PhpTabs

May 25, 2021

Contents

1 Overview **3**
- 1.1 Requirements . . . 3
- 1.2 Install . . . 3
- 1.3 Supported file formats . . . 3
- 1.4 Contribution and support . . . 3
- 1.5 Running the test suite . . . 4
- 1.6 License . . . 4

2 Parse from files **5**
- 2.1 PhpTabs . . . 5
- 2.2 IOFactory . . . 5

3 Parse from strings **7**
- 3.1 IOFactory . . . 7
- 3.2 PhpTabs . . . 8

4 Export to files **9**
- 4.1 PhpTabs::save($filename, $format = null) . . . 9

5 Export to variables **11**
- 5.1 Phptabs::convert($format) . . . 11
- 5.2 PhpTabs shortcuts . . . 12
- 5.3 PHP array . . . 12

6 Traverse a whole song **15**
- 6.1 Getter and counter rules . . . 15
- 6.2 Traversing example . . . 16

7 Target a track or a measure **19**
- 7.1 onlyTrack . . . 19
- 7.2 onlyMeasure . . . 19
- 7.3 Chaining onlyTrack and onlyMeasure . . . 20

8 Slice tracks or measures **21**
- 8.1 sliceTracks . . . 21
- 8.2 sliceMeasures . . . 22
- 8.3 Chaining sliceTracks and sliceMeasures . . . 22

9 Render a song **23**
9.1 ASCII tabs . . . 23
9.2 Vextab rendering . . . 23

10 Render a song as ASCII **25**
10.1 Quick usage . . . 25
10.2 Available options . . . 26
10.3 Slice and render . . . 28

11 Render a song as Vextab **29**
11.1 Quick Usage . . . 29
11.2 Customize VexTab options . . . 29
11.3 Supported VexTab features . . . 31

12 Calculate measure and beat durations in seconds **35**
12.1 Measure duration . . . 35
12.2 Beat duration . . . 36
12.3 Song duration . . . 36

13 Create a song from scratch **39**
13.1 Very minimalistic tablature . . . 39
13.2 A minimal and working tablature . . . 39
13.3 A working tablature with several tracks and measures . . . 41

14 Performance & caching **45**
14.1 Context . . . 45
14.2 Slicing tracks . . . 46
14.3 Exporting to JSON . . . 47
14.4 Caching . . . 47

15 Architecture **51**
15.1 Component roles . . . 51

16 IOFactory **53**
16.1 create() . . . 53
16.2 fromArray($data) . . . 54
16.3 fromFile($filename, $type) . . . 55
16.4 fromString($content, $type) . . . 55
16.5 fromJsonFile($filename) . . . 56
16.6 fromSerializedFile($filename) . . . 56
16.7 fromJson($string) . . . 57
16.8 fromSerialized($string) . . . 57

17 PhpTabs **59**
17.1 Read song informations . . . 59
17.2 Write song informations . . . 60
17.3 Channels . . . 60
17.4 Measure headers . . . 61
17.5 Tracks . . . 61

18 Music model **63**
18.1 Tree . . . 63
18.2 Traversing the tree is made simple . . . 64
18.3 Traversing the first level . . . 66
18.4 Traversing Channels . . . 67

18.5 Traversing MeasureHeaders 67
18.6 Traversing Tracks 67

PhpTabs is a PHP library for reading, writing and rendering tabs and MIDI files.

It provides direct methods to *read a song name*, get a list of instruments or whatever be your needs.

Phptabs is built on the top of a *music stack* that lets you create or modify your songs.

```php
use PhpTabs\PhpTabs;

$filename = 'my-file.gp5';

$song = new PhpTabs($filename);

// Display some metadata
echo $song->getName();

// Display the number of measures
// for the first track
echo $song->getTrack(0)->countMeasures();
```

CHAPTER 1

Overview

1.1 Requirements

Support for PHP 7.2+ and 8.0

1.2 Install

```
composer require stdtabs/phptabs
```

1.3 Supported file formats

PhpTabs currently supports the following file formats:

- GuitarPro 3 (.gp3)
- GuitarPro 4 (.gp4)
- GuitarPro 5 (.gp5)
- MIDI files (.mid, .midi)
- JSON (.json)
- XML (.xml)

1.4 Contribution and support

The source code is hosted on github at https://github.com/stdtabs/phptabs.

If you have any questions, please open an issue.

You want to write another parser, to fix a bug? Please open a pull request.

To discuss new features, make feedback or simply to share ideas, you can contact me on Mastodon at https://cybre.space/@landrok

1.5 Running the test suite

```
git clone https://github.com/stdtabs/phptabs.git
cd phptabs
composer require phpunit/phpunit
vendor/bin/phpunit
```

1.6 License

PhpTabs is licensed under LGPL2.1+.

CHAPTER 2

Parse from files

There are 2 ways to parse a file.

2.1 PhpTabs

The easiest way is to instanciate a PhpTabs with a filename as first argument.

```
use PhpTabs\PhpTabs;

$filename = 'my-file.gp5';

$song = new PhpTabs($filename);

echo $song->getName();
```

It supports Guitar Pro 3, 4 and 5, MIDI, JSON and PHP serialized files.

The file format is recognized by the file extension (gp3, gp4, gp5, mid, midi, json, ser).

See *all available PhpTabs methods*.

2.2 IOFactory

If you need more control, IOFactory is preferred.

After a read operation, a `PhpTabs` containing the entire song is returned.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';
```

(continues on next page)

(continued from previous page)

```
$song = IOFactory::fromFile($filename);

echo $song->getName();
```

If the file extension is not standard, a parser can be specified as the second parameter to force a file format.

```
use PhpTabs\IOFactory;

$filename = 'my-file.dat';

// The file is PHP serialized
$song = IOFactory::fromFile($filename, 'ser');

echo $song->getName();
```

`IOFactory` offers some other shortcuts to load from a specified parser.

```
use PhpTabs\IOFactory;

// Try to read a JSON file
$tab = IOFactory::fromJsonFile('mytabs.json');

// Try to read a serialized file
$tab = IOFactory::fromSerializedFile('mytabs.dat');
```

See *all available shortcuts for IOFactory*.

CHAPTER 3

Parse from strings

Sometimes, you may need to parse a song from a string (binary or not).

3.1 IOFactory

It's made with the `fromString()` method.

After a read operation, a *PhpTabs* containing the entire song is returned.

```
use PhpTabs\IOFactory;

$content = file_get_contents('my-file.gp5');

$song = IOFactory::fromString($content, 'gp5');

echo $song->getName();
```

The file format is given as second parameter (gp3, gp4, gp5, mid, midi, json, ser).

IOFactory offers some other shortcuts to force a parser.

```
use PhpTabs\IOFactory;

// Parse a JSON content
$song = IOFactory::fromJson($content);

// Parse a PHP serialized content
$song = IOFactory::fromSerialized($content);
```

See *all available shortcuts for IOFactory*.

3.2 PhpTabs

After PhpTabs has been instanciated, you may call a parser.

```
use PhpTabs\PhpTabs;

$content = file_get_contents('my-file.gp5');

$song = new PhpTabs();

// Parse a Guitar Pro string
$song->fromString($content, 'gp5');

echo $song->getName();
```

Warning: All modifications that you made before a `fromString()` call will be erased, including meta informations.

The *fromString()* method returns a PhpTabs instance.

```
use PhpTabs\IOFactory;

$content = file_get_contents('my-file.gp5');

// Render as ASCII in one line
echo IOFactory::create()                    // PhpTabs
              ->fromString($content, 'gp5') // PhpTabs
              ->toAscii();                  // string
```

CHAPTER 4

Export to files

4.1 PhpTabs::save($filename, $format = null)

The `save()` method may be used to store file contents on a disk.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

$song->save('my-file.mid');
```

The destination format is recognized by the file extension (gp3, gp4, gp5, mid, midi, json, ser, yml, xml) and the song is implicitly converted to this format.

The following formats are available:

- `gp3` for Guitar Pro 3
- `gp4` for Guitar Pro 4
- `gp5` for Guitar Pro 5
- `mid` or `midi` for MIDI
- `json`
- `xml`
- `ser` for PHP serialized string
- `txt` or `text` for a textual representation
- `yml` or `yaml`

If the file extension is not standard, a format may be passed as the second parameter.

Of course, you may read, convert and save in one line.

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// The Guitar Pro file is parsed, converted
// and recorded as a JSON string
IOFactory::fromFile($filename)
         ->save('my-file.dat', 'json');
```

CHAPTER 5

Export to variables

5.1 Phptabs::convert($format)

Sometimes, for debugging or storing contents another way, you may want to output a song to a variable.

You may make an explicit conversion with the `convert()` method.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// The Guitar Pro file is parsed, converted and returned as MIDI
// content
$midi = IOFactory::fromFile($filename)->convert('mid');
```

The following parameters are available:

- `gp3` for Guitar Pro 3
- `gp4` for Guitar Pro 4
- `gp5` for Guitar Pro 5
- `mid` or `midi` for MIDI
- `json`
- `xml`
- `ser` for PHP serialized string
- `txt` or `text` for a textual representation
- `yml` or `yaml`

5.2 PhpTabs shortcuts

There are some shortcuts to do that.

```
use PhpTabs\PhpTabs;

$filename = 'my-file.gp5';
$song = new PhpTabs($filename);

// Guitar Pro 3
$gp3 = $song->toGuitarPro3();

// Guitar Pro 4
$gp4 = $song->toGuitarPro4();

// Guitar Pro 5
$gp5 = $song->toGuitarPro5();

// MIDI
$midi = $song->toMidi();

// JSON
$json = $song->toJson();

// XML
$xml = $song->toXml();

// YAML
$yml = $song->toYaml();

// Text
$txt = $song->toText();

// PHP Serialized
$ser = $song->toSerialized();
```

All these methods return strings (binary or not).

5.3 PHP array

You may export a whole song as a PHP array with the `toArray()` method.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';
$song = IOFactory::fromFile($filename);

$array = $song->toArray();
```

Exports are made to visualize the internal music-tree or to communicate with a third-party application.

Exported arrays may be imported with `fromArray()` method.

```php
use PhpTabs\IOFactory;

$song = IOFactory::fromArray($array);
```

This way of reading data is bypassing entire parsing and may lead to better performances for large files.

For those who are interested, there is a *manual dedicated to performances* issues.

Warning: All modifications that you made before a `fromArray()` call will be erased, including meta informations.

CHAPTER 6

Traverse a whole song

PhpTabs makes a song fully-traversable.

Starting from one point, you may find your way with the *Music tree*.

Traversing data is made with getter and counter methods.

A traversal is done in read-write mode

6.1 Getter and counter rules

There are 4 rules for getter names:

1. `get + {objectName} + ()`

   It's a `property getter` method.

   ie: a measure header may only contain one Tempo, so the method name to get the tempo for a given measure is `$header->getTempo()`.

2. `count + {objectName} + s()`

   It's a `nodes counter` method.

   ie: a track may contain several measures, so the method name to count them is `$track->countMeasures()`

3. `get + {objectName} + s()`

   It's a collection getter method, it returns an array with all nodes.

   ie: a track may contain several measures, so the method name to get them is `$track->getMeasures()`.

4. `get + {objectName} + ($index)`

   It gets a node from a collection by its index. `$index` is starting from 0 to n-1, with n=child count (returned by the counter method)

ie: there can be several measures per Track, so the method name to get one measure (the first) is `$track->getMeasure(0)`

When in doubt, reference should be made to the *Music-Model reference*

6.2 Traversing example

In the following example, we'll traverse all tracks, all measures and all beats, the goal is to print all notes.

```
use PhpTabs\Music\Note;
use PhpTabs\PhpTabs;

$tab = new PhpTabs('mytab.gp4');

# Get all tracks
foreach ($tab->getTracks() as $track) {
    # Get all measures
    foreach ($track->getMeasures() as $measure) {
        # Get all beats
        foreach ($measure->getBeats() as $beat) {
            # Get all voices
            foreach ($beat->getVoices() as $voice) {
                # Get all notes
                foreach ($voice->getNotes() as $note) {

                    printNote($note);

                }
            }
        }
    }
}

/**
 * Print all referential
 * based on the note model
 *
 * @param \PhpTabs\Music\Note $note
 */
function printNote(Note $note)
{
    echo sprintf(
        "\nTrack %d - Measure %d - Beat %d - Voice %d - Note %s/%s",
        $note->getVoice()->getBeat()->getMeasure()->getTrack()->getNumber(),
        $note->getVoice()->getBeat()->getMeasure()->getNumber(),
        $note->getVoice()->getBeat()->getStart(),
        $note->getVoice()->getIndex(),
        $note->getValue(),
        $note->getString()
  );
}
```

will output something like

```
Track 1 - Measure 1 - Beat 6240 - Voice 0 - Note 11/3
Track 1 - Measure 1 - Beat 6480 - Voice 0 - Note 0/2

[...]

Track 2 - Measure 1 - Beat 960 - Voice 0 - Note 5/2
Track 2 - Measure 1 - Beat 1920 - Voice 0 - Note 5/2
Track 2 - Measure 1 - Beat 2880 - Voice 0 - Note 5/2
Track 2 - Measure 1 - Beat 3840 - Voice 0 - Note 5/2

[...]
```

All referential can be accessed starting from a note.

Let's rewrite the printNote function in a more readable way.

```php
/**
 * Print all referential
 *
 * @param \PhpTabs\Music\Track   $track
 * @param \PhpTabs\Music\Measure $measure
 * @param \PhpTabs\Music\Beat    $beat
 * @param \PhpTabs\Music\Voice   $voice
 * @param \PhpTabs\Music\Note    $note
 */
function printNote($track, $measure, $beat, $voice, $note)
{
    echo sprintf(
        "\nTrack %d - Measure %d - Beat %d - Voice %d - Note %s/%s",
        $track->getNumber(),
        $measure->getNumber(),
        $beat->getStart(),
        $voice->getIndex(),
        $note->getValue(),
        $note->getString()
  );
}
```

This example does not take into account some aspects of the referential such as rest beats, durations, dead notes, note effects and chord beats.

CHAPTER 7

Target a track or a measure

You may need to target a track or a measure to generate a new complete song.

PhpTabs provides 2 methods to extract a single track or a single measure and obtain a new PhpTabs instance.

7.1 onlyTrack

The method `onlyTrack` returns a new `PhpTabs` only with the targeted track. It accepts a track index as parameter.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Get the new song with only the third track
$new = $song->onlyTrack(2);

// Saving to Guitar Pro 5 file
$new->save('3rd-track-of-my-file.gp5');
```

If you only want to work with a particular track without generating a new song, you may need to have a look to `PhpTabs->getTrack()` method.

7.2 onlyMeasure

The method `onlyMeasure` returns a new `PhpTabs` only with the targeted measure for each track. It accepts a measure index as parameter.

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Get the new song with only the third measure for each track
$new = $song->onlyMeasure(2);
```

If you only want to work with a particular measure without generating a new song, you may need to have a look to `PhpTabs->getTrack(0)->getMeasure(0)` method.

7.3 Chaining onlyTrack and onlyMeasure

You may want to display only one measure for a particular track. In the example below, we'll render the first measure of the first track as an ASCII tab.

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Display track#0 measure#0 as ASCII
echo $song->onlyTrack(0)->onlyMeasure(0)->toAscii();
```

Of course, you may do the same thing in one line (Parse file, target a track, target a measure and render).

```php
echo PhpTabs\IOFactory::fromFile('my-file.gp5')
                ->onlyTrack(0)
                ->onlyMeasure(0)
                ->toAscii();
```

CHAPTER 8

Slice tracks or measures

You may need to slice tracks or measures to generate a new complete song.

PhpTabs provides 2 methods to extract ranges of tracks or measures to obtain a new PhpTabs instance.

8.1 sliceTracks

The method `sliceTracks` returns a new `PhpTabs` with the targeted tracks.

It requires 2 parameters :

- fromTrackIndex
- toTrackIndex

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Get a new song with third and fourth tracks
$new = $song->sliceTracks(2, 3);

// Saving to Guitar Pro 5 file
$new->save('3rd-and-4th-tracks-of-my-file.gp5');
```

If you only want to work with tracks without generating a new song, you may need to have a look to `PhpTabs->getTracks()` method.

8.2 sliceMeasures

The method `sliceMeasures` returns a new `PhpTabs` with the targeted measures for each track.

It accepts 2 parameters :

- fromMeasureIndex
- toMeasureIndex

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Get a new song with the third, fourth
// and fifth measures for each track
$new = $song->sliceMeasures(2, 4);
```

If you only want to work with measures without generating a new song, you may need to have a look to `PhpTabs->getTrack(0)->getMeasures()` method.

8.3 Chaining sliceTracks and sliceMeasures

You may want to display only some measures from particular tracks.

In the example below, we'll render the first and second measures of the first and second tracks as an ASCII tab.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Display tracks #0 and #1, measures #0 and #1 as ASCII
echo $song->sliceTracks(0, 1)->sliceMeasures(0, 1)->toAscii();
```

You may do the same thing in one line (Parse file, slice tracks, slice measures and render).

```
echo PhpTabs\IOFactory::fromFile('my-file.gp5')
                ->sliceTracks(0, 1)
                ->sliceMeasures(0, 1)
                ->toAscii();
```

CHAPTER 9

Render a song

9.1 ASCII tabs

ASCII tablature is a must-have feature, PhpTabs (>= 0.6.0) can render a whole song or a single track as an ASCII string.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Read and parse file
$song = IOFactory::fromFile($filename);

// Render the whole song
echo $song->toAscii();
```

There are a couple of options that can be passed to the `toAscii` method.

See *ASCII renderer* for more options.

9.2 Vextab rendering

PhpTabs (>= 0.5.0) can render a track as a VexTab string.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Render the first track
echo $song->toVextab();
```

Some options can be passed to the `toVextab` method.

See *Vextab renderer* for more options.

CHAPTER 10

Render a song as ASCII

ASCII tablature is a must-have feature, PhpTabs (>= 0.6.0) can *render* a whole song, some measures or tracks as ASCII strings.

10.1 Quick usage

The following code prints the whole song's tabstaves. All tracks are printed.

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Parse and render
// and render as ASCII tabs
echo IOFactory::fromFile($filename)->toAscii();
```

This example will ouput something like:

```
E|------------------------|-10-----10-------------------------------------------------
 ↪----|
B|--------------------X---|-------------13-----------------------------------------
 ↪----|
G|-%-------%------11------|------------------12---12--10------------10----------%-----
 ↪----|
D|-%-------%--------------|------------------------------12--12--------12--10--%---
 ↪12----|
A|------------------------|-------------------------------------------------------
 ↪----|
E|------------------------|-------------------------------------------------------
 ↪----|


E|-----------------------------------|-0-----------3-----------------------|
```

(continues on next page)

(continued from previous page)

```
B|-5-----5-----5-----5-----5-----5-----|-5-----5-----5-----5-----5-----5-----|
G|-------------------------------------|-------------5-----------------------|
D|-------------------------------------|-------------5-----------------------|
A|-------------------------------------|-------------3-----------------------|
E|-------------------------------------|-------------3-----------------------|
```

10.2 Available options

The `toAscii()` method may take an array of parameters.

Name	Default	Description
songHeader	false	Display song metadata
trackHeader	false	Display track number and name
maxLineLength	80	Max length for staves in characters

Track informations can be printed with `trackHeader` option.

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// trackHeader
echo IOFactory::fromFile($filename)->toAscii([
    'trackHeader' => true,
]);
```

This example will ouput something like:

```
Track 1: Guitar
E|-----------------------|-10-----10-------------------------------------------------
 ↪----|
B|--------------------X---|-------------13-----------------------------------------
 ↪----|
G|-%-------%------11------|-------------------12---12--10------------10----------%-----
 ↪----|
D|-%-------%--------------|-------------------------------12--12--------12--10--%---
 ↪12----|
A|-----------------------|-------------------------------------------------------------
 ↪----|
E|-----------------------|-------------------------------------------------------------
 ↪----|


Track 2: Voice
E|-------------------------------------|-0-----------3-----------------------|
B|-5-----5-----5-----5-----5-----5-----|-5-----5-----5-----5-----5-----5-----|
G|-------------------------------------|-------------5-----------------------|
D|-------------------------------------|-------------5-----------------------|
A|-------------------------------------|-------------3-----------------------|
E|-------------------------------------|-------------3-----------------------|
```

Song informations can be printed with `songHeader` option.

```php
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// trackHeader
echo IOFactory::fromFile($filename)->toAscii([
    'songHeader' => true,
    'trackHeader' => true,
]);
```

This example will ouput something like:

```
Title: Testing name
Album: Testing album
Artist: Testing artist
Author: Testing author

Track 1: Guitar
E|-----------------------|-10-----10-------------------------------------------------
 ↪----|
B|--------------------X---|-------------13-----------------------------------------
 ↪----|
G|-%-------%------11------|------------------12---12--10------------10----------%-----
 ↪----|
D|-%-------%--------------|-------------------------------12--12--------12--10--%---
 ↪12----|
A|-----------------------|-------------------------------------------------------------
 ↪----|
E|-----------------------|-------------------------------------------------------------
 ↪----|


Track 2: Voice
E|-----------------------------------|-0-----------3----------------------|
B|-5-----5-----5-----5-----5-----5---|-5-----5-----5-----5-----5-----5-----|
G|-----------------------------------|-------------5----------------------|
D|-----------------------------------|-------------5----------------------|
A|-----------------------------------|-------------3----------------------|
E|-----------------------------------|-------------3----------------------|
```

To format line length as you want, a `maxLineLength` option is available. It represents how many characters can be printed before going to a new line.

```php
use PhpTabs\PhpTabs;

$song = new PhpTabs('my-file.gp5');

// trackHeader
echo $song->toAscii([
    'maxLineLength' => 10,
]);
```

This example will ouput something like:

```
E|-----------------------|
B|--------------------X---|
G|-%-------%------11------|
```

(continues on next page)

(continued from previous page)

```
D|-%-------%--------------|
A|------------------------|
E|------------------------|


E|-10-----10-----------------------------------------------------|
B|-------------13-----------------------------------------------|
G|------------------12---12--10------------10----------%---------|
D|--------------------------------12--12--------12--10--%---12----|
A|-------------------------------------------------------------|
E|-------------------------------------------------------------|
```

10.3 Slice and render

By default, the whole song is rendered. Using *slice* and *only* methods may be useful to target only what you want to display.

Let's see how to render only the first track.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Parse, slice first track
// and render as ASCII tabs
echo IOFactory::fromFile($filename)  // Parse
                 ->onlyTrack(0)      // Slice
                 ->toAscii();        // Render
```

Even better, sometimes a track can be so long that you may want to render only some measures.

In the example below, only the first and second measures of the first track are rendered.

```
use PhpTabs\IOFactory;

$filename = 'my-file.gp5';

// Parse, target the first track,
// slice 2 measures
// and render as ASCII tabs
echo IOFactory::fromFile($filename)     // Parse
                 ->onlyTrack(0)         // Slice
                 ->sliceMeasures(0, 1)  // Slice
                 ->toAscii();           // Render
```

If you need more explanation, let's have a look at their manual.

Slicing tracks and measures

Target a single track or a single measure

CHAPTER 11

Render a song as Vextab

VexTab format is provided by vexflow.com. If you want to know more about VexTab format, there is a good tutorial.

PhpTabs (>= 0.5.0) can render a track as a VexTab string.

11.1 Quick Usage

The following code prints all tabstaves of the first track.

```php
use PhpTabs\PhpTabs;

$song = new PhpTabs('mytab.gp4');

// Render track 0
echo $song->toVextab();
```

This example will ouput something like:

```
options scale=1 space=16 width=520 tempo=66

tabstave notation=true time=4/4

notes | :8 0/5 9/3 7/4 7/3 T7/3 7/4 0/5 5/3
```

With a bit of VexFlow JS, that renders:

11.2 Customize VexTab options

Some options can be passed to override default VexTab options.

If the value is the same as VexTab defaults, it won't be printed.

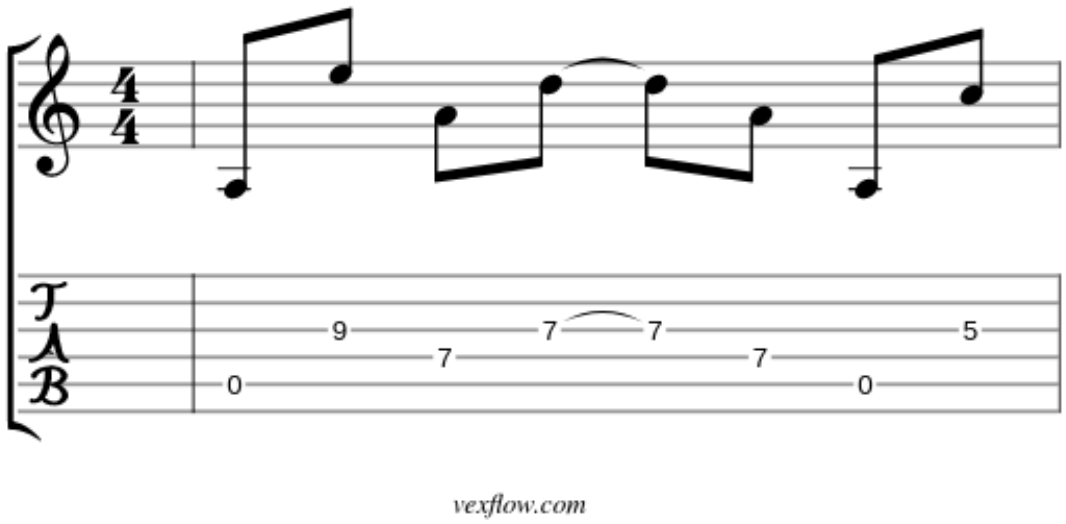


Fig. 1: Ex1: Rendering with defaults

```php
use PhpTabs\PhpTabs;

$song = new PhpTabs('mytab.gp4');

// Available options
$options = [
  // Renderer options
  'measures_per_stave'  => 1,

  // Global options
  'space'               => 16,        # An integer
  'scale'               => 0.8,       # A float or an integer
  'stave-distance'      => 20,        # An integer
  'width'               => 500,       # An integer, in pixel

  'font-size'           => 12,        # An integer
  'font-face'           => 'times',   # A string
  'font-style'          => 'italic',  # A string

  'tab-stems'           => true,      # A boolean, default: false
  'tab-stem-direction'  => 'down',    # A string up|down, default: up
  'player'              => false,     # A boolean, default: false

  // Tabstaves options
  'notation'            => true,       # A boolean, default: false
  'tablature'           => true,       # A boolean, default: true
];

// Rendering
echo $song->toVextab($options);
```

Will ouput something like:

```
options scale=0.8 space=16 width=500 tab-stems=true tab-stem-direction=down stave-
↪distance=20 font-size=12 font-face=times font-style=italic tempo=66

tabstave notation=true time=4/4

notes | :8 0/5 9/3 7/4 7/3 T7/3 7/4 0/5 5/3
```

That renders:

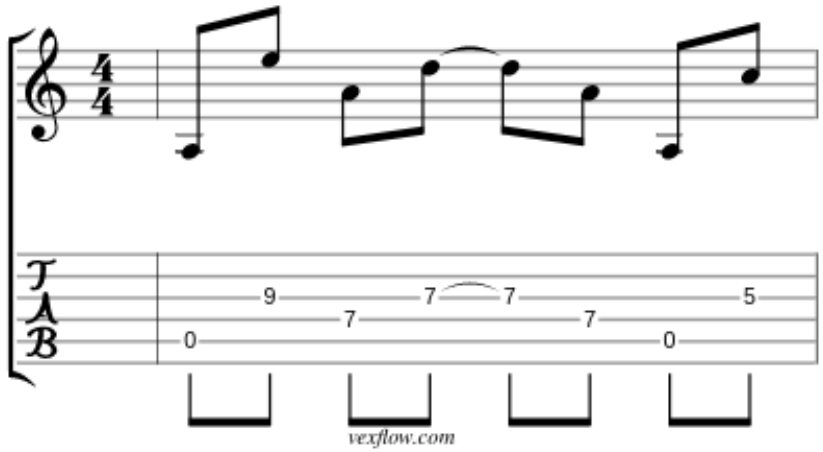


Fig. 2: Ex2: Rendering with custom options

Other options (tempo, clef, key, etc. . .) will be set by the tab object.

11.3 Supported VexTab features

11.3.1 Global features

All options rendered as `options ...`.

Feature	Example	Supported
tempo	tempo=192	OK
player	player=true	OK
tab-stems	tab-stems=true	OK
tab-stem-direction	tab-stem-direction=up	OK
width	width=1024	OK
scale	scale=0.8	OK
space	space=16	OK
stave-distance	stave-distance=16	OK
font-face	font-face=times	OK
font-style	font-style=italic	OK
font-size	font-size=12	OK

11.3.2 Stave features

All options rendered as `tabstave ...`.

Feature	Example	Supported
notation	notation=true	OK
tablature	tablature=true	OK
clef	clef=treble	OK
key	key=Ab	@todo
time	time=4/4	OK
tuning	tuning=eb	@todo

11.3.3 Measure and beat features

All options rendered as `notes ...`.

Bars

Feature	Notation	Supported
Bar		OK
Double Bar	||	@todo
Repeat Begin	=|:	OK
Repeat End	=:|	OK
Double Repeat	=::	@todo
End Bar	=|=	@todo

Beats and notes

Feature	Notation	Supported
Rest Beat	##	OK
Bend	b	OK
Dead Note	X	OK
Vibrato	v	OK
Harsh Vibrato	V	@todo
Hammer-on	h	OK
Pull-off	p	OK
Taps	t	OK
Slide	s	OK
Tied Note	T	OK
Upstroke	u	OK
Downstroke	d	OK
Chord Beat	(0/6.2/5.2/4)	OK
Tuplets	^n^	OK
Durations	w h q 8 16 32 64	OK
Annotations	$.$	@todo
Staccato	$a./bottom.$	@todo
Staccatissimo	$av/bottom.$	@todo
Accent	$a>/bottom.$	@todo
Tenuto	$a-/bottom.$	@todo
marcato	$a^/bottom.$	@todo
LH pizzicato	$a+/bottom.$	@todo
snap pizzicato	$ao/bottom.$	@todo
open note	$ah/bottom.$	@todo
up fermata	$a@a/bottom.$	@todo
down fermata	$a@u/bottom.$	@todo
bow up	$al/bottom.$	@todo
bow down	$am/bottom.$	@todo

Lyrics

Lyrics integration still has to be done.

11.3.4 Musical symbols

Feature	Notation	Supported
Trills	#tr	@todo
Codas	#coda	@todo
Segnos	#segno	@todo
Forte	#f	@todo

CHAPTER 12

Calculate measure and beat durations in seconds

PhpTabs provides some useful methods to calculate durations.

12.1 Measure duration

To calculate duration in seconds, the formula is `(Number of beats) * 60 / Tempo`.

The following code prints duration for the first measure of the first track.

```
use PhpTabs\PhpTabs;

// Instanciate a tablature
$song = new PhpTabs('myTab.gp5');

// Take a measure
$measure = $song->getTrack(0)->getMeasure(0);

// Calculate duration in seconds
$duration = 60
    * $measure->getTimeSignature()->getNumerator()
    / $measure->getTempo()->getValue();

// Print duration
echo sprintf("Duration=%ss", $duration);
```

With a 4/4 time signature and a tempo of 120, this example will ouput:

```
Duration=2s
```

12.2 Beat duration

Calculating beat duration in seconds is quite more complex. It depends on:

- tempo
- time signature
- non dotted, dotted or double dotted beat
- division type

PhpTabs (>=0.6.0) provides a shortcut method to easily get this value.

The `getTime()` method is provided by the `PhpTabs\Music\Voice` model.

The following code prints duration for the first beat of the first measure of the first track.

```
use PhpTabs\PhpTabs;

// Instanciate a tablature
$song = new PhpTabs('myTab.gp5');

// Take a beat
$beat = $song->getTrack(0)->getMeasure(0)->getBeat(0);

// Get duration in seconds for the first voice
$duration = $beat->getVoice(0)->getTime();

// Print duration
echo sprintf("Duration=%ss", $duration);
```

With a 4/4 time signature, a tempo of 120 and for a quarter beat, this example will ouput:

```
Duration=0.5s
```

12.3 Song duration

In order to calculate the duration of the complete song, we will add up the durations of each measure.

For one measure, the duration is calculated with the formula: `(Number of beats) * 60 / Tempo`.

```
use PhpTabs\PhpTabs;

// Instanciate a tablature
$song = new PhpTabs('myTab.gp5');

$total = 0;

// Take all measures for the first track
$measures = $song->getTrack(0)->getMeasures();

// Add up durations for each measure
foreach ($measures as $measure) {

    // Calculate duration in seconds
```

(continues on next page)

(continued from previous page)

```
    $total += 60
        * $measure->getTimeSignature()->getNumerator()
        / $measure->getTempo()->getValue();

}

// Print duration
echo sprintf("Duration=%ss\n", $total);
```

With a 4/4 time signature, a tempo of 120 and 10 measures, this example would ouput:

```
Duration=20s
```

CHAPTER 13

Create a song from scratch

13.1 Very minimalistic tablature

The shortest way to create a tablature is to instanciate a PhpTabs.

```
use PhpTabs\PhpTabs;

// Instanciate a tablature
$song = new PhpTabs();

// Read some information
echo $song->getName();
```

This is only sufficient to make basic operations but it failed if we want to save it as Guitar Pro or MIDI format.

13.2 A minimal and working tablature

We will create a tablature of one track with one measure.

In addition to the track, we have to define a channel, a measure header and a measure.

```
use PhpTabs\Music\Channel;
use PhpTabs\Music\Measure;
use PhpTabs\Music\MeasureHeader;
use PhpTabs\Music\TabString;
use PhpTabs\Music\Track;
use PhpTabs\PhpTabs;

// Instanciates a tablature
$tablature = new PhpTabs();

/* ------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
 | Create basic objects
 | ------------------------------------------------- */

// Create one track
$track = new Track();
$track->setName('My first track');

// Define 6 strings
foreach ([64, 59, 55, 50, 45, 40] as $index => $value) {
    $string = new TabString($index + 1, $value);
    $track->addString($string);
}

// One channel
$channel = new Channel();
$channel->setId(1);

// One measure header, will be shared
// by all first measures of all tracks
$mh = new MeasureHeader();
$mh->setNumber(1);

// One specific measure for the first track,
// with a MeasureHeader as only parameter
$measure = new Measure($mh);

/* -------------------------------------------------
 | Bound them together
 | ------------------------------------------------- */

// Attach channel to the song
$tablature->addChannel($channel);

// Attach measure header to the song
$tablature->addMeasureHeader($mh);

// Add measure to the track
$track->addMeasure($measure);

// Bound track and its channel configuration
$track->setChannelId($channel->getId());

// Finally, attach Track to the Tablature container
$tablature->addTrack($track);


// Now we can save, convert, export a fully functional song
$tablature->save('test.gp5');
```

Note that objects could have been instanciated in a different order. An approach would have been to create all measure headers first. Then to create measures for several tracks. Finally, we could have created the tracks and their channels in order to integrate everything.

13.3 A working tablature with several tracks and measures

We've seen how to create a basic tablature. It's time to build a more complex tablature.

Let's set our goals:

- One song called 'My song with notes'
- 2 tracks, one for a Piano and one for a Contrabass
- 2 measures per track and one note per measure

```
use PhpTabs\Music\Beat;
use PhpTabs\Music\Channel;
use PhpTabs\Music\Measure;
use PhpTabs\Music\MeasureHeader;
use PhpTabs\Music\Note;
use PhpTabs\Music\TabString;
use PhpTabs\Music\Track;
use PhpTabs\PhpTabs;

// Instanciate a tablature
$tablature = new PhpTabs();

// Set song name
$tablature->setName('My song with notes');

/* ------------------------------------------------
 | Create basic objects
 | ------------------------------------------------ */

// Create tracks
$piano_track = new Track();
$piano_track->setName('Piano track');

$contrabass_track = new Track();
$contrabass_track->setName('Contrabass track');


// Create channels
$channel0 = new Channel();
$channel0->setId(1);
$channel0->setProgram(0); // This program is for piano
$channel1 = new Channel();
$channel1->setId(2);
$channel1->setProgram(43); // This program is for contrabass

// One measure header for each measure
$mh0 = new MeasureHeader();
$mh0->setNumber(1);
$mh1 = new MeasureHeader();
$mh1->setNumber(2);

// 2 measures for the first track
$track0_measure0 = new Measure($mh0);
$track0_measure1 = new Measure($mh1);

// 2 measures for the second track
$track1_measure0 = new Measure($mh0);
```

(continues on next page)

(continued from previous page)

```php
$track1_measure1 = new Measure($mh1);

/* ------------------------------------------------
 | Add notes for each measure
 | ------------------------------------------------ */
foreach ([
    $track0_measure0,
    $track0_measure1,
    $track1_measure0,
    $track1_measure1
] as $measure) {
    // Create a Beat and a Note
    $beat = new Beat();
    $note = new Note();
    // Attach note to the beat
    $beat->getVoice(0)->addNote($note);
    // Make a random value for the note
    $note->setValue(rand(0, 5));
    // Attach beat to the measure
    $measure->addBeat($beat);
}


/* ------------------------------------------------
 | Bound headers, channels, measures and tracks
 | ------------------------------------------------ */

// Attach channels to the song
$tablature->addChannel($channel0);
$tablature->addChannel($channel1);

// Attach measure headers to the song
$tablature->addMeasureHeader($mh0);
$tablature->addMeasureHeader($mh1);

// Add measures to the first track
$piano_track->addMeasure($track0_measure0);
$piano_track->addMeasure($track0_measure1);
$piano_track->addString(new TabString(1, 64));

// Add measures to the second track
$contrabass_track->addMeasure($track1_measure0);
$contrabass_track->addMeasure($track1_measure1);
$contrabass_track->addString(new TabString(1, 64));

// Bound tracks and their channel configurations
$piano_track->setChannelId($channel0->getId());
$contrabass_track->setChannelId($channel1->getId());

// Finally, attach Tracks to the Tablature container
$tablature->addTrack($piano_track);
$tablature->addTrack($contrabass_track);

/* ------------------------------------------------
 | Now that we have a functionnal song, we can work
 | with it
 | ------------------------------------------------ */
```

(continues on next page)

(continued from previous page)

```
// Render the first track as a vextab string
echo $tablature->toVextab();

// Render the second track as an ASCII string
echo $tablature->onlyTrack(1)->toAscii();

// Save it as a Guitar Pro 5 file
$tablature->save('song-2-tracks-2-measures.gp5');
```

Some important things to keep in mind:

- Measure headers are defined globally (attached to the Song)
- Measures are defined per track
- you MUST have the same number of measures for each track
- This number of measures MUST be equal to the number of measure headers
- For all tracks, a measure number 1 MUST be bound to the measure header number 1, and so on for all measures
- To understand how elements are built on each other and how to be on the right scope to interact with them, refer to the *Music stack tree* and to the *getting/setting/counting rules*.

CHAPTER 14

Performance & caching

There are some cases when it's useful to improve performances.

It largely depends on the context but it may be good to know that PhpTabs provides some tools for this purpose.

It is often possible to summarize performance issues in 2 types:

- IO struggling
- Software issues

To fix IO issues, we'll try to put in cache (memory) some data.

But, first, let's look at what we will cache.

14.1 Context

For this example, we'll take a real-life Guitar Pro file which contains:

- 6 tracks
- 849 measures (Oh!)
- A little bit less than 1MB

Let's parse it !

```
$filename = 'big-file.gp5';

// Start
$start = microtime(true);

// Parse
$song = new PhpTabs($filename);

// Stop
$stop = microtime(true);
```

(continues on next page)

(continued from previous page)

```
// Display parsing time
echo round($stop - $start, 2) . 's';
```

And the result is:

```
5.78s
```

Woh! I don't know what the subject of your app is but we can tell it's going to be slow.

As usual for performance issues, you have to make choices.

We have `6 tracks * 849 measures = 5094 measures`. Do you want to display all of these in a webpage ? In a mobile app ?

Let's say that we only want to display one track.

PhpTabs provides features *to target a single track* and to generate a new file.

14.2 Slicing tracks

In this example, starting from the whole file, we'll create 6 files with only one track in each.

```
$filename = 'big-file.gp5';

// Parse
$song = new PhpTabs($filename);

// Generate one file per track
for ($i = 0; $i < $song->countTracks(); $i++) {
    $song->onlyTrack($i)->save("track-{$i}-{$filename}");
}
```

Now, we're going to test parsing for one of these files.

```
$filename = 'track-0-big-file.gp5';

// Start
$start = microtime(true);

// Parse
$song = new PhpTabs($filename);

// Stop
$stop = microtime(true);

// Display parsing time
echo "\nParsing a track file: " . round($stop - $start, 2) . 's';
echo "\n" . $song->getName();
```

```
Parsing a track file: 0.52s
My song title
```

Ok, that's better. At the end of this script, you may have seen that we've printed out the song title. Indeed, *slicing* or *targetting* a track does not loose global song informations.

14.3 Exporting to JSON

Is it possible to make it faster ?

We're going to make the same thing than before but instead of saving the track into in a Guitar Pro file, we're going to save it in JSON.

```
$filename = 'big-file.gp5';

// Parse
$song = new PhpTabs($filename);

// Generate one JSON file per track
for ($i = 0; $i < $song->countTracks(); $i++) {
    $song->onlyTrack($i)->save("track-{$i}-{$filename}.json");
}
```

Now, we're going to test parsing for one of these files.

```
$filename = 'track-0-big-file.gp5.json';

// Start
$start = microtime(true);

// Parse
$song = new PhpTabs($filename);

// Stop
$stop = microtime(true);

// Display parsing time
echo "\nParsing a JSON file: " . round($stop - $start, 2) . 's';
echo "\n" . $song->getName();
```

```
Parsing a JSON file: 0.21s
My song title
```

It's good for the moment.

JSON file is bigger than Guitar Pro file. Under the hood, it makes a `PhpTabs::toArray()` call, then it converts it to JSON.

As data is stored in a native Phptabs export, it makes it faster.

The idea here was to parse the whole song only once and split it into several files with sliced tracks.

The new problem is that we have 6 files for tracks.

What about pushing `toArray()` results into a cache system ?

14.4 Caching

We're going to take all the work done before in order to keep only the best parts.

Best parts are:

- Parsing only once the whole song
- Splitting a song into smaller units (tracks) for later use

What we're introducing here is:

- Exporting tracks to arrays
- Saving them into cache
- Importing an array into PhpTabs

Importing from an array is blazingly fast. There is no parsing time, it's like re-importing a part already analyzed previously.

You may have to install Memcache server and client before. Of course, you may use another caching system.

```
use PhpTabs\IOFactory;

$memcache = new Memcache;
$memcache->connect('localhost', 11211)
        or die ("Connection failed");

$filename = 'track-0-big-file.gp5';

// Parse
$song = IOFactory::create($filename);

// Generate one array for this track
$array = $song->toArray();

// Put in cache
$memcache->set($filename, $array);
```

And now, we may load this track from cache.

```
use PhpTabs\IOFactory;

$memcache = new Memcache;
$memcache->connect('localhost', 11211)
        or die ("Connection failed");

$filename = 'track-0-big-file.gp5';

// Start
$start = microtime(true);

// Get from cache
$song = IOFactory::fromArray(
    $memcache->get($filename)
);

$stop = microtime(true);

// Display loading time
echo "\nLoading time : " . round($stop - $start, 2) . 's';
```

```
Loading time : 0.13s
```

It's a quick example on how to tackle some performance issues. You may not use these scripts without adapting them to your own context.

However, with that in mind, you have an idea of how to successfully meet production constraints.

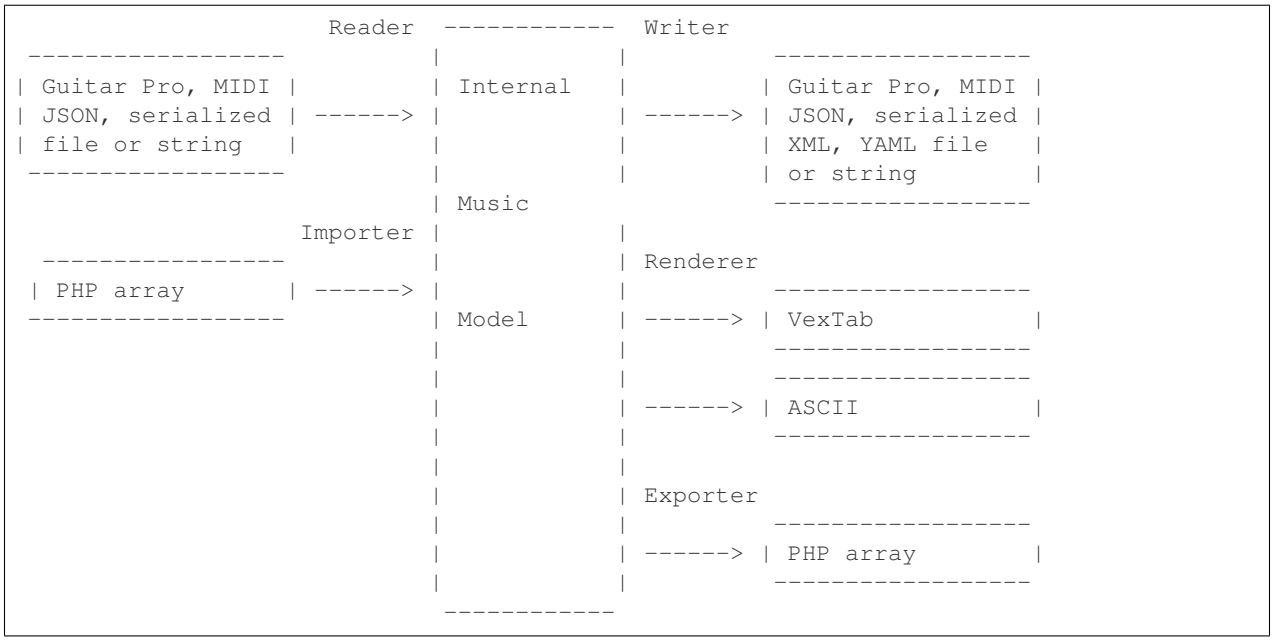
If you have any questions or some feedbacks, feel free to open issues or contribute to this manual.

CHAPTER 15

Architecture

In the scheme below, you may see how PhpTabs core is structured.

It may be useful for all PhpTabs users. Nevertheless, this is more for those who plan to contribute.



15.1 Component roles

- **Reader** imports data from *files* and *strings* into the internal model
- **Writer** exports data to *files* or *strings*
- **Renderer** exports data to a *human-readable representation*,

- **Exporter** exports data as PHP arrays for caching or various usages
- **Importer** imports data from internal exports (array)

With the *internal music model*, you can easily convert files from one type to another.

CHAPTER 16

IOFactory

IOFactory is done in order to:

- create empty PhpTabs,
- load PhpTabs from files, strings and arrays

```php
use PhpTabs\IOFactory;

$song = IOFactory::create();

$song = IOFactory::fromArray($array);

$song = IOFactory::fromFile($filename);

$song = IOFactory::fromString($string, $format);

$song = IOFactory::fromJsonFile($jsonFile);

$song = IOFactory::fromSerializedFile($phpSerializedFilename);

$song = IOFactory::fromJson($jsonString);

$song = IOFactory::fromSerialized($phpSerializedString);
```

All these methods return a `PhpTabs` instance.

16.1 create()

This method returns an empty `PhpTabs` instance.

16.1.1 Type

`PhpTabs\PhpTabs`

16.1.2 Example

```php
use PhpTabs\IOFactory;

// Equivalent to 'new PhpTabs()'
$tab = IOFactory::create();

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=0"
```

16.2 fromArray($data)

This method returns a `PhpTabs` resource, loaded from a PHP array.

16.2.1 Parameters

array `$data` An array previously exported with `$phptabs->toArray()`

16.2.2 Type

`PhpTabs\PhpTabs`

16.2.3 Example

```php
use PhpTabs\IOFactory;

// Create an empty tabs
$tab = IOFactory::create();

// Export as an array
$data = $tab->export();

// Now you can reimport as an array
$tab = IOFactory::fromArray($data);

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=0"
```

16.3 fromFile($filename, $type)

This method returns a `PhpTabs` instance, loaded from a file.

16.3.1 Parameters

string `$filename` **string** `$type` *Optional*

16.3.2 Type

`PhpTabs\PhpTabs`

16.3.3 Example

```
use PhpTabs\IOFactory;

// Create a PhpTabs instance
$tab = IOFactory::fromFile('mytabs.gp4');

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=2"
```

In case you need to force a parser type, use the second parameter.

```
use PhpTabs\IOFactory;

// Create a PhpTabs instance from a JSON file
$tab = IOFactory::fromFile('mytabs.dat', 'json');

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=2"
```

16.4 fromString($content, $type)

This method returns a `PhpTabs` instance, loaded from a string (binary or not).

16.4.1 Parameters

string `$content` **string** `$type`

16.4.2 Type

`PhpTabs\PhpTabs`

16.4.3 Example

```
use PhpTabs\IOFactory;

$content = file_get_contents('mytabs.gp4');

// Create a PhpTabs instance
$song = IOFactory::fromString($content, 'gp4');

// Work with the song
echo $song->getName();
```

In case you need to force a parser type, use the second parameter.

16.5 fromJsonFile($filename)

This method returns a `PhpTabs` resource, loaded from a JSON file.

16.5.1 Parameters

string `$filename`

16.5.2 Type

`PhpTabs\PhpTabs`

16.5.3 Example

```
use PhpTabs\IOFactory;

// Create a PhpTabs instance
$tab = IOFactory::fromJsonFile('mytabs.json');

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=2"
```

16.6 fromSerializedFile($filename)

This method returns a `PhpTabs` resource, loaded from a PHP serialized file.

16.6.1 Parameters

string `$filename`

16.6.2 Type

`PhpTabs\PhpTabs`

16.6.3 Example

```
use PhpTabs\IOFactory;

// Create a PhpTabs instance
$tab = IOFactory::fromSerializedFile('mytabs.ser');

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=2"
```

16.7 fromJson($string)

This method returns a `PhpTabs` instance loaded from a JSON string.

16.7.1 Parameters

string `string`

16.7.2 Type

`PhpTabs\PhpTabs`

16.7.3 Example

```
use PhpTabs\IOFactory;

// Create a PhpTabs instance
$tab = IOFactory::fromJson('{"song":{"name":null,"artist":null,"album":null,"author
↪":null,"copyright":null,"writer":null,"comments":null,"channels":[],"measureHeaders
↪":[],"tracks":[]}}');

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=0"
```

16.8 fromSerialized($string)

This method returns a `PhpTabs` instance, loaded from a PHP serialized string.

16.8.1 Parameters

string `string`

16.8.2 Type

`PhpTabs\PhpTabs`

16.8.3 Example

```php
use PhpTabs\IOFactory;

// Create a PhpTabs instance
$tab = IOFactory::fromSerialized('a:1:{s:4:"song";a:10:{s:4:"name";N;s:6:"artist";N;
↪s:5:"album";N;s:6:"author";N;s:9:"copyright";N;s:6:"writer";N;s:8:"comments";N;s:8:
↪"channels";a:0:{}s:14:"measureHeaders";a:0:{}s:6:"tracks";a:0:{}}}');

// Print track number
echo "Track count=" . $tab->countTracks();

// Should return "Track count=0"
```

CHAPTER 17

PhpTabs

PhpTabs provides some methods to access metadata, attributes and nodes.

17.1 Read song informations

You may read metadata with the following methods. They all return string or null.

```
use PhpTabs\PhpTabs;

$song = new PhpTabs('my-song.gp5');

// Display all metas
echo sprintf("
Title: %s
Album: %s
Artist: %s
Author: %s
Writer: %s
Date: %s
Copyright: %s
Transcriber: %s
Comments: %s",

    $song->getName(),
    $song->getAlbum(),
    $song->getArtist(),
    $song->getAuthor(),
    $song->getWriter(),
    $song->getDate(),
    $song->getCopyright(),
    $song->getTranscriber(),
    $song->getComments(),
);
```

It will ouput something like:

```
Title: Song title
Album: My album
Artist: Me and my band
Author: Me and my band too
Writer: A writer
Date: A long time ago
Copyright: So cheap
Transcriber:
Comments: Some multiline comments
```

17.2 Write song informations

For each getter method, a setter one is available.

```
$song->setName('New song title');
$song->setAlbum('Song album');
$song->setArtist('Song artist');
$song->setAuthor('Song author');
$song->setWriter('Song writer');
$song->setDate('Song date');
$song->setCopyright('Song copyright');
$song->setTranscriber('Song transcriber');
$song->setComments('Song comments');
```

17.3 Channels

You may handle channels.

```
// Number of channels
$count = $song->countChannels();

// Get an array of channels
$channels = $song->getChannels();

// Get a single channel by its index
// starting from 0 to n-1
$channel = $song->getChannel(0);

// Get a single channel by its id (integer)
$channel = $song->getChannelById(1);

// Remove a channel
$song->removeChannel($channel);

// Add a channel
$song->addChannel($channel);
```

17.4 Measure headers

You may handle measure headers.

```
// Number of measure headers
$count = $song->countMeasureHeaders();

// Get an array of measure headers
$measureHeaders = $song->getMeasureHeaders();

// Get a single measure header by its index
// starting from 0 to n-1
$measureHeader = $song->getMeasureHeader(0);

// Remove a measure header
$song->removeMeasureHeader($measureHeader);

// Add a measure header
$song->addMeasureHeader($measureHeader);
```

17.5 Tracks

You may handle tracks.

```
// Number of tracks
$count = $song->countTracks();

// Get an array of tracks
$tracks = $song->getTracks();

// Get a single track by its index
// starting from 0 to n-1
$track = $song->getTrack(0);

// Remove a track
$song->removeTrack($track);

// Add a track
$song->addTrack($track);
```

CHAPTER 18

Music model

PhpTabs builds a musical tree which is called the **Music-Model** (MM).

18.1 Tree

Song (= A *PhpTabs* instance)

- []Channel
  - []ChannelParameter

  [... channels]
- []MeasureHeader
  - Song [parent]
  - Tempo
  - TimeSignature
    - Duration
      - DivisionType

  [... measureHeaders]
- []Track
  - Song [parent]
  - Color
  - Lyric
  - []Measure
    - Track [parent]
    - Marker

        * MeasureHeader
        * []Beat
            - Measure [parent]
            - Stroke
            - Chord
            - Text
            - []Voice
            - Beat [parent]
            - Duration
            - []Note
            - Voice [parent]
            - NoteEffect
            - EffectBend
            - EffectGrace
            - EffectHarmonic
            - EffectTremoloBar
            - EffectTremoloPicking
            - EffectTrill

              [... notes]

          [... voices]

      [... beats]

  [... measures]

    – []TabString

[... tracks]

18.2 Traversing the tree is made simple

In this example, we read the fret value and string number, for the first note of the first track.

```
$song = new PhpTabs('mytab.gp4');

// We read a note
$note = $song
  ->getTrack(0)      # Track 0
  ->getMeasure(0)    # Measure 0
  ->getBeat(0)       # Beat 0
  ->getVoice(0)      # Voice 0
  ->getNote(0);      # Note 0
```

(continues on next page)

(continued from previous page)

```
// Print fret and string numbers
echo sprintf(
  "Note: %s/%d",
  $note->getValue(),
  $note->getString()
);
```

It will ouput something like:

```
Note: 13/2
```

Below, we make the same thing, for all tracks.

```
$tab = new PhpTabs('mytab.gp4');

foreach ($tab->getTracks() as $track) {

  // We read a note
  $note = $track
    ->getMeasure(0)    # Measure 0
    ->getBeat(0)       # Beat 0
    ->getVoice(0)      # Voice 0
    ->getNote(0);      # Note 0

  // Print track, fret and string numbers
  echo sprintf(
    "\nTrack %d - Note: %s/%d ",
    $track->getNumber(),
    $note->getValue(),
    $note->getString()
  );
}
```

It will ouput something like:

```
Track 1 - Note: 13/2
Track 2 - Note: 5/2
```

Now, we read all the beats for the first measure of all tracks.

```
$tab = new PhpTabs('mytab.gp4');

foreach ($tab->getTracks() as $track) {

  foreach ($track->getMeasure(0)->getBeats() as $idxBeat => $beat) {

    // We read a note
    $note = $beat
      ->getVoice(0)      # Voice 0
      ->getNote(0);      # Note 0

    // Print Track, Beat, fret and string numbers
    echo sprintf(
      "\nTrack %d - Beat %d - Note: %s/%d ",
      $track->getNumber(),
```

(continues on next page)

(continued from previous page)

```
      $idxBeat,
      null !== $note ? $note->getValue() : '-',
      null !== $note ? $note->getString(): '-'
    );
  }
}
```

Outputs:

```
Track 1 - Beat 0 - Note: -/0
Track 1 - Beat 1 - Note: -/0
Track 1 - Beat 2 - Note: 11/3
Track 1 - Beat 3 - Note: 0/2
Track 2 - Beat 0 - Note: 5/2
Track 2 - Beat 1 - Note: 5/2
Track 2 - Beat 2 - Note: 5/2
Track 2 - Beat 3 - Note: 5/2
Track 2 - Beat 4 - Note: 5/2
Track 2 - Beat 5 - Note: 5/2
```

Note the first two beats, they must be rest beats.

A short but useful view of the MOM is :

- Song
  - Track
    - Measure
      - Beat
      - Voice
      - Note

You can traverse it this way:

```
$tab
  ->getTrack(0)
  ->getMeasure(0)
  ->getBeat(0)
  ->getVoice(0)
  ->getNote(0);
```

18.3 Traversing the first level

A Song object contains:

- meta data (Name, artist, etc. . .)
- channels
- measure headers
- tracks

Channel, MeasureHeader and Track can be accessed with following methods:

18.4 Traversing Channels

`getChannels()`, `getChannel()` and `getChannelById()` methods

In this example, we print the channel names.

```
// Working with all channels
foreach ($song->getChannels as $channel) {
  echo $channel->getName() . PHP_EOL;
}

// Accessing by index
echo $song->getChannel(0)->getName() . PHP_EOL;
// Outputs something like "Clean Guitar 1"

// Accessing by id
echo $song->getChannelById(1)->getName() . PHP_EOL;
// Outputs something like "Clean Guitar 1"
```

18.5 Traversing MeasureHeaders

`getMeasureHeaders()` and `getMeasureHeader()` methods

In this example, we print the tempo for each measure.

```
// Working with all measure headers
foreach ($song->getMeasureHeaders() as $header) {
  echo $header->getTempo()->getValue() . PHP_EOL;
}

// Accessing by index to the first header
echo $song->getMeasureHeader(0)->getTempo()->getValue() . PHP_EOL;
// Outputs something like "90"
```

18.6 Traversing Tracks

`getTracks()` and `getTrack()` methods

In this example, we print the number of measures by track.

```
// Working with all tracks
foreach ($song->getTracks() as $track) {
  echo $track->countMeasures() . PHP_EOL;
}
```

(continues on next page)

(continued from previous page)

```
// Accessing by index to the first track
echo $song->getTrack(0)->countMeasures() . PHP_EOL;
// Outputs something like "4" (small tab!)
```