
















Reference

CONTENTS:

- [Arrangements](#)
 -  [Arrangement](#)
 -  [Playlist](#)
 - [Track](#)
-  [Channel Rack](#)
 -  [Automation](#)
 - [Channel](#)
 - [DisplayGroup](#)
 - [Instrument](#)
 -  [Layer](#)
 -  [Sampler](#)
 - [Shared](#)
-  [Mixer](#)
 -  [Insert](#)
 -  [Slot](#)
-  [Patterns](#)
 - [Pattern](#)
-  [Plugins](#)
 - [Effects](#)
 - [Generators](#)
 - [VST](#)
-  [Controllers](#)
-  [Events](#)
-  [Exceptions](#)
-  [Project](#)
-  [Timemarkers](#)

API

PyFLP provides a low-level events-based API and a high-level API. Generally, you should only need the high level API though.

`pyflp.parse(file: Path | str) → Project`

[\[source\]](#)

Parses an FL Studio project file and returns a parsed `Project`.

PARAMETERS:

file – Path to the FLP.

RAISES:

- [HeaderCorrupted](#) – When an invalid value is found in the file header.
- [VersionNotDetected](#) – A correct string type couldn't be determined.

`pyflp.save(project: Project, file: Path | str) → None`

[\[source\]](#)

Save a parsed project back into a file.

Caution

Always have a backup ready, just in case 😊

PARAMETERS:

- **project** – The object returned by `parse()`.
- **file** – The file in which the contents of `project` are serialised back.