

# File Types

- [Naming Conventions](#)
- [Pickling](#)
- [JSON](#)
- [Plaintext](#)
- [Encryption](#)

# Naming Conventions

The proper (though not required) filename extension for a PYNDatabase is `.pyndb`

If saving to a JSON file, the prefix `.json` should be used instead.

As of version 3.2.0, pyndb will now automatically set the filetype for you based on the file's extension. The recognized extensions and their filetypes are as follows:

- `.pyndb` → pickled
- `.json` → json
- `.txt` or `.pydb` → plaintext (`.pydb` is included here for greater backwards compatibility)

All other files default to a pickled filetype.

“ Fun fact: In earlier versions not released to the public, the project was named DataManager and did not use the Node structure. Thus the proper filename extension was `.pydb`. This is partially how the name pyndb came to be.

# Pickling

pyndb saves data using `pickle` by default. `pickle` is installed with python, and thus should not require installation.

## Why pickle it?

Saving the data to file with `pickle` allows objects to be saved to file without force injecting a module into pyndb.

## Here's an example:

plaintext solution:

```
# Saving the class to file
import pyndb
import datetime
db = pyndb.PYNDatabase('example.pyndb', filetype='plaintext')
db.set('test', datetime.datetime.now())
db.save()

import pyndb
import datetime
pyndb.datetime = datetime
# This injection is necessary for plaintext, as the __repr__ value of the datetime object is what's saved to file.
# It also must be performed before you load the database.
db = pyndb.PYNDatabase('example.pyndb', filetype='plaintext')
print(db.test.val) # Will print the datetime object
```

pickled solution:

```
# Saving the class to file
import pyndb
import datetime
db = pyndb.PYNDatabase('example.pyndb')
db.set('test', datetime.datetime.now())
db.save()

import pyndb
```

```
db = pyndb.PYNDatabase('example.pyndb')  
print(db.test.val) # Will print the datetime object
```

## How will this affect code from v2.x?

For now, backwards compatibility has been added, so when you attempt to open a plaintext file using the default setting (pickled) it will switch to plaintext for you. This will, however, throw a warning when the database loads, as this compatibility may be removed in the future. The option to set the file type to plaintext will remain, though.

## Converting plaintext to pickle

To convert a plaintext file to pickled data, simply set the `filetype` flag to `'pickled'` after initializing the `PYNDatabase`, and then call the `save` method.

## Using plaintext instead

The master class has a `filetype` flag, which can be changed like so:

```
# Change on initialization  
PYNDatabase(file, autosave=False, filetype='pickled')  
  
# Change after initialization  
PYNDatabase.filetype = 'plaintext'
```

# JSON

JSON files are useful when you want your data to be saved as simple, readable, and portable code. This makes it useful in things such as configuration files. JSON files, like plaintext, cannot store objects. To save a PYNDatabase to a JSON file, simply set the `filetype` flag to `json`.

# Plaintext

Plaintext format stores data exactly how you would expect it to - as plain text. What's actually written to the file is the string representation of the fileObj variable. Opening the file will reveal a single line of text being a python dictionary. This is useful for porting dictionaries across python scripts without pyndb, and also for large configurations that aren't meant to be read unless absolutely necessary. Like JSON data, it cannot store objects\*.

\*Objects can be stored if their `__str__` value is a command that recreates the object.

# Encryption

You can use encryption on any file type supported by pyndb.

You must have an up-to-date version of the cryptography module to use encryption features. Run "pip -U install cryptography" to update.

pyndb allows you to encrypt your databases using Fernet.

## Encrypting a new database

Simply specify a password when creating a database to encrypt it.

```
from pyndb import PYNDatabase
new_db = PYNDatabase('new_db.pyndb', password='your-password-here')
```

If you would like, you can also specify a salt and/or a number of iterations.

```
from pyndb import PYNDatabase
new_db = PYNDatabase('new_db.pyndb', password='a really long password', salt=b'A bytes object',
iterations=390000)
```

Due to limitations of PBKDF2, a salt and number of iterations must be provided. By default, pyndb uses `pyndb_default` as the salt, and 390000 as the number of iterations.

## Encrypting an existing database

if you load a database with the wrong password, it will try to load it without one. This is so that unencrypted databases can be loaded even if a password is specified. The password will also be set to `None` so that the database is not accidentally encrypted. Therefore, to encrypt an existing database, you can do as follows:

```
from pyndb import PYNDatabase
old_db = PYNDatabase('old_db.pyndb')
old_db.password = 'a very long password'
old_db.save()
```

## Removing encryption

To remove encryption, you can load the encrypted database with the password and then set it to None.

```
from pyndb import PYNDatabase  
db = PYNDatabase('db.pyndb', password='your-password-here')  
db.password = None  
db.save()
```