Importing Modules

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

Unknown directive type "highlight".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) import.rst, line 9)

Unknown directive type "c:function".

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

   .. index::
      single: package variable; __all__
      single: __all__ (package variable)
      single: modules (in module sys)

   This is a simplified interface to :c:func:`PyImport_ImportModuleEx` below,
   leaving the *globals* and *locals* arguments set to ``NULL`` and *level* set
   to 0.  When the *name*
   argument contains a dot (when it specifies a submodule of a package), the
   *fromlist* argument is set to the list ``['*']`` so that the return value is the
   named module rather than the top-level package containing it as would otherwise
   be the case.  (Unfortunately, this has an additional side effect when *name* in
   fact specifies a subpackage instead of a submodule: the submodules specified in
   the package's ``__all__`` variable are  loaded.)  Return a new reference to the
   imported module, or ``NULL`` with an exception set on failure.  A failing
   import of a module doesn't leave the module in :data:`sys.modules`.

   This function always uses absolute imports.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) import.rst, line 31)

Unknown directive type "c:function".

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

   This function is a deprecated alias of :c:func:`PyImport_ImportModule`.

   .. versionchanged:: 3.3
      This function used to fail immediately when the import lock was held
      by another thread.  In Python 3.3 though, the locking scheme switched
      to per-module locks for most purposes, so this function's special
      behaviour isn't needed anymore.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) import.rst, line 42)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ImportModuleEx(const char *name, PyObject *globals, PyObject *locals, PyObject *fromli

   .. index:: builtin: __import__

   Import a module.  This is best described by referring to the built-in Python
   function :func:`__import__`.

   The return value is a new reference to the imported module or top-level
   package, or ``NULL`` with an exception set on failure.  Like for
   :func:`__import__`, the return value when a submodule of a package was
   requested is normally the top-level package, unless a non-empty *fromlist*
   was given.

   Failing imports remove incomplete module objects, like with
   :c:func:`PyImport_ImportModule`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) import.rst, line 59)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ImportModuleLevelObject(PyObject *name, PyObject *globals, PyObject *locals, PyObject

   Import a module.  This is best described by referring to the built-in Python
   function :func:`__import__`, as the standard :func:`__import__` function calls
   this function directly.

   The return value is a new reference to the imported module or top-level package,
   or ``NULL`` with an exception set on failure.  Like for :func:`__import__`,
   the return value when a submodule of a package was requested is normally the
   top-level package, unless a non-empty *fromlist* was given.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api) import.rst, line 73)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ImportModuleLevel(const char *name, PyObject *globals, PyObject *locals, PyObject *fr

   Similar to :c:func:`PyImport_ImportModuleLevelObject`, but the name is a
```

```
   UTF-8 encoded string instead of a Unicode object.

   .. versionchanged:: 3.3
         Negative values for *level* are no longer accepted.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 81)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_Import(PyObject *name)

   This is a higher-level interface that calls the current "import hook
   function" (with an explicit *level* of 0, meaning absolute import).  It
   invokes the :func:`__import__` function from the ``__builtins__`` of the
   current globals.  This means that the import is done using whatever import
   hooks are installed in the current environment.

   This function always uses absolute imports.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 92)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ReloadModule(PyObject *m)

   Reload a module.  Return a new reference to the reloaded module, or ``NULL`` with
   an exception set on failure (the module still exists in this case).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 98)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_AddModuleObject(PyObject *name)

   Return the module object corresponding to a module name.  The *name* argument
   may be of the form ``package.module``. First check the modules dictionary if
   there's one there, and if not, create a new one and insert it in the modules
   dictionary. Return ``NULL`` with an exception set on failure.

   .. note::

      This function does not load or import the module; if the module wasn't already
      loaded, you will get an empty module object. Use :c:func:`PyImport_ImportModule`
      or one of its variants to import a module.  Package structures implied by a
      dotted name for *name* are not created if not already present.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 115)

Unknown directive type "c:function".

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

   Similar to :c:func:`PyImport_AddModuleObject`, but the name is a UTF-8
   encoded string instead of a Unicode object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 121)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ExecCodeModule(const char *name, PyObject *co)

   .. index:: builtin: compile

   Given a module name (possibly of the form ``package.module``) and a code object
   read from a Python bytecode file or obtained from the built-in function
   :func:`compile`, load the module.  Return a new reference to the module object,
   or ``NULL`` with an exception set if an error occurred.  *name*
   is removed from :attr:`sys.modules` in error cases, even if *name* was already
   in :attr:`sys.modules` on entry to :c:func:`PyImport_ExecCodeModule`.  Leaving
   incompletely initialized modules in :attr:`sys.modules` is dangerous, as imports of
   such modules have no way to know that the module object is an unknown (and
   probably damaged with respect to the module author's intents) state.

   The module's :attr:`__spec__` and :attr:`__loader__` will be set, if
   not set already, with the appropriate values.  The spec's loader will
   be set to the module's ``__loader__`` (if set) and to an instance of
   :class:`SourceFileLoader` otherwise.

   The module's :attr:`__file__` attribute will be set to the code object's
   :c:member:`co_filename`.  If applicable, :attr:`__cached__` will also
   be set.

   This function will reload the module if it was already imported.  See
   :c:func:`PyImport_ReloadModule` for the intended way to reload a module.

   If *name* points to a dotted name of the form ``package.module``, any package
   structures not already created will still not be created.

   See also :c:func:`PyImport_ExecCodeModuleEx` and
   :c:func:`PyImport_ExecCodeModuleWithPathnames`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 154)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ExecCodeModuleEx(const char *name, PyObject *co, const char *pathname)

   Like :c:func:`PyImport_ExecCodeModule`, but the :attr:`__file__` attribute of
   the module object is set to *pathname* if it is non-``NULL``.

   See also :c:func:`PyImport_ExecCodeModuleWithPathnames`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 162)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ExecCodeModuleObject(PyObject *name, PyObject *co, PyObject *pathname, PyObject *cpat

   Like :c:func:`PyImport_ExecCodeModuleEx`, but the :attr:`__cached__`
   attribute of the module object is set to *cpathname* if it is
   non-``NULL``.  Of the three functions, this is the preferred one to use.

   .. versionadded:: 3.3
```

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

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_ExecCodeModuleWithPathnames(const char *name, PyObject *co, const char *pathname, cor

   Like :c:func:`PyImport_ExecCodeModuleObject`, but *name*, *pathname* and
   *cpathname* are UTF-8 encoded strings. Attempts are also made to figure out
   what the value for *pathname* should be from *cpathname* if the former is
   set to ``NULL``.

   .. versionadded:: 3.2
   .. versionchanged:: 3.3
      Uses :func:`imp.source_from_cache()` in calculating the source path if
      only the bytecode path is provided.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 184)

Unknown directive type "c:function".

```
.. c:function:: long PyImport_GetMagicNumber()

   Return the magic number for Python bytecode files (a.k.a. :file:`.pyc` file).
   The magic number should be present in the first four bytes of the bytecode
   file, in little-endian byte order. Returns ``-1`` on error.

   .. versionchanged:: 3.3
      Return value of ``-1`` upon failure.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 194)

Unknown directive type "c:function".

```
.. c:function:: const char * PyImport_GetMagicTag()

   Return the magic tag string for :pep:`3147` format Python bytecode file
   names.  Keep in mind that the value at ``sys.implementation.cache_tag`` is
   authoritative and should be used instead of this function.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 202)

Unknown directive type "c:function".

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

   Return the dictionary used for the module administration (a.k.a.
   ``sys.modules``).  Note that this is a per-interpreter variable.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 207)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyImport_GetModule(PyObject *name)

   Return the already imported module with the given name.  If the
   module has not been imported yet then returns ``NULL`` but does not set
   an error.  Returns ``NULL`` and sets an error if the lookup failed.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 215)

Unknown directive type "c:function".

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

   Return a finder object for a :data:`sys.path`/:attr:`pkg.__path__` item
   *path*, possibly by fetching it from the :data:`sys.path_importer_cache`
   dict.  If it wasn't yet cached, traverse :data:`sys.path_hooks` until a hook
   is found that can handle the path item.  Return ``None`` if no hook could;
   this tells our caller that the :term:`path based finder` could not find a
   finder for this path item. Cache the result in :data:`sys.path_importer_cache`.
   Return a new reference to the finder object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 226)

Unknown directive type "c:function".

```
.. c:function:: int PyImport_ImportFrozenModuleObject(PyObject *name)

   Load a frozen module named *name*.  Return ``1`` for success, ``0`` if the
   module is not found, and ``-1`` with an exception set if the initialization
   failed.  To access the imported module on a successful load, use
   :c:func:`PyImport_ImportModule`.  (Note the misnomer --- this function would
   reload the module if it was already imported.)

   .. versionadded:: 3.3

   .. versionchanged:: 3.4
      The ``__file__`` attribute is no longer set on the module.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 240)

Unknown directive type "c:function".

```
.. c:function:: int PyImport_ImportFrozenModule(const char *name)

   Similar to :c:func:`PyImport_ImportFrozenModuleObject`, but the name is a
   UTF-8 encoded string instead of a Unicode object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 246)

Unknown directive type "c:type".

```
.. c:type:: struct _frozen

   .. index:: single: freeze utility

   This is the structure type definition for frozen module descriptors, as
   generated by the :program:`freeze` utility (see :file:`Tools/freeze/` in the
   Python source distribution).  Its definition, found in :file:`Include/import.h`,
   is::

      struct _frozen {
          const char *name;
          const unsigned char *code;
          int size;
          bool is_package;
      };

   .. versionchanged:: 3.11
      The new ``is_package`` field indicates whether the module is a package or not.
      This replaces setting the ``size`` field to a negative value.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 266)

Unknown directive type "c:var".

```
.. c:var:: const struct _frozen* PyImport_FrozenModules

   This pointer is initialized to point to an array of :c:type:`struct _frozen`
   records, terminated by one whose members are all ``NULL`` or zero.  When a frozen
   module is imported, it is searched in this table.  Third-party code could play
   tricks with this to provide a dynamically created collection of frozen modules.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 274)

Unknown directive type "c:function".

```
.. c:function:: int PyImport_AppendInittab(const char *name, PyObject* (*initfunc)(void))

   Add a single module to the existing table of built-in modules.  This is a
   convenience wrapper around :c:func:`PyImport_ExtendInittab`, returning ``-1`` if
   the table could not be extended.  The new module can be imported by the name
   *name*, and uses the function *initfunc* as the initialization function called
   on the first attempted import.  This should be called before
   :c:func:`Py_Initialize`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)import.rst, line 284)

Unknown directive type "c:type".

```
.. c:type:: struct _inittab

   Structure describing a single entry in the list of built-in modules.  Each of
```

```
these structures gives the name and initialization function for a module built
into the interpreter.  The name is an ASCII encoded string.  Programs which
embed Python may use an array of these structures in conjunction with
:c:func:`PyImport_ExtendInittab` to provide additional built-in modules.
The structure is defined in :file:`Include/import.h` as::

   struct _inittab {
       const char *name;           /* ASCII encoded string */
       PyObject* (*initfunc)(void);
   };
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main)(Doc)(c-api)import.rst, line 299)

Unknown directive type "c:function".

```
.. c:function:: int PyImport_ExtendInittab(struct _inittab *newtab)

   Add a collection of modules to the table of built-in modules.  The *newtab*
   array must end with a sentinel entry which contains ``NULL`` for the :attr:`name`
   field; failure to provide the sentinel value can result in a memory fault.
   Returns ``0`` on success or ``-1`` if insufficient memory could be allocated to
   extend the internal table.  In the event of failure, no modules are added to the
   internal table.  This must be called before :c:func:`Py_Initialize`.

   If Python is initialized multiple times, :c:func:`PyImport_AppendInittab` or
   :c:func:`PyImport_ExtendInittab` must be called before each Python
   initialization.
```