25

Extending and Embedding Classic Python

CPython runs on a portable, C-coded virtual machine. Python's built-in objects—such as numbers, sequences, dictionaries, sets, and files—are coded in C, as are several modules in Python's standard library. Modern platforms support dynamic-load libraries, with file extensions such as *.dll* on Windows, *.so* on Linux, and *.dylib* on Mac: building Python produces such binary files. You can code your own extension modules for Python in C, using the Python C API covered in this chapter, to produce and deploy dynamic libraries that Python scripts and interactive sessions can later use with the `import` statement, covered in "The import Statement".

Extending Python means building modules that Python code can `import` to access the features the modules supply. *Embedding* Python means executing Python code from an application coded in another language. For such execution to be useful, Python code must in turn be able to access some of your application's functionality. In practice, therefore, embedding implies some extending, as well as a few embedding-specific operations. The three main reasons for wishing to extend Python can be summarized as follows:

- Reimplementing some functionality (originally coded in Python) in a lower-level language, hoping to get better performance
- Letting Python code access some existing functionality supplied by libraries coded in (or, at any rate, callable from) lower-level languages
- Letting Python code access some existing functionality of an application that is in the process of embedding Python as the application's scripting language

[Placeholder for PDF of finished chapter produced by O'Reilly]