importlib — Python's Import Mechanism

Purpose: The importlib module exposes the implementation of Python's import statement.

The `importlib` module includes functions that implement Python's import mechanism for loading code in packages and modules. It is one access point to importing modules dynamically, and useful in some cases where the name of the module that needs to be imported is unknown when the code is written (for example, for plugins or extensions to an application).

Example Package

The examples in this section use a package called `example` with `__init__.py`.

```
# example/__init__.py

print('Importing example package')
```

The package also contains `submodule.py`.

```
# example/submodule.py

print('Importing submodule')
```

Watch for the text from the `print()` calls in the sample output when the package or module are imported.

Module Types

Python supports several styles of modules. Each requires its own handling when opening the module and adding it to the namespace, and support for the formats varies by platform. For example, under Microsoft Windows, shared libraries are loaded from files with extensions `.dll` or `.pyd`, instead of `.so`. The extensions for C modules may also change when using a debug build of the interpreter instead of a normal release build, since they can be compiled with debug information included as well. If a C extension library or other module is not loading as expected, use the constants defined in `importlib.machinery` to find the supported types for the current platform, and the parameters for loading them.

```
# importlib_suffixes.py

import importlib.machinery

SUFFIXES = [
    ('Source:', importlib.machinery.SOURCE_SUFFIXES),
    ('Debug:',
     importlib.machinery.DEBUG_BYTECODE_SUFFIXES),
    ('Optimized:',
     importlib.machinery.OPTIMIZED_BYTECODE_SUFFIXES),
    ('Bytecode:', importlib.machinery.BYTECODE_SUFFIXES),
    ('Extension:', importlib.machinery.EXTENSION_SUFFIXES),
]


def main():
    tmpl = '{:<10}  {}'
    for name, value in SUFFIXES:
        print(tmpl.format(name, value))


if __name__ == '__main__':
    main()
```

The return value is a sequence of tuples containing the file extension, mode to use for opening the file containing the module, and a type code from a constant defined in the module. This table is incomplete, because some of the importable module or package types do not correspond to single files.

```
$ python3 importlib_suffixes.py

Source:     ['.py']
Debug:      ['.pyc']
Optimized:  ['.pyc']
Bytecode:   ['.pyc']
Extension:  ['.cpython-37m-darwin.so', '.abi3.so', '.so']
```

Importing Modules

The high level API in `importlib` makes it simple to import a module given an absolute or relative name. When using a relative module name, specify the package containing the module as a separate argument.

```
# importlib_import_module.py

import importlib


m1 = importlib.import_module('example.submodule')
print(m1)

m2 = importlib.import_module('.submodule', package='example')
print(m2)

print(m1 is m2)
```

The return value from `import_module()` is the module object that was created by the import.

```
$ python3 importlib_import_module.py

Importing example package
Importing submodule
<module 'example.submodule' from '.../example/submodule.py'>
<module 'example.submodule' from '.../example/submodule.py'>
True
```

If the module cannot be imported, `import_module()` raises `ImportError`.

```
# importlib_import_module_error.py

import importlib


try:
    importlib.import_module('example.nosuchmodule')
except ImportError as err:
    print('Error:', err)
```

The error message includes the name of the missing module.

```
$ python3 importlib_import_module_error.py

Importing example package
Error: No module named 'example.nosuchmodule'
```

To reload an existing module, use `reload()`.

```
# importlib_reload.py

import importlib


m1 = importlib.import_module('example.submodule')
print(m1)

m2 = importlib.reload(m1)
print(m1 is m2)
```

The return value from `reload()` is the new module. Depending on which type of loader was used, it may be the same module instance.

```
$ python3 importlib_reload.py

Importing example package
Importing submodule
<module 'example.submodule' from '.../example/submodule.py'>
Importing submodule
True
```

Loaders

The lower-level API in `importlib.util` provides access to the loader objects, as described in Modules and Imports from the section on the `sys` module. To get the information needed to load the module for a module, use `find_spec()` to find the "import spec". Then to retrieve the module, use the loader's `load_module()` method.

```
# importlib_find_loader.py

import importlib

spec = importlib.util.find_spec('example')
print('Loader:', spec.loader)

m = spec.loader.load_module()
print('Module:', m)
```

This example loads the top level of the `example` package.

```
$ python3 importlib_find_loader.py

Loader: <_frozen_importlib_external.SourceFileLoader object at
0x104d31208>
Importing example package
Module: <module 'example' from '.../example/__init__.py'>
```

Submodules within packages need to be loaded separately using the path from the package. In the following example, the `example` package is is passed to `find_spec()` to create a loader capable of loading the submodule.

```
# importlib_submodule.py

import importlib

spec = importlib.util.find_spec('.submodule', package='example')
print('Loader:', spec.loader)

m = spec.loader.load_module()
print('Module:', m)
```

As with `import_module()`, the name of the submodule should be given wit the relative path prefix.

```
$ python3 importlib_submodule.py

Importing example package
Loader: <_frozen_importlib_external.SourceFileLoader object at
0x108b465f8>
Importing submodule
Module: <module 'example.submodule' from
'.../example/submodule.py'>
```

See also

- Standard library documentation for importlib
- Modules and Imports – Import hooks, the module search path, and other related machinery in the `sys` module.
- `inspect` – Load information from a module programmatically.
- **PEP 302** – New import hooks.
- **PEP 369** – Post import hooks.

- **PEP 369** – Post import hooks.
- **PEP 488** – Elimination of PYO files.
  - Python 2 to 3 porting notes for importlib

Modules and Packages
pkgutil — Package Utilities

Quick Links

This page was last updated 2018-12-09.

Navigation

Modules and Packages
pkgutil — Package Utilities

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?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example