System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 1)

Unknown directive type "highlight".

```
.. highlight:: c
```

Operating System Utilities

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 8)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyOS_FSPath(PyObject *path)

   Return the file system representation for *path*. If the object is a
   :class:`str` or :class:`bytes` object, then its reference count is
   incremented. If the object implements the :class:`os.PathLike` interface,
   then :meth:`~os.PathLike.__fspath__` is returned as long as it is a
   :class:`str` or :class:`bytes` object. Otherwise :exc:`TypeError` is raised
   and ``NULL`` is returned.

   .. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 20)

Unknown directive type "c:function".

```
.. c:function:: int Py_FdIsInteractive(FILE *fp, const char *filename)

   Return true (nonzero) if the standard I/O file *fp* with name *filename* is
   deemed interactive.  This is the case for files for which ``isatty(fileno(fp))``
   is true.  If the global flag :c:data:`Py_InteractiveFlag` is true, this function
   also returns true if the *filename* pointer is ``NULL`` or if the name is equal to
   one of the strings ``'<stdin>'`` or ``'???'``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 29)

Unknown directive type "c:function".

```
.. c:function:: void PyOS_BeforeFork()

   Function to prepare some internal state before a process fork.  This
   should be called before calling :c:func:`fork` or any similar function
   that clones the current process.
   Only available on systems where :c:func:`fork` is defined.

   .. warning::
      The C :c:func:`fork` call should only be made from the
      :ref:`"main" thread <fork-and-threads>` (of the
      :ref:`"main" interpreter <sub-interpreter-support>`).  The same is
      true for ``PyOS_BeforeFork()``.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 45)

Unknown directive type "c:function".

```
.. c:function:: void PyOS_AfterFork_Parent()

   Function to update some internal state after a process fork.  This
   should be called from the parent process after calling :c:func:`fork`
```

```
   or any similar function that clones the current process, regardless
   of whether process cloning was successful.
   Only available on systems where :c:func:`fork` is defined.

   .. warning::
      The C :c:func:`fork` call should only be made from the
      :ref:`"main" thread <fork-and-threads>` (of the
      :ref:`"main" interpreter <sub-interpreter-support>`).  The same is
      true for ``PyOS_AfterFork_Parent()``.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 62)

Unknown directive type "c:function".

```
.. c:function:: void PyOS_AfterFork_Child()

   Function to update internal interpreter state after a process fork.
   This must be called from the child process after calling :c:func:`fork`,
   or any similar function that clones the current process, if there is
   any chance the process will call back into the Python interpreter.
   Only available on systems where :c:func:`fork` is defined.

   .. warning::
      The C :c:func:`fork` call should only be made from the
      :ref:`"main" thread <fork-and-threads>` (of the
      :ref:`"main" interpreter <sub-interpreter-support>`).  The same is
      true for ``PyOS_AfterFork_Child()``.

   .. versionadded:: 3.7

   .. seealso::
      :func:`os.register_at_fork` allows registering custom Python functions
      to be called by :c:func:`PyOS_BeforeFork()`,
      :c:func:`PyOS_AfterFork_Parent` and  :c:func:`PyOS_AfterFork_Child`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 84)

Unknown directive type "c:function".

```
.. c:function:: void PyOS_AfterFork()

   Function to update some internal state after a process fork; this should be
   called in the new process if the Python interpreter will continue to be used.
   If a new executable is loaded into the new process, this function does not need
   to be called.

   .. deprecated:: 3.7
      This function is superseded by :c:func:`PyOS_AfterFork_Child()`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 95)

Unknown directive type "c:function".

```
.. c:function:: int PyOS_CheckStack()

   Return true when the interpreter runs out of stack space.  This is a reliable
   check, but is only available when :const:`USE_STACKCHECK` is defined (currently
   on Windows using the Microsoft Visual C++ compiler).  :const:`USE_STACKCHECK`
   will be defined automatically; you should never change the definition in your
   own code.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 104)

Unknown directive type "c:function".

```
.. c:function:: PyOS_sighandler_t PyOS_getsig(int i)

   Return the current signal handler for signal *i*.  This is a thin wrapper around
   either :c:func:`sigaction` or :c:func:`signal`.  Do not call those functions
   directly! :c:type:`PyOS_sighandler_t` is a typedef alias for :c:type:`void
   (\*)(int)`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 112)

Unknown directive type "c:function".

```
.. c:function:: PyOS_sighandler_t PyOS_setsig(int i, PyOS_sighandler_t h)

   Set the signal handler for signal *i* to be *h*; return the old signal handler.
   This is a thin wrapper around either :c:func:`sigaction` or :c:func:`signal`.  Do
   not call those functions directly!  :c:type:`PyOS_sighandler_t` is a typedef
   alias for :c:type:`void (\*)(int)`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 119)

Unknown directive type "c:function".

```
.. c:function:: wchar_t* Py_DecodeLocale(const char* arg, size_t *size)

   .. warning::
      This function should not be called directly: use the :c:type:`PyConfig`
      API with the :c:func:`PyConfig_SetBytesString` function which ensures
      that :ref:`Python is preinitialized <c-preinit>`.

      This function must not be called before :ref:`Python is preinitialized
      <c-preinit>` and so that the LC_CTYPE locale is properly configured: see
      the :c:func:`Py_PreInitialize` function.

   Decode a byte string from the :term:`filesystem encoding and error handler`.
   If the error handler is :ref:`surrogateescape error handler
   <surrogateescape>`, undecodable bytes are decoded as characters in range
   U+DC80..U+DCFF; and if a byte sequence can be decoded as a surrogate
   character, the bytes are escaped using the surrogateescape error handler
   instead of decoding them.

   Return a pointer to a newly allocated wide character string, use
   :c:func:`PyMem_RawFree` to free the memory. If size is not ``NULL``, write
   the number of wide characters excluding the null character into ``*size``

   Return ``NULL`` on decoding error or memory allocation error. If *size* is
   not ``NULL``, ``*size`` is set to ``(size_t)-1`` on memory error or set to
   ``(size_t)-2`` on decoding error.

   The :term:`filesystem encoding and error handler` are selected by
   :c:func:`PyConfig_Read`: see :c:member:`~PyConfig.filesystem_encoding` and
   :c:member:`~PyConfig.filesystem_errors` members of :c:type:`PyConfig`.

   Decoding errors should never happen, unless there is a bug in the C
   library.

   Use the :c:func:`Py_EncodeLocale` function to encode the character string
   back to a byte string.

   .. seealso::

      The :c:func:`PyUnicode_DecodeFSDefaultAndSize` and
      :c:func:`PyUnicode_DecodeLocaleAndSize` functions.

   .. versionadded:: 3.5

   .. versionchanged:: 3.7
      The function now uses the UTF-8 encoding in the :ref:`Python UTF-8 Mode
      <utf8-mode>`.

   .. versionchanged:: 3.8
      The function now uses the UTF-8 encoding on Windows if
      :c:data:`Py_LegacyWindowsFSEncodingFlag` is zero;
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 171)

Unknown directive type "c:function".

```
.. c:function:: char* Py_EncodeLocale(const wchar_t *text, size_t *error_pos)

   Encode a wide character string to the :term:`filesystem encoding and error
   handler`. If the error handler is :ref:`surrogateescape error handler
   <surrogateescape>`, surrogate characters in the range U+DC80..U+DCFF are
   converted to bytes 0x80..0xFF.

   Return a pointer to a newly allocated byte string, use :c:func:`PyMem_Free`
   to free the memory. Return ``NULL`` on encoding error or memory allocation
   error.

   If error_pos is not ``NULL``, ``*error_pos`` is set to ``(size_t)-1`` on
   success,  or set to the index of the invalid character on encoding error.

   The :term:`filesystem encoding and error handler` are selected by
   :c:func:`PyConfig_Read`: see :c:member:`~PyConfig.filesystem_encoding` and
   :c:member:`~PyConfig.filesystem_errors` members of :c:type:`PyConfig`.

   Use the :c:func:`Py_DecodeLocale` function to decode the bytes string back
   to a wide character string.

   .. warning::
      This function must not be called before :ref:`Python is preinitialized
      <c-preinit>` and so that the LC_CTYPE locale is properly configured: see
      the :c:func:`Py_PreInitialize` function.

   .. seealso::

      The :c:func:`PyUnicode_EncodeFSDefault` and
      :c:func:`PyUnicode_EncodeLocale` functions.

   .. versionadded:: 3.5

   .. versionchanged:: 3.7
      The function now uses the UTF-8 encoding in the :ref:`Python UTF-8 Mode
      <utf8-mode>`.

   .. versionchanged:: 3.8
      The function now uses the UTF-8 encoding on Windows if
      :c:data:`Py_LegacyWindowsFSEncodingFlag` is zero.
```

System Functions

These are utility functions that make functionality from the :mod:`sys` module accessible to C code. They all work with the current interpreter thread's :mod:`sys` module's dict, which is contained in the internal thread state structure.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 218); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 218); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 222)

Unknown directive type "c:function".

```
.. c:function:: PyObject *PySys_GetObject(const char *name)

   Return the object *name* from the :mod:`sys` module or ``NULL`` if it does
   not exist, without setting an exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 227)

Unknown directive type "c:function".

```
.. c:function:: int PySys_SetObject(const char *name, PyObject *v)

   Set *name* in the :mod:`sys` module to *v* unless *v* is ``NULL``, in which
   case *name* is deleted from the sys module. Returns ``0`` on success, ``-1``
   on error.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 233)

Unknown directive type "c:function".

```
.. c:function:: void PySys_ResetWarnOptions()

   Reset :data:`sys.warnoptions` to an empty list. This function may be
   called prior to :c:func:`Py_Initialize`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 238)

Unknown directive type "c:function".

```
.. c:function:: void PySys_AddWarnOption(const wchar_t *s)

   This API is kept for backward compatibility: setting
   :c:member:`PyConfig.warnoptions` should be used instead, see :ref:`Python
   Initialization Configuration <init-config>`.

   Append *s* to :data:`sys.warnoptions`. This function must be called prior
   to :c:func:`Py_Initialize` in order to affect the warnings filter list.

   .. deprecated:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 249)

Unknown directive type "c:function".

```
.. c:function:: void PySys_AddWarnOptionUnicode(PyObject *unicode)

   This API is kept for backward compatibility: setting
   :c:member:`PyConfig.warnoptions` should be used instead, see :ref:`Python
   Initialization Configuration <init-config>`.

   Append *unicode* to :data:`sys.warnoptions`.

   Note: this function is not currently usable from outside the CPython
   implementation, as it must be called prior to the implicit import of
   :mod:`warnings` in :c:func:`Py_Initialize` to be effective, but can't be
   called until enough of the runtime has been initialized to permit the
   creation of Unicode objects.

   .. deprecated:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) sys.rst, line 265)

Unknown directive type "c:function".

```
.. c:function:: void PySys_SetPath(const wchar_t *path)

   Set :data:`sys.path` to a list object of paths found in *path* which should
   be a list of paths separated with the platform's search path delimiter
   (``:`` on Unix, ``;`` on Windows).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 271)

Unknown directive type "c:function".

```
.. c:function:: void PySys_WriteStdout(const char *format, ...)

   Write the output string described by *format* to :data:`sys.stdout`.  No
   exceptions are raised, even if truncation occurs (see below).

   *format* should limit the total size of the formatted output string to
   1000 bytes or less -- after 1000 bytes, the output string is truncated.
   In particular, this means that no unrestricted "%s" formats should occur;
   these should be limited using "%.<N>s" where <N> is a decimal number
   calculated so that <N> plus the maximum size of other formatted text does not
   exceed 1000 bytes.  Also watch out for "%f", which can print hundreds of
   digits for very large numbers.

   If a problem occurs, or :data:`sys.stdout` is unset, the formatted message
   is written to the real (C level) *stdout*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 287)

Unknown directive type "c:function".

```
.. c:function:: void PySys_WriteStderr(const char *format, ...)

   As :c:func:`PySys_WriteStdout`, but write to :data:`sys.stderr` or *stderr*
   instead.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 292)

Unknown directive type "c:function".

```
.. c:function:: void PySys_FormatStdout(const char *format, ...)

   Function similar to PySys_WriteStdout() but format the message using
   :c:func:`PyUnicode_FromFormatV` and don't truncate the message to an
   arbitrary length.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 300)

Unknown directive type "c:function".

```
.. c:function:: void PySys_FormatStderr(const char *format, ...)

   As :c:func:`PySys_FormatStdout`, but write to :data:`sys.stderr` or *stderr*
   instead.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 307)

Unknown directive type "c:function".

```
.. c:function:: void PySys_AddXOption(const wchar_t *s)

   This API is kept for backward compatibility: setting
   :c:member:`PyConfig.xoptions` should be used instead, see :ref:`Python
   Initialization Configuration <init-config>`.

   Parse *s* as a set of :option:`-X` options and add them to the current
   options mapping as returned by :c:func:`PySys_GetXOptions`. This function
   may be called prior to :c:func:`Py_Initialize`.

   .. versionadded:: 3.2
```

```
   .. deprecated:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 321)

Unknown directive type "c:function".

```
.. c:function:: PyObject *PySys_GetXOptions()

   Return the current dictionary of :option:`-X` options, similarly to
   :data:`sys._xoptions`.  On error, ``NULL`` is returned and an exception is
   set.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 330)

Unknown directive type "c:function".

```
.. c:function:: int PySys_Audit(const char *event, const char *format, ...)

   Raise an auditing event with any active hooks. Return zero for success
   and non-zero with an exception set on failure.

   If any hooks have been added, *format* and other arguments will be used
   to construct a tuple to pass. Apart from ``N``, the same format characters
   as used in :c:func:`Py_BuildValue` are available. If the built value is not
   a tuple, it will be added into a single-element tuple. (The ``N`` format
   option consumes a reference, but since there is no way to know whether
   arguments to this function will be consumed, using it may cause reference
   leaks.)

   Note that ``#`` format characters should always be treated as
   ``Py_ssize_t``, regardless of whether ``PY_SSIZE_T_CLEAN`` was defined.

   :func:`sys.audit` performs the same function from Python code.

   .. versionadded:: 3.8

   .. versionchanged:: 3.8.2

      Require ``Py_ssize_t`` for ``#`` format characters. Previously, an
      unavoidable deprecation warning was raised.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)sys.rst, line 356)

Unknown directive type "c:function".

```
.. c:function:: int PySys_AddAuditHook(Py_AuditHookFunction hook, void *userData)

   Append the callable *hook* to the list of active auditing hooks.
   Return zero on success
   and non-zero on failure. If the runtime has been initialized, also set an
   error on failure. Hooks added through this API are called for all
   interpreters created by the runtime.

   The *userData* pointer is passed into the hook function. Since hook
   functions may be called from different runtimes, this pointer should not
   refer directly to Python state.

   This function is safe to call before :c:func:`Py_Initialize`. When called
   after runtime initialization, existing audit hooks are notified and may
   silently abort the operation by raising an error subclassed from
   :class:`Exception` (other errors will not be silenced).

   The hook function is of type :c:type:`int (*)(const char *event, PyObject
   *args, void *userData)`, where *args* is guaranteed to be a
   :c:type:`PyTupleObject`. The hook function is always called with the GIL
   held by the Python interpreter that raised the event.

   See :pep:`578` for a detailed description of auditing.  Functions in the
```

```
runtime and standard library that raise events are listed in the
:ref:`audit events table <audit-events>`.
Details are in each function's documentation.

.. audit-event:: sys.addaudithook "" c.PySys_AddAuditHook

   If the interpreter is initialized, this function raises a auditing event
   ``sys.addaudithook`` with no arguments. If any existing hooks raise an
   exception derived from :class:`Exception`, the new hook will not be
   added and the exception is cleared. As a result, callers cannot assume
   that their hook has been added unless they control all existing hooks.

.. versionadded:: 3.8
```

Process Control

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 400)

Unknown directive type "c:function".

```
.. c:function:: void Py_FatalError(const char *message)

   .. index:: single: abort()

   Print a fatal error message and kill the process.  No cleanup is performed.
   This function should only be invoked when a condition is detected that would
   make it dangerous to continue using the Python interpreter; e.g., when the
   object administration appears to be corrupted.  On Unix, the standard C library
   function :c:func:`abort` is called which will attempt to produce a :file:`core`
   file.

   The ``Py_FatalError()`` function is replaced with a macro which logs
   automatically the name of the current function, unless the
   ``Py_LIMITED_API`` macro is defined.

   .. versionchanged:: 3.9
      Log the function name automatically.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 419)

Unknown directive type "c:function".

```
.. c:function:: void Py_Exit(int status)

   .. index::
      single: Py_FinalizeEx()
      single: exit()

   Exit the current process.  This calls :c:func:`Py_FinalizeEx` and then calls the
   standard C library function ``exit(status)``.  If :c:func:`Py_FinalizeEx`
   indicates an error, the exit status is set to 120.

   .. versionchanged:: 3.6
      Errors from finalization no longer ignored.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)sys.rst, line 433)

Unknown directive type "c:function".

```
.. c:function:: int Py_AtExit(void (*func) ())

   .. index::
      single: Py_FinalizeEx()
      single: cleanup functions

   Register a cleanup function to be called by :c:func:`Py_FinalizeEx`.  The cleanup
   function will be called with no arguments and should return no value.  At most
   32 cleanup functions can be registered.  When the registration is successful,
```

```
:c:func:`Py_AtExit` returns ``0``; on failure, it returns ``-1``.  The cleanup
function registered last is called first. Each cleanup function will be called
at most once.  Since Python's internal finalization will have completed before
the cleanup function, no Python APIs should be called by *func*.
```