

audiotags Manual

Tianyi Shi

2020-10-29

Contents

Preface	5
1 Start Simple	7
2 Conversion and Downcast	9
2.1 Converting from a <code>Box<dyn AudioTag></code> to Another	9
2.2 Converting into a Concrete Type (Downcasting)	10
2.3 Upcasting	11
3 AnyTag	13

Preface

Thank you for considering **audiotags**!

Examples in this manual

If you want to run the examples in this book:

1. clone the repo and navigate into it
2. create `src/main.rs`
3. all examples, unless otherwise specified, can be copied verbatim from this book to `src/main.rs` and run with `cargo run` (if you're reading it online, the copy button wil show if you hover over a code block)

Chapter 1

Start Simple

The following example shows how you can read an audio file, parse, set, and save its metadata:

```
fn main() {  
    // using `default()` or `new()` alone so that the metadata format is  
    // guessed (from the file extension) (in this case, Id3v2 tag is read)  
    let mut tag = Tag::new().read_from_path(MP3_FILE).unwrap();  
    // You can also specify the metadata format (tag type):  
    let _tag = Tag::new()  
        .with_tag_type(TagType::Id3v2)  
        .read_from_path(MP3_FILE)  
        .expect("Fail to read!");  
  
    tag.set_title("foo title");  
    assert_eq!(tag.title(), Some("foo title"));  
    tag.remove_title();  
    assert!(tag.title().is_none());  
    tag.remove_title();  
    // trying to remove a field that's already empty won't hurt  
  
    let cover = Picture {  
        mime_type: MimeType::Jpeg,  
        data: &vec![0u8; 10],  
    };  
  
    tag.set_album_cover(cover.clone());  
    assert_eq!(tag.album_cover(), Some(cover));  
    tag.remove_album_cover();  
    assert!(tag.album_cover().is_none());  
}
```

```

tag.remove_album_cover();

tag.write_to_path(MP3_FILE).expect("Fail to save");
// TASK: reload the file and prove the data have been saved
}

```

Note that Tag always reads into a `Box<dyn AudioTag>`. If you do not want a trait object, you can use the underlying concrete types. However, you'll also need to manually bring the traits into scope if you prefer not to write `audiotags::*`.

```

use audiotags::{traits::*, FlacTag, Id3v2Tag, Mp4Tag};
// or alternatively `use audiotags::*`

fn main() {
    let mut tag = FlacTag::read_from_path("assets/a.flac").unwrap();
    tag.set_title("foo");
    assert_eq!(tag.title(), Some("foo"));
    let mut tag = Mp4Tag::read_from_path("assets/a.m4a").unwrap();
    tag.set_title("foo");
    assert_eq!(tag.title(), Some("foo"));
    let mut tag = Id3v2Tag::read_from_path("assets/a.mp3").unwrap();
    tag.set_title("foo");
    assert_eq!(tag.title(), Some("foo"));
    // all other methods in trait `AudioTagEdit` are available, not just title
}

```


Chapter 2

Conversion and Downcast

The following example shows how you can read the tag in an mp3 file, convert it into an mp4 tag, and write it to an m4a file.

2.1 Converting from a Box<dyn AudioTag to Another

```
use audiotags::{Config, Tag, TagType};

fn main() {
    // we have an mp3 and an m4a file
    const MP3_FILE: &'static str = "assets/a.mp3";
    const M4A_FILE: &'static str = "assets/a.m4a";
    // read tag from the mp3 file. Using `default()` so that the
    // type of tag is guessed from the file extension
    let mut mp3tag = Tag::default().read_from_path(MP3_FILE).unwrap();
    // set the title
    mp3tag.set_title("title from mp3 file");
    // we can convert it to an mp4 tag and save it to an m4a file.
    let mut mp4tag = mp3tag.to_dyn_tag(TagType::Mp4);
    mp4tag.write_to_path(M4A_FILE).unwrap();

    // reload the tag from the m4a file; this time specifying the
    // tag type (you can also use `default()`)
    let mut mp4tag = Tag::new()
        .with_tag_type(TagType::Mp4)
        .read_from_path(M4A_FILE)
```

```

        .unwrap();
// the tag originated from an mp3 file is successfully written
// to an m4a file!
        assert_eq!(mp4tag.title(), Some("title from mp3 file"));
// multiple artists
        mp4tag.add_artist("artist1 of mp4");
        mp4tag.add_artist("artist2 of mp4");
        assert_eq!(
            mp4tag.artists(),
            Some(vec!["artist1 of mp4", "artist2 of mp4"])
        );
// convert to id3 tag, which does not support multiple artists
        mp4tag.set_config(Config::default().sep_artist("/"));
// separator is by default `;` but we can customise it
        let mp3tag = mp4tag.to_dyn_tag(TagType::Id3v2);
        assert_eq!(mp3tag.artist(), Some("artist1 of mp4/artist2 of mp4"));
    }

```

2.2 Converting into a Concrete Type (Downcasting)

Can I convert into a concrete type?

Yes, you can directly convert `.into()` it (this is technically known as a “downcast”):

```

use audiotags::{FlacTag, Tag};

fn main() {
    let id3v2tag = Tag::default().read_from_path("assets/a.mp3").unwrap();
    let _flactag: FlacTag = id3v2tag.into();
    // of course, you can `let id3v2tag_concrete: Id3v2Tag = id3v2tag.into();`
}

```

You can even convert `.into()` the ‘backend’ tag type:

```

use audiotags::Tag;

fn main() {
    let mp3tag = Tag::default().read_from_path("assets/a.mp3").unwrap();
    let flactag: metaflac::Tag = mp3tag.into(); // into the 'backend' tag
    // then you can use methods specific to metaflac
}

```

```
let _ = flactag.get_streaminfo();  
}
```

This is useful when you really need to use the methods not provided by `audiotags::traits::*`.

You can also downcast the concrete `audiotags::FlacTag` to `metaflac::Tag` and so on.

2.3 Upcasting

Since you're allowed to downcast, naturally you can also upcast:

```
use audiotags::*;  
  
fn main() {  
    let mut innertag = metaflac::Tag::default();  
    innertag  
        .vorbis_comments_mut()  
        .set_title(vec!["title from metaflac::Tag"]);  
    let tag: FlacTag = innertag.into();  
    let _id3tag = tag.to_dyn_tag(TagType::Id3v2);  
    // in this case the "title" metadata will be  
    // losslessly written into the id3tag.  
    // However, if you have FLAC-specific fields,  
    // they will be lost upon conversion  
}
```


Chapter 3

AnyTag

The following example shows how you can create a “generic” `AnyTag` and convert it into a specific tag type.

```
use audiotags::{AnyTag, AudioTagEdit, Id3v2Tag};

fn main() {
    let mut tag = AnyTag::default();
    tag.set_title("foo");
    tag.set_year(2001);
    let tag: Id3v2Tag = tag.into();
    assert_eq!(tag.year(), Some(2001));
    tag.write_to_path("assets/a.mp3").unwrap();
}
```