betterpath Documentation

Release latest

Corbin Simpson

Sep 27, 2017

Contents

Contents:

CHAPTER 1

FilePath

class bp.filepath.**FilePath**(*path*, *alwaysCreate=False*)

I am a path on the filesystem that only permits "downwards" access.

Instantiate me with a pathname, e.g. FilePath('/home/myuser/public_html'), and I will attempt to only provide access to files which reside inside that path. I may be a path to a file, a directory, or a file which does not exist.

The correct way to use me is to instantiate me, and then do *all* filesystem access through me. In other words, do not import the os module; if you need to open a file, call my *open()* method. If you need to list a directory, call my *listdir()* method.

Even if you pass me a relative path, I will convert that to an absolute path internally.

Note: although time-related methods do return floating-point results, they may still be only second resolution depending on the platform and the last value passed to os.stat_float_times. If you want greater-than-second precision, call os.stat_float_times(True), or use Python 2.5. Greater-than-second precision is only available in Windows on Python 2.5 and later.

On both Python 2 and Python 3, paths can only be bytes.

Variables

- **alwaysCreate** (*bool*) – When opening this file, only succeed if the file does not already exist.
- **path** (*bytes*) – The path from which "downward" traversal is permitted.
- **statinfo** (*os.stat_result*) – The currently cached status information about the file on the filesystem that this FilePath points to. This attribute is C{None} if the file is in an indeterminate state (either this FilePath has not yet had cause to call C{stat()} yet or L{FilePath.changed} indicated that new information is required), 0 if C{stat()} was called and returned an error (i.e. the path did not exist when C{stat()} was called), or a C{stat_result} object that describes the last known status of the underlying file (or directory, as the case may be). Trust me when I tell you that you do not want to use this attribute. Instead, use the methods on FilePath which give you information about it, like C{getsize()}, C{isdir()}, C{getModificationTime()}, and so on.

Warning: Do not use `statinfo`. Trust me when I tell you that you do not want to use this attribute.

__getstate__()
Support serialization by discarding cached `os.stat()` results and returning everything else.

__hash__()
Hash the same as another `FilePath` with the same path as mine.

__init__(*path*, *alwaysCreate=False*)
Convert a path string to an absolute path if necessary and initialize the `FilePath` with the result.

__weakref__
list of weak references to the object (if defined)

basename()
Retrieve the final component of the file path's path (everything after the final path separator).

Returns The final component of the `FilePath`'s path (Everything after the final path separator).

Return type L{bytes}

changed()
Clear any cached information about the state of this path on disk.

child(*path*)
Create and return a new `FilePath` representing a path contained by this path.

Parameters **path** (*bytes*) – The base name of the new `FilePath`. If it contains directory separators or parent references, it will be rejected.

Raises **InsecurePath** – If the result of combining this path with the given path would result in a path which is not a direct child of this path.

Returns The child path

Return type `FilePath`

childSearchPreauth(**paths*)
Return my first existing child with a name in C{paths}.

C{paths} is expected to be a list of *pre-secured* path fragments; in most cases this will be specified by a system administrator and not an arbitrary user.

If no appropriately-named children exist, this will return C{None}.

Returns C{None} or the child path.

Return type L{types.NoneType} or `FilePath`

children(*path*)
List the children of the given path.

Returns an iterable of all currently-existing children of the path.

Return type iterable

chmod(*mode*)
Changes the permissions on self, if possible. Propagates errors from L{os.chmod} up.

Parameters **mode** (*int*) – the new permissions desired (same as the command line chmod)

clonePath
alias of `FilePath`

copyTo(*destination*, *followLinks=True*)
Copies self to destination.

If self doesn't exist, an OSError is raised.

If self is a directory, this method copies its children (but not itself) recursively to destination - if destination does not exist as a directory, this method creates it. If destination is a file, an IOError will be raised.

If self is a file, this method copies it to destination. If destination is a file, this method overwrites it. If destination is a directory, an IOError will be raised.

If self is a link (and followLinks is False), self will be copied over as a new symlink with the same target as returned by os.readlink. That means that if it is absolute, both the old and new symlink will link to the same thing. If it's relative, then perhaps not (and it's also possible that this relative link will be broken).

File/directory permissions and ownership will NOT be copied over.

If followLinks is True, symlinks are followed so that they're treated as their targets. In other words, if self is a link, the link's target will be copied. If destination is a link, self will be copied to the destination's target (the actual destination will be destination's target). Symlinks under self (if self is a directory) will be followed and its target's children be copied recursively.

If followLinks is False, symlinks will be copied over as symlinks.

Parameters

- **destination** (FilePath) – the destination (a FilePath) to which self should be copied
- **followLinks** (*bool*) – whether symlinks in self should be treated as links or as their targets

create()
Exclusively create a file, only if this file previously did not exist.

Returns A file-like object opened from this path.

createDirectory()
Create the directory the FilePath refers to.

@see: L{makedirs}

Raises **OSError** – If the directory cannot be created.

descendant(*path*, *segments*)
Retrieve a child or child's child of the given path.

Parameters **segments** (*iterable*) – A sequence of path segments as L{str} instances.

Returns A L{FilePath} constructed by looking up the C{segments[0]} child of this path, the C{segments[1]} child of that path, and so on.

dirname()
Retrieve all of the components of the FilePath's path except the last one (everything up to the final path separator).

Returns All of the components of the FilePath's path except the last one (everything up to the final path separator).

Return type L{bytes}

exists()
Check if this FilePath exists.

Returns Whether this path definitely exists.

Return type bool

getAccessTime()
Retrieve the time that this file was last accessed.

Returns a number of seconds from the epoch.

Return type float

getContent(*path*)
Retrieve the data from a given file path.

getDevice()
Retrieves the device containing the file. The inode number and device number together uniquely identify the file, but the device number is not necessarily consistent across reboots or system crashes.

Raises **NotImplementedError** – if the platform is Windows, since the device number would be 0 for all partitions on a Windows platform

Returns a number representing the device

Return type int

getGroupID()
Returns the group ID of the file.

Raises **NotImplementedError** – if the platform is Windows, since the GID is always 0 on windows

Returns the group ID of the file

Return type int

getInodeNumber()
Retrieve the file serial number, also called inode number, which distinguishes this file from all other files on the same device.

Raises **NotImplementedError** – if the platform is Windows, since the inode number would be a dummy value for all files in Windows

Returns a number representing the file serial number

Return type int

getModificationTime()
Retrieve the time of last access from this file.

Returns a number of seconds from the epoch.

Return type float

getNumberOfHardLinks()
Retrieves the number of hard links to the file.

This count keeps track of how many directories have entries for this file. If the count is ever decremented to zero then the file itself is discarded as soon as no process still holds it open. Symbolic links are not counted in the total.

Raises **NotImplementedError** – if the platform is Windows, since Windows doesn't maintain a link count for directories, and `os.stat()` does not set C{st_nlink} on Windows anyway.

Returns the number of hard links to the file

Return type int

getPermissions()
Returns the permissions of the file. Should also work on Windows; however, those permissions may not be what is expected in Windows.

Returns the permissions for the file

Return type Permissions

getStatusChangeTime()
Retrieve the time of the last status change for this file.

Returns a number of seconds from the epoch.

Return type float

getUserID()
Returns the user ID of the file's owner.

Raises **NotImplementedError** – if the platform is Windows, since the UID is always 0 on Windows

Returns the user ID of the file's owner

Return type L{int}

getsize()
Retrieve the size of this file in bytes.

Returns The size of the file at this file path in bytes.

Raises **Exception** – if the size cannot be obtained.

Return type int

globChildren(*pattern*)
Assuming I am representing a directory, return a list of FilePaths representing my children that match the given pattern.

@param pattern: A glob pattern to use to match child paths. @type pattern: L{bytes}

Returns A L{list} of matching children.

Return type L{list}

isBlockDevice()
Returns whether the underlying path is a block device.

Returns C{True} if it is a block device, C{False} otherwise

Return type L{bool}

isSocket()
Returns whether the underlying path is a socket.

Returns C{True} if it is a socket, C{False} otherwise

Return type L{bool}

isabs()
Check if this FilePath refers to an absolute path.

Deprecated since version 0.2: This method always returns True. To replace this method, simply replace its usage in code with True and then simplify as needed.

Returns True

Return type bool

isdir()
Check if this FilePath refers to a directory.

Returns Whether this FilePath refers to a directory

Return type bool

isfile()
Check if this file path refers to a regular file.

Returns C{True} if this FilePath points to a regular file (not a directory, socket, named pipe, etc), C{False} otherwise.

Return type L{bool}

islink()
Check if this FilePath points to a symbolic link.

Returns C{True} if this FilePath points to a symbolic link, C{False} otherwise.

Return type L{bool}

linkTo(*linkFilePath*)
Creates a symlink to self to at the path in the FilePath C{linkFilePath}.

Only works on posix systems due to its dependence on L{os.symlink}. Propagates L{OSError}s up from L{os.symlink} if C{linkFilePath.parent()} does not exist, or C{linkFilePath} already exists.

Parameters **linkFilePath** (FilePath) – the link to be created.

listdir()
List the base names of the direct children of this FilePath.

Returns A L{list} of L{bytes} giving the names of the contents of the directory this FilePath refers to. These names are relative to this FilePath.

Return type L{list}

Raises

- **OSError** – If an error occurs while listing the directory. If the error is 'serious', meaning that the operation failed due to an access violation, exhaustion of some kind of resource (file descriptors or memory), OSError or a platform-specific variant will be raised.
- **UnlistableError** – If the inability to list the directory is due to this path not existing or not being a directory, the more specific OSError subclass L{UnlistableError} is raised instead.

Raise Anything the platform L{os.listdir} implementation might raise (typically L{OSError}).

makedirs()
Create all directories not yet existing in C{path} segments, using L{os.makedirs}.

Returns C{None}

moveTo(*destination*, *followLinks=True*)
Move self to destination - basically renaming self to whatever destination is named.

If destination is an already-existing directory, moves all children to destination if destination is empty. If destination is a non-empty directory, or destination is a file, an OSError will be raised.

If moving between filesystems, self needs to be copied, and everything that applies to copyTo applies to moveTo.

@param destination: the destination (a FilePath) to which self should be copied

@param followLinks: whether symlinks in self should be treated as links or as their targets (only applicable when moving between filesystems)

open(*mode='r'*)
Open this file using C{mode} or for writing if C{alwaysCreate} is C{True}.

In all cases the file is opened in binary mode, so it is not necessary to include C{"b"} in C{mode}.

Parameters **mode** (*str*) – The mode to open the file in. Default is C{"r"}.

Raises **AssertionError** – If C{"a"} is included in the mode and C{alwaysCreate} is C{True}.

Return type L{file}

Returns An open L{file} object.

parent()
A file path for the directory containing the file at this file path.

Returns A `FilePath` representing the path which directly contains this `FilePath`.

Return type `FilePath`

parents(*path*)
Retrieve an iterator of all the ancestors of the given path.

Returns An iterator of all the ancestors of the given path, from the most recent (its immediate parent) to the root of its filesystem.

Return type iterator

preauthChild(*path*)
Use me if C{path} might have slashes in it, but you know they're safe.

Parameters **path** (*bytes*) – A relative path (ie, a path not starting with C{"/"}) which will be interpreted as a child or descendant of this path.

Returns The child path.

Return type `FilePath`

realpath()
Returns the absolute target as a `FilePath` if self is a link, self otherwise.

The absolute link is the ultimate file or directory the link refers to (for instance, if the link refers to another link, and another...). If the filesystem does not support symlinks, or if the link is cyclical, raises a `LinkError`.

Returns `FilePath` of the target path.

Return type `FilePath`

Raises **LinkError** – if links are not supported or links are cyclical.

remove()
Removes the file or directory that is represented by self. If C{self.path} is a directory, recursively remove all its children before removing the directory. If it's a file or link, just delete it.

requireCreate(*val=True*)
Sets the C{alwaysCreate} variable.

Parameters **val** (*bool*) – C{True} or C{False}, indicating whether opening this path will be required to create the file or not.

restat (*reraise=True*)

Re-calculate cached effects of 'stat'. To refresh information on this path after you know the filesystem may have changed, call this method.

Parameters **reraise** (*bool*) – If true, re-raise exceptions from `os.stat()`; otherwise, mark this path as not existing, and remove any cached stat information.

Raises **Exception** – If C{reraise} is C{True} and an exception occurs while reloading metadata.

Note: Please do not use this method.

Deprecated since version 0.2.

segmentsFrom (*path*, *ancestor*)

Return a list of segments between a child and its ancestor.

For example, in the case of a path X representing /a/b/c/d and a path Y representing /a/b, C{Y.segmentsFrom(X)} will return C{['c', 'd']}.

Parameters **ancestor** – an instance of the same class as self, ostensibly an ancestor of self.

Raises **ValueError** – When the 'ancestor' parameter is not actually an ancestor, i.e. a path for /x/y/z is passed as an ancestor for /a/b/c/d.

Returns a list of segments

Return type list

setContent (*content*, *ext='.new'*)

Replace the file at this path with a new file that contains the given bytes, trying to avoid data-loss in the meanwhile.

On UNIX-like platforms, this method does its best to ensure that by the time this method returns, either the old contents *or* the new contents of the file will be present at this path for subsequent readers regardless of premature device removal, program crash, or power loss, making the following assumptions:

•your filesystem is journaled (i.e. your filesystem will not I{itself} lose data due to power loss)

•your filesystem's C{rename()} is atomic

•your filesystem will not discard new data while preserving new metadata (see U{http://mjg59.livejournal.com/108257.html} for more detail)

On most versions of Windows there is no atomic C{rename()} (see U{http://bit.ly/win32-overwrite} for more information), so this method is slightly less helpful. There is a small window where the file at this path may be deleted before the new file is moved to replace it: however, the new file will be fully written and flushed beforehand so in the unlikely event that there is a crash at that point, it should be possible for the user to manually recover the new version of their data. In the future, Twisted will support atomic file moves on those versions of Windows which *do* support them: see U{Twisted ticket 3004<http://twistedmatrix.com/trac/ticket/3004>}.

This method should be safe for use by multiple concurrent processes, but note that it is not easy to predict which process's contents will ultimately end up on disk if they invoke this method at close to the same time.

Parameters

- **content** (*bytes*) – The desired contents of the file at this path.

- **ext** (*bytes*) – An extension to append to the temporary filename used to store the bytes while they are being written. This can be used to make sure that temporary files can be identified by their suffix, for cleanup in case of crashes.

sibling(*path*, *segment*)
Return an L{IFilePath} with the same directory as the given path, but with a basename of C{segment}.

Parameters **segment** (*str*) – The basename of the L{IFilePath} to return.

Returns The sibling path.

Return type L{IFilePath}

siblingExtension(*ext*)
Attempt to return a path with my name, given the extension at C{ext}.

Parameters **ext** (*str*) – File-extension to search for.

Returns The sibling path.

Return type FilePath

siblingExtensionSearch(**exts*)
Attempt to return a path with my name, given multiple possible extensions.

Each extension in C{exts} will be tested and the first path which exists will be returned. If no path exists, C{None} will be returned. If C{''} is in C{exts}, then if the file referred to by this path exists, C{self} will be returned.

The extension '*' has a magic meaning, which means "any path that begins with C{self.path + '.'} is acceptable".

splitext()
Split the file path into a pair C{(root, ext)} such that C{root + ext == path}.

Returns Tuple where the first item is the filename and second item is the file extension. See Python docs for L{os.path.splitext}.

Return type L{tuple}

temporarySibling(*extension=''*)
Construct a path referring to a sibling of this path.

The resulting path will be unpredictable, so that other subprocesses should neither accidentally attempt to refer to the same path before it is created, nor they should other processes be able to guess its name in advance.

Parameters **extension** (*bytes*) – A suffix to append to the created filename. (Note that if you want an extension with a '.' you must include the '.' yourself.)

Returns A FilePath with the given extension suffix and with C{alwaysCreate} set to True.

Return type FilePath

touch()
Updates the access and last modification times of the file at this file path to the current time. Also creates the file if it does not already exist.

@raise Exception: if unable to create or modify the last modification time of the file.

walk(*path*, *descend=None*)
Yield a path, then each of its children, and each of those children's children in turn.

Parameters **descend** (*callable*) – A one-argument callable that will return True for FilePaths that should be traversed and False otherwise. It will be called with each path

for which *isdir()* returns True. If omitted, all directories will be traversed, including symbolic links.

Raises **LinkError** – A cycle of symbolic links was found

Returns a generator yielding FilePath-like objects

Return type generator

CHAPTER 2

MemoryPath

class bp.memory.**MemoryPath**(*fs*, *path=()*)

An IFilePath which shows a view into a MemoryFS.

children(*path*)

List the children of the given path.

Returns an iterable of all currently-existing children of the path.

Return type iterable

listdir()

Pretend that we are a directory and get a listing of child names.

parents(*path*)

Retrieve an iterator of all the ancestors of the given path.

Returns An iterator of all the ancestors of the given path, from the most recent (its immediate parent) to the root of its filesystem.

Return type iterator

segmentsFrom(*path*, *ancestor*)

Return a list of segments between a child and its ancestor.

For example, in the case of a path X representing /a/b/c/d and a path Y representing /a/b, C{Y.segmentsFrom(X)} will return C{['c', 'd']}.

Parameters **ancestor** – an instance of the same class as self, ostensibly an ancestor of self.

Raises **ValueError** – When the 'ancestor' parameter is not actually an ancestor, i.e. a path for /x/y/z is passed as an ancestor for /a/b/c/d.

Returns a list of segments

Return type list

sibling(*path*, *segment*)

Return an L{IFilePath} with the same directory as the given path, but with a basename of C{segment}.

Parameters **segment** (*str*) – The basename of the L{IFilePath} to return.

Returns The sibling path.

Return type L{IFilePath}

walk(*path*, *descend=None*)

Yield a path, then each of its children, and each of those children's children in turn.

Parameters **descend** (*callable*) – A one-argument callable that will return True for FilePaths that should be traversed and False otherwise. It will be called with each path for which *isdir()* returns True. If omitted, all directories will be traversed, including symbolic links.

Raises **LinkError** – A cycle of symbolic links was found

Returns a generator yielding FilePath-like objects

Return type generator

CHAPTER 3

ReadOnlyPath

class bp.readonly.**ReadOnlyPath**(*fp*)

An IFilePath which is intrinsically read-only in every aspect.

children(*path*)

List the children of the given path.

Returns an iterable of all currently-existing children of the path.

Return type iterable

descendant(*path*, *segments*)

Retrieve a child or child's child of the given path.

Parameters **segments** (*iterable*) – A sequence of path segments as L{str} instances.

Returns A L{FilePath} constructed by looking up the C{segments[0]} child of this path, the C{segments[1]} child of that path, and so on.

parents(*path*)

Retrieve an iterator of all the ancestors of the given path.

Returns An iterator of all the ancestors of the given path, from the most recent (its immediate parent) to the root of its filesystem.

Return type iterator

segmentsFrom(*path*, *ancestor*)

Return a list of segments between a child and its ancestor.

For example, in the case of a path X representing /a/b/c/d and a path Y representing /a/b, C{Y.segmentsFrom(X)} will return C{['c', 'd']}.

Parameters **ancestor** – an instance of the same class as self, ostensibly an ancestor of self.

Raises **ValueError** – When the 'ancestor' parameter is not actually an ancestor, i.e. a path for /x/y/z is passed as an ancestor for /a/b/c/d.

Returns a list of segments

Return type list

sibling (*path*, *segment*)
Return an L{IFilePath} with the same directory as the given path, but with a basename of C{segment}.

Parameters **segment** (*str*) – The basename of the L{IFilePath} to return.

Returns The sibling path.

Return type L{IFilePath}

walk (*path*, *descend=None*)
Yield a path, then each of its children, and each of those children's children in turn.

Parameters **descend** (*callable*) – A one-argument callable that will return True for FilePaths that should be traversed and False otherwise. It will be called with each path for which *isdir()* returns True. If omitted, all directories will be traversed, including symbolic links.

Raises **LinkError** – A cycle of symbolic links was found

Returns a generator yielding FilePath-like objects

Return type generator

CHAPTER 4

ZipPath

class bp.zippath.**ZipPath**(*archive*, *pathInArchive*)

I am a file or directory contained within a zip file.

__init__(*archive*, *pathInArchive*)

Don't construct me directly. Use ZipArchive.child().

Parameters

- **archive** (*ZipArchive*) – a ZipArchive instance.
- **pathInArchive** (*str*) – a ZIP_PATH_SEP-separated string.

__weakref__

list of weak references to the object (if defined)

child(*path*)

Return a new ZipPath representing a path in C{self.archive} which is a child of this path.

Note: Requesting the C{".."} (or other special name) child will **not** cause L{InsecurePath} to be raised since these names do not have any special meaning inside a zip archive. Be particularly careful with the C{path} attribute (if you absolutely must use it) as this means it may include special names with special meaning outside of the context of a zip archive.

---

children(*path*)

List the children of the given path.

Returns an iterable of all currently-existing children of the path.

Return type iterable

descendant(*path*, *segments*)

Retrieve a child or child's child of the given path.

Parameters **segments** (*iterable*) – A sequence of path segments as L{str} instances.

Returns A L{FilePath} constructed by looking up the C{segments[0]} child of this path, the C{segments[1]} child of that path, and so on.

getAccessTime()
Retrieve this file's last access-time.

This is the same as the last access time for the archive.

Returns a number of seconds since the epoch

Return type int

getModificationTime()
Retrieve this file's last modification time.

This is the time of modification recorded in the zipfile.

Returns a number of seconds since the epoch.

Return type int

getStatusChangeTime()
Retrieve this file's last modification time.

This is the time of modification recorded in the zipfile.

Returns a number of seconds since the epoch.

Return type int

getsize()
Retrieve this file's size.

@return: file size, in bytes

parents(*path*)
Retrieve an iterator of all the ancestors of the given path.

Returns An iterator of all the ancestors of the given path, from the most recent (its immediate parent) to the root of its filesystem.

Return type iterator

segmentsFrom(*path*, *ancestor*)
Return a list of segments between a child and its ancestor.

For example, in the case of a path X representing /a/b/c/d and a path Y representing /a/b, C{Y.segmentsFrom(X)} will return C{['c', 'd']}.

Parameters **ancestor** – an instance of the same class as self, ostensibly an ancestor of self.

Raises **ValueError** – When the 'ancestor' parameter is not actually an ancestor, i.e. a path for /x/y/z is passed as an ancestor for /a/b/c/d.

Returns a list of segments

Return type list

splitext()
Return a value similar to that returned by `os.path.splitext()`.

walk(*path*, *descend=None*)
Yield a path, then each of its children, and each of those children's children in turn.

Parameters **descend** (*callable*) – A one-argument callable that will return True for FilePaths that should be traversed and False otherwise. It will be called with each path for which *isdir()* returns True. If omitted, all directories will be traversed, including symbolic links.

Raises **LinkError** – A cycle of symbolic links was found

Returns a generator yielding FilePath-like objects

Return type generator

CHAPTER 5

Generic Helpers

bp.generic.**genericChildren**(*path*)
List the children of the given path.

Returns an iterable of all currently-existing children of the path.

Return type iterable

bp.generic.**genericDescendant**(*path*, *segments*)
Retrieve a child or child's child of the given path.

Parameters **segments** (*iterable*) – A sequence of path segments as L{str} instances.

Returns A L{FilePath} constructed by looking up the C{segments[0]} child of this path, the C{segments[1]} child of that path, and so on.

bp.generic.**genericGetContent**(*path*)
Retrieve the data from a given file path.

bp.generic.**genericParents**(*path*)
Retrieve an iterator of all the ancestors of the given path.

Returns An iterator of all the ancestors of the given path, from the most recent (its immediate parent) to the root of its filesystem.

Return type iterator

bp.generic.**genericSegmentsFrom**(*path*, *ancestor*)
Return a list of segments between a child and its ancestor.

For example, in the case of a path X representing /a/b/c/d and a path Y representing /a/b, C{Y.segmentsFrom(X)} will return C{['c', 'd']}.

Parameters **ancestor** – an instance of the same class as self, ostensibly an ancestor of self.

Raises **ValueError** – When the 'ancestor' parameter is not actually an ancestor, i.e. a path for /x/y/z is passed as an ancestor for /a/b/c/d.

Returns a list of segments

Return type list

bp.generic.**genericSibling**(*path*, *segment*)
Return an L{IFilePath} with the same directory as the given path, but with a basename of C{segment}.

Parameters **segment** (*str*) – The basename of the L{IFilePath} to return.

Returns The sibling path.

Return type L{IFilePath}

bp.generic.**genericWalk**(*path*, *descend=None*)
Yield a path, then each of its children, and each of those children's children in turn.

Parameters **descend** (*callable*) – A one-argument callable that will return True for FilePaths that should be traversed and False otherwise. It will be called with each path for which *isdir()* returns True. If omitted, all directories will be traversed, including symbolic links.

Raises **LinkError** – A cycle of symbolic links was found

Returns a generator yielding FilePath-like objects

Return type generator

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

Symbols

B

C

D

E

F

G

I

L

M

O

P

R

S

T

W

Z