**16.1. os — Miscellaneous operating system interfaces**

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

Notes on the availability of these functions:

•The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about path in the same format (which happens to have originated with the POSIX interface).

•Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.

•All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.

•An “Availability: Unix” note means that this function is commonly found on Unix systems. It does not make any claims about its existence on a specific operating system.

•If not separately noted, all functions that claim “Availability: Unix” are supported on Mac OS X, which builds on a Unix core.

Note:

All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

**16.1.5. Files and Directories**

On some Unix platforms, many of these functions support one or more of these features:

•specifying a file descriptor: For some functions, the path argument can be not only a string giving a path name, but also a file descriptor. The function will then operate on the file referred to by the descriptor. (For POSIX systems, Python will call the f... version of the function.)

You can check whether or not path can be specified as a file descriptor on your platform using os.supports\_fd. If it is unavailable, using it will raise a NotImplementedError.

If the function also supports dir\_fd or follow\_symlinks arguments, it is an error to specify one of those when supplying path as a file descriptor.

•paths relative to directory descriptors: If dir\_fd is not None, it should be a file descriptor referring to a directory, and the path to operate on should be relative; path will then be relative to that directory. If the path is absolute, dir\_fd is ignored. (For POSIX systems, Python will call the ...at or f...at version of the function.)

You can check whether or not dir\_fd is supported on your platform using os.supports\_dir\_fd. If it is unavailable, using it will raise a NotImplementedError.

•not following symlinks: If follow\_symlinks is False, and the last element of the path to operate on is a symbolic link, the function will operate on the symbolic link itself instead of the file the link points to. (For POSIX systems, Python will call the l... version of the function.)

You can check whether or not follow\_symlinks is supported on your platform using os.supports\_follow\_symlinks. If it is unavailable, using it will raise a NotImplementedError.

**os.access(path, mode, \*, dir\_fd=None, effective\_ids=False, follow\_symlinks=True)**

Use the real uid/gid to test for access to path. Note that most operations will use the effective uid/gid, therefore this routine can be used in a suid/sgid environment to test if the invoking user has the specified access to path. mode should be F\_OK to test the existence of path, or it can be the inclusive OR of one or more of R\_OK, W\_OK, and X\_OK to test permissions. Return True if access is allowed, False if not. See the Unix man page access(2) for more information.

This function can support specifying paths relative to directory descriptors and not following symlinks.

If effective\_ids is True, access() will perform its access checks using the effective uid/gid instead of the real uid/gid. effective\_ids may not be supported on your platform; you can check whether or not it is available using os.supports\_effective\_ids. If it is unavailable, using it will raise a NotImplementedError.

Availability: Unix, Windows.

Note:

Using access() to check if a user is authorized to e.g. open a file before actually doing so using open() creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. It’s preferable to use EAFP techniques. For example:

if os.access("myfile", os.R\_OK):

with open("myfile") as fp:

return fp.read()

return "some default data"

is better written as:

try:

fp = open("myfile")

except PermissionError:

return "some default data"

else:

with fp:

return fp.read()

Note:

I/O operations may fail even when access() indicates that they would succeed, particularly for operations on network filesystems which may have permissions semantics beyond the usual POSIX permission-bit model.

Changed in version 3.3: Added the dir\_fd, effective\_ids, and follow\_symlinks parameters.

os.F\_OKos.R\_OKos.W\_OKos.X\_OK

Values to pass as the mode parameter of access() to test the existence, readability, writability and executability of path, respectively.

**os.listdir(path='.')**

Return a list containing the names of the entries in the directory given by path. The list is in arbitrary order, and does not include the special entries '.' and '..' even if they are present in the directory.

path may be either of type str or of type bytes. If path is of type bytes, the filenames returned will also be of type bytes; in all other circumstances, they will be of type str.

This function can also support specifying a file descriptor; the file descriptor must refer to a directory.

Note:

To encode str filenames to bytes, use fsencode().

Availability: Unix, Windows.

Changed in version 3.2: The path parameter became optional.

New in version 3.3: Added support for specifying an open file descriptor for path.