# Bethesda Structs Documentation

***Release 0.1.4***

**Stephen Bunn**

**Dec 23, 2020**

# Contents

*A wrapper for some of Bethesda's more popular archive and plugin file formats.*

In my pursuit to create a better method of mod distribution, the issue of processing and accurately representing mods is a major annoyance. Being able to analyze and potentially discover relationships between Bethesda based mods is something that better mod distribution methods desperately need.

Because of the lack of any **simple** and **lightweight** libraries built to address this issue, I decided to take a stab at handling it myself. *Please* let me know if you find any issues with my parsing logic as both the documentation and my knowledge on the file formats is very much thrown together.

To get started using this package, please see the *Getting Started* page!

CHAPTER 1

# Getting Started

**Welcome to Bethesda-Structs!**
This page should hopefully provide you with enough information to get you started interacting with some of Bethesda's file formats.

## 1.1 Installation and Setup

Installing the package should be super duper simple as we utilize Python's setuptools.

```
$ pipenv install bethesda-structs
$ # or if you're old school...
$ pip install bethesda-structs
```

Or you can build and install the package from the git repo.

```
$ git clone https://github.com/stephen-bunn/bethesda-structs.git
$ cd ./bethesda-structs
$ python setup.py install
```

## 1.2 Example Usage

All of the supported file type classes are subclasses of *BaseFiletype*.
These include *BaseArchive* and *BasePlugin* and any other one-off file types.

Because this package is just a parser around some of Bethesda's popular file formats, its hopefully extendable to different purposes you might have. Below are a few quick examples of how you might use this package, **but** remember to keep the *autodocs* handy!

### 1.2.1 Checking if a file can be parsed

Because many of the structures used to parse these filetypes are bulit using construct, the parser **silently** fails if parsing raises an error. For this reason, all file types include the classmethod *can_handle()*, which takes a filepath and returns either `True` or `False` depending on if the filetype class believes it can parse the file.

This method can be executed similar to the following:

```
>>> from bethesda_structs.archive import BSAArchive
>>> BSAArchive.can_handle('/media/sf_VMShared/bsa/Campfire.bsa')
True
>>> from bethesda_structs.plugin import FNVPlugin
>>> FNVPlugin.can_handle('/media/sf_VMShared/bsa/Campfire.bsa')
False
```

### 1.2.2 Parsing a file

We try to keep the api simple and consistent across all of our parsable filetypes. For example, the following methods should be available on all parseable filetypes:

- *parse()* — *parses bytes*
- *parse_stream()* — *parses bytes from a file stream*
- *parse_file()* — *parses bytes from a file path*

This makes parsing of a given file super easy.
For example:

```
>>> from bethesda_structs.archive import BSAArchive
>>> archive = BSAArchive.parse_file('/media/sf_VMShared/bsa/Campfire.bsa')
>>> archive
BSAArchive(filepath=PosixPath('/media/sf_VMShared/bsa/Campfire.bsa'))
>>> archive.filepath
PosixPath('/media/sf_VMShared/bsa/Campfire.bsa')
```

The `filepath` attribute is automatically transformed into a `pathlib.Path` instance on initialization of the filetype. This is really helpful for quickly obtaining file information if you need it.

```
>>> archive.filepath.stat()
os.stat_result(st_mode=33272, st_ino=6, st_dev=46, st_nlink=1, st_uid=0, st_gid=999,␣
↪st_size=25097317, st_atime=1522860401, st_mtime=1522475016, st_ctime=1522475016)
```

**Note:** Parsed files *only* have the `filepath` attribute populated if either the *parse_file()* method was used, or the `filepath` named argument was passed into the other parsing methods.

### 1.2.3 Examining a parsed file

All parsed files should contain the `container` attribute which is a root `Container` instance. This contains the required information for examining a parsed file's contents.

For example you can view the header *sub*-container of a *BSAArchive* like the following:

```
>>> from bethesda_structs.archive import BSAArchive
>>> archive = BSAArchive.parse_file('/media/sf_VMShared/bsa/Campfire.bsa')
>>> print(archive.container.header)
Container:
    magic = b'BSA\x00' (total 4)
    version = 105
    directory_offset = 36
    archive_flags = Container:
        directories_named = True
        files_named = True
    directory_count = 4
    file_count = 493
    directory_names_length = 50
    file_names_length = 14839
    file_flags = Container:
```

Every different subclass of *BaseFiletype* most likely implements a different `container` structure.
For example, every *BSAArchive* has `directory_records`, `directory_blocks`, and `file_names` sub-containers:

```
>>> from bethesda_structs.archive import BSAArchive
>>> archive = BSAArchive.parse_file('/media/sf_VMShared/bsa/Campfire.bsa')
>>> print(archive.container.directory_records)
ListContainer:
    Container:
        hash = 1948419268744733541
        file_count = 155
        name_offset = 14971
    Container:
        hash = 2736503539341685349
        file_count = 174
        name_offset = 17467
    Container:
        hash = 3940292978845119603
        file_count = 163
        name_offset = 20268
    Container:
        hash = 11606842737777340531
        file_count = 1
        name_offset = 22885
>>> print(archive.container.directory_blocks[3])  # contains 1 file
Container:
    name = u'meshes\\mps\x00' (total 11)
    file_records = ListContainer:
        Container:
            hash = 161837549572200789631
            size = 2384
            offset = 25094933
>>> print(archive.container.file_names[:5])
['_camp_objectplacementindicatorthread01.psc', '_camp_
↪objectplacementindicatorthread02.psc', '_camp_objectplacementindicatorthread03.psc',
↪ '_camp_tentsitlayscript.psc', 'campcampfire.psc']
```

However, *BTDXArchive* has a completely different structure just using a `files` attribute.

```
>>> from bethesda_structs.archive import BTDXArchive
>>> archive = BTDXArchive.parse_file('/media/sf_VMShared/ba2/Immersive HUD - Main.bsa
↪')
>>> print(archive.container.header)
Container:
    magic = b'BTDX' (total 4)
    version = 1
    type = u'GNRL' (total 4)
    file_count = 16
    names_offset = 39979
>>> print(archive.container.files[0])
Container:
    hash = 2246534376
    ext = u'swf' (total 3)
    directory_hash = 3539859571
    offset = 600
    packed_size = 0
    unpacked_size = 5011
```

**So knowledge of what you are parsing is important, but how it's being done shouldn't be.**

### 1.2.4 Extracting an archive

All subclasses of *BaseArchive* contain a method *extract()*. This method makes it incredibly simple to extract the content of a parsed archive into a directory.

```
>>> from bethesda_structs.archive import BSAArchive
>>> archive = BSAArchive.parse_file('/media/sf_VMShared/bsa/Campfire.bsa')
>>> archive.extract('/home/stephen-bunn/Downloads/campfire-extracted/')
>>> from pathlib import Path
>>> for path in pathlib.Path('/home/stephen-bunn/Downloads/campfire-extracted/').glob(
↪'**/*'):
...     print(path.as_posix())
/home/stephen-bunn/Downloads/campfire-extracted/scripts/_camp_indicatortrigger.pexl
/home/stephen-bunn/Downloads/campfire-extracted/scripts/_camp_objectplacementthread30.
↪pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/campconjuredshelter.pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/_camp_
↪objectplacementindicatorthread02.pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/_camp_instinctseffects.pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/_camp_legacymenu.pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/_camp_
↪objectplacementindicatorthread.pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/bladessparringscript.pex
/home/stephen-bunn/Downloads/campfire-extracted/scripts/tentsystem.pex
... <only first 9 files> ...
```

Both variants of *BTDXArchive* can also be extracted (`GNRL` and `DX10`).

#### Getting extraction progress

The *extract()* method takes a named argument `progress_hook` that acts as a (pre/post) callback function taking 3 positional arguments:

- `current` — *current extracted bytes*

- `total` — *total bytes to extract*
- `filepath` — *current filepath being extracted*

```
>>> from bethesda_structs.archive import BSAArchive
>>> archive = BSAArchive.parse_file('/media/sf_VMShared/bsa/Campfire.bsa')
>>> def progress_hook(current, total, filepath):
...     print(((current / total) * 100.0, filepath))
>>> archive.extract('/home/stephen-bunn/Downloads/campfire-extracted/', progress_
↪hook=progress_hook)
(0.0, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/source/_camp_
↪objectplacementindicatorthread01.psc')
(0.0003748842843881753, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_objectplacementindicatorthread01.psc')
(0.0003748842843881753, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_objectplacementindicatorthread02.psc')
(0.0007497685687763506, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_objectplacementindicatorthread02.psc')
(0.0007497685687763506, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_objectplacementindicatorthread03.psc')
(0.0011246528531645259, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_objectplacementindicatorthread03.psc')
(0.0011246528531645259, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_tentsitlayscript.psc')
(0.004051940775940278, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/_camp_tentsitlayscript.psc')
(0.004051940775940278, '/home/stephen-bunn/Downloads/campfire-extracted/scripts/
↪source/campcampfire.psc')
... <only first 9 callbacks> ...
```

### 1.2.5 Working with plugins

So far in these quick examples we've only seen examples using archives, but we can also parse and examine plugins as well! Well parsing these file types is done in the same way as archives.

```
>>> from bethesda_structs.plugin import FNVPlugin
>>> plugin = FNVPlugin.parse_file('/media/sf_VMShared/esp/fnv/NVWillow.esp')
>>> print(plugin.container.header)
Container:
    type = u'TES4' (total 4)
    data_size = 163
    flags = Container:
        master = True
    id = 0
    revision = 0
    version = 15
    data = b'HEDR\x0c\x00\x1f\x85\xab?\x97\x12\x00\x00#\xad'... (truncated, total 163)
    subrecords = ListContainer:
        Container:
            type = u'HEDR' (total 4)
            data_size = 12
            data = b'\x1f\x85\xab?\x97\x12\x00\x00#\xad\r\x00' (total 12)
            parsed = Container:
                value = Container:
                    version = 1.340000033378601
                    num_records = 4759
                    next_object_id = 896291
```

(continues on next page)

(continued from previous page)

```
                description = u'Header' (total 6)
        Container:
            type = u'CNAM' (total 4)
            data_size = 9
            data = b'llamaRCA\x00' (total 9)
            parsed = Container:
                value = u'llamaRCA' (total 8)
                description = u'Author' (total 6)
        Container:
            type = u'SNAM' (total 4)
            data_size = 16
            data = b'NVWillow v.1.10\x00' (total 16)
            parsed = Container:
                value = u'NVWillow v.1.10' (total 15)
                description = u'Description' (total 11)
        Container:
            type = u'MAST' (total 4)
            data_size = 14
            data = b'FalloutNV.esm\x00' (total 14)
            parsed = Container:
                value = u'FalloutNV.esm' (total 13)
                description = u'Master Plugin' (total 13)
        Container:
            type = u'DATA' (total 4)
            data_size = 8
            data = b'\x00\x00\x00\x00\x00\x00\x00\x00' (total 8)
            parsed = Container:
                value = 0
                description = u'File Size' (total 9)
        Container:
            type = u'ONAM' (total 4)
            data_size = 68
            data = b'V\xe3\x0c\x00\xc3\xe3\x0c\x00\xc4\xe3\x0c\x00\xc5\xe3\x0c\x00'...
↪ (truncated, total 68)
            parsed = Container:
                value = ListContainer:
                    844630
                    844739
                    844740
                    844741
                    1372461
                    1372463
                    1383111
                    1385321
                    1387301
                    1387302
                    1387303
                    1387304
                    1387906
                    1457771
                    1479505
                    1520201
                    1544392
                description = u'Overridden Records' (total 18)
```

### Getting plugin masters

One of the most common tasks in plugin analysis is determining the masters of a plugin. The names of a plugin's masters are stored within the `MAST` subrecords in the plugin's header record. Using the information above, you can get these names like the following:

```
>>> from bethesda_structs.plugin import FNVPlugin
>>> masters = []
>>> for subrecord in plugin.container.header.subrecords:
...     if subrecord.type == 'MAST':
...         masters.append(subrecord.parsed.value)
>>> print(masters)
['FalloutNV.esm']
```

You can also use the *`iter_subrecords()`* helper method to simplify your code:

```
>>> masters = [
...     subrecord.parsed.value
...     for subrecord in plugin.iter_subrecords(
...         'MAST', 'TES4',
...         include_header=True
...     )
... ]
>>> print(masters)
['FalloutNV.esm']
```

### Getting key (`KEYM`) records

Here is a quick example at getting the data for the first `KEYM` record (an in-game key). This probably really isn't that helpful to you, but I think an example was needed on how to iterate over specific records (as they can become quite large).

```
>>> from bethesda_structs.plugin import FNVPlugin
>>> for record in plugin.iter_records('KEYM'):
...     print(record)
...     break
Container:
    type = u'KEYM' (total 4)
    data_size = 279
    flags = Container:
    id = 17415634
    revision = 0
    version = 15
    data = b'EDID\x17\x00WillowNova'... (truncated, total 279)
    subrecords = ListContainer:
        Container:
            type = u'EDID' (total 4)
            data_size = 23
            data = b'WillowNovacBunga'... (truncated, total 23)
            parsed = Container:
                value = u'WillowNovacBungalowKey' (total 22)
                description = u'Editor ID' (total 9)
        Container:
            type = u'OBND' (total 4)
            data_size = 12
            data = b'\xff\xff\xfc\xff\x00\x00\x01\x00\x04\x00\x00\x00' (total 12)
```

(continues on next page)

(continued from previous page)

```
            parsed = Container:
                value = Container:
                    X1 = -1
                    Y1 = -4
                    Z1 = 0
                    X2 = 1
                    Y2 = 4
                    Z2 = 0
                description = u'Object Bounds' (total 13)
        Container:
            type = u'FULL' (total 4)
            data_size = 27
            data = b'Dino Dee-lite Bu'... (truncated, total 27)
            parsed = Container:
                value = u'Dino Dee-lite Bungalow Key' (total 26)
                description = u'Name' (total 4)
        Container:
            type = u'MODL' (total 4)
            data_size = 23
            data = b'Clutter\\Key01Dir'... (truncated, total 23)
            parsed = Container:
                value = u'Clutter\\Key01Dirty.NIF' (total 22)
                description = u'Model Filename' (total 14)
        Container:
            type = u'ICON' (total 4)
            data_size = 48
            data = b'Interface\\Icons\\'... (truncated, total 48)
            parsed = Container:
                value = u'Interface\\Icons\\PipboyImages\\Ite'... (truncated, total␣
↪47)
                description = u'Large Icon Filename' (total 19)
        Container:
            type = u'MICO' (total 4)
            data_size = 66
            data = b'Interface\\Icons\\'... (truncated, total 66)
            parsed = Container:
                value = u'Interface\\Icons\\PipboyImages_sma'... (truncated, total 65)
                description = u'Small Icon Filename' (total 19)
        Container:
            type = u'SCRI' (total 4)
            data_size = 4
            data = b'T.\n\x01' (total 4)
            parsed = Container:
                value = FormID(form_id=17444436, forms=['SCPT'])
                description = u'Script' (total 6)
        Container:
            type = u'YNAM' (total 4)
            data_size = 4
            data = b'\xbb\x10\x07\x00' (total 4)
            parsed = Container:
                value = FormID(form_id=463035, forms=['SOUN'])
                description = u'Sound - Pick Up' (total 15)
        Container:
            type = u'ZNAM' (total 4)
            data_size = 4
            data = b'\xbc\x10\x07\x00' (total 4)
            parsed = Container:
```

(continues on next page)

(continued from previous page)

```
            value = FormID(form_id=463036, forms=['SOUN'])
            description = u'Sound - Drop' (total 12)
        Container:
            type = u'DATA' (total 4)
            data_size = 8
            data = b'\x00\x00\x00\x00\x00\x00\x00\x00' (total 8)
            parsed = Container:
                value = Container:
                    value = 0
                    weight = 0.0
                description = u'Data' (total 4)
```

CHAPTER 2

# Contributing

When contributing to this repository, please first discuss the change you wish to make via an issue to the owners of this repository before submitting a pull request.

**Important: We have an enforced style guide and a code of conduct.** Please follow them in all your interactions with this project.

## 2.1 Style Guide

- We somewhat follow PEP8 and utilize Sphinx docstrings on **all** classes and functions.
- **We employee flake8 as our linter with exceptions to the following rules:**
  - D203
  - F401
  - E123
  - W503
  - E203
- Maximum line length is 88 characters.
- Maxiumum McCabe complexity is 12.
- Linting and test environments are configured via `tox.ini`.
- Imports are sorted using isort with multi-line output mode `3`.
- An `.editorconfig` file is included in this repository which dictates whitespace, indentation, and file encoding rules.
- Although `requirements.txt` and `requirements_dev.txt` do exist, Pipenv is utilized as the primary virtual environment and package manager for this project.

- We strictly utilize Semantic Versioning as our version specification.
- All Python source files are post-processed using ambv/black.

## 2.2 Issues

Issues should follow the included `ISSUE_TEMPLATE` found in `.github/ISSUE_TEMPLATE.md`.

- **Issues should contain the following sections:**
  - Expected Behavior
  - Current Behavior
  - Possible Solution
  - Steps to Reproduce (for bugs)
  - Context
  - Your Environment

These sections help the developers greatly by providing a large understanding of the context of the bug or requested feature without having to launch a full fleged discussion inside of the issue.

## 2.3 Pull Requests

Pull requests should follow the included `PULL_REQUEST_TEMPLATE` found in `.github/PULL_REQUEST_TEMPLATE.md`.

- Pull requests should always be from a **topic/feature/bugfix** (left side) branch. *Pull requests from master branches will not be merged.*
- Pull requests should not fail our requested style guidelines or linting checks.

## 2.4 Code of Conduct

Our code of conduct is taken directly from the Contributor Covenant since it directly hits all of the points we find necessary to address.

### 2.4.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

### 2.4.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences

- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 2.4.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

### 2.4.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

### 2.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at stephen@bunn.io. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

### 2.4.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

# CHAPTER 3

# Credit

This project was aided by the research and resources of many other projects.
*For their great contributions, they definitely deserve to be noted.*

**TES5Edit/fopdoc**

> This is probably the most useful resource that I have found documenting the structures being used by plugins. The documentation focuses mostly on Fallout plugins, but these are easily translated over to The Elder Scrolls plugins as they share the same structure.
>
> **Best for resources about Fallout plugin strucutres and subrecord formats.**

**jonwd7/bae**

> This project helped me better understand the structures and fields that are used in Bethesda's archives. Being written in C++ the logic is not *super* structured, but seems to be correct.
>
> **Best for learning how to handle BSA and BTDX archives.**

**Unofficial Elder Scrolls Pages**

> This documenation helps with some of the specific structures used in "The Elder Scrolls" series. Although the documentation of records and subrecord structures is sometimes hard to understand, contains a lot of useful information that *Fopdoc* does not cover
>
> **Best for filling in the blanks regarding TES that Fopdoc doesn't cover.**

## 3.1 The Icon

I know that the icon doesn't 100% match the title of the project, but I really enjoy Bethesda Game Studio's logo. The reason the logo contains the text "Structs for Bethesda" instead of "Bethesda Structs" is because I didn't want to use "Bethesda" as the primary text.
*This might allow the project be misconstrued as a product of Bethesda.*

The font used in the icon is Microsquare.

The partial gear icon is contributed to Bethesda Game Studios.

CHAPTER 4

# Changelog

All notable changes to this project will be documented in this file.
The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

## 4.1 WIP: 0.1.5 (*unreleased*)

- proper plugin parsing logic

## 4.2 0.1.4 (*2019-08-18*)

- fixed `RBG` flag naming for DDS pixel format headers in BTDX archives

## 4.3 0.1.3 (*2018-04-22*)

- full rewrite using construct.
- added complete support for BTDX archives
- added complete support for BSA archives (*not including Morrowind*)
- added initial support for FNV plugins
- added basic support for subrecord structure parsing (*not easy* )
- added initial archive tests
- added fancy new logos and badges
- added beginnings of generic structure documentation
- **deprecated** all old structures and files from versions `<0.1.0`

## 4.4 0.0.1 (*2018-01-12*)

- added basic UserWarning for WIP TES5 plugins

## 4.5 0.0.0 (*2017-12-19*)

*this is the first pre-alpha release, so the only other prior change history is the git commit log*

- added initial support for TES4 plugins
- added initial support for *basic* TES5 plugins
- added initial support for BSA archives
- added initial support for BA2 archives
- added guess-work methods for guessing plugin/archive objects from a file
- added abstract class for checksumming classes with filepaths
- added abstract class for auto handling structures with prefixes
- added all of the documentations

CHAPTER 5

# Project Reference

## 5.1 Bethesda Structs Package

The following is the automatically built documentation for the entire *bethesda_structs* package. All objects that are considered to be handlers for a Bethesda file format are subclasses of *BaseFiletype*.

**`Plugins` and `Archives` adhere to the following naming conventions.**

- `{GAME_PREFIX}Plugin` - *FNVPlugin*
- `{TYPE_PREFIX}Archive` - *BSAArchive*

In some cases (such as *BSAArchive*) this may seem repetitive (`BSA[rchive]Archive`). But, in this project, repetition is sacrificed for standardization.

---

**Important:** Common resources for each module (including submodules) are placed in a file named `_common.py` for each module. Typically you will see abstract objects or module helper methods defined in these files.

They are important for **contributors**, but shouldn't ever need to be seen by users.

---

### 5.1.1 bethesda_structs.plugin

This module contains structures that can read and extract Bethesda's plugin file formats. These are files typically denoted with the following extensions:

- `.esp` - *Elder Scrolls Plugin*
- `.esm` - *Elder Scrolls Master*
- `.esl` - *Elder Scrolls Light*

These files contain and register logic and resources in order to extend the content and logic of an "Elder Scrolls" (engine) game. Typically the frontline of runtime errors, being able to understand what data the plugin contains is critical to understanding how to fix errors.

bethesda_structs.plugin.**get_plugin**(*filepath*)
Get an instance of the first plugin that can handle a given file.

**Parameters** **filepath** (*str*) – The path of the file to handle

**Returns** The base plugin

**Return type** *BasePlugin*

### Examples

This method simply returns the first encountered plugin that can handle a given file.

```
>>> FILEPATH = ""  # absolute filepath to some FNV plugin
>>> plugin = bethesda_structs.plugin.get_plugin(FILEPATH)
>>> plugin
FNVPlugin(filepath=PosixPath(...))
```

## Common

Below is a listing of common objects, resources, etc. that can are are probably used throughout the other Plugin objects. An example of this is the *BasePlugin* class which provides an abstract class which all valid plugins should extend.

**class** bethesda_structs.plugin._common.**FormID**(*form_id*, *forms=NOTHING*)
The standardized form id object.

**Raises** ValueError – If forms is not a list of uppercase strings

**validate**(*attribute*, *value*)
Validates forms.

**Parameters**

- **attribute** (*str*) – Should always be forms
- **value** (*Any*) – Should be a list of uppercase strings

**Raises**

- ValueError – If *value* is not a list
- ValueError – If all entries in *value* are not uppercase strings

**class** bethesda_structs.plugin._common.**Subrecord**(*name*, *struct*, *optional=False*, *multiple=False*)
Defines a subrecord that can be further parsed using the supplied struct.

**name_validator**(*attribute*, *value*)
Ensures that the name attribute is valid.

**Parameters**

- **attribute** (*str*) – The attribute name
- **value** (*str*) – The attribute value

**Raises** ValueError – - When the name is not of length 4

**classmethod parse_flag**(*flag*)
Parses a given flag into a tuple for optional and multiple.

**Parameters** **flag** (*str*) – The flag to parse

**Returns** A tuple containing the new (optional, multiple) booleans

**Return type** Tuple[bool, bool]

**classmethod from_dict**(*data*)
Builds a subrecord from a given dictionary.

**Parameters** **data** (*dict*) – The dictionary to use

**Returns** A new instance of a subrecord

**Return type** T_Subrecord

**classmethod from_definition**(*definition*, *struct*)
Builds a subrecord from a given definition.

**Parameters**

- **definition** (*str*) – The definition to build from
- **struct** (*Construct*) – The structure of the subrecord

**Returns** A new instance of a subrecord

**Return type** T_Subrecord

**to_dict**()
Serializes the current subrecord as a dictionary.

---

**Note:** Currently not JSON serializable due to structs requiring the use of lambda functions and self refrences.

---

**Returns** The resulting dictionary.

**Return type** dict

**to_definition**()
Returns the definition of the subrecord.

**Returns** The definition, a tuple of (definition, struct)

**Return type** Tuple[str, Construct]

**be**(*flag*)
Set the optional and multiple arguments.

**Parameters** **flag** (*str*) – The flag to set for the current collection

**Returns** The current subrecord

**Return type** T_Subrecord

**class** bethesda_structs.plugin._common.**SubrecordCollection**(*name*, *items=NOTHING*, *optional=False*, *multiple=False*)
Defines a collection of subrecords.

**items_validator**(*attribute*, *value*)
Ensures that the items attribute is valid.

**Parameters**

- **attribute** (*str*) – The attribute name
- **value** (*list*) – The attribute value

**Raises**

- ValueError – - When the length of the items is not greater than 0
- **TypeError – - When not all of the items within the list are not of instance** Subrecord or SubrecordCollection

**classmethod parse_flag**(*flag*)
Parses a given flag into a tuple for optional and multiple.

**Parameters** **flag** (*str*) – The flag to parse

**Returns** A tuple containing the new (optional, multiple) booleans

**Return type** Tuple[bool, bool]

**classmethod from_dict**(*data*)
Builds a subrecord collection from a given dictionary.

**Parameters** **data** (*dict*) – The dictionary to use

**Returns** A new instance of a collection

**Return type** T_SubrecordCollection

**classmethod from_definition**(*definition*, *data*)
Builds a subrecord collection from a given definition.

**Parameters**

- **definition** (*str*) – The definition to build from
- **data** (*list*) – The data of the definition

**Returns** A new instance of a SubrecordCollection

**Return type** T_SubrecordCollection

**to_dict**()
Serializes the current collection as a dictionary.

---

**Note:** Not JSON serializable due to structs requiring lambda functions and self references.

---

**Returns** The resulting dictionary

**Return type** Dict[str, Any]

**to_definition**()
Returns the definition of the collection.

**Returns** The definition instance

**Return type** Tuple[str, list]

**be**(*flag*)
Set the optional and multiple arguments.

**Parameters** **flag** (*str*) – Teh flag to set for the current collection

**Returns** The current subrecord collection

**Return type** T_SubrecordCollection

**discover**(*names*, *target*, *strict=True*)
Discovers the next expected subrecord given a target.

**Parameters**

- **names** (*list*) – The previously discovered subrecord names
- **target** (*str*) – The target to discover next
- **strict** (*bool, optional*) – Defaults to True. Enforce that required subrecords should appear before the target

**Raises** exceptions.UnexpectedSubrecord – - When nothing is expected next but target requested - When requested target does not match next expected subrecord

**Returns** The resulting discovered subrecord, or None

**Return type** *Subrecord*

**handle_working**(*subrecord_name*, *subrecord_data*, *working_record*, *strict=True*)

**Handles discovering and parsing a given subrecord using a list of already** handled subrecord names.

---

**Note:** Subrecords that cannot be correctly discovered by the collection's discovery process utilize a default GreedyBytes * "Not Handled struct. So any subrecord that cannot be discovered correctly or isn't handled correctly with simply be a Container with a value that matches the subrecord's data and a description of Not Handled.

---

**Parameters**

- **subrecord_name** (*str*) – The name of the subrecord to discover and parse
- **subrecord_data** (*bytes*) – The data of the subrecord to discover and parse
- **working_record** (*list*) – The list of names that have already been handled in the working record
- **strict** (*bool*) – Defaults to True, If True, enforce strict discovery

**Returns**

**A tuple of** (parsed container, new handled names to extend the working record with)

**Return type** Tuple[Container, List[str]]

**class** bethesda_structs.plugin._common.**BasePlugin**(*content*, *filepath=None*, *record_registry=<CIMultiDict()>*)
The base class all Plugins should subclass.

**plugin_struct**
The base plugin structure to use for parsing a plugin.

**Returns** The plugin structure

**Return type** Construct

**classmethod parse**(*content*, *filepath=None*)
Create a *BasePlugin* from a byte array.

**Parameters**

- **content** (*bytes*) – The byte content of the archive

- **filepath** (*str, optional*) – Defaults to None. Sets the filepath attribute for user's reference

**Raises** ValueError – If the given content is not of bytes

**Returns** A created *BasePlugin*

**Return type** T_BasePlugin

**iter_records**(*record_type=None*, *include_header=False*)
Iterates over the container's records.

**record_type (str, optional): Defaults to None. Filters the record types to** yield

**include_header (bool, optional): Defaults to False. Includes the header** record (regardless of `record_type`)

**Yields** *Container* – A record's container

**Return type** Generator[Container, None, None]

**iter_subrecords**(*subrecord_type=None*, *record_type=None*, *include_header=False*)
Iterates over the container's subrecords.

**subrecord_type (str, optional): Defaults to None. Filters the subrecord** types to yield

**record_type (str, optional): Defaults to None. Filters the record types to** look for subrecords in

**include_header (bool, optional): Defaults to False. Includes the header** record in the filter (regardless of `record_type`)

**Yields** *Container* – A subrecord's container

**Return type** Generator[Container, None, None]

### Fallout: New Vegas

This module contains all the required structures to parse FNV plugins.

**class** bethesda_structs.plugin.fnv.**FNVPlugin**(*content*, *filepath=None*, *record_registry=<CIMultiDict()>*)
Bases: *bethesda_structs.plugin._common.BasePlugin*

The plugin for Fallout: New Vegas.

**This plugin structure *should* handle plugins for the games:**

- Fallout 3
- Fallout: New Vegas

---

**Note:** This structure *currently* reads all data on initialization. This may appear as *slower* initialization times for larger plugins. Should be fixed by lazy constructs when they become more stable.

---

**Credit:**

- FopDoc

**subrecord_struct = <Struct>**
The structure for FO3/FNV subrecords.

**Returns** The structure of FO3/FNV subrecords

**Return type** Struct

**record_struct = <Struct>**
The structure for FO3/FNV records.

**Returns** The structure of FO3/FNV records

**Return type** Struct

**group_struct = <Struct>**
The structure for FO3/FNV groups.

**Returns** The structure of FO3/FNV groups

**Return type** Struct

**plugin_struct = <Struct>**
The structure for FO3/FNV plugins.

**Returns** The structure of FO3/FNV plugins

**Return type** Struct

**classmethod can_handle**(*filepath*)
Determines if a given file can be handled by the plugin.

**Parameters** **filepath** (*str*) – The filepath to evaluate

**Raises** FileNotFoundError – When the given *filepath* cannot be found

**Returns** True if file can be handled, otherwise False

**Return type** bool

**classmethod parse_subrecord**(*record_id*, *record_type*, *subrecord_type*, *subrecord_data*, *strict=True*)
Parses a subrecord's data.

**Parameters**

- **record_type** (*str*) – The parent record type
- **subrecord_type** (*str*) – The subrecord type
- **subrecord_data** (*bytes*) – The subrecord data to parse
- **strict** (*bool*) – Defaults to True, If True, enforce strict subrecord discovery

**Returns** The resulting parsed container

**Return type** Container

### Fallout 3

This module contains all the required structures to parse FO3 plugins.

**class** bethesda_structs.plugin.fo3.**FO3FormID**(*forms*, **args*, ***kwargs*)
Bases: bethesda_structs.plugin.fnv._common.FNVFormID

A formid wrapper for Fallout 3.

**Note:** Because the logic for parsing Fallout 3 form ids is the same as Fallout: New Vegas form ids, this class is simply a subclass of `FNVFormID`.

**Credit:**

- FopDoc

**class** bethesda_structs.plugin.fo3.**FO3Plugin**(*content*, *filepath=None*, *record_registry=<CIMultiDict()>*)

Bases: *bethesda_structs.plugin.fnv.FNVPlugin*

The plugin for Fallout 3.

**Note:** Because the logic for parsing Fallout 3 plugins is the same as Fallout: New Vegas plugins, this class is simply a subclass of *FNVPlugin*.

**Credit:**

- FopDoc

### 5.1.2 bethesda_structs.archive

This module contains structures that can read and extract Bethesda's archive file formats.

bethesda_structs.archive.**get_archive**(*filepath*)

Get an instance of the first archive that can handle a given file.

**Parameters** **filepath** (*str*) – The path of the file to handle

**Returns** The base archive

**Return type** *BaseArchive*

#### Examples

This method simply returns the first encountered archive that can handle a given file.

```
>>> FILEPATH = ""  # absolute filepath to some BSA
>>> archive = bethesda_structs.archive.get_archve(FILEPATH)
>>> archive
BSAArchive(filepath=PosixPath(...))
```

#### Common

Below is a listing of common objects, resources, etc. that can and are probably used throughout the other Archive objects. An example of this is the *BaseArchive* class which provides an abstract class which all valid archives should extend.

**class** bethesda_structs.archive._common.**ArchiveFile**(*filepath*, *data*)

A generic archive file object that can be used for extracting.

The purpose of this object is to provide some generic format for *iter_files()* to yield so that the *extract()* method can be abstracted away from the archive subclasses.

**filepath = None**
The relative filepath of the archived file.

**Returns** The relative filepath of the archived file

**Return type** str

**data = None**
The raw data of the archived file.

**Returns** The raw data of the archived file

**Return type** bytes

**size**
The size of the raw data.

**Returns** The size of the raw data

**Return type** int

**class** bethesda_structs.archive._common.**BaseArchive**(*content*, *filepath=None*)
The base class all Archives should subclass.

**archive_struct**
The base archive structure to use for parsing the **full** archive.

**Raises** NotImplementedError – Subclasses must implement

**Returns** The archive structure

**Return type** construct.Construct

**classmethod parse**(*content*, *filepath=None*)
Create a *BaseArchive* from a byte array.

**Parameters**

- **content** (*bytes*) – The byte content of the archive
- **filepath** (*str, optional*) – Defaults to None. Sets the filepath attribute for user's reference

**Raises** ValueError – If the given content is not of bytes

**Returns** An archive instance

**Return type** *BaseArchive*

**iter_files**()
Iterates over the available files in the archive.

**Yields** *ArchiveFile* – An archive file

**Raises** NotImplementedError – Subclasses must implement

**Return type** Generator[*ArchiveFile*, None, None]

**extract**(*to_dir*, *progress_hook=None*)
Extracts the content of the *BaseArchive* to the given directory.

**Parameters**

- **to_dir** (*str*) – The directory to extract the content to
- **progress_hook** (*Callable[[int, int, str], None], optional*) – Defaults to None. A progress hook that should expect (current, total, current_filepath) as arguments

#### Example

```
>>> FILEPATH = ""  # absolute path to BSA/BTDX archive
>>> archive = bethesda_structs.archive.get_archive(FILEPATH)
>>> archive.extract('/home/username/Downloads/extracted')
```

#### Example

```
>>> def progress_hook(current, total, filepath):
...     print((current / total) * 100.0)
>>> FILEPATH = ""  # absolute path to BSA/BTDX archive
>>> archive = bethesda_structs.archive.get_archive(FILEPATH)
>>> archive.extract(
...     '/home/username/Downloads/extracted',
...     progress_hook=progress_hook
... )
0.0
12.2
12.2
12.67
12.67
50.4443
50.4443
70.0
70.0
92.1
92.1
100.0
```

**Note:** The provided progress hook is simple and two-stage. It is called once before a file is being written and once after the same file is done being written.

### BSA Archives

This module contains all the required structures to extract BSA archives.

**class** bethesda_structs.archive.bsa.**LZ4CompressedAdapter**(*subcon*)
Bases: construct.core.Adapter

An adapter for LZ4 compressed data.

**Note:** v105 BSA's utilize LZ4 compression instead of zlib.

**class** bethesda_structs.archive.bsa.**BSAArchive**(*content*, *filepath=None*)
Bases: *bethesda_structs.archive._common.BaseArchive*

Archive type for BSA files.

BSA stands for "Bethesda ? Archive". These archives are compressed meshes, textures and other static resources that can be loaded as a single file instead of a directory of files (loose files).

**There are currently 4 versions of BSA:**

- ???: Morrowind
- 103: Oblivion
- 104: Fallout 3, Fallout: New Vegas, and Skyrim
- 105: Skyrim: Special Edition

---

**Note:** BSA archives to not read the file data on initialization. Header's, records and names are read in and files are built during *iter_files()*.

---

**Credit:**

- BAE

**header_struct = <Struct>**
The structure of BSA headers.

**Returns** The structure of BSA headers

**Return type** Struct

**directory_record_struct = <Struct>**
The structure of directory records.

**Returns** The structure of directory records

**Return type** Struct

**file_record_struct = <Struct>**
The structure of file records.

**Returns** The structure of file records

**Return type** Struct

**directory_block_struct = <Struct>**
The structure of directory blocks.

**Returns** The structure of directory blocks

**Return type** Struct

**archive_struct = <Struct>**
The **partial** structure of BSA archives.

**Returns** The **partial** structure of BSA archives

**Return type** Struct

**uncompressed_file_struct**
The uncompressed file structure for uncompressed files.

**Returns** The uncompressed file structure for uncompressed files.

**Return type** Struct

**compressed_file_struct**
The compressed file structure for compressed files.

**Returns** The compressed file structure for compressed files.

**Return type** Struct

**classmethod can_handle**(*filepath*)
Determines if a given file can be handled by the current archive.

**Parameters** **filepath** (*str*) – The filepath to check if can be handled

**Return type** bool

**iter_files**()
Iterates over the parsed data and yields instances of *ArchiveFile*.

**Yields** *ArchiveFile* – An file contained within the archive

**Return type** Generator[*ArchiveFile*, None, None]

### BTDX Archives

This module (along with *dds*) contains all the required structures to read and extract BTDX (.ba2) archives.

**class** bethesda_structs.archive.btdx.**BTDXArchive**(*content*, *filepath=None*)
Bases: *bethesda_structs.archive._common.BaseArchive*

Archive type for BTDX files (aka. BA2).

BTDX files (utilize the extension .ba2) are Bethesda's second framework revision for archives. These files have virtually the same goal as *BSAArchive* but for optimized loading of archived textures directly into the engine instead of simply compressing the files.

**This is done by splitting the BTDX archive into 2 different types:**

- GNRL: Storage for general files that are simply compressed
- DX10: Storage for Microsoft DirectDraw textures in an optimized format

The extraction for GNRL files is simple. But the extraction for DX10 requires rebuliding the DDS headers for each of the texture chunks. For this reason the *bethesda_structs.contrib.dds* module was added.

---

**Note:** BTDX archives to not read the file data on initialization. Header's, records and names are read in and files are built during *iter_files()*.

---

**Reference:**

- BAE

**header_struct = <Struct>**
The structure of BTDX headers.

**Returns** The structure of BTDX headers

**Return type** Struct

**file_struct = <Struct>**
The structure of GNRL files.

**Returns** The structure of GNRL files

**Return type** Struct

**tex_header_struct = <Struct>**
The structure of DX10 file headers.

**Returns** The structure of DX10 file headers.

**Return type** Struct

**tex_chunk_struct = <Struct>**
The structure of DX10 chunks.

**Returns** The structure of DX10 chunks

**Return type** Struct

**tex_struct = <Struct>**
The structure of DX10 tex files.

**Returns** The structure of DX10 tex files

**Return type** Struct

**archive_struct = <Struct>**
The **partial** structure of BTDX archives.

**Returns** The **partial** structure of BTDX archives

**Return type** Struct

**classmethod can_handle**(*filepath*)
Determines if a given file can be handled by the current archive.

**Parameters** **filepath** (*str*) – The filepath to check if can be handled

**Return type** bool

**iter_files**()
Iterates over the parsed data and yields instances of *ArchiveFile*

**Raises** ValueError – If a filename cannot be determined for a specific file record

**Yields** *ArchiveFile* – A file contained within the archive

**Return type** Generator[*ArchiveFile*, None, None]

### 5.1.3 bethesda_structs.contrib

This module contains various resources that required by some of Bethesda's file formats.

#### DDS

This module provides structures, and resources for Microsoft's Direct Draw Surface formats. Required by BTDXArchive as DDS headers have to be rebuilt for extraction.

bethesda_structs.contrib.dds.**MAKEFOURCC**(*ch0*, *ch1*, *ch2*, *ch3*)
Implementation of Window's *MAKEFOURCC*.

This is simply just returning the bytes of the joined characters. *MAKEFOURCC(*"DX10")* can also be implemented by *Bytes(b"DX10")*.

**Parameters**

- **ch0** (*str*) – First char
- **ch1** (*str*) – Second char
- **ch2** (*str*) – Third char
- **ch3** (*str*) – Fourth char

**Returns** The integer representation of given characters.

**Return type** int

**Reference:** Microsoft

**class** bethesda_structs.contrib.dds.**DXGIFormats**
The format enum for DXGI files.

**class** bethesda_structs.contrib.dds.**D3D10ResourceDimension**
The dimension enum for D3D10 resources.

**class** bethesda_structs.contrib.dds.**D3D10ResourceMiscFlag**
The miscellaneous flags for D3D10 resources.

bethesda_structs.contrib.dds.**DXGI_FORMAT = <Enum <FormatField>>**
The DXGI_FORMAT structure.

**Reference:** Microsoft

bethesda_structs.contrib.dds.**D3D10_RESOURCE_DIMENSION = <Enum <FormatField>>**
The D3D10_RESOURCE_DIMENSION structure.

**Reference:** Microsoft

bethesda_structs.contrib.dds.**D3D10_RESOURCE_MISC_FLAG = <FlagsEnum <FormatField>>**
The D3D10_RESOURCE_MISC_FLAG structure.

**Reference:** Microsoft

bethesda_structs.contrib.dds.**DDS_PIXELFORMAT = <Struct +nonbuild>**
The DDS_PIXELFORMAT structure.

**Reference:** Microsoft

bethesda_structs.contrib.dds.**DDS_HEADER = <Struct>**
The DDS_HEADER structure.

**Reference:** Microsoft

bethesda_structs.contrib.dds.**DDS_HEADER_DX10 = <Struct +nonbuild>**
The DDS_HEADER_DX10 structure.

**Reference:** Microsoft

---

**class** bethesda_structs._common.**BaseFiletype**
Bases: abc.ABC

The base filetype for all supported file parsers.

**classmethod can_handle**(*filepath*)
Determines if a given *filepath* can be handled by the archive.

**Parameters** **filepath** (*str*) – The filepath to evaluate

**Raises** NotImplementedError – Subclasses must implement

**Return type** bool

**classmethod parse**(*content*, *filepath=None*)
Create a *BaseFiletype* from a byte array.

**Parameters**

- **content** (*bytes*) – The byte content
- **filepath** (*str, optional*) – Defaults to None. Sets the filepath attribute for user's reference

**Raises** NotImplementedError – Subclasses must implement

**Returns** A filetype instance

**Return type** *BaseFiletype*

**classmethod parse_stream**(*stream*, *filepath=None*)
Create a *BaseFiletype* from a file stream.

**Parameters**

- **stream** (*io.BufferedReader*) – A file stream to read from.
- **filepath** (*str, optional*) – Defaults to None. Sets the filepath attribute for user's reference.

**Raises** ValueError – If the given stream is not of bytes

**Returns** A filetype instance

**Return type** *BaseFiletype*

**classmethod parse_file**(*filepath*)
Create a *BaseFiletype* from a given filepath.

**Parameters** **filepath** (*str*) – The filepath to read from

**Raises** FileNotFoundError – If the given filepath does not exist

**Returns** A filetype instance

**Return type** *BaseFiletype*

# Python Module Index

## b

# Index

## U



## V