Extending/Embedding FAQ

- Can I create my own functions in C?
- Can I create my own functions in C++?
- Writing C is hard; are there any alternatives?
- How can I execute arbitrary Python statements from C?
- How can I evaluate an arbitrary Python expression from C?
- How do I extract C values from a Python object?
- How do I use Py_BuildValue() to create a tuple of arbitrary length?
- How do I call an object's method from C?
- How do I catch the output from PyErr_Print() (or anything that prints to stdout/stderr)?
- How do I access a module written in Python from C?
- How do I interface to C++ objects from Python?
- I added a module using the Setup file and the make fails; why?
- How do I debug an extension?
- I want to compile a Python module on my Linux system, but some files are missing. Why?
- How do I tell "incomplete input" from "invalid input"?
- How do I find undefined g++ symbols __builtin_new or __pure_virtual?
- Can I create an object class with some methods implemented in C and others in Python (e.g. through i

html

Can I create my own functions in C?

Yes, you can create built-in modules containing functions, variables, exceptions and even new types in C. This is explained in the document `extending-index`.

Most intermediate or advanced Python books will also cover this topic.

Can I create my own functions in C++?

Yes, using the C compatibility features found in C++. Place `extern "C" { ... }` around the Python include files and put `extern "C"` before each function that is going to be called by the Python interpreter. Global or static C++ objects with constructors are probably not a good idea.

Writing C is hard; are there any alternatives?

There are a number of alternatives to writing your own C extensions, depending on what you're trying to do. `Recommended third party tools <c-api-tools>` offer both simpler and more sophisticated approaches to creating C and C++ extensions for Python.

How can I execute arbitrary Python statements from C?

The highest-level function to do this is `PyRun_SimpleString` which takes a single string argument to be executed in the context of the module `__main__` and returns `0` for success and `-1` when an exception occurred (including `SyntaxError`). If you want more control, use `PyRun_String`; see the source for `PyRun_SimpleString` in `Python/pythonrun.c`.

How can I evaluate an arbitrary Python expression from C?

Call the function `PyRun_String` from the previous question with the start symbol `Py_eval_input`; it parses an expression, evaluates it and returns its value.

How do I extract C values from a Python object?

That depends on the object's type. If it's a tuple, `PyTuple_Size` returns its length and `PyTuple_GetItem` returns the item at a specified index. Lists have similar functions, `PyList_Size` and `PyList_GetItem`.

For bytes, `PyBytes_Size` returns its length and `PyBytes_AsStringAndSize` provides a pointer to its value and its length. Note that Python bytes objects may contain null bytes so C's `!strlen` should not be used.

To test the type of an object, first make sure it isn't `NULL`, and then use `PyBytes_Check`, `PyTuple_Check`, `PyList_Check`, etc.

There is also a high-level API to Python objects which is provided by the so-called 'abstract' interface -- read `Include/abstract.h` for further details. It allows interfacing with any kind of Python sequence using calls like `PySequence_Length`, `PySequence_GetItem`, etc. as well as many other useful protocols such as numbers (`PyNumber_Index` et al.) and mappings in the PyMapping APIs.

How do I use Py_BuildValue() to create a tuple of arbitrary length?

You can't. Use `PyTuple_Pack` instead.

How do I call an object's method from C?

The `PyObject_CallMethod` function can be used to call an arbitrary method of an object. The parameters are the object, the name of the method to call, a format string like that used with `Py_BuildValue`, and the argument values:

```
PyObject *
PyObject_CallMethod(PyObject *object, const char *method_name,
                    const char *arg_format, ...);
```

This works for any object that has methods -- whether built-in or user-defined. You are responsible for eventually `Py_DECREF`'ing the return value.

To call, e.g., a file object's "seek" method with arguments 10, 0 (assuming the file object pointer is "f"):

```
res = PyObject_CallMethod(f, "seek", "(ii)", 10, 0);
if (res == NULL) {
        ... an exception occurred ...
}
else {
        Py_DECREF(res);
}
```

Note that since `PyObject_CallObject` *always* wants a tuple for the argument list, to call a function without arguments, pass "()" for the format, and to call a function with one argument, surround the argument in parentheses, e.g. "(i)".

How do I catch the output from PyErr_Print() (or anything that prints to stdout/stderr)?

In Python code, define an object that supports the `write()` method. Assign this object to `sys.stdout` and `sys.stderr`. Call print_error, or just allow the standard traceback mechanism to work. Then, the output will go wherever your `write()` method sends it.

The easiest way to do this is to use the `io.StringIO` class:

```
>>> import io, sys
>>> sys.stdout = io.StringIO()
>>> print('foo')
>>> print('hello world!')
>>> sys.stderr.write(sys.stdout.getvalue())
foo
hello world!
```

A custom object to do the same would look like this:

```
>>> import io, sys
>>> class StdoutCatcher(io.TextIOBase):
...     def __init__(self):
...         self.data = []
...     def write(self, stuff):
...         self.data.append(stuff)
...
>>> import sys
>>> sys.stdout = StdoutCatcher()
>>> print('foo')
>>> print('hello world!')
>>> sys.stderr.write(''.join(sys.stdout.data))
foo
hello world!
```

How do I access a module written in Python from C?

You can get a pointer to the module object as follows:

```
module = PyImport_ImportModule("<modulename>");
```

If the module hasn't been imported yet (i.e. it is not yet present in `sys.modules`), this initializes the module; otherwise it simply returns the value of `sys.modules["<modulename>"]`. Note that it doesn't enter the module into any namespace -- it only ensures it has been initialized and is stored in `sys.modules`.

You can then access the module's attributes (i.e. any name defined in the module) as follows:

```
attr = PyObject_GetAttrString(module, "<attrname>");
```

Calling `PyObject_SetAttrString` to assign to variables in the module also works.

How do I interface to C++ objects from Python?

Depending on your requirements, there are many approaches. To do this manually, begin by reading `the "Extending and Embedding" document <extending-index>`. Realize that for the Python run-time system, there isn't a whole lot of difference between C and C++ -- so the strategy of building a new Python type around a C structure (pointer) type will also work for C++ objects.

For C++ libraries, see `c-wrapper-software`.

I added a module using the Setup file and the make fails; why?

Setup must end in a newline, if there is no newline there, the build process fails. (Fixing this requires some ugly shell script hackery, and this bug is so minor that it doesn't seem worth the effort.)

How do I debug an extension?

When using GDB with dynamically loaded extensions, you can't set a breakpoint in your extension until your extension is loaded.

In your `.gdbinit` file (or interactively), add the command:

```
br _PyImport_LoadDynamicModule
```

Then, when you run GDB:

```
$ gdb /local/bin/python
gdb) run myscript.py
gdb) continue # repeat until your extension is loaded
gdb) finish   # so that your extension is loaded
gdb) br myfunction.c:50
gdb) continue
```

I want to compile a Python module on my Linux system, but some

files are missing. Why?

Most packaged versions of Python omit some files required for compiling Python extensions.

For Red Hat, install the python3-devel RPM to get the necessary files.

For Debian, run `apt-get install python3-dev`.

How do I tell "incomplete input" from "invalid input"?

Sometimes you want to emulate the Python interactive interpreter's behavior, where it gives you a continuation prompt when the input is incomplete (e.g. you typed the start of an "if" statement or you didn't close your parentheses or triple string quotes), but it gives you a syntax error message immediately when the input is invalid.

In Python you can use the `codeop` module, which approximates the parser's behavior sufficiently. IDLE uses this, for example.

The easiest way to do it in C is to call `PyRun_InteractiveLoop` (perhaps in a separate thread) and let the Python interpreter handle the input for you. You can also set the `PyOS_ReadlineFunctionPointer` to point at your custom input function. See `Modules/readline.c` and `Parser/myreadline.c` for more hints.

How do I find undefined g++ symbols __builtin_new or __pure_virtual?

To dynamically load g++ extension modules, you must recompile Python, relink it using g++ (change LINKCC in the Python Modules Makefile), and link your extension module using g++ (e.g., `g++ -shared -o mymodule.so mymodule.o`).

Can I create an object class with some methods implemented in C and others in Python (e.g. through inheritance)?

Yes, you can inherit from built-in classes such as `int`, `list`, `dict`, etc.

The Boost Python Library (BPL, https://www.boost.org/libs/python/doc/index.html) provides a way of doing this from C++ (i.e. you can inherit from an extension class written in C++ using the BPL).

□□□ #2
□ □□□ □□□ 2026-10-31 16:57:48 UTC
□ □□□ □□□ 2026-10-31 16:57:51 UTC