sysconfig — Interpreter Compile-time Configuration

Purpose: Access the configuration settings used to build Python.

The features of `sysconfig` have been extracted from `distutils` to create a stand-alone module. It includes functions for determining the settings used to compile and install the current interpreter.

Configuration Variables

Access the build-time configuration settings is provided through two functions. `get_config_vars()` returns a dictionary mapping the configuration variable names to values.

```
# sysconfig_get_config_vars.py

import sysconfig

config_values = sysconfig.get_config_vars()
print('Found {} configuration settings'.format(
    len(config_values.keys())))

print('\nSome highlights:\n')

print(' Installation prefixes:')
print('  prefix={prefix}'.format(**config_values))
print('  exec_prefix={exec_prefix}'.format(**config_values))

print('\n Version info:')
print('  py_version={py_version}'.format(**config_values))
print('  py_version_short={py_version_short}'.format(
    **config_values))
print('  py_version_nodot={py_version_nodot}'.format(
    **config_values))

print('\n Base directories:')
print('  base={base}'.format(**config_values))
print('  platbase={platbase}'.format(**config_values))
print('  userbase={userbase}'.format(**config_values))
print('  srcdir={srcdir}'.format(**config_values))

print('\n Compiler and linker flags:')
print('  LDFLAGS={LDFLAGS}'.format(**config_values))
print('  BASECFLAGS={BASECFLAGS}'.format(**config_values))
print('  Py_ENABLE_SHARED={Py_ENABLE_SHARED}'.format(
    **config_values))
```

The level of detail available through the `sysconfig` API depends on the platform where a program is running. On POSIX systems such as Linux and OS X, the `Makefile` used to build the interpreter and `config.h` header file generated for the build are parsed and all of the variables found within are available. On non-POSIX systems such as Windows, the settings are limited to a few paths, filename extensions, and version details.

```
$ python3 sysconfig_get_config_vars.py

Found 668 configuration settings

Some highlights:

 Installation prefixes:
  prefix=/Library/Frameworks/Python.framework/Versions/3.7
  exec_prefix=/Library/Frameworks/Python.framework/Versions/3.7

 Version info:
  py_version=3.7.1
  py_version_short=3.7
  py_version_nodot=37

 Base directories:
  base=/Users/dhellmann/Envs/pymotw37
  platbase=/Users/dhellmann/Envs/pymotw37
  userbase=/Users/dhellmann/Library/Python/3.7
  srcdir=/Library/Frameworks/Python.framework/Versions/3.7/lib/p
ython3.7/config-3.7m-darwin

 Compiler and linker flags:
  LDFLAGS=-arch x86_64 -g
  BASECFLAGS=-Wno-unused-result -Wsign-compare -Wunreachable-
code -fno-common -dynamic
  Py_ENABLE_SHARED=0
```

Passing variable names to `get_config_vars()` changes the return value to a `list` created by appending all of the values for those variables together.

```
# sysconfig_get_config_vars_by_name.py

import sysconfig

bases = sysconfig.get_config_vars('base', 'platbase', 'userbase')
print('Base directories:')
for b in bases:
    print('  ', b)
```

This example builds a list of all of the installation base directories where modules can be found on the current system.

```
$ python3 sysconfig_get_config_vars_by_name.py

Base directories:
   /Users/dhellmann/Envs/pymotw37
   /Users/dhellmann/Envs/pymotw37
   /Users/dhellmann/Library/Python/3.7
```

When only a single configuration value is needed, use `get_config_var()` to retrieve it.

```
# sysconfig_get_config_var.py

import sysconfig

print('User base directory:',
      sysconfig.get_config_var('userbase'))
print('Unknown variable   :',
      sysconfig.get_config_var('NoSuchVariable'))
```

If the variable is not found, `get_config_var()` returns None instead of raising an exception.

```
$ python3 sysconfig_get_config_var.py

User base directory: /Users/dhellmann/Library/Python/3.7
Unknown variable   : None
```

Installation Paths

`sysconfig` is primarily meant to be used by installation and packaging tools. As a result, while it provides access to general configuration settings such as the interpreter version, it is focused on the information needed to locate parts of the Python distribution currently installed on a system. The locations used for installing a package depend on the *scheme* used.

A scheme is a set of platform-specific default directories organized based on the platform's packaging standards and guidelines. There are different schemes for installing into a site-wide location or a private directory owned by the user. The full set of schemes can be accessed with `get_scheme_names()`.

```
# sysconfig_get_scheme_names.py

import sysconfig
```

```
for name in sysconfig.get_scheme_names():
    print(name)
```

There is no concept of a “current scheme” per se. The default scheme depends on the platform, and the actual scheme used depends on options given to the installation program. If the current system is running a POSIX-compliant operating system, the default is `posix_prefix`. Otherwise the default is the operating system name, as defined by `os.name`.

```
$ python3 sysconfig_get_scheme_names.py

nt
nt_user
osx_framework_user
posix_home
posix_prefix
posix_user
```

Each scheme defines a set of paths used for installing packages. For a list of the path names, use `get_path_names()`.

```
# sysconfig_get_path_names.py

import sysconfig

for name in sysconfig.get_path_names():
    print(name)
```

Some of the paths may be the same for a given scheme, but installers should not make any assumptions about what the actual paths are. Each name has a particular semantic meaning, so the correct name should be used to find the path for a given file during installation. Refer to the table below for a complete list of the path names and their meaning.

Path Names Used in sysconfig

Name	Description
`stdlib`	Standard Python library files, not platform-specific
`platstdlib`	Standard Python library files, platform-specific
`platlib`	Site-specific, platform-specific files
`purelib`	Site-specific, non-platform-specific files
`include`	Header files, not platform-specific
`platinclude`	Header files, platform-specific
`scripts`	Executable script files
`data`	Data files

```
$ python3 sysconfig_get_path_names.py

stdlib
platstdlib
purelib
platlib
include
scripts
data
```

Use `get_paths()` to retrieve the actual directories associated with a scheme.

```
# sysconfig_get_paths.py

import sysconfig
import pprint
import os

for scheme in ['posix_prefix', 'posix_user']:
    print(scheme)
    print('=' * len(scheme))
    paths = sysconfig.get_paths(scheme=scheme)
    prefix = os.path.commonprefix(list(paths.values()))
    print('prefix = {}\n'.format(prefix))
    for name, path in sorted(paths.items()):
        print('{}\n  .{}'.format(name, path[len(prefix):]))
    print()
```

This example shows the difference between the system-wide paths used for posix_prefix under a framework build on Mac OS X, and the user-specific values for posix_user.

```
$ python3 sysconfig_get_paths.py

posix_prefix
============
prefix = /Users/dhellmann/Envs/pymotw37

data
  .
include
  ./include/python3.7m
platinclude
  ./include/python3.7m
platlib
  ./lib/python3.7/site-packages
platstdlib
  ./lib/python3.7
purelib
  ./lib/python3.7/site-packages
scripts
  ./bin
stdlib
  ./lib/python3.7

posix_user
==========
prefix = /Users/dhellmann/Library/Python/3.7

data
  .
include
  ./include/python3.7
platlib
  ./lib/python3.7/site-packages
platstdlib
  ./lib/python3.7
purelib
  ./lib/python3.7/site-packages
scripts
  ./bin
stdlib
  ./lib/python3.7
```

For an individual path, call get_path().

```
# sysconfig_get_path.py

import sysconfig
import pprint

for scheme in ['posix_prefix', 'posix_user']:
    print(scheme)
    print('=' * len(scheme))
    print('purelib =', sysconfig.get_path(name='purelib',
                                          scheme=scheme))
    print()
```

Using get_path() is equivalent to saving the value of get_paths() and looking up the individual key in the dictionary. If several paths are needed, get_paths() is more efficient because it does not recompute all of the paths each time.

```
$ python3 sysconfig_get_path.py

posix_prefix
```

```
============
purelib = /Users/dhellmann/Envs/pymotw37/lib/python3.7/site-pack
ages

posix_user
==========
purelib = /Users/dhellmann/Library/Python/3.7/lib/python3.7/site
-packages
```

Python Version and Platform

While sys includes some basic platform identification (see Build-time Version Information), it is not specific enough to be used for installing binary packages because `sys.platform` does not always include information about hardware architecture, instruction size, or other values that effect the compatibility of binary libraries. For a more precise platform specifier, use `get_platform()`.

```python
# sysconfig_get_platform.py
```

```python
import sysconfig

print(sysconfig.get_platform())
```

The interpreter used to prepare this sample output was compiled for Mac OS X 10.6 compatibility, so that is the version number included in the platform string.

```
$ python3 sysconfig_get_platform.py

macosx-10.9-x86_64
```

As a convenience, the interpreter version from `sys.version_info` is also available through `get_python_version()` in `sysconfig`.

```python
# sysconfig_get_python_version.py
```

```python
import sysconfig
import sys

print('sysconfig.get_python_version():',
      sysconfig.get_python_version())
print('\nsys.version_info:')
print('  major       :', sys.version_info.major)
print('  minor       :', sys.version_info.minor)
print('  micro       :', sys.version_info.micro)
print('  releaselevel:', sys.version_info.releaselevel)
print('  serial      :', sys.version_info.serial)
```

`get_python_version()` returns a string suitable for use when building a version-specific path.

```
$ python3 sysconfig_get_python_version.py

sysconfig.get_python_version(): 3.7

sys.version_info:
  major       : 3
  minor       : 7
  micro       : 1
  releaselevel: final
  serial      : 0
```

See also

- Standard library documentation for sysconfig
- distutils – sysconfig used to be part of the `distutils` package.
- site – The site module describes the paths searched when importing in more detail.
- os – Includes `os.name`, the name of the current operating system.
- sys – Includes other build-time information such as the platform.

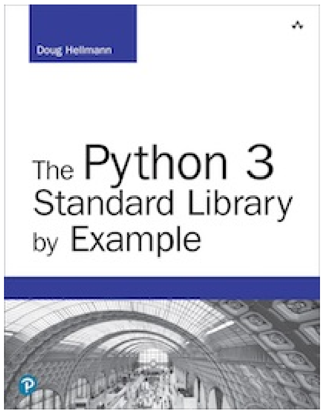
Quick Links

Configuration Variables
Installation Paths
Python Version and Platform

This page was last updated 2018-12-09.

Navigation

gc — Garbage Collector
Language Tools



Get the book

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?