31.1. `zipimport` — Import modules from Zip archives

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 only files `.py` and `.pyc` are available for import. 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.

ZIP archives with an archive comment are currently not supported.

See also:

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

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

PEP 302 - New Import Hooks
The PEP to add the import hooks that help this module work.

This module defines an exception:

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

31.1.1. zipimporter Objects

`zipimporter` is the class for importing ZIP files.

class `zipimport.`**`zipimporter`**(*archivepath*)

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.

`find_module`(*fullname*[, *path*])

Search for a module specified by *fullname*. *fullname* must be the fully qualified (dotted) module name. It returns the zipimporter instance itself if the module was found, or `None` if it wasn't. The optional *path* argument is ignored—it's there for compatibility with the importer protocol.

`get_code`(*fullname*)

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

`get_data`(*pathname*)

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

Changed in version 3.3: `IOError` used to be raised instead 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.

New in version 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.

`load_module`(*fullname*)

Load the module specified by *fullname*. *fullname* must be the fully qualified (dotted) module name. It returns the imported module, or raises `ZipImportError` if it wasn't found.

archive

The file name of the importer's associated ZIP file, without a possible sub-path.

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.

31.1.2. 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.zip
Archive:  example.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
     8467  11-26-02 22:30   jwzthreading.py
 --------                   -------
     8467                   1 file
$ ./python
Python 2.3 (#1, Aug 1 2003, 19:54:32)
>>> import sys
>>> sys.path.insert(0, 'example.zip')  # Add .zip file to front of pat
>>> import jwzthreading
>>> jwzthreading.__file__
'example.zip/jwzthreading.py'
```