ptulsconv Documentation

Release 2.1.0

Jamie Hardt

Nov 16, 2023

USER DOCUMENTATION

ptulsconv is a tool for converting Pro Tools text exports into PDF reports for ADR spotting. It can also be used for converting text exports into JSON documents for processing by other applications.

CHAPTER

ONE

QUICK START

The workflow for creating ADR reports in *ptulsconv* is similar to other ADR spotting programs: spot ADR lines in Pro Tools with clips using a special code to take notes, export the tracks as text and then run the program.

1.1 Step 1: Use Pro Tools to Spot ADR Lines

ptulsconv can be used to spot ADR lines similarly to other programs.

1. Create a new Pro Tools session, name this session after your project.
2. Create new tracks, one for each character. Name each track after a character.
3. On each track, create a clip group (or edit in some audio) at the time you would like an ADR line to appear in the report. Name the clip after the dialogue you are replacing at that time.

1.2 Step 2: Add More Information to Your Spots

Clips, tracks and markers in your session can contain additional information to make your ADR reports more complete and useful. You add this information with *tagging*.

- **Every ADR clip must have a unique cue number.** After the name of each clip, add the letters `$QN=` and then a unique number (any combination of letters or numbers that don't contain a space). You can type these yourself or add them with batch-renaming when you're done spotting.
- ADR spots should usually have a reason indicated, so you can remember exactly why you're replacing a particular line. Do this by adding the the text `{R=` to your clip names after the prompt and then some short text describing the reason, and then a closing `}`. You can type anything, including spaces.
- If, for example, a line is a TV cover line, you can add the text `[TV]` to the end.

So for example, some ADR spot's clip name might look like:

```
Get to the ladder! {R=Noise} $QN=J1001
"Forget your feelings! {R=TV Cover} $QN=J1002 [TV]
```

These tags can appear in any order.

- You can add the name of an actor to a character's track, so this information will appear on your reports. In the track name, or in the track comments, type `{Actor=xxx}` replacing the xxx with the actor's name.
- Characters need to have a number (perhaps from the cast list) to express how they should be collated. Add `$CN=xxx` with a unique number to each track (or the track's comments.)

- Set the scene for each line with markers. Create a marker at the beginning of a scene and make it's name `{Sc=xxx}`, replacing the xxx with the scene number and name.

1.3 Step 3: Run *ptulsconv*

In Pro Tools, select the tracks that contain your spot clips.

Then, in your Terminal, run the following command:

```
ptulsconv
```

ptulsconv will connect to Pro Tools and read all of the clips on the selected track. It will then create a folder named "Title_CURRENT_DATE", and within that folder it will create several PDFs and folders:

- "TITLE ADR Report" a PDF tabular report of every ADR line you've spotted.
- "TITLE Continuity" a PDF listing every scene you have indicated and its timecode.
- "TITLE Line Count" a PDF tabular report giving line counts by reel, and the time budget per character and reel (if provided in the tagging).
- "CSV/" a folder containing CSV documents of all spotted ADR, groupd by character and reel.
- "Director Logs/" a folder containing PDF tabular reports, like the overall report except groupd by character.
- "Supervisor Logs/" a folder containing PDF reports, one page per line, designed for note taking during a session, particularly on an iPad.
- "Talent Scripts/" a folder containing PDF scripts or sides, with the timecode and prompts for each line, grouped by character but with most other information suppressed.

CHAPTER

TWO

TAGGING

Tags are used to add additional data to a clip in an organized way. The tagging system in *ptulsconv* is flexible and can be used to add any kind of extra data to a clip.

2.1 Fields in Clip Names

Track names, track comments, and clip names can also contain meta-tags, or "fields," to add additional columns to the output. Thus, if a clip has the name::

```
Fireworks explosion {note=Replace for final} $V=1 [FX] [DESIGN]
```

The row output for this clip will contain columns for the values:

Clip Name	note	V	FX	DESIGN
Fireworks explosion	Replace for final	1	FX	DESIGN

These fields can be defined in the clip name in three ways:

- `$NAME=VALUE` creates a field named `NAME` with a one-word value `VALUE`.
- `{NAME=VALUE}` creates a field named `NAME` with the value `VALUE`. `VALUE` in this case may contain spaces or any chartacter up to the closing bracket.
- `[NAME]` creates a field named `NAME` with a value `NAME`. This can be used to create a boolean-valued field; in the output, clips with the field will have it, and clips without will have the column with an empty value.

For example, if three clips are named::

```
"Squad fifty-one, what is your status?" [FUTZ] {Ch=Dispatcher} [ADR]

"We are ten-eight at Rampart Hospital." {Ch=Gage} [ADR]

(1M) FC callouts rescuing trapped survivors. {Ch=Group} $QN=1001 [GROUP]
```

The output will contain the range:

Clip Name	Ch	FUTZ	ADR	QN	GROUP
"Squad fifty-one, what is your status?"	Dispatcher	FUTZ	ADR		
"We are ten-eight at Rampart Hospital."	Gage		ADR		
(1M) FC callouts rescuing trapped survivors.	Group			1001	GROUP

2.2 Fields in Track Names and Markers

Fields set in track names, and in track comments, will be applied to *each* clip on that track. If a track comment contains the text `{Dept=Foley}` for example, every clip on that track will have a "Foley" value in a "Dept" column.

Likewise, fields set on the session name will apply to all clips in the session.

Fields set in markers, and in marker comments, will be applied to all clips whose finish is *after* that marker. Fields in markers are applied cumulatively from breakfast to dinner in the session. The latest marker applying to a clip has precedence, so if one marker comes after the other, but both define a field, the value in the later marker

An important note here is that, always, fields set on the clip name have the highest precedence. If a field is set in a clip name, the same field set on the track, the value set on the clip will prevail.

2.3 Apply Fields to a Time Range of Clips

A clip name beginning with `@` will not be included in the output, but its fields will be applied to clips within its time range on lower tracks.

If track 1 has a clip named `@ {Sc=1- The House}`, any clips beginning within that range on lower tracks will have a field `Sc` with that value.

2.4 Combining Clips with Long Names or Many Tags

A clip name beginning with `&` will have its parsed clip name appended to the preceding cue, and the fields of following cues will be applied, earlier clips having precedence. The clips need not be touching, and the clips will be combined into a single row of the output. The start time of the first clip will become the start time of the row, and the finish time of the last clip will become the finish time of the row.

2.5 Setting Document Options

Note: Document options are not yet implemented.

CHAPTER

THREE

PTULSCONV FOR ADR REPORT GENERATION

3.1 Reports Created by the ADR Report Generator

(FIXME: write this)

3.2 Tags Used by the ADR Report Generator

3.2.1 Project-Level Tags

It usually makes sense to place these either in the session name, or on a *marker* at the beginning of the session, so it will apply to all of the clips in the session.

Title

The title of the project. This will appear at the top of every report.

Warning: *ptulsconv* at this time only supports one title per export. If you attempt to use multiple titles in one export it will fail.

Supv

The supervisor of the project. This appears at the bottom of every report.

Client

The client of the project. This will often appear under the title on every report.

Spot

The date or version number of the spotting report.

3.2.2 Time Range Tags

All of these tags can be set to different values on each clip, but it often makes sense to use these tags in a *time range*.

Sc

The scene description. This appears on the continuity report and is used in the Director's logs.

Ver

The picture version. This appears beside the spot timecodes on most reports.

Reel

The reel. This appears beside the spot timecodes on most reports and is used to summarize line totals on the line count report.

3.2.3 Line tags

P
: Priority.

QN
: Cue number. This appears on all reports.

Warning: *ptulsconv* will verify that all cue numbers in a given title are unique.

All lines must have a cue number in order to generate reports, if any lines do not have a cue number set, *ptulsconv* will fail.

CN
: Character number. This is used to collate character records and will appear on the line count and in character-collated reports.

Char
: Character name. By default, a clip will set this to the name of the track it appears on, but the track name can be overridden here.

Actor
: Actor name.

Line
: The prompt to appear for this ADR line. By default, this will be whatever text appears in a clip name prior to the first tag.

R
: Reason.

Mins
: Time budget for this line, in minutes. This is used in the line count report to give estimated times for each character. This can be set for the entire project (with a *marker*), or for individual actors (with a tag in the *track comments*), or can be set for individual lines to override these.

Shot
: Shot. A Date or other description indicating the line has been recorded.

3.2.4 Boolean-valued ADR Tag Fields

EFF
: Effort. Lines with this tag are subtotaled in the line count report.

TV
: TV line. Lines with this tag are subtotaled in the line count report.

TBW
: To be written.

ADLIB
: Ad-lib.

OPT
: Optional. Lines with this tag are subtotaled in the line count report.

CHAPTER

FOUR

COMMAND-LINE REFERENCE

4.1 Usage Form

Invocations of ptulsconv take the following form:

```
ptulsconv [options] IN_FILE
```

4.2 Flags

-h, –help
: Show the help message.

f FMT, –format=FMT
: Select the output format. By default this is *doc*, which will generate *ADR reports*.

  The *other available options* are *raw* and *tagged*.

4.2.1 Informational Options

These options display information and exit without processing any input documents.

–show-formats
: Display information about available output formats.

–show-available-tags
: Display information about tags that are used by the report generator.

4.3 Alternate Output Formats

4.3.1 *raw* Output

The "raw" output format is a JSON document of the parsed input data.

The document is a top-level dictionary with keys for the main sections of the text export: *header*, *files*, *clips*, *plugins*, *tracks* and *markers*, and the values for these are a list of section entries, or a dictionary of values, in the case of *header*.

The text values of each record and field in the text export is read and output verbatim, no further processing is done.

4.3.2 *tagged* Output

The “tagged” output format is also a JSON document based on the parsed input data, after the additional step of processing all of the *tags* in the document.

The document is a top-level array of dictionaries, one for each recognized ADR spotting clip in the session. Each dictionary has a *clip_name*, *track_name* and *session_name* key, a *tags* key that contains a dictionary of every parsed tag (after applying tags from all tracks and markers), and a *start* and *end* key. The *start* and *end* key contain the parsed timecode representations of these values in rational number form, as a dictionary with *numerator* and *denominator* keys.

CHAPTER

FIVE

CONTRIBUTING

5.1 Testing

Before submitting PRs or patches, please make sure your branch passes all of the unit tests by running Pytest.

CHAPTER SIX

THEORY OF OPERATION

6.1 Execution Flow When Producing "doc" Output

1. The command line argv is read in `ptulsconv.__main__.main()`, which calls *`ptulsconv.commands.convert()`*
2. *`ptulsconv.commands.convert()`* reads the input with `ptuslconv.docparser.doc_parser_visitor()`, which uses the `parsimonious` library to parse the input into an abstract syntax tree, which the parser visitor uses to convert into a `ptulsconv.docparser.doc_entity.SessionDescriptor`, which structures all of the data in the session output.
3. The next action based on the output format. In the case of the "doc" output format, it runs some validations on the input, and calls *`ptulsconv.commands.generate_documents()`*.
4. *`ptulsconv.commands.generate_documents()`* creates the output folder, creates the Continuity report with `ptulsconv.pdf.continuity.output_continuity()` (this document requires some special-casing), and at the tail calls…
5. *`ptulsconv.commands.create_adr_reports()`*, which creates folders for

(FIXME finish this)

CHAPTER

SEVEN

PARSING

7.1 Docparser Classes

class ptulsconv.docparser.adr_entity.**ADRLine**(*title: str = '', supervisor: Optional[str] = None, client: Optional[str] = None, scene: Optional[str] = None, version: Optional[str] = None, reel: Optional[str] = None, start: fractions.Fraction = Fraction(0, 1), finish: fractions.Fraction = Fraction(0, 1), omitted: bool = False, note: Optional[str] = None, requested_by: Optional[str] = None, priority: Optional[int] = None, cue_number: Optional[str] = None, character_id: Optional[str] = None, character_name: Optional[str] = None, actor_name: Optional[str] = None, prompt: Optional[str] = None, reason: Optional[str] = None, time_budget_mins: Optional[float] = None, spot: Optional[str] = None, shot: Optional[str] = None, effort: bool = False, tv: bool = False, tbw: bool = False, adlib: bool = False, optional: bool = False*)

actor_name: Optional[str] = None

adlib: bool = False

character_id: Optional[str] = None

character_name: Optional[str] = None

cue_number: Optional[str] = None

effort: bool = False

optional: bool = False

priority: Optional[int] = None

prompt: Optional[str] = None

reason: Optional[str] = None

shot: Optional[str] = None

spot: Optional[str] = None

```
tag_mapping = [<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>,
<ptulsconv.docparser.tag_mapping.TagMapping object>]
```

`tbw:  bool = False`

`time_budget_mins:  Optional[float] = None`

`tv:  bool = False`

CHAPTER

EIGHT

AUXILIARY AND HELPER MODULES

8.1 Commands Module

This module provides the main input document parsing and transform implementation.

class ptulsconv.commands.**FractionEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

A subclass of `JSONEncoder` which encodes `Fraction` objects as a dict.

default(*o*)

ptulsconv.commands.**convert**(*major_mode, input_file=None, output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>, warnings=True*)

Primary worker function, accepts the input file and decides what to do with it based on the *major_mode*.

Parameters

- **input_file** – a path to the input file.
- **major_mode** – the selected output mode, 'raw', 'tagged' or 'doc'.

ptulsconv.commands.**create_adr_reports**(*lines: List[ADRLine], tc_display_format: TimecodeFormat, reel_list: List[str]*)

Creates a directory heirarchy and a respective set of ADR reports, given a list of lines.

ptulsconv.commands.**generate_documents**(*session_tc_format, scenes, adr_lines: List[ADRLine], title*)

Create PDF output.

ptulsconv.commands.**output_adr_csv**(*lines: List[ADRLine], time_format: TimecodeFormat*)

Writes ADR lines as CSV to the current working directory. Creates directories for each character number and name pair, and within that directory, creates a CSV file for each reel.

ptulsconv.commands.**perform_adr_validations**(*lines: Iterator[ADRLine]*)

Performs validations on the input.

8.2 Broadcast Timecode Module

Useful functions for parsing and working with timecode.

class `ptulsconv.broadcast_timecode.`**`TimecodeFormat`**(*frame_duration*, *logical_fps*, *drop_frame*)

A struct repperesenting a timecode datum.

`ptulsconv.broadcast_timecode.`**`smpte_to_frame_count`**(*smpte_rep_string: str*, *frames_per_logical_second: int*, *drop_frame_hint=False*) → Optional[int]

Convert a string with a SMPTE timecode representation into a frame count.

Parameters

- **`smpte_rep_string`** – The timecode string
- **`frames_per_logical_second`** – Num of frames in a logical second. This is asserted to be in one of *[24,25,30,48,50,60]*
- **`drop_frame_hint`** – *True* if the timecode rep is drop frame. This is ignored (and implied *True*) if the last separator in the timecode string is a semicolon. This is ignored (and implied *False*) if *frames_per_logical_second* is not 30 or 60.

8.3 Footage Module

Methods for converting string reprentations of film footage.

`ptulsconv.footage.`**`footage_to_seconds`**(*footage: str*) → Optional[Fraction]

Converts a string representation of a footage (35mm, 24fps) into a `Fraction`, this fraction being a some number of seconds.

Parameters

`footage` – A string reprenentation of a footage of the form resembling "90+01".

8.4 Reporting Module

Reporting logic. These methods provide reporting methods to the package and take some pains to provide nice-looking escape codes if we're writing to a tty.

`ptulsconv.reporting.`**`print_advisory_tagging_error`**(*failed_string*, *position*, *parent_track_name=None*, *clip_time=None*)

`ptulsconv.reporting.`**`print_banner_style`**(*message*)

`ptulsconv.reporting.`**`print_fatal_error`**(*message*)

`ptulsconv.reporting.`**`print_section_header_style`**(*message*)

`ptulsconv.reporting.`**`print_status_style`**(*message*)

`ptulsconv.reporting.`**`print_warning`**(*warning_string*)

8.5 Validations Module

Validation logic for enforcing various consistency rules.

class ptulsconv.validations.**ValidationError**(*message: str*, *event: Optional[ptulsconv.docparser.adr_entity.ADRLine] = None*)

  event: Optional[*ADRLine*] = None

  message: str

  report_message()

ptulsconv.validations.**validate_dependent_value**(*input_lines: Iterator[ADRLine]*, *key_field*, *dependent_field*)

  Validates that two events with the same value in *key_field* always have the same value in *dependent_field*

ptulsconv.validations.**validate_non_empty_field**(*input_lines: Iterator[ADRLine]*, *field='cue_number'*)

ptulsconv.validations.**validate_unique_count**(*input_lines: Iterator[ADRLine]*, *field='title'*, *count=1*)

ptulsconv.validations.**validate_unique_field**(*input_lines: Iterator[ADRLine]*, *field='cue_number'*, *scope=None*)

ptulsconv.validations.**validate_value**(*input_lines: Iterator[ADRLine]*, *key_field*, *predicate*)

CHAPTER NINE

INDICES AND TABLES

- modindex
- genindex
- search

PYTHON MODULE INDEX

INDEX