12.4. `shelve` — Python object persistence¶

A “shelf” is a persistent, dictionary-like object. The difference with “dbm” databases is that the values (not the keys!) in a shelf can be essentially arbitrary Python objects — anything that the `pickle` module can handle. This includes most class instances, recursive data types, and objects containing lots of shared sub-objects. The keys are ordinary strings.

`shelve.open`(*filename*[, *flag='c'*[, *protocol=None*[, *writeback=False*]]])¶

Open a persistent dictionary. The filename specified is the base filename for the underlying database. As a side-effect, an extension may be added to the filename and more than one file may be created. By default, the underlying database file is opened for reading and writing. The optional *flag* parameter has the same interpretation as the *flag* parameter of `anydbm.open()`.

By default, version 0 pickles are used to serialize values. The version of the pickle protocol can be specified with the *protocol* parameter.

Changed in version 2.3: The *protocol* parameter was added.

Because of Python semantics, a shelf cannot know when a mutable persistent-dictionary entry is modified. By default modified objects are written *only* when assigned to the shelf (see *Example*). If the optional *writeback* parameter is set to *True*, all entries accessed are also cached in memory, and written back on `sync()` and `close()`; this can make it handier to mutate mutable entries in the persistent dictionary, but, if many entries are accessed, it can consume vast amounts of memory for the cache, and it can make the close operation very slow since all accessed entries are written back (there is no way to determine which accessed entries are mutable, nor which ones were actually mutated).

Note

Do not rely on the shelf being closed automatically; always call `close()` explicitly when you don’t need it any more, or use a `with` statement with `contextlib.closing()`.

Shelf objects support all methods supported by dictionaries. This eases the transition from dictionary based scripts to those requiring persistent storage.

Two additional methods are supported:

`Shelf.sync`()¶
Write back all entries in the cache if the shelf was opened with *writeback* set to `True`. Also empty the cache and synchronize the persistent dictionary on disk, if feasible. This is called automatically when the shelf is closed with `close()`.

`Shelf.close`()¶
Synchronize and close the persistent *dict* object. Operations on a closed shelf will fail with a `ValueError`.

See also

Persistent dictionary recipe with widely supported storage formats and having the speed of native dictionaries.

12.4.1. Restrictions¶

- The choice of which database package will be used (such as `dbm`, `gdbm` or `bsddb`) depends on which interface is available. Therefore it is not safe to open the database directly using `dbm`. The database is also (unfortunately) subject to the limitations of `dbm`, if it is used — this means that (the pickled representation of) the objects stored in the database should be fairly small, and in rare cases key collisions may cause the database to refuse updates.
- The `shelve` module does not support *concurrent* read/write access to shelved objects. (Multiple simultaneous read accesses are safe.) When a program has a shelf open for writing, no other program should have it open for reading or writing. Unix file locking can be used to solve this, but this differs across Unix versions and requires knowledge about the database implementation used.

class `shelve.Shelf`(*dict*[, *protocol=None*[, *writeback=False*]])¶

A subclass of `UserDict.DictMixin` which stores pickled values in the *dict* object.

By default, version 0 pickles are used to serialize values. The version of the pickle protocol can be specified with the *protocol* parameter. See the `pickle` documentation for a discussion of the pickle protocols.

Changed in version 2.3: The *protocol* parameter was added.

If the *writeback* parameter is `True`, the object will hold a cache of all entries accessed and write them back to the *dict* at sync and close times. This allows natural operations on mutable entries, but can consume much more memory and make sync and close take a long time.

class `shelve.BsdDbShelf`(*dict*[, *protocol=None*[, *writeback=False*]])¶
A subclass of `Shelf` which exposes `first()`, `next()`, `previous()`, `last()` and `set_location()` which are available in the `bsddb` module but not in other database modules. The *dict* object passed to the constructor must support those methods. This is generally accomplished by calling one of `bsddb.hashopen()`, `bsddb.btopen()` or `bsddb.rnopen()`. The optional *protocol* and *writeback* parameters have the same interpretation as for the `Shelf` class.

class `shelve.DbfilenameShelf`(*filename*[, *flag='c'*[, *protocol=None*[, *writeback=False*]]])¶
A subclass of `Shelf` which accepts a *filename* instead of a dict-like object. The underlying file will be opened using `anydbm.open()`. By default, the file will be created and opened for both read and write. The optional *flag* parameter has the same interpretation as for the `open()` function. The optional *protocol* and *writeback* parameters have the same interpretation as for the `Shelf` class.

12.4.2. Example¶

To summarize the interface (`key` is a string, `data` is an arbitrary object):

```
import shelve

d = shelve.open(filename) # open -- file may get suffix added by low-level
                         # library

d[key] = data   # store data at key (overwrites old data if
               # using an existing key)
data = d[key]   # retrieve a COPY of data at key (raise KeyError if no
               # such key)
del d[key]      # delete data stored at key (raises KeyError
               # if no such key)
flag = d.has_key(key)   # true if the key exists
klist = d.keys() # a list of all existing keys (slow!)

# as d was opened WITHOUT writeback=True, beware:
d['xx'] = range(4)  # this works as expected, but...
d['xx'].append(5)   # *this doesn't!* -- d['xx'] is STILL range(4)!

# having opened d without writeback=True, you need to code carefully:
temp = d['xx']      # extracts the copy
temp.append(5)      # mutates the copy
d['xx'] = temp      # stores the copy right back, to persist it

# or, d=shelve.open(filename,writeback=True) would let you just code
# d['xx'].append(5) and have it work as expected, BUT it would also
# consume more memory and make the d.close() operation slower.

d.close()       # close it
```

See also

Module `anydbm`
Generic interface to `dbm`-style databases.

Module `bsddb`
BSD `db` database interface.

Module `dbhash`
Thin layer around the `bsddb` which provides an `open()` function like the other database modules.

Module `dbm`
Standard Unix database interface.

Module `dumbdbm`
Portable implementation of the `dbm` interface.

Module `gdbm`
GNU database interface, based on the `dbm` interface.

Module `pickle`
Object serialization used by `shelve`.

Module `cPickle`
High-performance version of `pickle`.

Table Of Contents

Previous topic

12.3. `copy_reg` — Register `pickle` support functions

Next topic

12.5. `marshal` — Internal Python object serialization

This Page

- Show Source

Navigation

- index
- modules |
- next |
- previous |
- Python v2.6.4 documentation »
- The Python Standard Library »
- 12. Data Persistence »