site — Site-wide Configuration

The `site` module handles site-specific configuration, especially the import path.

Import Path

`site` is automatically imported each time the interpreter starts up. On import, it extends `sys.path` with site-specific names constructed by combining the prefix values `sys.prefix` and `sys.exec_prefix` with several suffixes. The prefix values used are saved in the module-level variable PREFIXES for reference later. Under Windows, the suffixes are an empty string and `lib/site-packages`. For Unix-like platforms, the values are `lib/python$version/site-packages` (where `$version` is replaced by the major and minor version number of the interpreter, such as `3.5`) and `lib/site-python`.

```
# site_import_path.py

import sys
import os
import site

if 'Windows' in sys.platform:
    SUFFIXES = [
        '',
        'lib/site-packages',
    ]
else:
    SUFFIXES = [
        'lib/python{}/site-packages'.format(sys.version[:3]),
        'lib/site-python',
    ]

print('Path prefixes:')
for p in site.PREFIXES:
    print('  ', p)

for prefix in sorted(set(site.PREFIXES)):
    print()
    print(prefix)
    for suffix in SUFFIXES:
        print()
        print(' ', suffix)
        path = os.path.join(prefix, suffix).rstrip(os.sep)
        print('   exists :', os.path.exists(path))
        print('   in path:', path in sys.path)
```

Each of the paths resulting from the combinations is tested, and those that exist are added to `sys.path`. This output shows the framework version of Python installed on a Mac OS X system.

```
$ python3 site_import_path.py

Path prefixes:
   /Library/Frameworks/Python.framework/Versions/3.5
   /Library/Frameworks/Python.framework/Versions/3.5

/Library/Frameworks/Python.framework/Versions/3.5

  lib/python3.5/site-packages
   exists : True
   in path: True

  lib/site-python
   exists : False
   in path: False
```

User Directories

User Directories

In addition to the global site-packages paths, `site` is responsible for adding the user-specific locations to the import path. The user-specific paths are all based on the `USER_BASE` directory, which usually located in a part of the file system owned (and writable) by the current user. Inside the `USER_BASE` directory is a `site-packages` directory, with the path accessible as `USER_SITE`.

```
# site_user_base.py

import site

print('Base:', site.USER_BASE)
print('Site:', site.USER_SITE)
```

The `USER_SITE` path name is created using the same platform-specific suffix values described earlier.

```
$ python3 site_user_base.py

Base: /Users/dhellmann/.local
Site: /Users/dhellmann/.local/lib/python3.7/site-packages
```

The user base directory can be set through the `PYTHONUSERBASE` environment variable, and has platform-specific defaults (`~/Python$version/site-packages` for Windows and `~/.local` for non-Windows).

```
$ PYTHONUSERBASE=/tmp/$USER python3 site_user_base.py

Base: /tmp/dhellmann
Site: /tmp/dhellmann/lib/python3.7/site-packages
```

The user directory is disabled under some circumstances that would pose security issues (for example, if the process is running with a different effective user or group id than the actual user that started it). An application can check the setting by examining `ENABLE_USER_SITE`.

```
# site_enable_user_site.py

import site

status = {
    None: 'Disabled for security',
    True: 'Enabled',
    False: 'Disabled by command-line option',
}

print('Flag   :', site.ENABLE_USER_SITE)
print('Meaning:', status[site.ENABLE_USER_SITE])
```

The user directory can also be explicitly disabled on the command line with `-s`.

```
$ python3 site_enable_user_site.py

Flag   : True
Meaning: Enabled

$ python3 -s site_enable_user_site.py

Flag   : False
Meaning: Disabled by command-line option
```

Path Configuration Files

As paths are added to the import path, they are also scanned for *path configuration files*. A path configuration file is a plain text file with the extension `.pth`. Each line in the file can take one of four forms.

- A full or relative path to another location that should be added to the import path.
- A Python statement to be executed. All such lines must begin with an `import` statement.
- Blank lines are ignored.
- A line starting with # is treated as a comment and ignored.

Path configuration files can be used to extend the import path to look in locations that would not have been added

Path configuration files can be used to extend the import path to look in locations that would not have been added automatically. For example, the `setuptools` package adds a path to `easy-install.pth` when it installs a package in development mode using `python setup.py develop`.

The function for extending `sys.path` is public, and it can be used in example programs to show how the path configuration files work. Given a directory named `with_modules` containing the file `mymodule.py` with this `print` statement showing how the module was imported:

```
# with_modules/mymodule.py

import os
print('Loaded {} from {}'.format(
    __name__, __file__[len(os.getcwd()) + 1:])
)
```

This script shows how `addsitedir()` extends the import path so the interpreter can find the desired module.

```
# site_addsitedir.py

import site
import os
import sys

script_directory = os.path.dirname(__file__)
module_directory = os.path.join(script_directory, sys.argv[1])

try:
    import mymodule
except ImportError as err:
    print('Could not import mymodule:', err)

print()
before_len = len(sys.path)
site.addsitedir(module_directory)
print('New paths:')
for p in sys.path[before_len:]:
    print(p.replace(os.getcwd(), '.'))  # shorten dirname

print()
import mymodule
```

After the directory containing the module is added to `sys.path`, the script can import `mymodule` without issue.

```
$ python3 site_addsitedir.py with_modules

Could not import mymodule: No module named 'mymodule'

New paths:
./with_modules

Loaded mymodule from with_modules/mymodule.py
```

The path changes by `addsitedir()` go beyond simply appending the argument to `sys.path`. If the directory given to `addsitedir()` includes any files matching the pattern `*.pth`, they are loaded as path configuration files. Given a directory structure like the following

```
with_pth
├── pymotw.pth
└── subdir
    └── mymodule.py
```

If `with_pth/pymotw.pth` contains

```
# Add a single subdirectory to the path.
./subdir
```

then `with_pth/subdir/mymodule.py` can be imported by adding `with_pth` as a site directory, even though the module is not in that directory because both `with_pth` and `with_pth/subdir` are added to the import path.

```
$ python3 site_addsitedir.py with_pth

Could not import mymodule: No module named 'mymodule'

New paths:
./with_pth
./with_pth/subdir

Loaded mymodule from with_pth/subdir/mymodule.py
```

If a site directory contains multiple `.pth` files, they are processed in alphabetical order.

```
$ ls -F multiple_pth

a.pth
b.pth
from_a/
from_b/

$ cat multiple_pth/a.pth

./from_a

$ cat multiple_pth/b.pth

./from_b
```

In this case, the module is found in `multiple_pth/from_a` because `a.pth` is read before `b.pth`.

```
$ python3 site_addsitedir.py multiple_pth

Could not import mymodule: No module named 'mymodule'

New paths:
./multiple_pth
./multiple_pth/from_a
./multiple_pth/from_b

Loaded mymodule from multiple_pth/from_a/mymodule.py
```

Customizing Site Configuration

The `site` module is also responsible for loading site-wide customization defined by the local site owner in a `sitecustomize` module. Uses for `sitecustomize` include extending the import path and enabling coverage, profiling, or other development tools.

For example, this `sitecustomize.py` script extends the import path with a directory based on the current platform. The platform-specific path in `/opt/python` is added to the import path, so any packages installed there can be imported. A system like this is useful for sharing packages containing compiled extension modules between hosts on a network via a shared file system. Only the `sitecustomize.py` script needs to be installed on each host, and the other packages can be accessed from the file server.

```
# with_sitecustomize/sitecustomize.py

print('Loading sitecustomize.py')

import site
import platform
import os
import sys

path = os.path.join('/opt',
                    'python',
                    sys.version[:3],
                    platform.platform(),
                    )
print('Adding new path', path)

site.addsitedir(path)
```

A simple script can be used to show that `sitecustomize.py` is imported before Python starts running your own code.

```
# with_sitecustomize/site_sitecustomize.py

import sys

print('Running main program from\n{}'.format(sys.argv[0]))

print('End of path:', sys.path[-1])
```

Since `sitecustomize` is meant for system-wide configuration, it should be installed somewhere in the default path (usually in the `site-packages` directory). This example sets `PYTHONPATH` explicitly to ensure the module is picked up.

```
$ PYTHONPATH=with_sitecustomize python3 with_sitecustomize/sit\
e_sitecustomize.py

Loading sitecustomize.py
Adding new path /opt/python/3.7/Darwin-18.0.0-x86_64-i386-64bit
Running main program from
with_sitecustomize/site_sitecustomize.py
End of path: /opt/python/3.7/Darwin-18.0.0-x86_64-i386-64bit
```

Customizing User Configuration

Similar to `sitecustomize`, the `usercustomize` module can be used to set up user-specific settings each time the interpreter starts up. `usercustomize` is loaded after `sitecustomize`, so site-wide customizations can be overridden.

In environments where a user's home directory is shared on several servers running different operating systems or versions, the standard user directory mechanism may not work for user-specific installations of packages. In these cases, a platform-specific directory tree can be used instead.

```
# with_usercustomize/usercustomize.py

print('Loading usercustomize.py')

import site
import platform
import os
import sys

path = os.path.expanduser(os.path.join('~',
                                       'python',
                                       sys.version[:3],
                                       platform.platform(),
                                       ))
print('Adding new path', path)

site.addsitedir(path)
```

Another simple script, similar to the one used for `sitecustomize`, can be used to show that `usercustomize.py` is imported before Python starts running other code.

```
# with_usercustomize/site_usercustomize.py

import sys

print('Running main program from\n{}'.format(sys.argv[0]))

print('End of path:', sys.path[-1])
```

Since `usercustomize` is meant for user-specific configuration for a user, it should be installed somewhere in the user's default path, but not on the site-wide path. The default `USER_BASE` directory is a good location. This example sets `PYTHONPATH` explicitly to ensure the module is picked up.

```
$ PYTHONPATH=with_usercustomize python3 with_usercustomize/site\
_usercustomize.py
```

```
Loading usercustomize.py
Adding new path /Users/dhellmann/python/3.5/Darwin-15.5.0-x86_64\
-i386-64bit
Running main program from
with_usercustomize/site_usercustomize.py
End of path: /Users/dhellmann/python/3.5/Darwin-15.5.0-x86_64\
-i386-64bit
```

When the user site directory feature is disabled, `usercustomize` is not imported, whether it is located in the user site directory or elsewhere.

```
$ PYTHONPATH=with_usercustomize python3 -s with_usercustomize/s\
ite_usercustomize.py

Running main program from
with_usercustomize/site_usercustomize.py
End of path: /Users/dhellmann/Envs/pymotw35/lib/python3.5/site-
packages
```

Disabling the site Module

To maintain backwards-compatibility with versions of Python from before the automatic import was added, the interpreter accepts an `-S` option.

```
$ python3 -S site_import_path.py

Path prefixes:
   /Users/dhellmann/Envs/pymotw37/bin/..
   /Users/dhellmann/Envs/pymotw37/bin/..

/Users/dhellmann/Envs/pymotw37/bin/..

  lib/python3.7/site-packages
   exists : True
   in path: False

  lib/site-python
   exists : False
   in path: False
```

See also

- Standard library documentation for site
- Modules and Imports – Description of how the import path defined in `sys` works.
- setuptools – Packaging library and installation tool `easy_install`.
- Running code at Python startup – Post from Ned Batchelder discussing ways to cause the Python interpreter to run custom initialization code before starting the main program execution.

Runtime Features

sys — System-specific Configuration

Quick Links
Import Path
User Directories
Path Configuration Files
Customizing Site Configuration
Customizing User Configuration
Disabling the site Module

This page was last updated 2018-12-09.

Navigation
Runtime Features
sys — System-specific Configuration

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