:mod:`platform` --- Access to underlying platform's identifying data

Source code: :source:`Lib/platform.py`

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

Cross Platform

and then only if the executable points to the Python interpreter. Reasonable
defaults are used when the above needs are not met.

```
.. note::

   On macOS (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 :attr:`sys.maxsize` attribute::

      is_64bits = sys.maxsize > 2**32
```

```
.. function:: machine()

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

```
.. function:: node()

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

```
.. function:: 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 :func:`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.

   .. versionchanged:: 3.8
      On macOS, the function now uses :func:`mac_ver`, if it returns a
      non-empty release string, to get the macOS version rather than the darwin
      version.
```

```
.. function:: 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
:func:`machine`. NetBSD does this.

```
.. function:: python_build()

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

```
.. function:: python_compiler()

   Returns a string identifying the compiler used for compiling Python.
```

```
.. function:: python_branch()

   Returns a string identifying the Python implementation SCM branch.
```

```
.. function:: python_implementation()

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

```
.. function:: python_revision()

   Returns a string identifying the Python implementation SCM revision.
```

```
.. function:: 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).
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)platform.rst`**, line 132)**

Unknown directive type "function".

```
.. function:: 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'``).
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)platform.rst`**, line 140)**

Unknown directive type "function".

```
.. function:: release()

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

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)platform.rst`**, line 146)**

Unknown directive type "function".

```
.. function:: system()

   Returns the system/OS name, such as ``'Linux'``, ``'Darwin'``, ``'Java'``,
   ``'Windows'``. An empty string is returned if the value cannot be determined.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)platform.rst`**, line 152)**

Unknown directive type "function".

```
.. function:: 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.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)platform.rst`**, line 159)**

Unknown directive type "function".

```
.. function:: version()

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

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)platform.rst`**, line 165)**

Unknown directive type "function".

```
.. function:: uname()

   Fairly portable uname interface. Returns a :func:`~collections.namedtuple`
   containing six attributes: :attr:`system`, :attr:`node`, :attr:`release`,
   :attr:`version`, :attr:`machine`, and :attr:`processor`.

   Note that this adds a sixth attribute (:attr:`processor`) not present
   in the :func:`os.uname` result.  Also, the attribute names are different
```

```
for the first two attributes; :func:`os.uname` names them
:attr:`sysname` and :attr:`nodename`.

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

.. versionchanged:: 3.3
   Result changed from a tuple to a namedtuple.
```

Java Platform

```
.. function:: 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 ``''``).
```

Windows Platform

```
.. function:: 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.
```

```
.. function:: win32_edition()

   Returns a string representing the current Windows edition.  Possible
   values include but are not limited to ``'Enterprise'``, ``'IoTUAP'``,
   ``'ServerStandard'``, and ``'nanoserver'``.

   .. versionadded:: 3.8
```

```
.. function:: win32_is_iot()

   Return ``True`` if the Windows edition returned by :func:`win32_edition`
   is recognized as an IoT edition.

   .. versionadded:: 3.8
```

macOS Platform

```
.. function:: mac_ver(release='', versioninfo=('','',''), machine='')

   Get macOS 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.
```

Unix Platforms

```
.. function:: libc_ver(executable=sys.executable, lib='', version='', chunksize=16384)

   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 :program:`gcc`.

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

Linux Platforms

```
.. function:: freedesktop_os_release()

   Get operating system identification from ``os-release`` file and return
   it as a dict. The ``os-release`` file is a `freedesktop.org standard
   <https://www.freedesktop.org/software/systemd/man/os-release.html>`_ and
   is available in most Linux distributions. A noticeable exception is
   Android and Android-based distributions.

   Raises :exc:`OSError` or subclass when neither ``/etc/os-release`` nor
   ``/usr/lib/os-release`` can be read.

   On success, the function returns a dictionary where keys and values are
   strings. Values have their special characters like ``"`` and ``$``
   unquoted. The fields ``NAME``, ``ID``, and ``PRETTY_NAME`` are always
   defined according to the standard. All other fields are optional. Vendors
   may include additional fields.

   Note that fields like ``NAME``, ``VERSION``, and ``VARIANT`` are strings
   suitable for presentation to users. Programs should use fields like
   ``ID``, ``ID_LIKE``, ``VERSION_ID``, or ``VARIANT_ID`` to identify
   Linux distributions.

   Example::

      def get_like_distro():
          info = platform.freedesktop_os_release()
          ids = [info["ID"]]
          if "ID_LIKE" in info:
              # ids are space separated and ordered by precedence
```

```
            ids.extend(info["ID_LIKE"].split())
        return ids
```

.. versionadded:: 3.10