Commit

gh-123270: Replaced SanitizedNames with a more surgical fix. (#123354)

Browse files

Applies changes from zipp 3.20.1 and jaraco/zipp#124

main (#123354)

jaraco committed last week Verified 1 parent 7e38e67 commit 2231286

Showing **3 changed files** with **87 additions** and **71 deletions**.

Whitesp... Ignore whitesp... S... Uni...

Filter changed files

- Lib
  - test/test_zipfile/_path
    - test_path.py
  - zipfile/_path
    - __init__.py
- Misc/NEWS.d/next/Library
  - 2024-08-26-13-45-2...

73 Lib/test/test_zipfile/_path/test_path.py

```
 5   5    import pickle
 6   6    import stat
 7   7    import sys
     8  + import time
 8   9    import unittest
 9  10    import zipfile
10  11    import zipfile._path
592 593
593 594        def test_malformed_paths(self):
594 595            """
595     -          Path should handle malformed paths.
        596 +      Path should handle malformed paths gracefully.
        597 +
        598 +      Paths with leading slashes are not visible.
        599 +
        600 +      Paths with dots are treated like regular files.
596 601            """
597 602            data = io.BytesIO()
598 603            zf = zipfile.ZipFile(data, "w")
601 606            zf.writestr("../parent.txt", b"content")
602 607            zf.filename = ''
603 608            root = zipfile.Path(zf)
604     -          assert list(map(str, root.iterdir())) == [
605     -              'one-slash.txt',
606     -              'two-slash.txt',
607     -              'parent.txt',
608     -          ]
```

```
+        assert list(map(str, root.iterdir())) == ['../']
+        assert root.joinpath('..').joinpath('parent.txt').read_bytes() == b'content'
+
+    def test_unsupported_names(self):
+        """
+        Path segments with special characters are readable.
+
+        On some platforms or file systems, characters like
+        ``:`` and ``?`` are not allowed, but they are valid
+        in the zip file.
+        """
+        data = io.BytesIO()
+        zf = zipfile.ZipFile(data, "w")
+        zf.writestr("path?", b"content")
+        zf.writestr("V: NMS.flac", b"fLaC...")
+        zf.filename = ''
+        root = zipfile.Path(zf)
+        contents = root.iterdir()
+        assert next(contents).name == 'path?'
+        assert next(contents).name == 'V: NMS.flac'
+        assert root.joinpath('V: NMS.flac').read_bytes() == b"fLaC..."
+
+    def test_backslash_not_separator(self):
+        """
+        In a zip file, backslashes are not separators.
+        """
+        data = io.BytesIO()
+        zf = zipfile.ZipFile(data, "w")
+        zf.writestr(DirtyZipInfo.for_name("foo\\bar", zf), b"content")
+        zf.filename = ''
+        root = zipfile.Path(zf)
+        (first,) = root.iterdir()
+        assert not first.is_dir()
+        assert first.name == 'foo\\bar'
```

```
    @pass_alpharep
    def test_interface(self, alpharep):
        from importlib.resources.abc import Traversable

        zf = zipfile.Path(alpharep)
        assert isinstance(zf, Traversable)
```

```
+
+
+ class DirtyZipInfo(zipfile.ZipInfo):
```

```
+    """
+    Bypass name sanitization.
+    """
+
+    def __init__(self, filename, *args,
**kwargs):
+        super().__init__(filename, *args,
**kwargs)
+        self.filename = filename
+
+    @classmethod
+    def for_name(cls, name, archive):
+        """
+        Construct the same way that
ZipFile.writestr does.
+
+        TODO: extract this functionality and
re-use
+        """
+        self = cls(filename=name,
date_time=time.localtime(time.time())[:6])
+        self.compress_type =
archive.compression
+        self.compress_level =
archive.compresslevel
+        if self.filename.endswith('/'):  #
pragma: no cover
+            self.external_attr = 0o40775 << 16
# drwxrwxr-x
+            self.external_attr |= 0x10  # MS-
DOS directory flag
+        else:
+            self.external_attr = 0o600 << 16  #
?rw-------
+        return self
```

82 Lib/zipfile/_path/__init__.py

```
@@ -1,3 +1,12 @@
+"""
+A Path-like interface for zipfiles.
+
+This codebase is shared between zipfile.Path in
the stdlib
+and zipp in PyPI. See
+https://github.com/python/importlib_metadata/wi
ki/Development-Methodology
+for more detail.
+"""
+
 import io
 import posixpath
 import zipfile
 def _ancestry(path):
     """
     Given a path with elements separated by
```

```
-     posixpath.sep, generate all elements of that path
+     posixpath.sep, generate all elements of that path.

      >>> list(_ancestry('b/d'))
      ['b/d', 'b']
      ['b']
      >>> list(_ancestry(''))
      []
+
+     Multiple separators are treated like a single.
+
+     >>> list(_ancestry('//b//d///f//'))
+     ['//b//d///f', '//b//d', '//b']
      """
      path = path.rstrip(posixpath.sep)
-     while path and path != posixpath.sep:
+     while path.rstrip(posixpath.sep):
          yield path
          path, tail = posixpath.split(path)

          super().__init__(*args, **kwargs)


- class SanitizedNames:
-     """
-     ZipFile mix-in to ensure names are sanitized.
-     """
-
-     def namelist(self):
-         return list(map(self._sanitize, super().namelist()))
-
-     @staticmethod
-     def _sanitize(name):
-         r"""
-         Ensure a relative path with posix separators and no dot names.
-
-         Modeled after
-         https://github.com/python/cpython/blob/bcc1be39cb1d04ad9fc0bd1b9193d3972835a57c/Lib/zipfile/__init__.py#L1799-L1813
-         but provides consistent cross-platform behavior.
-
-         >>> san = SanitizedNames._sanitize
-         >>> san('/foo/bar')
-         'foo/bar'
-         >>> san('//foo.txt')
-         'foo.txt'
-         >>> san('foo/.././bar.txt')
```

```
-         'foo/bar.txt'
-         >>> san('foo../.bar.txt')
-         'foo../.bar.txt'
-         >>> san('\\foo\\bar.txt')
-         'foo/bar.txt'
-         >>> san('D:\\foo.txt')
-         'D/foo.txt'
-         >>> san('\\\\server\\share\\file.txt')
-         'server/share/file.txt'
-         >>> san('\\\\?\\GLOBALROOT\\Volume3')
-         '?/GLOBALROOT/Volume3'
-         >>> san('\\\\.\\PhysicalDrive1\\root')
-         'PhysicalDrive1/root'
-
-         Retain any trailing slash.
-         >>> san('abc/')
-         'abc/'
-
-         Raises a ValueError if the result is empty.
-         >>> san('../..')
-         Traceback (most recent call last):
-         ...
-         ValueError: Empty filename
-         """
-
-         def allowed(part):
-             return part and part not in {'..', '.'}
-
-         # Remove the drive letter.
-         # Don't use ntpath.splitdrive, because that also strips UNC paths
-         bare = re.sub('^([A-Z]):', r'\1', name, flags=re.IGNORECASE)
-         clean = bare.replace('\\', '/')
-         parts = clean.split('/')
-         joined = '/'.join(filter(allowed, parts))
-         if not joined:
-             raise ValueError("Empty filename")
-         return joined + '/' * name.endswith('/')
-
-
- class CompleteDirs(InitializedState, SanitizedNames, zipfile.ZipFile):
+ class CompleteDirs(InitializedState, zipfile.ZipFile):
      """
      A ZipFile subclass that ensures that implied directories
      are always included in the namelist.
```

3

Misc/NEWS.d/next/Library/2024-08-26-13-45-20.gh-issue-1232…

```
...  ...  @@ -0,0 +1,3 @@
     1  + Applied a more surgical fix for malformed payloads
          in :class:`zipfile.Path`
     2  + causing infinite loops (gh-122905) without
          breaking contents using
     3  + legitimate characters.
```

0 comments on commit `2231286`

Please sign in to comment.