Extending and Embedding the Python Interpreter

- Recommended third party tools
- C API Tutorial
- Guides for intermediate topics
- Embedding the CPython runtime in a larger application

This document describes how to write modules in C or C++ to extend the Python interpreter with new modules. Those modules can do what Python code does -- define functions, object types and methods -- but also interact with native libraries or achieve better performance by avoiding the overhead of an interpreter. The document also describes how to embed the Python interpreter in another application, for use as an extension language. Finally, it shows how to compile and link extension modules so that they can be loaded dynamically (at run time) into the interpreter, if the underlying operating system supports this feature.

This document assumes basic knowledge about C and Python. For an informal introduction to Python, see `tutorial-index`. `reference-index` gives a more formal definition of the language. `builtins-index` documents the built-in functions and object types, and `library-index` documents the modules (both built-in and written in Python) that give the language its wide application range.

For a detailed description of the whole Python/C API, see the separate `c-api-index`.

To support extensions, Python's C API (Application Programmers Interface) defines a set of functions, macros and variables that provide access to most aspects of the Python run-time system. The Python API is incorporated in a C source file by including the header `"Python.h"`.

> The C extension interface is specific to CPython, and extension modules do not work on other Python implementations. In many cases, it is possible to avoid writing C extensions and preserve portability to other implementations. For example, if your use case is calling C library functions or system calls, you should consider using the `ctypes` module or the cffi library rather than writing custom C code. These modules let you write Python code to interface with C code and are more portable between implementations of Python than writing and compiling a C extension module.

first-extension-module.rst extending.rst newtypes_tutorial.rst newtypes.rst building.rst windows.rst embedding.rst

Recommended third party tools

This document only covers the basic tools for creating extensions provided as part of this version of CPython. Some `third party tools <c-api-tools>` offer both simpler and more sophisticated approaches to creating C and C++ extensions for Python.

While this document is aimed at extension authors, it should also be helpful to the authors of such tools. For example, the tutorial module can serve as a simple test case for a build tool or sample expected output of a code generator.

C API Tutorial

This tutorial describes how to write a simple module in C or C++, using the Python C API -- that is, using the basic tools provided as part of this version of CPython.

1. `first-extension-module`

Guides for intermediate topics

This section of the guide covers creating C and C++ extensions without assistance from third party tools. It is intended primarily for creators of those tools, rather than being a recommended way to create your own C extensions.

- `extending-intro`
- `defining-new-types`
- `new-types-topics`
- `building`
- `building-on-windows`

Embedding the CPython runtime in a larger application

Sometimes, rather than creating an extension that runs inside the Python interpreter as the main application, it is desirable to instead embed the CPython runtime inside a larger application. This section covers some of the details involved in doing that successfully.

- `embedding`

□□□ #2
□ □□□ □□□ 2026-10-31 16:57:22 UTC
□ □□□ □□□ 2026-10-31 16:57:24 UTC