shelve — Persistent Storage of Objects

Purpose: The shelve module implements persistent storage for arbitrary Python objects that can be pickled, using a dictionary-like API.

The `shelve` module can be used as a simple persistent storage option for Python objects when a relational database is not required. The shelf is accessed by keys, just as with a dictionary. The values are pickled and written to a database created and managed by dbm.

Creating a new Shelf

The simplest way to use `shelve` is via the `DbfilenameShelf` class. It uses dbm to store the data. The class can be used directly, or by calling `shelve.open()`.

```
# shelve_create.py

import shelve

with shelve.open('test_shelf.db') as s:
    s['key1'] = {
        'int': 10,
        'float': 9.5,
        'string': 'Sample data',
    }
```

To access the data again, open the shelf and use it like a dictionary.

```
# shelve_existing.py

import shelve

with shelve.open('test_shelf.db') as s:
    existing = s['key1']

print(existing)
```

Running both sample scripts produces the following output.

```
$ python3 shelve_create.py
$ python3 shelve_existing.py

{'int': 10, 'float': 9.5, 'string': 'Sample data'}
```

The dbm module does not support multiple applications writing to the same database at the same time, but it does support concurrent read-only clients. If a client will not be modifying the shelf, tell `shelve` to open the database read-only by passing `flag='r'`.

```
# shelve_readonly.py

import dbm
import shelve

with shelve.open('test_shelf.db', flag='r') as s:
    print('Existing:', s['key1'])
    try:
        s['key1'] = 'new value'
    except dbm.error as err:
        print('ERROR: {}'.format(err))
```

If the program tries to modify the database while it is opened read-only, an access error exception is generated. The exception type depends on the database module selected by dbm when the database was created.

```
$ python3 shelve_readonly.py

Existing: {'int': 10, 'float': 9.5, 'string': 'Sample data'}
ERROR: cannot add item to database
```

Write-back

Shelves do not track modifications to volatile objects, by default. That means if the contents of an item stored in the shelf are changed, the shelf must be updated explicitly by storing the entire item again.

```python
# shelve_withoutwriteback.py

import shelve

with shelve.open('test_shelf.db') as s:
    print(s['key1'])
    s['key1']['new_value'] = 'this was not here before'

with shelve.open('test_shelf.db', writeback=True) as s:
    print(s['key1'])
```

In this example, the dictionary at `'key1'` is not stored again, so when the shelf is re-opened, the changes have not been preserved.

```
$ python3 shelve_create.py
$ python3 shelve_withoutwriteback.py

{'int': 10, 'float': 9.5, 'string': 'Sample data'}
{'int': 10, 'float': 9.5, 'string': 'Sample data'}
```

To automatically catch changes to volatile objects stored in the shelf, open it with writeback enabled. The `writeback` flag causes the shelf to remember all of the objects retrieved from the database using an in-memory cache. Each cache object is also written back to the database when the shelf is closed.

```python
# shelve_writeback.py

import shelve
import pprint

with shelve.open('test_shelf.db', writeback=True) as s:
    print('Initial data:')
    pprint.pprint(s['key1'])

    s['key1']['new_value'] = 'this was not here before'
    print('\nModified:')
    pprint.pprint(s['key1'])

with shelve.open('test_shelf.db', writeback=True) as s:
    print('\nPreserved:')
    pprint.pprint(s['key1'])
```

Although it reduces the chance of programmer error, and can make object persistence more transparent, using writeback mode may not be desirable in every situation. The cache consumes extra memory while the shelf is open, and pausing to write every cached object back to the database when it is closed slows down the application. All of the cached objects are written back to the database because there is no way to tell if they have been modified. If the application reads data more than it writes, writeback will impact performance unnecessarily.

```
$ python3 shelve_create.py
$ python3 shelve_writeback.py

Initial data:
{'float': 9.5, 'int': 10, 'string': 'Sample data'}

Modified:
{'float': 9.5,
 'int': 10,
 'new_value': 'this was not here before',
 'string': 'Sample data'}

Preserved:
{'float': 9.5,
 'int': 10,
 'new_value': 'this was not here before',
 'string': 'Sample data'}
```

Specific Shelf Types

The earlier examples all used the default shelf implementation. Using `shelve.open()` instead of one of the shelf implementations directly is a common usage pattern, especially if it does not matter what type of database is used to store the data. There are times, however, when the database format is important. In those situations, use `DbfilenameShelf` or `BsdDbShelf` directly, or even subclass `Shelf` for a custom solution.

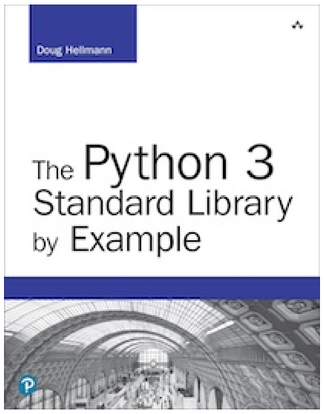
See also

- Standard library documentation for shelve
- dbm – The `dbm` module finds an available DBM library to create a new database.
- feedcache – The `feedcache` module uses `shelve` as a default storage option.
- shove – Shove implements a similar API with more back-end formats.

pickle — Object Serialization   dbm — Unix Key-Value Databases

This page was last updated 2018-03-18.

Navigation
pickle — Object Serialization
dbm — Unix Key-Value Databases



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site
Module Index
Index

Other Writing

- Blog
- The Python Standard Library By Example