]

## S3 replacement method for class 'pt2pat'
x[i, j, ...] <- value

## S3 replacement method for class 'pt2celllist'
x[[i, ...]] <- value

## S3 replacement method for class 'pt2celllist'
x[i, ...] <- value
```

```
## S3 method for class 'pt2celllist'
x[[i, ...]]

## S3 method for class 'pt2celllist'
x[i, ...]

## S3 method for class 'pt2command'
x[[i, ...]]

## S3 method for class 'pt2command'
x[i, ...]

## S3 replacement method for class 'pt2command'
x[[i, ...]] <- value

## S3 replacement method for class 'pt2command'
x[i, ...] <- value
```

Arguments

x	Object to apply S3 method to. See 'usage' section for allowed object types.
i, j	Indices for extracting or replacing ProTrackR2 object elements
...	Passed on to other methods.
value	Replacement value for the selected element(s).

Value

Returns the selected object in case of a selection ([, [[, or $) operator. Returns the updated object x in case of an assignment (<-) operator.

Index