Module Objects

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

Unknown directive type "index".

```
.. index:: object: module
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 11)

Unknown directive type "c:var".

```
.. c:var:: PyTypeObject PyModule_Type

   .. index:: single: ModuleType (in module types)

   This instance of :c:type:`PyTypeObject` represents the Python module type.  This
   is exposed to Python programs as ``types.ModuleType``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 19)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_Check(PyObject *p)

   Return true if *p* is a module object, or a subtype of a module object.
   This function always succeeds.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 25)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_CheckExact(PyObject *p)

   Return true if *p* is a module object, but not a subtype of
   :c:data:`PyModule_Type`.  This function always succeeds.
```

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

Unknown directive type "c:function".

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

   .. index::
      single: __name__ (module attribute)
      single: __doc__ (module attribute)
      single: __file__ (module attribute)
      single: __package__ (module attribute)
      single: __loader__ (module attribute)

   Return a new module object with the :attr:`__name__` attribute set to *name*.
   The module's :attr:`__name__`, :attr:`__doc__`, :attr:`__package__`, and
   :attr:`__loader__` attributes are filled in (all but :attr:`__name__` are set
   to ``None``); the caller is responsible for providing a :attr:`__file__`
   attribute.

   .. versionadded:: 3.3

   .. versionchanged:: 3.4
      :attr:`__package__` and :attr:`__loader__` are set to ``None``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 52)

Unknown directive type "c:function".

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

   Similar to :c:func:`PyModule_NewObject`, 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]module.rst, line 58)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyModule_GetDict(PyObject *module)

   .. index:: single: __dict__ (module attribute)

   Return the dictionary object that implements *module*'s namespace; this object
   is the same as the :attr:`~object.__dict__` attribute of the module object.
   If *module* is not a module object (or a subtype of a module object),
   :exc:`SystemError` is raised and ``NULL`` is returned.

   It is recommended extensions use other :c:func:`PyModule_\*` and
   :c:func:`PyObject_\*` functions rather than directly manipulate a module's
   :attr:`~object.__dict__`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 72)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyModule_GetNameObject(PyObject *module)

   .. index::
      single: __name__ (module attribute)
      single: SystemError (built-in exception)

   Return *module*'s :attr:`__name__` value.  If the module does not provide one,
   or if it is not a string, :exc:`SystemError` is raised and ``NULL`` is returned.

   .. versionadded:: 3.3
```

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

Unknown directive type "c:function".

```
.. c:function:: const char* PyModule_GetName(PyObject *module)

   Similar to :c:func:`PyModule_GetNameObject` but return the name encoded to
   ``'utf-8'``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 89)

Unknown directive type "c:function".

```
.. c:function:: void* PyModule_GetState(PyObject *module)

   Return the "state" of the module, that is, a pointer to the block of memory
   allocated at module creation time, or ``NULL``.  See
   :c:member:`PyModuleDef.m_size`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 96)

Unknown directive type "c:function".

```
.. c:function:: PyModuleDef* PyModule_GetDef(PyObject *module)

   Return a pointer to the :c:type:`PyModuleDef` struct from which the module was
```

```
created, or ``NULL`` if the module wasn't created from a definition.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 102)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyModule_GetFilenameObject(PyObject *module)

   .. index::
      single: __file__ (module attribute)
      single: SystemError (built-in exception)

   Return the name of the file from which *module* was loaded using *module*'s
   :attr:`__file__` attribute.  If this is not defined, or if it is not a
   unicode string, raise :exc:`SystemError` and return ``NULL``; otherwise return
   a reference to a Unicode object.

   .. versionadded:: 3.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 116)

Unknown directive type "c:function".

```
.. c:function:: const char* PyModule_GetFilename(PyObject *module)

   Similar to :c:func:`PyModule_GetFilenameObject` but return the filename
   encoded to 'utf-8'.

   .. deprecated:: 3.2
      :c:func:`PyModule_GetFilename` raises :c:type:`UnicodeEncodeError` on
      unencodable filenames, use :c:func:`PyModule_GetFilenameObject` instead.
```

Initializing C modules

Modules objects are usually created from extension modules (shared libraries which export an initialization function), or compiled-in modules (where the initialization function is added using :c:func:`PyImport_AppendInittab`). See :ref:`building` or :ref:`extending-with-embedding` for details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 131); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 131); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 131); *backlink*

Unknown interpreted text role "ref".

The initialization function can either pass a module definition instance to :c:func:`PyModule_Create`, and return the resulting module object, or request "multi-phase initialization" by returning the definition struct itself.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 136); *backlink*

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 140)

Unknown directive type "c:type".

```
.. c:type:: PyModuleDef

   The module definition struct, which holds all information needed to create
   a module object. There is usually only one statically initialized variable
   of this type for each module.
```

```
.. c:member:: PyModuleDef_Base m_base

   Always initialize this member to :const:`PyModuleDef_HEAD_INIT`.

.. c:member:: const char *m_name

   Name for the new module.

.. c:member:: const char *m_doc

   Docstring for the module; usually a docstring variable created with
   :c:macro:`PyDoc_STRVAR` is used.

.. c:member:: Py_ssize_t m_size

   Module state may be kept in a per-module memory area that can be
   retrieved with :c:func:`PyModule_GetState`, rather than in static globals.
   This makes modules safe for use in multiple sub-interpreters.

   This memory area is allocated based on *m_size* on module creation,
   and freed when the module object is deallocated, after the
   :c:member:`m_free` function has been called, if present.

   Setting ``m_size`` to ``-1`` means that the module does not support
   sub-interpreters, because it has global state.

   Setting it to a non-negative value means that the module can be
   re-initialized and specifies the additional amount of memory it requires
   for its state. Non-negative ``m_size`` is required for multi-phase
   initialization.

   See :PEP:`3121` for more details.

.. c:member:: PyMethodDef* m_methods

   A pointer to a table of module-level functions, described by
   :c:type:`PyMethodDef` values.  Can be ``NULL`` if no functions are present.

.. c:member:: PyModuleDef_Slot* m_slots

   An array of slot definitions for multi-phase initialization, terminated by
   a ``{0, NULL}`` entry.
   When using single-phase initialization, *m_slots* must be ``NULL``.

   .. versionchanged:: 3.5

      Prior to version 3.5, this member was always set to ``NULL``,
      and was defined as:

        .. c:member:: inquiry m_reload

.. c:member:: traverseproc m_traverse

   A traversal function to call during GC traversal of the module object, or
   ``NULL`` if not needed.

   This function is not called if the module state was requested but is not
   allocated yet. This is the case immediately after the module is created
   and before the module is executed (:c:data:`Py_mod_exec` function). More
   precisely, this function is not called if :c:member:`m_size` is greater
   than 0 and the module state (as returned by :c:func:`PyModule_GetState`)
   is ``NULL``.

   .. versionchanged:: 3.9
      No longer called before the module state is allocated.

.. c:member:: inquiry m_clear

   A clear function to call during GC clearing of the module object, or
   ``NULL`` if not needed.

   This function is not called if the module state was requested but is not
   allocated yet. This is the case immediately after the module is created
   and before the module is executed (:c:data:`Py_mod_exec` function). More
   precisely, this function is not called if :c:member:`m_size` is greater
   than 0 and the module state (as returned by :c:func:`PyModule_GetState`)
   is ``NULL``.

   Like :c:member:`PyTypeObject.tp_clear`, this function is not *always*
   called before a module is deallocated. For example, when reference
   counting is enough to determine that an object is no longer used,
   the cyclic garbage collector is not involved and
   :c:member:`~PyModuleDef.m_free` is called directly.

   .. versionchanged:: 3.9
      No longer called before the module state is allocated.

.. c:member:: freefunc m_free

   A function to call during deallocation of the module object, or ``NULL``
```

```
if not needed.

This function is not called if the module state was requested but is not
allocated yet. This is the case immediately after the module is created
and before the module is executed (:c:data:`Py_mod_exec` function). More
precisely, this function is not called if :c:member:`m_size` is greater
than 0 and the module state (as returned by :c:func:`PyModule_GetState`)
is ``NULL``.

.. versionchanged:: 3.9
   No longer called before the module state is allocated.
```

Single-phase initialization

The module initialization function may create and return the module object directly. This is referred to as "single-phase initialization", and uses one of the following two module creation functions:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 255)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyModule_Create(PyModuleDef *def)

   Create a new module object, given the definition in *def*.  This behaves
   like :c:func:`PyModule_Create2` with *module_api_version* set to
   :const:`PYTHON_API_VERSION`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 262)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyModule_Create2(PyModuleDef *def, int module_api_version)

   Create a new module object, given the definition in *def*, assuming the
   API version *module_api_version*.  If that version does not match the version
   of the running interpreter, a :exc:`RuntimeWarning` is emitted.

   .. note::

      Most uses of this function should be using :c:func:`PyModule_Create`
      instead; only use this if you are sure you need it.
```

Before it is returned from in the initialization function, the resulting module object is typically populated using functions like :c:func:`PyModule_AddObjectRef`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 273); *backlink*

Unknown interpreted text role "c:func".

Multi-phase initialization

An alternate way to specify extensions is to request "multi-phase initialization". Extension modules created this way behave more like Python modules: the initialization is split between the *creation phase*, when the module object is created, and the *execution phase*, when it is populated. The distinction is similar to the :py:meth:`__new__` and :py:meth:`__init__` methods of classes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 281); *backlink*

Unknown interpreted text role "py:meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 281); *backlink*

Unknown interpreted text role "py:meth".

Unlike modules created using single-phase initialization, these modules are not singletons: if the *sys.modules* entry is removed and the module is re-imported, a new module object is created, and the old module is subject to normal garbage collection -- as with Python modules. By default, multiple modules created from the same definition should be independent: changes to one should not affect the others. This means that all state should be specific to the module object (using e.g. using :c:func:`PyModule_GetState`), or its contents (such as the module's :attr:`__dict__` or individual classes created with :c:func:`PyType_FromSpec`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 288); backlink

Unknown interpreted text role "c:func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 288); backlink

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 288); backlink

Unknown interpreted text role "c:func".

All modules created using multi-phase initialization are expected to support :ref:`sub-interpreters <sub-interpreter-support>`. Making sure multiple modules are independent is typically enough to achieve this.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 298); backlink

Unknown interpreted text role "ref".

To request multi-phase initialization, the initialization function (PyInit_modulename) returns a :c:type:`PyModuleDef` instance with non-empty :c:member:`~PyModuleDef.m_slots`. Before it is returned, the `PyModuleDef` instance must be initialized with the following function:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 302); backlink

Unknown interpreted text role "c:type".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 302); backlink

Unknown interpreted text role "c:member".

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

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyModuleDef_Init(PyModuleDef *def)

   Ensures a module definition is a properly initialized Python object that
   correctly reports its type and reference count.

   Returns *def* cast to ``PyObject*``, or ``NULL`` if an error occurred.

   .. versionadded:: 3.5
```

The *m_slots* member of the module definition must point to an array of `PyModuleDef_Slot` structures:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 319)

Unknown directive type "c:type".

```
.. c:type:: PyModuleDef_Slot

   .. c:member:: int slot

      A slot ID, chosen from the available values explained below.

   .. c:member:: void* value

      Value of the slot, whose meaning depends on the slot ID.

   .. versionadded:: 3.5
```

The *m_slots* array must be terminated by a slot with id 0.

The available slot types are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 335)

Unknown directive type "c:macro".

```
.. c:macro:: Py_mod_create

   Specifies a function that is called to create the module object itself.
   The *value* pointer of this slot must point to a function of the signature:

   .. c:function:: PyObject* create_module(PyObject *spec, PyModuleDef *def)

   The function receives a :py:class:`~importlib.machinery.ModuleSpec`
   instance, as defined in :PEP:`451`, and the module definition.
   It should return a new module object, or set an error
   and return ``NULL``.

   This function should be kept minimal. In particular, it should not
   call arbitrary Python code, as trying to import the same module again may
   result in an infinite loop.

   Multiple ``Py_mod_create`` slots may not be specified in one module
   definition.

   If ``Py_mod_create`` is not specified, the import machinery will create
   a normal module object using :c:func:`PyModule_New`. The name is taken from
   *spec*, not the definition, to allow extension modules to dynamically adjust
   to their place in the module hierarchy and be imported under different
   names through symlinks, all while sharing a single module definition.

   There is no requirement for the returned object to be an instance of
   :c:type:`PyModule_Type`. Any type can be used, as long as it supports
   setting and getting import-related attributes.
   However, only ``PyModule_Type`` instances may be returned if the
   ``PyModuleDef`` has non-``NULL`` ``m_traverse``, ``m_clear``,
   ``m_free``; non-zero ``m_size``; or slots other than ``Py_mod_create``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 367)

Unknown directive type "c:macro".

```
.. c:macro:: Py_mod_exec

   Specifies a function that is called to *execute* the module.
   This is equivalent to executing the code of a Python module: typically,
   this function adds classes and constants to the module.
   The signature of the function is:

   .. c:function:: int exec_module(PyObject* module)

   If multiple ``Py_mod_exec`` slots are specified, they are processed in the
   order they appear in the *m_slots* array.
```

See PEP 489 for more details on multi-phase initialization.

Low-level module creation functions

The following functions are called under the hood when using multi-phase initialization. They can be used directly, for example when creating module objects dynamically. Note that both `PyModule_FromDefAndSpec` and `PyModule_ExecDef` must be called to fully initialize a module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 389)

Unknown directive type "c:function".

```
.. c:function:: PyObject * PyModule_FromDefAndSpec(PyModuleDef *def, PyObject *spec)

   Create a new module object, given the definition in *module* and the
   ModuleSpec *spec*.  This behaves like :c:func:`PyModule_FromDefAndSpec2`
   with *module_api_version* set to :const:`PYTHON_API_VERSION`.

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 397)

Unknown directive type "c:function".

```
.. c:function:: PyObject * PyModule_FromDefAndSpec2(PyModuleDef *def, PyObject *spec, int module_api_ve

   Create a new module object, given the definition in *module* and the
   ModuleSpec *spec*, assuming the API version *module_api_version*.
   If that version does not match the version of the running interpreter,
   a :exc:`RuntimeWarning` is emitted.

   .. note::
```

```
   Most uses of this function should be using :c:func:`PyModule_FromDefAndSpec`
   instead; only use this if you are sure you need it.

.. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 411)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_ExecDef(PyObject *module, PyModuleDef *def)

   Process any execution slots (:c:data:`Py_mod_exec`) given in *def*.

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 417)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_SetDocString(PyObject *module, const char *docstring)

   Set the docstring for *module* to *docstring*.
   This function is called automatically when creating a module from
   ``PyModuleDef``, using either ``PyModule_Create`` or
   ``PyModule_FromDefAndSpec``.

   .. versionadded:: 3.5
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 426)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddFunctions(PyObject *module, PyMethodDef *functions)

   Add the functions from the ``NULL`` terminated *functions* array to *module*.
   Refer to the :c:type:`PyMethodDef` documentation for details on individual
   entries (due to the lack of a shared module namespace, module level
   "functions" implemented in C typically receive the module as their first
   parameter, making them similar to instance methods on Python classes).
   This function is called automatically when creating a module from
   ``PyModuleDef``, using either ``PyModule_Create`` or
   ``PyModule_FromDefAndSpec``.

   .. versionadded:: 3.5
```

Support functions

The module initialization function (if using single phase initialization) or a function called from a module execution slot (if using multi-phase initialization), can use the following functions to help initialize the module state:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 447)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddObjectRef(PyObject *module, const char *name, PyObject *value)

   Add an object to *module* as *name*.  This is a convenience function which
   can be used from the module's initialization function.

   On success, return ``0``. On error, raise an exception and return ``-1``.

   Return ``NULL`` if *value* is ``NULL``. It must be called with an exception
   raised in this case.

   Example usage::

       static int
       add_spam(PyObject *module, int value)
       {
           PyObject *obj = PyLong_FromLong(value);
           if (obj == NULL) {
               return -1;
           }
           int res = PyModule_AddObjectRef(module, "spam", obj);
           Py_DECREF(obj);
           return res;
       }
```

```
   The example can also be written without checking explicitly if *obj* is
   ``NULL``::

       static int
       add_spam(PyObject *module, int value)
       {
           PyObject *obj = PyLong_FromLong(value);
           int res = PyModule_AddObjectRef(module, "spam", obj);
           Py_XDECREF(obj);
           return res;
        }

   Note that ``Py_XDECREF()`` should be used instead of ``Py_DECREF()`` in
   this case, since *obj* can be ``NULL``.

   .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 489)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddObject(PyObject *module, const char *name, PyObject *value)

   Similar to :c:func:`PyModule_AddObjectRef`, but steals a reference to
   *value* on success (if it returns ``0``).

   The new :c:func:`PyModule_AddObjectRef` function is recommended, since it is
   easy to introduce reference leaks by misusing the
   :c:func:`PyModule_AddObject` function.

   .. note::

      Unlike other functions that steal references, ``PyModule_AddObject()``
      only decrements the reference count of *value* **on success**.

      This means that its return value must be checked, and calling code must
      :c:func:`Py_DECREF` *value* manually on error.

   Example usage::

      static int
      add_spam(PyObject *module, int value)
      {
          PyObject *obj = PyLong_FromLong(value);
          if (obj == NULL) {
              return -1;
          }
          if (PyModule_AddObject(module, "spam", obj) < 0) {
              Py_DECREF(obj);
              return -1;
          }
          // PyModule_AddObject() stole a reference to obj:
          // Py_DECREF(obj) is not needed here
          return 0;
      }

   The example can also be written without checking explicitly if *obj* is
   ``NULL``::

      static int
      add_spam(PyObject *module, int value)
      {
          PyObject *obj = PyLong_FromLong(value);
          if (PyModule_AddObject(module, "spam", obj) < 0) {
              Py_XDECREF(obj);
              return -1;
          }
          // PyModule_AddObject() stole a reference to obj:
          // Py_DECREF(obj) is not needed here
          return 0;
      }

   Note that ``Py_XDECREF()`` should be used instead of ``Py_DECREF()`` in
   this case, since *obj* can be ``NULL``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 544)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddIntConstant(PyObject *module, const char *name, long value)
```

```
   Add an integer constant to *module* as *name*.  This convenience function can be
   used from the module's initialization function. Return ``-1`` on error, ``0`` on
   success.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 551)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddStringConstant(PyObject *module, const char *name, const char *value)

   Add a string constant to *module* as *name*.  This convenience function can be
   used from the module's initialization function.  The string *value* must be
   ``NULL``-terminated.  Return ``-1`` on error, ``0`` on success.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 558)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddIntMacro(PyObject *module, macro)

   Add an int constant to *module*. The name and the value are taken from
   *macro*. For example ``PyModule_AddIntMacro(module, AF_INET)`` adds the int
   constant *AF_INET* with the value of *AF_INET* to *module*.
   Return ``-1`` on error, ``0`` on success.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 566)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddStringMacro(PyObject *module, macro)

   Add a string constant to *module*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 570)

Unknown directive type "c:function".

```
.. c:function:: int PyModule_AddType(PyObject *module, PyTypeObject *type)

   Add a type object to *module*.
   The type object is finalized by calling internally :c:func:`PyType_Ready`.
   The name of the type object is taken from the last component of
   :c:member:`~PyTypeObject.tp_name` after dot.
   Return ``-1`` on error, ``0`` on success.

   .. versionadded:: 3.9
```

Module lookup

Single-phase initialization creates singleton modules that can be looked up in the context of the current interpreter. This allows the module object to be retrieved later with only a reference to the module definition.

These functions will not work on modules created using multi-phase initialization, since multiple such modules can be created from a single definition.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 591)

Unknown directive type "c:function".

```
.. c:function:: PyObject* PyState_FindModule(PyModuleDef *def)

   Returns the module object that was created from *def* for the current interpreter.
   This method requires that the module object has been attached to the interpreter state with
   :c:func:`PyState_AddModule` beforehand. In case the corresponding module object is not
   found or has not been attached to the interpreter state yet, it returns ``NULL``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 598)

Unknown directive type "c:function".

```
.. c:function:: int PyState_AddModule(PyObject *module, PyModuleDef *def)

   Attaches the module object passed to the function to the interpreter state. This allows
   the module object to be accessible via :c:func:`PyState_FindModule`.

   Only effective on modules created using single-phase initialization.

   Python calls ``PyState_AddModule`` automatically after importing a module,
   so it is unnecessary (but harmless) to call it from module initialization
   code. An explicit call is needed only if the module's own init code
   subsequently calls ``PyState_FindModule``.
   The function is mainly intended for implementing alternative import
   mechanisms (either by calling it directly, or by referring to its
   implementation for details of the required state updates).

   The caller must hold the GIL.

   Return 0 on success or -1 on failure.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\[cpython-main][Doc][c-api]module.rst, line 619)

Unknown directive type "c:function".

```
.. c:function:: int PyState_RemoveModule(PyModuleDef *def)

   Removes the module object created from *def* from the interpreter state.
   Return 0 on success or -1 on failure.

   The caller must hold the GIL.

   .. versionadded:: 3.3
```