Introduction

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

Unknown directive type "highlight".

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

The Application Programmer's Interface to Python gives C and C++ programmers access to the Python interpreter at a variety of levels. The API is equally usable from C++, but for brevity it is generally referred to as the Python/C API. There are two fundamentally different reasons for using the Python/C API. The first reason is to write *extension modules* for specific purposes; these are C modules that extend the Python interpreter. This is probably the most common use. The second reason is to use Python as a component in a larger application; this technique is generally referred to as :dfn:`embedding` Python in an application.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 10); *backlink*

Unknown interpreted text role "dfn".

Writing an extension module is a relatively well-understood process, where a "cookbook" approach works well. There are several tools that automate the process to some extent. While people have embedded Python in other applications since its early existence, the process of embedding Python is less straightforward than writing an extension.

Many API functions are useful independent of whether you're embedding or extending Python; moreover, most applications that embed Python will need to provide a custom extension as well, so it's probably a good idea to become familiar with writing an extension before attempting to embed Python in a real application.

Coding standards

If you're writing C code for inclusion in CPython, you **must** follow the guidelines and standards defined in PEP 7. These guidelines apply regardless of the version of Python you are contributing to. Following these conventions is not necessary for your own third party extension modules, unless you eventually expect to contribute them to Python.

Include Files

All function, type and macro definitions needed to use the Python/C API are included in your code by the following line:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
```

This implies inclusion of the following standard headers: `<stdio.h>`, `<string.h>`, `<errno.h>`, `<limits.h>`, `<assert.h>` and `<stdlib.h>` (if available).

Note

Since Python may define some pre-processor definitions which affect the standard headers on some systems, you *must* include :file:`Python.h` before any standard headers are included.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 60); *backlink*

Unknown interpreted text role "file".

It is recommended to always define `PY_SSIZE_T_CLEAN` before including `Python.h`. See :ref:`arg-parsing` for a description of this macro.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 64); *backlink*

Unknown interpreted text role "ref".

All user visible names defined by Python.h (except those defined by the included standard headers) have one of the prefixes `Py` or `_Py`. Names beginning with `_Py` are for internal use by the Python implementation and should not be used by extension writers. Structure member names do not have a reserved prefix.

> **Note**
>
> User code should never define names that begin with `Py` or `_Py`. This confuses the reader, and jeopardizes the portability of the user code to future Python versions, which may define additional names beginning with one of these prefixes.

The header files are typically installed with Python. On Unix, these are located in the directories :file:`{prefix}/include/pythonversion/` and :file:`{exec_prefix}/include/pythonversion/`, where :envvar:`prefix` and :envvar:`exec_prefix` are defined by the corresponding parameters to Python's :program:`configure` script and *version* is `'%d.%d' % sys.version_info[:2]`. On Windows, the headers are installed in :file:`{prefix}/include`, where :envvar:`prefix` is the installation directory specified to the installer.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "program".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 79); *backlink*

Unknown interpreted text role "envvar".

To include the headers, place both directories (if different) on your compiler's search path for includes. Do *not* place the parent directories on the search path and then use `#include <pythonX.Y/Python.h>`; this will break on multi-platform builds since the platform independent headers under :envvar:`prefix` include the platform specific headers from :envvar:`exec_prefix`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 88); *backlink*

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 88); *backlink*

Unknown interpreted text role "envvar".

C++ users should note that although the API is defined entirely using C, the header files properly declare the entry points to be extern "C". As a result, there is no need to do anything special to use the API from C++.

Useful macros

Several useful macros are defined in the Python header files. Many are defined closer to where they are useful (e.g. :c:macro:`Py_RETURN_NONE`). Others of a more general utility are defined here. This is not necessarily a complete listing.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst, line 103); ***backlink***

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst, line 108)

Unknown directive type "c:macro".

```
.. c:macro:: Py_ABS(x)

   Return the absolute value of ``x``.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst, line 114)

Unknown directive type "c:macro".

```
.. c:macro:: Py_ALWAYS_INLINE

   Ask the compiler to always inline a static inline function. The compiler can
   ignore it and decides to not inline the function.

   It can be used to inline performance critical static inline functions when
   building Python in debug mode with function inlining disabled. For example,
   MSC disables function inlining when building in debug mode.

   Marking blindly a static inline function with Py_ALWAYS_INLINE can result in
   worse performances (due to increased code size for example). The compiler is
   usually smarter than the developer for the cost/benefit analysis.

   If Python is :ref:`built in debug mode <debug-build>` (if the ``Py_DEBUG``
   macro is defined), the :c:macro:`Py_ALWAYS_INLINE` macro does nothing.

   It must be specified before the function return type. Usage::

       static inline Py_ALWAYS_INLINE int random(void) { return 4; }

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst, line 136)

Unknown directive type "c:macro".

```
.. c:macro:: Py_CHARMASK(c)

   Argument must be a character or an integer in the range [-128, 127] or [0,
   255].  This macro returns ``c`` cast to an ``unsigned char``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst, line 141)

Unknown directive type "c:macro".

```
.. c:macro:: Py_DEPRECATED(version)

   Use this for deprecated declarations.  The macro must be placed before the
   symbol name.

   Example::
      Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);

   .. versionchanged:: 3.8
      MSVC support was added.
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 153)

Unknown directive type "c:macro".

```
.. c:macro:: Py_GETENV(s)

   Like ``getenv(s)``, but returns ``NULL`` if :option:`-E` was passed on the
   command line (i.e. if ``Py_IgnoreEnvironmentFlag`` is set).
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 158)

Unknown directive type "c:macro".

```
.. c:macro:: Py_MAX(x, y)

   Return the maximum value between ``x`` and ``y``.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 164)

Unknown directive type "c:macro".

```
.. c:macro:: Py_MEMBER_SIZE(type, member)

   Return the size of a structure (``type``) ``member`` in bytes.

   .. versionadded:: 3.6
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 170)

Unknown directive type "c:macro".

```
.. c:macro:: Py_MIN(x, y)

   Return the minimum value between ``x`` and ``y``.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 176)

Unknown directive type "c:macro".

```
.. c:macro:: Py_NO_INLINE

   Disable inlining on a function. For example, it reduces the C stack
   consumption: useful on LTO+PGO builds which heavily inline code (see
   :issue:`33720`).

   Usage::

       Py_NO_INLINE static int random(void) { return 4; }

   .. versionadded:: 3.11
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 188)

Unknown directive type "c:macro".

```
.. c:macro:: Py_STRINGIFY(x)

   Convert ``x`` to a C string.  E.g. ``Py_STRINGIFY(123)`` returns
   ``"123"``.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 195)

Unknown directive type "c:macro".

```
.. c:macro:: Py_UNREACHABLE()

   Use this when you have a code path that cannot be reached by design.
   For example, in the ``default:`` clause in a ``switch`` statement for which
   all possible values are covered in ``case`` statements.  Use this in places
   where you might be tempted to put an ``assert(0)`` or ``abort()`` call.

   In release mode, the macro helps the compiler to optimize the code, and
   avoids a warning about unreachable code.  For example, the macro is
   implemented with ``__builtin_unreachable()`` on GCC in release mode.

   A use for ``Py_UNREACHABLE()`` is following a call a function that
   never returns but that is not declared :c:macro:`_Py_NO_RETURN`.

   If a code path is very unlikely code but can be reached under exceptional
   case, this macro must not be used.  For example, under low memory condition
   or if a system call returns a value out of the expected range.  In this
   case, it's better to report the error to the caller.  If the error cannot
   be reported to caller, :c:func:`Py_FatalError` can be used.

   .. versionadded:: 3.7
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 217)

Unknown directive type "c:macro".

```
.. c:macro:: Py_UNUSED(arg)

   Use this for unused arguments in a function definition to silence compiler
   warnings. Example: ``int func(int a, int Py_UNUSED(b)) { return a; }``.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 224)

Unknown directive type "c:macro".

```
.. c:macro:: PyDoc_STRVAR(name, str)

   Creates a variable with name ``name`` that can be used in docstrings.
   If Python is built without docstrings, the value will be empty.

   Use :c:macro:`PyDoc_STRVAR` for docstrings to support building
   Python without docstrings, as specified in :pep:`7`.

   Example::

      PyDoc_STRVAR(pop_doc, "Remove and return the rightmost element.");

      static PyMethodDef deque_methods[] = {
          // ...
          {"pop", (PyCFunction)deque_pop, METH_NOARGS, pop_doc},
          // ...
      }
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 242)

Unknown directive type "c:macro".

```
.. c:macro:: PyDoc_STR(str)

   Creates a docstring for the given input string or an empty string
   if docstrings are disabled.

   Use :c:macro:`PyDoc_STR` in specifying docstrings to support
   building Python without docstrings, as specified in :pep:`7`.

   Example::

      static PyMethodDef pysqlite_row_methods[] = {
          {"keys", (PyCFunction)pysqlite_row_keys, METH_NOARGS,
              PyDoc_STR("Returns the keys of the row.")},
          {NULL, NULL}
      };
```

Objects, Types and Reference Counts

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 264)

Unknown directive type "index".

```
.. index:: object: type
```

Most Python/C API functions have one or more arguments as well as a return value of type :c:type:`PyObject*`. This type is a pointer to an opaque data type representing an arbitrary Python object. Since all Python object types are treated the same way by the Python language in most situations (e.g., assignments, scope rules, and argument passing), it is only fitting that they should be represented by a single C type. Almost all Python objects live on the heap: you never declare an automatic or static variable of type :c:type:`PyObject`, only pointer variables of type :c:type:`PyObject*` can be declared. The sole exception are the type objects; since these must never be deallocated, they are typically static :c:type:`PyTypeObject` objects.

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

Unknown interpreted text role "c:type".

All Python objects (even Python integers) have a :dfn:`type` and a :dfn:`reference count`. An object's type determines what kind of object it is (e.g., an integer, a list, or a user-defined function; there are many more as explained in :ref:`types`). For each of the well-known types there is a macro to check whether an object is of that type; for instance, `PyList_Check(a)` is true if (and only if) the object pointed to by *a* is a Python list.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 277); *backlink*

Unknown interpreted text role "dfn".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 277); *backlink*

Unknown interpreted text role "dfn".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 277); *backlink*

Unknown interpreted text role "ref".

Reference Counts

The reference count is important because today's computers have a finite (and often severely limited) memory size; it counts how many different places there are that have a reference to an object. Such a place could be another object, or a global (or static) C variable, or a local variable in some C function. When an object's reference count becomes zero, the object is deallocated. If it contains references to other objects, their reference count is decremented. Those other objects may be deallocated in turn, if this decrement makes their reference count become zero, and so on. (There's an obvious problem with objects that reference each other here; for now, the solution is "don't do that.")

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 301)

Unknown directive type "index".

```
.. index::
   single: Py_INCREF()
   single: Py_DECREF()
```

Reference counts are always manipulated explicitly. The normal way is to use the macro :c:func:`Py_INCREF` to increment an object's reference count by one, and :c:func:`Py_DECREF` to decrement it by one. The :c:func:`Py_DECREF` macro is considerably more complex than the incref one, since it must check whether the reference count becomes zero and then cause the object's deallocator to be called. The deallocator is a function pointer contained in the object's type structure. The type-specific deallocator takes care of decrementing the reference counts for other objects contained in the object if this is a compound object type, such as a list, as well as performing any additional finalization that's needed. There's no chance that the reference count can overflow; at least as many bits are used to hold the reference count as there are distinct memory locations in virtual memory (assuming `sizeof(Py_ssize_t) >= sizeof(void*)`). Thus, the reference count increment is a simple operation.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 305); *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)intro.rst`, line 305); *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)intro.rst`, line 305); *backlink*

Unknown interpreted text role "c:func".

It is not necessary to increment an object's reference count for every local variable that contains a pointer to an object. In theory, the object's reference count goes up by one when the variable is made to point to it and it goes down by one when the variable goes out of scope. However, these two cancel each other out, so at the end the reference count hasn't changed. The only real reason to use the reference count is to prevent the object from being deallocated as long as our variable is pointing to it. If we know that there is at least one other reference to the object that lives at least as long as our variable, there is no need to increment the reference count temporarily. An important situation where this arises is in objects that are passed as arguments to C functions in an extension module that are called from Python; the call mechanism guarantees to hold a reference to every argument for the duration of the call.

However, a common pitfall is to extract an object from a list and hold on to it for a while without incrementing its reference count. Some other operation might conceivably remove the object from the list, decrementing its reference count and possibly deallocating it. The real danger is that innocent-looking operations may invoke arbitrary Python code which could do this; there is a code path which allows control to flow back to the user from a :c:func:`Py_DECREF`, so almost any operation is potentially dangerous.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 333); *backlink*

Unknown interpreted text role "c:func".

A safe approach is to always use the generic operations (functions whose name begins with `PyObject_`, `PyNumber_`, `PySequence_` or `PyMapping_`). These operations always increment the reference count of the object they return. This leaves the caller with the responsibility to call :c:func:`Py_DECREF` when they are done with the result; this soon becomes second nature.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 341); *backlink*

Unknown interpreted text role "c:func".

Reference Count Details

The reference count behavior of functions in the Python/C API is best explained in terms of *ownership of references*. Ownership pertains to references, never to objects (objects are not owned: they are always shared). "Owning a reference" means being responsible for calling Py_DECREF on it when the reference is no longer needed. Ownership can also be transferred, meaning that the code that receives ownership of the reference then becomes responsible for eventually decref'ing it by calling :c:func:`Py_DECREF` or :c:func:`Py_XDECREF` when it's no longer needed---or passing on this responsibility (usually to its caller). When a function passes ownership of a reference on to its caller, the caller is said to receive a *new* reference. When no ownership is transferred, the caller is said to *borrow* the reference. Nothing needs to be done for a :term:`borrowed reference`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 353); *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)intro.rst`, line 353); *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)intro.rst`, line 353); *backlink*

Unknown interpreted text role "term".

Conversely, when a calling function passes in a reference to an object, there are two possibilities: the function *steals* a reference to the object, or it does not. *Stealing a reference* means that when you pass a reference to a function, that function assumes that it now owns that reference, and you are not responsible for it any longer.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 372)

Unknown directive type "index".

```
.. index::
   single: PyList_SetItem()
   single: PyTuple_SetItem()
```

Few functions steal references; the two notable exceptions are :c:func:`PyList_SetItem` and :c:func:`PyTuple_SetItem`, which steal a reference to the item (but not to the tuple or list into which the item is put!). These functions were designed to steal a reference because of a common idiom for populating a tuple or list with newly created objects; for example, the code to create the tuple `(1, 2, "three")` could look like this (forgetting about error handling for the moment; a better way to code this is shown below):

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 376); *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)intro.rst`, line 376); *backlink*

Unknown interpreted text role "c:func".

```
PyObject *t;

t = PyTuple_New(3);
PyTuple_SetItem(t, 0, PyLong_FromLong(1L));
PyTuple_SetItem(t, 1, PyLong_FromLong(2L));
PyTuple_SetItem(t, 2, PyUnicode_FromString("three"));
```

Here, :c:func:`PyLong_FromLong` returns a new reference which is immediately stolen by :c:func:`PyTuple_SetItem`. When you want to keep using an object although the reference to it will be stolen, use :c:func:`Py_INCREF` to grab another reference before calling the reference-stealing function.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 391); *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)intro.rst`, line 391); *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)intro.rst`, line 391); *backlink*

Unknown interpreted text role "c:func".

Incidentally, :c:func:`PyTuple_SetItem` is the *only* way to set tuple items; :c:func:`PySequence_SetItem` and :c:func:`PyObject_SetItem` refuse to do this since tuples are an immutable data type. You should only use :c:func:`PyTuple_SetItem` for tuples that you are creating yourself.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 396); *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)intro.rst`, line 396); *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)intro.rst`, line 396); *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)intro.rst`, line 396); *backlink*

Unknown interpreted text role "c:func".

Equivalent code for populating a list can be written using :c:func:`PyList_New` and :c:func:`PyList_SetItem`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 401); *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)intro.rst`, line 401); *backlink*

Unknown interpreted text role "c:func".

However, in practice, you will rarely use these ways of creating and populating a tuple or list. There's a generic function, :c:func:`Py_BuildValue`, that can create most common objects from C values, directed by a :dfn:`format string`. For example, the above two blocks of code could be replaced by the following (which also takes care of the error checking):

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 404); *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)intro.rst`, line 404); *backlink*

Unknown interpreted text role "dfn".

```
PyObject *tuple, *list;

tuple = Py_BuildValue("(iis)", 1, 2, "three");
list = Py_BuildValue("[iis]", 1, 2, "three");
```

It is much more common to use :c:func:`PyObject_SetItem` and friends with items whose references you are only borrowing, like arguments that were passed in to the function you are writing. In that case, their behaviour regarding reference counts is much saner, since you don't have to increment a reference count so you can give a reference away ("have it be stolen"). For example, this function sets all items of a list (actually, any mutable sequence) to a given item:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 415); *backlink*

Unknown interpreted text role "c:func".

```
int
set_all(PyObject *target, PyObject *item)
{
    Py_ssize_t i, n;

    n = PyObject_Length(target);
    if (n < 0)
        return -1;
    for (i = 0; i < n; i++) {
        PyObject *index = PyLong_FromSsize_t(i);
        if (!index)
            return -1;
        if (PyObject_SetItem(target, index, item) < 0) {
            Py_DECREF(index);
            return -1;
        }
        Py_DECREF(index);
    }
    return 0;
}
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 443)

Unknown directive type "index".

```
.. index:: single: set_all()
```

The situation is slightly different for function return values. While passing a reference to most functions does not change your ownership responsibilities for that reference, many functions that return a reference to an object give you ownership of the reference. The reason is simple: in many cases, the returned object is created on the fly, and the reference you get is the only reference to the object. Therefore, the generic functions that return object references, like :c:func:`PyObject_GetItem` and :c:func:`PySequence_GetItem`, always return a new reference (the caller becomes the owner of the reference).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 445); *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)intro.rst`, line 445); *backlink*

Unknown interpreted text role "c:func".

It is important to realize that whether you own a reference returned by a function depends on which function you call only --- *the plumage* (the type of the object passed as an argument to the function) *doesn't enter into it!* Thus, if you extract an item from a list using :c:func:`PyList_GetItem`, you don't own the reference --- but if you obtain the same item from the same list using :c:func:`PySequence_GetItem` (which happens to take exactly the same arguments), you do own a reference to the returned object.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\ (cpython-main) (Doc) (c-api) intro.rst, line 454); *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) intro.rst, line 454); *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) intro.rst, line 462)

Unknown directive type "index".

```
.. index::
   single: PyList_GetItem()
   single: PySequence_GetItem()
```

Here is an example of how you could write a function that computes the sum of the items in a list of integers; once using :c:func:`PyList_GetItem`, and once using :c:func:`PySequence_GetItem`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\ (cpython-main) (Doc) (c-api) intro.rst, line 466); *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) intro.rst, line 466); *backlink*

Unknown interpreted text role "c:func".

```
long
sum_list(PyObject *list)
{
    Py_ssize_t i, n;
    long total = 0, value;
    PyObject *item;

    n = PyList_Size(list);
    if (n < 0)
        return -1; /* Not a list */
    for (i = 0; i < n; i++) {
        item = PyList_GetItem(list, i); /* Can't fail */
        if (!PyLong_Check(item)) continue; /* Skip non-integers */
        value = PyLong_AsLong(item);
        if (value == -1 && PyErr_Occurred())
            /* Integer too big to fit in a C long, bail out */
            return -1;
        total += value;
    }
    return total;
}
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\ (cpython-main) (Doc) (c-api) intro.rst, line 492)

Unknown directive type "index".

```
.. index:: single: sum_list()
```

```
long
sum_sequence(PyObject *sequence)
{
    Py_ssize_t i, n;
    long total = 0, value;
    PyObject *item;
    n = PySequence_Length(sequence);
    if (n < 0)
        return -1; /* Has no length */
    for (i = 0; i < n; i++) {
        item = PySequence_GetItem(sequence, i);
        if (item == NULL)
```

```
            return -1; /* Not a sequence, or other failure */
        if (PyLong_Check(item)) {
            value = PyLong_AsLong(item);
            Py_DECREF(item);
            if (value == -1 && PyErr_Occurred())
                /* Integer too big to fit in a C long, bail out */
                return -1;
            total += value;
        }
        else {
            Py_DECREF(item); /* Discard reference ownership */
        }
    }
    return total;
}
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 524)

Unknown directive type "index".

```
.. index:: single: sum_sequence()
```

Types

There are few other data types that play a significant role in the Python/C API; most are simple C types such as :c:type:`int`, :c:type:`long`, :c:type:`double` and :c:type:`char*`. A few structure types are used to describe static tables used to list the functions exported by a module or the data attributes of a new object type, and another is used to describe the value of a complex number. These will be discussed together with the functions that use them.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 532); *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)intro.rst`, line 532); *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)intro.rst`, line 532); *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)intro.rst`, line 532); *backlink*

Unknown interpreted text role "c:type".

Exceptions

The Python programmer only needs to deal with exceptions if specific error handling is required; unhandled exceptions are automatically propagated to the caller, then to the caller's caller, and so on, until they reach the top-level interpreter, where they are reported to the user accompanied by a stack traceback.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 552)

Unknown directive type "index".

```
.. index:: single: PyErr_Occurred()
```

For C programmers, however, error checking always has to be explicit. All functions in the Python/C API can raise exceptions, unless an explicit claim is made otherwise in a function's documentation. In general, when a function encounters an error, it sets an exception, discards any object references that it owns, and returns an error indicator. If not documented otherwise, this indicator is either `NULL` or `-1`, depending on the function's return type. A few functions return a Boolean true/false result, with false indicating an error. Very few functions return no explicit error indicator or have an ambiguous return value, and require explicit testing for errors with :c:func:`PyErr_Occurred`. These exceptions are always explicitly documented.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 554); *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)intro.rst`, line 565)

Unknown directive type "index".

```
.. index::
   single: PyErr_SetString()
   single: PyErr_Clear()
```

Exception state is maintained in per-thread storage (this is equivalent to using global storage in an unthreaded application). A thread can be in one of two states: an exception has occurred, or not. The function :c:func:`PyErr_Occurred` can be used to check for this: it returns a borrowed reference to the exception type object when an exception has occurred, and `NULL` otherwise. There are a number of functions to set the exception state: :c:func:`PyErr_SetString` is the most common (though not the most general) function to set the exception state, and :c:func:`PyErr_Clear` clears the exception state.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 569); *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)intro.rst`, line 569); *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)intro.rst`, line 569); *backlink*

Unknown interpreted text role "c:func".

The full exception state consists of three objects (all of which can be `NULL`): the exception type, the corresponding exception value, and the traceback. These have the same meanings as the Python result of `sys.exc_info()`; however, they are not the same: the Python objects represent the last exception being handled by a Python :keyword:`try` ... :keyword:`except` statement, while the C level exception state only exists while an exception is being passed on between C functions until it reaches the Python bytecode interpreter's main loop, which takes care of transferring it to `sys.exc_info()` and friends.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 579); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 579); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 589)

Unknown directive type "index".

```
.. index:: single: exc_info() (in module sys)
```

Note that starting with Python 1.5, the preferred, thread-safe way to access the exception state from Python code is to call the function :func:`sys.exc_info`, which returns the per-thread exception state for Python code. Also, the semantics of both ways to access the exception state have changed so that a function which catches an exception will save and restore its thread's exception state so as to preserve the exception state of its caller. This prevents common bugs in exception handling code caused by an innocent-looking function overwriting the exception being handled; it also reduces the often unwanted lifetime extension for objects that are referenced by the stack frames in the traceback.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, **line 591**); *backlink*

Unknown interpreted text role "func".

As a general principle, a function that calls another function to perform some task should check whether the called function raised an exception, and if so, pass the exception state on to its caller. It should discard any object references that it owns, and return an error indicator, but it should *not* set another exception --- that would overwrite the exception that was just raised, and lose important information about the exact cause of the error.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, **line 609**)

Unknown directive type "index".

```
.. index:: single: sum_sequence()
```

A simple example of detecting exceptions and passing them on is shown in the :c:func:`sum_sequence` example above. It so happens that this example doesn't need to clean up any owned references when it detects an error. The following example function shows some error cleanup. First, to remind you why you like Python, we show the equivalent Python code:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, **line 611**); *backlink*

Unknown interpreted text role "c:func".

```
def incr_item(dict, key):
    try:
        item = dict[key]
    except KeyError:
        item = 0
    dict[key] = item + 1
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, **line 624**)

Unknown directive type "index".

```
.. index:: single: incr_item()
```

Here is the corresponding C code, in all its glory:

```
int
incr_item(PyObject *dict, PyObject *key)
{
    /* Objects all initialized to NULL for Py_XDECREF */
    PyObject *item = NULL, *const_one = NULL, *incremented_item = NULL;
    int rv = -1; /* Return value initialized to -1 (failure) */

    item = PyObject_GetItem(dict, key);
    if (item == NULL) {
        /* Handle KeyError only: */
        if (!PyErr_ExceptionMatches(PyExc_KeyError))
            goto error;

        /* Clear the error and use zero: */
        PyErr_Clear();
        item = PyLong_FromLong(0L);
        if (item == NULL)
            goto error;
    }
    const_one = PyLong_FromLong(1L);
    if (const_one == NULL)
        goto error;

    incremented_item = PyNumber_Add(item, const_one);
    if (incremented_item == NULL)
        goto error;
    if (PyObject_SetItem(dict, key, incremented_item) < 0)
        goto error;
    rv = 0; /* Success */
    /* Continue with cleanup code */

 error:
    /* Cleanup code, shared by success and failure path */

    /* Use Py_XDECREF() to ignore NULL references */
    Py_XDECREF(item);
    Py_XDECREF(const_one);
    Py_XDECREF(incremented_item);

    return rv; /* -1 for error, 0 for success */
}
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 671)

Unknown directive type "index".

```
.. index:: single: incr_item()
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 673)

Unknown directive type "index".

```
.. index::
   single: PyErr_ExceptionMatches()
   single: PyErr_Clear()
   single: Py_XDECREF()
```

This example represents an endorsed use of the `goto` statement in C! It illustrates the use of :c:func:`PyErr_ExceptionMatches` and :c:func:`PyErr_Clear` to handle specific exceptions, and the use of :c:func:`Py_XDECREF` to dispose of owned references that may be `NULL` (note the `'X'` in the name; :c:func:`Py_DECREF` would crash when confronted with a `NULL` reference). It is important that the variables used to hold owned references are initialized to `NULL` for this to work; likewise, the proposed return value is initialized to `-1` (failure) and only set to success after the final call made is successful.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 678); *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)intro.rst`, line 678); *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)intro.rst`, line 678); *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)intro.rst`, line 678); *backlink*

Unknown interpreted text role "c:func".

Embedding Python

The one important task that only embedders (as opposed to extension writers) of the Python interpreter have to worry about is the initialization, and possibly the finalization, of the Python interpreter. Most functionality of the interpreter can only be used after the interpreter has been initialized.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 699)

Unknown directive type "index".

```
.. index::
   single: Py_Initialize()
   module: builtins
   module: __main__
   module: sys
   triple: module; search; path
   single: path (in module sys)
```

The basic initialization function is :c:func:`Py_Initialize`. This initializes the table of loaded modules, and creates the fundamental modules :mod:`builtins`, :mod:`__main__`, and :mod:`sys`. It also initializes the module search path (`sys.path`).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 707); *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)intro.rst`, line 707); *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)intro.rst`, line 707); *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)intro.rst`, line 707); *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)intro.rst`, line 712)

Unknown directive type "index".

```
.. index:: single: PySys_SetArgvEx()
```

:c:func:`Py_Initialize` does not set the "script argument list" (`sys.argv`). If this variable is needed by Python code that will be executed later, it must be set explicitly with a call to `PySys_SetArgvEx(argc, argv, updatepath)` after the call 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)intro.rst`, line 714); *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)intro.rst`, line 714); *backlink*

Unknown interpreted text role "c:func".

On most systems (in particular, on Unix and Windows, although the details are slightly different), :c:func:`Py_Initialize` calculates the module search path based upon its best guess for the location of the standard Python interpreter executable, assuming that the Python library is found in a fixed location relative to the Python interpreter executable. In particular, it looks for a directory named :file:`lib/python{X.Y}` relative to the parent directory where the executable named :file:`python` is found on the shell command search path (the environment variable :envvar:`PATH`).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 719); *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)intro.rst`, line 719); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 719); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 719); *backlink*

Unknown interpreted text role "envvar".

For instance, if the Python executable is found in :file:`/usr/local/bin/python`, it will assume that the libraries are in :file:`/usr/local/lib/python{X.Y}`. (In fact, this particular path is also the "fallback" location, used when no executable file named :file:`python` is found along :envvar:`PATH`.) The user can override this behavior by setting the environment variable :envvar:`PYTHONHOME`, or insert additional directories in front of the standard path by setting :envvar:`PYTHONPATH`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 728); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 728); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 728); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 728); *backlink*

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 728); *backlink*

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 728); *backlink*

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 736)

Unknown directive type "index".

```
.. index::
   single: Py_SetProgramName()
   single: Py_GetPath()
   single: Py_GetPrefix()
   single: Py_GetExecPrefix()
   single: Py_GetProgramFullPath()
```

The embedding application can steer the search by calling `Py_SetProgramName(file)` *before* calling :c:func:`Py_Initialize`. Note that :envvar:`PYTHONHOME` still overrides this and :envvar:`PYTHONPATH` is still inserted in front of the standard path. An application that requires total control has to provide its own implementation of :c:func:`Py_GetPath`, :c:func:`Py_GetPrefix`,

:c:func:`Py_GetExecPrefix`, and :c:func:`Py_GetProgramFullPath` (all defined in :file:`Modules/getpath.c`).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 743); ***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)intro.rst`, line 743); ***backlink***

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 743); ***backlink***

Unknown interpreted text role "envvar".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 743); ***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)intro.rst`, line 743); ***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)intro.rst`, line 743); ***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)intro.rst`, line 743); ***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)intro.rst`, line 743); ***backlink***

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 751)

Unknown directive type "index".

```
.. index:: single: Py_IsInitialized()
```

Sometimes, it is desirable to "uninitialize" Python. For instance, the application may want to start over (make another call to :c:func:`Py_Initialize`) or the application is simply done with its use of Python and wants to free memory allocated by Python. This can be accomplished by calling :c:func:`Py_FinalizeEx`. The function :c:func:`Py_IsInitialized` returns true if Python is currently in the initialized state. More information about these functions is given in a later chapter. Notice that :c:func:`Py_FinalizeEx` does *not* free all memory allocated by the Python interpreter, e.g. memory allocated by extension modules currently cannot be released.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 753); ***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)intro.rst`, line 753); ***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)intro.rst`, line 753); *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)intro.rst`, line 753); *backlink*

Unknown interpreted text role "c:func".

Debugging Builds

Python can be built with several macros to enable extra checks of the interpreter and extension modules. These checks tend to add a large amount of overhead to the runtime so they are not enabled by default.

A full list of the various types of debugging builds is in the file :file:`Misc/SpecialBuilds.txt` in the Python source distribution. Builds are available that support tracing of reference counts, debugging the memory allocator, or low-level profiling of the main interpreter loop. Only the most frequently-used builds will be described in the remainder of this section.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 773); *backlink*

Unknown interpreted text role "file".

Compiling the interpreter with the :c:macro:`Py_DEBUG` macro defined produces what is generally meant by :ref:`a debug build of Python <debug-build>`. :c:macro:`Py_DEBUG` is enabled in the Unix build by adding :option:`--with-pydebug` to the :file:`./configure` command. It is also implied by the presence of the not-Python-specific :c:macro:`_DEBUG` macro. When :c:macro:`Py_DEBUG` is enabled in the Unix build, compiler optimization is disabled.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 779); *backlink*

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 779); *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)intro.rst`, line 779); *backlink*

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 779); *backlink*

Unknown interpreted text role "option".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 779); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 779); *backlink*

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 779); *backlink*

Unknown interpreted text role "c:macro".

In addition to the reference count debugging described below, extra checks are performed, see :ref:`Python Debug Build <debug-

build>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 787); *backlink*

Unknown interpreted text role "ref".

Defining :c:macro:`Py_TRACE_REFS` enables reference tracing (see the :option:`configure --with-trace-refs option <--with-trace-refs>`). When defined, a circular doubly linked list of active objects is maintained by adding two extra fields to every :c:type:`PyObject`. Total allocations are tracked as well. Upon exit, all existing references are printed. (In interactive mode this happens after every statement run by the interpreter.)

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 790); *backlink*

Unknown interpreted text role "c:macro".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 790); *backlink*

Unknown interpreted text role "option".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 790); *backlink*

Unknown interpreted text role "c:type".

Please refer to :file:`Misc/SpecialBuilds.txt` in the Python source distribution for more detailed information.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\c-api\(cpython-main) (Doc) (c-api)intro.rst`, line 797); *backlink*

Unknown interpreted text role "file".