16.14. `platform` — Access to underlying platform's identifying data

Source code: Lib/platform.py

> **Note:** Specific platforms listed alphabetically, with Linux included in the Unix section.

16.14.1. Cross Platform

`platform.`**`architecture`**(*executable=sys.executable*, *bits=''*, *linkage=''*)

Queries the given executable (defaults to the Python interpreter binary) for various architecture information.

Returns a tuple `(bits, linkage)` which contain information about the bit architecture and the linkage format used for the executable. Both values are returned as strings.

Values that cannot be determined are returned as given by the parameter presets. If bits is given as `''`, the `sizeof(pointer)` (or `sizeof(long)` on Python version < 1.5.2) is used as indicator for the supported pointer size.

The function relies on the system's `file` command to do the actual work. This is available on most if not all Unix platforms and some non-Unix platforms and then only if the executable points to the Python interpreter. Reasonable defaults are used when the above needs are not met.

> **Note:** On Mac OS X (and perhaps other platforms), executable files may be universal files containing multiple architectures.
> To get at the "64-bitness" of the current interpreter, it is more reliable to query the `sys.maxsize` attribute:
>
> ```
> is_64bits = sys.maxsize > 2**32
> ```

`platform.`**`machine`**()

Returns the machine type, e.g. `'i386'`. An empty string is returned if the value cannot be determined.

`platform.`**`node`**()

Returns the computer's network name (may not be fully qualified!). An empty string is returned if the value cannot be determined.

`platform.`**`platform`**(*aliased=0*, *terse=0*)

Returns a single string identifying the underlying platform with as much useful information as possible.

The output is intended to be *human readable* rather than machine parseable. It may look different on different platforms and this is intended.

If *aliased* is true, the function will use aliases for various platforms that report system names which differ from their common names, for example SunOS will be reported as Solaris. The `system_alias()` function is used to implement this.

Setting *terse* to true causes the function to return only the absolute minimum information needed to identify the platform.

`platform.`**`processor`**()

Returns the (real) processor name, e.g. `'amdk6'`.

An empty string is returned if the value cannot be determined. Note that many platforms do not provide this information or simply return the same value as for `machine()`. NetBSD does this.

`platform.`**`python_build`**()

Returns a tuple `(buildno, builddate)` stating the Python build number and date as strings.

`platform.`**`python_compiler`**()

Returns a string identifying the compiler used for compiling Python.

`platform.`**`python_branch`**()

Returns a string identifying the Python implementation SCM branch.

`platform.`**`python_implementation`**()

Returns a string identifying the Python implementation. Possible return values are: 'CPython', 'IronPython', 'Jython', 'PyPy'.

`platform.`**`python_revision`**()

Returns a string identifying the Python implementation SCM revision.

`platform.`**`python_version`**()

Returns the Python version as string `'major.minor.patchlevel'`.

Note that unlike the Python `sys.version`, the returned value will always include the patchlevel (it defaults to 0).

`platform.`**`python_version_tuple`**()

Returns the Python version as tuple `(major, minor, patchlevel)` of strings.

Note that unlike the Python `sys.version`, the returned value will always include the patchlevel (it defaults to `'0'`).

`platform.`**`release`**()

Returns the system's release, e.g. `'2.2.0'` or `'NT'` An empty string is returned if the value cannot be determined.

`platform.`**`system`**()

Returns the system/OS name, e.g. `'Linux'`, `'Windows'`, or `'Java'`. An empty string is returned if the value cannot be determined.

`platform.`**`system_alias`**(*system*, *release*, *version*)

Returns `(system, release, version)` aliased to common marketing names used for some systems. It also does some reordering of the information in some cases where it would otherwise cause confusion.

`platform.`**`version`**()

Returns the system's release version, e.g. `'#3 on degas'`. An empty string is returned if the value cannot be determined.

`platform.`**`uname`**()

Fairly portable uname interface. Returns a `namedtuple()` containing six attributes: `system`, `node`, `release`, `version`, `machine`, and `processor`.

Note that this adds a sixth attribute (`processor`) not present in the `os.uname()` result. Also, the attribute names are different for the first two attributes; `os.uname()` names them `sysname` and `nodename`.

Entries which cannot be determined are set to `''`.

Changed in version 3.3: Result changed from a tuple to a namedtuple.

16.14.2. Java Platform

`platform.`**`java_ver`**(*release=''*, *vendor=''*, *vminfo=('', '', '')*, *osinfo=('', '', '')*)

Version interface for Jython.

Returns a tuple `(release, vendor, vminfo, osinfo)` with *vminfo* being a tuple `(vm_name, vm_release, vm_vendor)` and *osinfo* being a tuple `(os_name, os_version, os_arch)`. Values which cannot be determined are set to the defaults given as parameters (which all default to `''`).

16.14.3. Windows Platform

`platform.`**`win32_ver`**(*release=''*, *version=''*, *csd=''*, *ptype=''*)

Get additional version information from the Windows Registry and return a tuple `(release, version, csd, ptype)` referring to OS release, version number, CSD level (service pack) and OS type (multi/single processor).

As a hint: *ptype* is `'Uniprocessor Free'` on single processor NT machines and `'Multiprocessor Free'` on multi processor machines. The *'Free'* refers to the OS version being free of debugging code. It could also state *'Checked'* which means the OS version uses debugging code, i.e. code that checks arguments, ranges, etc.

> **Note:** This function works best with Mark Hammond's `win32all` package installed, but also on Python 2.3 and later (support for this was added in Python 2.6). It obviously only runs on Win32 compatible platforms.

16.14.3.1. Win95/98 specific

`platform.`**`popen`**(*cmd*, *mode='r'*, *bufsize=-1*)

Portable `popen()` interface. Find a working popen implementation preferring `win32pipe.popen()`. On Windows NT, `win32pipe.popen()` should work; on Windows 9x it hangs due to bugs in the MS C library.

Deprecated since version 3.3: This function is obsolete. Use the `subprocess` module. Check especially the Replacing Older Functions with the subprocess Module section.

16.14.4. Mac OS Platform

`platform.`**`mac_ver`**(*release=''*, *versioninfo=('', '', '')*, *machine=''*)

Get Mac OS version information and return it as tuple `(release, versioninfo, machine)` with *versioninfo* being a tuple `(version, dev_stage, non_release_version)`.

Entries which cannot be determined are set to `''`. All tuple entries are strings.

16.14.5. Unix Platforms

`platform.`**`dist`**(*distname=''*, *version=''*, *id=''*, *supported_dists=('SuSE'*, *'debian'*, *'redhat'*, *'mandrake'*, *...)*)

This is another name for `linux_distribution()`.

Deprecated since version 3.5, will be removed in version 3.8: See alternative like the distro package.

`platform.`**`linux_distribution`**(*distname=''*, *version=''*, *id=''*, *supported_dists=('SuSE'*, *'debian'*, *'redhat'*, *'mandrake'*, *...)*, *full_distribution_name=1*)

Tries to determine the name of the Linux OS distribution name.

`supported_dists` may be given to define the set of Linux distributions to look for. It defaults to a list of currently supported Linux distributions identified by their release file name.

If `full_distribution_name` is true (default), the full distribution read from the OS is returned. Otherwise the short name taken from `supported_dists` is used.

Returns a tuple `(distname,version,id)` which defaults to the args given as parameters. `id` is the item in parentheses after the version number. It is usually the version codename.

Deprecated since version 3.5, will be removed in version 3.8: See alternative like the distro package.

`platform.`**`libc_ver`**(*executable=sys.executable*, *lib=''*, *version=''*, *chunksize=2048*)

Tries to determine the libc version against which the file executable (defaults to the Python interpreter) is linked. Returns a tuple of strings `(lib, version)` which default to the given parameters in case the lookup fails.

Note that this function has intimate knowledge of how different libc versions add symbols to the executable is probably only usable for executables compiled using **gcc**.

The file is read and scanned in chunks of *chunksize* bytes.