zipimport — Load Python Code from ZIP Archives

Purpose: Import Python modules saved as members of ZIP archives.

The zipimport module implements the zipimporter class, which can be used to find and load Python modules inside ZIP archives. The zipimporter supports the import hooks API specified in PEP 302; this is how Python Eggs work.

It is not usually necessary to use the zipimport module directly, since it is possible to import directly from a ZIP archive as long as that archive appears in sys.path. However, it is instructive to study how the importer API can be used, to learn the features available and understand how module importing works. Knowing how the ZIP importer works will also help debug issues that may come up when distributing applications packaged as ZIP archives created with zipfile.PyZipFile.

Example

These examples reuse some of the code from the discussion of zipfile to create an example ZIP archive containing a few Python modules.

```
# zipimport_make_example.py

import sys
import zipfile

if __name__ == '__main__':
    zf = zipfile.PyZipFile('zipimport_example.zip', mode='w')
    try:
        zf.writepy('.')
        zf.write('zipimport_get_source.py')
        zf.write('example_package/README.txt')
    finally:
        zf.close()
    for name in zf.namelist():
        print(name)
```

Run zipimport_make_example.py before any of the rest of the examples to create a ZIP archive containing all of the modules in the example directory, along with some test data needed for the examples in this section.

```
$ python3 zipimport_make_example.py

__init__.pyc
example_package/__init__.pyc
zipimport_find_module.pyc
zipimport_get_code.pyc
zipimport_get_data.pyc
zipimport_get_data_nozip.pyc
zipimport_get_data_zip.pyc
zipimport_get_source.pyc
zipimport_is_package.pyc
zipimport_load_module.pyc
zipimport_make_example.pyc
zipimport_get_source.py
example_package/README.txt
```

Finding a Module

Given the full name of a module, find_module() will try to locate that module inside the ZIP archive.

```
# zipimport_find_module.py

import zipimport

importer = zipimport.zipimporter('zipimport_example.zip')
```

```
for module_name in ['zipimport_find_module', 'not_there']:
    print(module_name, ':', importer.find_module(module_name))
```

If the module is found, the `zipimporter` instance is returned. Otherwise, None is returned.

```
$ python3 zipimport_find_module.py

zipimport_find_module : <zipimporter object
"zipimport_example.zip">
not_there : None
```

Accessing Code

The `get_code()` method loads the code object for a module from the archive.

```
# zipimport_get_code.py

import zipimport

importer = zipimport.zipimporter('zipimport_example.zip')
code = importer.get_code('zipimport_get_code')
print(code)
```

The code object is not the same as a `module` object, but is used to create one.

```
$ python3 zipimport_get_code.py

<code object <module> at 0x1012b4ae0, file
"./zipimport_get_code.py", line 6>
```

To load the code as a usable module, use `load_module()` instead.

```
# zipimport_load_module.py

import zipimport

importer = zipimport.zipimporter('zipimport_example.zip')
module = importer.load_module('zipimport_get_code')
print('Name   :', module.__name__)
print('Loader :', module.__loader__)
print('Code   :', module.code)
```

The result is a module object configured as though the code had been loaded from a regular import.

```
$ python3 zipimport_load_module.py

<code object <module> at 0x1007b4c00, file
"./zipimport_get_code.py", line 6>
Name   : zipimport_get_code
Loader : <zipimporter object "zipimport_example.zip">
Code   : <code object <module> at 0x1007b4c00, file
"./zipimport_get_code.py", line 6>
```

Source

As with the inspect module, it is possible to retrieve the source code for a module from the ZIP archive, if the archive includes the source. In the case of the example, only `zipimport_get_source.py` is added to `zipimport_example.zip` (the rest of the modules are just added as the `.pyc` files).

```
# zipimport_get_source.py

import zipimport

modules = [
    'zipimport_get_code',
    'zipimport_get_source',
]

importer = zipimport.zipimporter('zipimport_example.zip')
for module_name in modules:
    source = importer.get_source(module_name)
    print('=' * 80)
    print(module_name)
    print('=' * 80)
    print(source)
    print()
```

If the source for a module is not available, `get_source()` returns None.

```
$ python3 zipimport_get_source.py

================================================================
zipimport_get_code
================================================================
None

================================================================
zipimport_get_source
================================================================
#!/usr/bin/env python3
#
# Copyright 2007 Doug Hellmann.
#
"""Retrieving the source code for a module within a zip archive.
"""

#end_pymotw_header
import zipimport

modules = [
    'zipimport_get_code',
    'zipimport_get_source',
]

importer = zipimport.zipimporter('zipimport_example.zip')
for module_name in modules:
    source = importer.get_source(module_name)
    print('=' * 80)
    print(module_name)
    print('=' * 80)
    print(source)
    print()
```

Packages

To determine if a name refers to a package instead of a regular module, use `is_package()`.

```
# zipimport_is_package.py

import zipimport

importer = zipimport.zipimporter('zipimport_example.zip')
for name in ['zipimport_is_package', 'example_package']:
    print(name, importer.is_package(name))
```

In this case, `zipimport_is_package` came from a module and the `example_package` is a package.

```
$ python3 zipimport_is_package.py

zipimport_is_package False
example_package True
```

Data

There are times when source modules or packages need to be distributed with non-code data. Images, configuration files, default data, and test fixtures are just a few examples of this. Frequently, the module `__path__` or `__file__` attributes are used to find these data files relative to where the code is installed.

For example, with a "normal" module, the file system path can be constructed from the `__file__` attribute of the imported package like this:

```python
# zipimport_get_data_nozip.py

import os
import example_package

# Find the directory containing the imported
# package and build the data filename from it.
pkg_dir = os.path.dirname(example_package.__file__)
data_filename = os.path.join(pkg_dir, 'README.txt')

# Read the file and show its contents.
print(data_filename, ':')
print(open(data_filename, 'r').read())
```

The output will depend on where the sample code is located on the file system.

```
$ python3 zipimport_get_data_nozip.py

.../example_package/README.txt :
This file represents sample data which could be embedded in the
ZIP archive.  You could include a configuration file, images, or
any other sort of noncode data.
```

If the example_package is imported from the ZIP archive instead of the file system, using `__file__` does not work.

```python
# zipimport_get_data_zip.py

import sys
sys.path.insert(0, 'zipimport_example.zip')

import os
import example_package
print(example_package.__file__)
data_filename = os.path.join(
    os.path.dirname(example_package.__file__),
    'README.txt',
)
print(data_filename, ':')
print(open(data_filename, 'rt').read())
```

The `__file__` of the package refers to the ZIP archive, and not a directory, so building up the path to the `README.txt` file gives the wrong value.

```
$ python3 zipimport_get_data_zip.py

zipimport_example.zip/example_package/__init__.pyc
zipimport_example.zip/example_package/README.txt :
Traceback (most recent call last):
  File "zipimport_get_data_zip.py", line 20, in <module>
    print(open(data_filename, 'rt').read())
NotADirectoryError: [Errno 20] Not a directory:
'zipimport_example.zip/example_package/README.txt'
```

A more reliable way to retrieve the file is to use the `get_data()` method. The `zipimporter` instance that loaded the module can be accessed through the `__loader__` attribute of the imported module:

```python
# zipimport_get_data.py

import sys
sys.path.insert(0, 'zipimport_example.zip')

```

```
import os
import example_package
print(example_package.__file__)
data = example_package.__loader__.get_data(
    'example_package/README.txt')
print(data.decode('utf-8'))
```

pkgutil.get_data() uses this interface to access data from within a package. The value returned is a byte string, which needs to be decoded to a unicode string before printing.

```
$ python3 zipimport_get_data.py

zipimport_example.zip/example_package/__init__.pyc
This file represents sample data which could be embedded in the
ZIP archive.  You could include a configuration file, images, or
any other sort of noncode data.
```

The `__loader__` is not set for modules not imported via `zipimport`.

See also

- Standard library documentation for zipimport
- Python 2 to 3 porting notes for zipimport
- imp – Other import-related functions.
- pkgutil – Provides a more generic interface to `get_data()`.
- zipfile – Read and write ZIP archive files.
- **PEP 302** – New Import Hooks

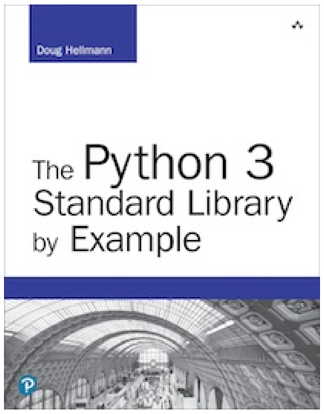
pkgutil — Package Utilities

Unix-specific Services

This page was last updated 2016-12-31.

Navigation

pkgutil — Package Utilities
Unix-specific Services



The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?