12.5. dbm — Interfaces to Unix “databases”

Source code: Lib/dbm/__init__.py

`dbm` is a generic interface to variants of the DBM database — `dbm.gnu` or `dbm.ndbm`. If none of these modules is installed, the slow-but-simple implementation in module `dbm.dumb` will be used. There is a third party interface to the Oracle Berkeley DB.

exception `dbm.`**`error`**

A tuple containing the exceptions that can be raised by each of the supported modules, with a unique exception also named `dbm.error` as the first item — the latter is used when `dbm.error` is raised.

`dbm.`**`whichdb`**(*filename*)

This function attempts to guess which of the several simple database modules available — `dbm.gnu`, `dbm.ndbm` or `dbm.dumb` — should be used to open a given file.

Returns one of the following values: `None` if the file can’t be opened because it’s unreadable or doesn’t exist; the empty string (`''`) if the file’s format can’t be guessed; or a string containing the required module name, such as `'dbm.ndbm'` or `'dbm.gnu'`.

`dbm.`**`open`**(*file*, *flag='r'*, *mode=0o666*)

Open the database file *file* and return a corresponding object.

If the database file already exists, the `whichdb()` function is used to determine its type and the appropriate module is used; if it does not exist, the first module listed above that can be imported is used.

The optional *flag* argument can be:

Value	Meaning
`'r'`	Open existing database for reading only (default)
`'w'`	Open existing database for reading and writing
`'c'`	Open database for reading and writing, creating it if it doesn’t exist
`'n'`	Always create a new, empty database, open for reading and writing

The optional *mode* argument is the Unix mode of the file, used only when the database has to be created. It defaults to octal `0o666` (and will be modified by the prevailing umask).

The object returned by `open()` supports the same basic functionality as dictionaries; keys and their corresponding values can be stored, retrieved, and deleted, and the `in` operator and the `keys()` method are available, as well as `get()` and `setdefault()`.

Changed in version 3.2: `get()` and `setdefault()` are now available in all database modules.

Key and values are always stored as bytes. This means that when strings are used they are implicitly converted to the default encoding before being stored.

These objects also support being used in a `with` statement, which will automatically close them when done.

Changed in version 3.4: Added native support for the context management protocol to the objects returned by `open()`.

The following example records some hostnames and a corresponding title, and then prints out the contents of the database:

```
import dbm

# Open database, creating it if necessary.
with dbm.open('cache', 'c') as db:

    # Record some values
    db[b'hello'] = b'there'
    db['www.python.org'] = 'Python Website'
    db['www.cnn.com'] = 'Cable News Network'

    # Note that the keys are considered bytes now.
    assert db[b'www.python.org'] == b'Python Website'
    # Notice how the value is now in bytes.
    assert db['www.cnn.com'] == b'Cable News Network'

    # Often-used methods of the dict interface work too.
    print(db.get('python.org', b'not present'))

    # Storing a non-string key or value will raise an exception (most
    # likely a TypeError).
    db['www.yahoo.com'] = 4

# db is automatically closed when leaving the with statement.
```

See also:

Module `shelve`
: Persistence module which stores non-string data.

The individual submodules are described in the following sections.

12.5.1. `dbm.gnu` — GNU's reinterpretation of dbm

Source code: Lib/dbm/gnu.py

This module is quite similar to the `dbm` module, but uses the GNU library `gdbm` instead to provide some additional functionality. Please note that the file formats created by `dbm.gnu` and `dbm.ndbm` are incompatible.

The `dbm.gnu` module provides an interface to the GNU DBM library. `dbm.gnu.gdbm` objects behave like mappings (dictionaries), except that keys and values are always converted to bytes before storing. Printing a `gdbm` object doesn't print the keys and values, and the `items()` and `values()` methods are not supported.

exception `dbm.gnu.`**`error`**
: Raised on `dbm.gnu`-specific errors, such as I/O errors. `KeyError` is raised for general mapping errors like specifying an incorrect key.

`dbm.gnu.`**`open`**(*filename*[, *flag*[, *mode*]])
: Open a `gdbm` database and return a `gdbm` object. The *filename* argument is the name of the database file.

  The optional *flag* argument can be:

Value	Meaning
`'r'`	Open existing database for reading only (default)
`'w'`	Open existing database for reading and writing
`'c'`	Open database for reading and writing, creating it if it doesn't exist
`'n'`	Always create a new, empty database, open for reading and writing

  The following additional characters may be appended to the flag to control how the database is opened:

Value	Meaning
`'f'`	Open the database in fast mode. Writes to the database will not be synchronized.
`'s'`	Synchronized mode. This will cause changes to the database to be immediately written to the file.
`'u'`	Do not lock database.

Not all flags are valid for all versions of `gdbm`. The module constant `open_flags` is a string of supported flag characters. The exception `error` is raised if an invalid flag is specified.

The optional *mode* argument is the Unix mode of the file, used only when the database has to be created. It defaults to octal `0o666`.

In addition to the dictionary-like methods, `gdbm` objects have the following methods:

`gdbm.`**`firstkey`**()

It's possible to loop over every key in the database using this method and the `nextkey()` method. The traversal is ordered by `gdbm`'s internal hash values, and won't be sorted by the key values. This method returns the starting key.

`gdbm.`**`nextkey`**(*key*)

Returns the key that follows *key* in the traversal. The following code prints every key in the database `db`, without having to create a list in memory that contains them all:

```
k = db.firstkey()
while k != None:
    print(k)
    k = db.nextkey(k)
```

`gdbm.`**`reorganize`**()

If you have carried out a lot of deletions and would like to shrink the space used by the `gdbm` file, this routine will reorganize the database. `gdbm` objects will not shorten the length of a database file except by using this reorganization; otherwise, deleted file space will be kept and reused as new (key, value) pairs are added.

`gdbm.`**`sync`**()

When the database has been opened in fast mode, this method forces any unwritten data to be written to the disk.

gdbm.**close**()

Close the gdbm database.

12.5.2. dbm.ndbm — Interface based on ndbm

Source code: Lib/dbm/ndbm.py

The dbm.ndbm module provides an interface to the Unix "(n)dbm" library. Dbm objects behave like mappings (dictionaries), except that keys and values are always stored as bytes. Printing a dbm object doesn't print the keys and values, and the items() and values() methods are not supported.

This module can be used with the "classic" ndbm interface or the GNU GDBM compatibility interface. On Unix, the **configure** script will attempt to locate the appropriate header file to simplify building this module.

exception dbm.ndbm.**error**

Raised on dbm.ndbm-specific errors, such as I/O errors. KeyError is raised for general mapping errors like specifying an incorrect key.

dbm.ndbm.**library**

Name of the ndbm implementation library used.

dbm.ndbm.**open**(*filename*[, *flag*[, *mode*]])

Open a dbm database and return a ndbm object. The *filename* argument is the name of the database file (without the .dir or .pag extensions).

The optional *flag* argument must be one of these values:

Value	Meaning
'r'	Open existing database for reading only (default)
'w'	Open existing database for reading and writing
'c'	Open database for reading and writing, creating it if it doesn't exist
'n'	Always create a new, empty database, open for reading and writing

The optional *mode* argument is the Unix mode of the file, used only when the database has to be created. It defaults to octal 0o666 (and will be modified by the prevailing umask).

In addition to the dictionary-like methods, `ndbm` objects provide the following method:

`ndbm.`**`close`**`()`

Close the `ndbm` database.

12.5.3. `dbm.dumb` — Portable DBM implementation

Source code: Lib/dbm/dumb.py

> **Note:** The `dbm.dumb` module is intended as a last resort fallback for the `dbm` module when a more robust module is not available. The `dbm.dumb` module is not written for speed and is not nearly as heavily used as the other database modules.

The `dbm.dumb` module provides a persistent dictionary-like interface which is written entirely in Python. Unlike other modules such as `dbm.gnu` no external library is required. As with other persistent mappings, the keys and values are always stored as bytes.

The module defines the following:

exception `dbm.dumb.`**`error`**

Raised on `dbm.dumb`-specific errors, such as I/O errors. `KeyError` is raised for general mapping errors like specifying an incorrect key.

`dbm.dumb.`**`open`**`(`*filename*`[, `*flag*`[, `*mode*`]])`

Open a `dumbdbm` database and return a dumbdbm object. The *filename* argument is the basename of the database file (without any specific extensions). When a dumbdbm database is created, files with `.dat` and `.dir` extensions are created.

The optional *flag* argument supports only the semantics of `'c'` and `'n'` values. Other values will default to database being always opened for update, and will be created if it does not exist.

The optional *mode* argument is the Unix mode of the file, used only when the database has to be created. It defaults to octal `0o666` (and will be modified by the prevailing umask).

Changed in version 3.5: `open()` always creates a new database when the flag has the value `'n'`.

Deprecated since version 3.6, will be removed in version 3.8: Creating database in `'r'` and `'w'` modes. Modifying database in `'r'` mode.

In addition to the methods provided by the `collections.abc.MutableMapping` class, `dumbdbm` objects provide the following methods:

`dumbdbm.`**`sync`**()

Synchronize the on-disk directory and data files. This method is called by the `Shelve.sync()` method.

`dumbdbm.`**`close`**()

Close the `dumbdbm` database.