`!zipimport` --- Import modules from Zip archives

- zipimporter Objects
- Examples

zipimport

Source code: `Lib/zipimport.py`

This module adds the ability to import Python modules (`\*.py`, `\*.pyc`) and packages from ZIP-format archives. It is usually not needed to use the `!zipimport` module explicitly; it is automatically used by the built-in `import` mechanism for `sys.path` items that are paths to ZIP archives.

Typically, `sys.path` is a list of directory names as strings. This module also allows an item of `sys.path` to be a string naming a ZIP file archive. The ZIP archive can contain a subdirectory structure to support package imports, and a path within the archive can be specified to only import from a subdirectory. For example, the path `example.zip/lib/` would only import from the `lib/` subdirectory within the archive.

Any files may be present in the ZIP archive, but importers are only invoked for `.py` and `.pyc` files. ZIP import of dynamic modules (`.pyd`, `.so`) is disallowed. Note that if an archive only contains `.py` files, Python will not attempt to modify the archive by adding the corresponding `.pyc` file, meaning that if a ZIP archive doesn't contain `.pyc` files, importing may be rather slow.

3.15 Zstandard (*zstd*) compressed zip file entries are supported.

3.13 ZIP64 is supported

3.8 Previously, ZIP archives with an archive comment were not supported.

PKZIP Application Note
Documentation on the ZIP file format by Phil Katz, the creator of the format and algorithms used.

`273` - Import Modules from Zip Archives
Written by James C. Ahlstrom, who also provided an implementation. Python 2.3 follows the specification in `273`, but uses an implementation written by Just van Rossum that uses the import hooks described in `302`.

`importlib` - The implementation of the import machinery
Package providing the relevant protocols for all importers to implement.

This module defines an exception:

ZipImportError

Exception raised by zipimporter objects. It's a subclass of `ImportError`, so it can be caught as `ImportError`, too.

zipimporter Objects

`zipimporter` is the class for importing ZIP files.

Create a new zipimporter instance. *archivepath* must be a path to a ZIP file, or to a specific path within a ZIP file. For example, an *archivepath* of `foo/bar.zip/lib` will look for modules in the `lib` directory inside the ZIP file `foo/bar.zip` (provided that it exists).

`ZipImportError` is raised if *archivepath* doesn't point to a valid ZIP archive.

3.12

Methods `find_loader()` and `find_module()`, deprecated in 3.10 are now removed. Use `find_spec` instead.

create_module(spec)

Implementation of `importlib.abc.Loader.create_module` that returns `None` to explicitly request the default semantics.

3.10

exec_module(module)

Implementation of `importlib.abc.Loader.exec_module`.

3.10

find_spec(fullname, target=None)

An implementation of `importlib.abc.PathEntryFinder.find_spec`.

3.10

get_code(fullname)

Return the code object for the specified module. Raise `ZipImportError` if the module couldn't be imported.

get_data(pathname)

Return the data associated with *pathname*. Raise `OSError` if the file wasn't found.

3.3 `IOError` used to be raised, it is now an alias of `OSError`.

get_filename(fullname)

Return the value `__file__` would be set to if the specified module was imported. Raise `ZipImportError` if the module couldn't be found.

3.1

get_source(fullname)

Return the source code for the specified module. Raise `ZipImportError` if the module couldn't be found, return `None` if the archive does contain the module, but has no source for it.

is_package(fullname)

Return `True` if the module specified by *fullname* is a package. Raise `ZipImportError` if the module couldn't be found.

invalidate_caches()

Clear out the internal cache of information about files found within the ZIP archive.

3.10

archive

The file name of the importer's associated ZIP file, without a possible subpath.

prefix

The subpath within the ZIP file where modules are searched. This is the empty string for zipimporter objects which point to the root of the ZIP file.

The `archive` and `prefix` attributes, when combined with a slash, equal the original *archivepath* argument given to the `zipimporter` constructor.

Examples

Here is an example that imports a module from a ZIP archive - note that the `!zipimport` module is not explicitly used.

```
$ unzip -l example_archive.zip
Archive:  example_archive.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
     8467  01-01-00 12:30   example.py
 --------                   -------
     8467                   1 file
```

```
>>> import sys
>>> # Add the archive to the front of the module search path
>>> sys.path.insert(0, 'example_archive.zip')
>>> import example
>>> example.__file__
'example_archive.zip/example.py'
```

□□□ #2
□ □□□ □□□ 2026-10-31 17:25:16 UTC
□ □□□ □□□ 2026-10-31 17:25:18 UTC