The import system

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

Unknown directive type "index".

```
.. index:: single: import machinery
```

Python code in one :term:`module` gains access to the code in another module by the process of :term:`importing` it. The :keyword:`import` statement is the most common way of invoking the import machinery, but it is not the only way. Functions such as :func:`importlib.import_module` and built-in :func:`__import__` can also be used to invoke the import machinery.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 10); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 10); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 10); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 10); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 10); *backlink*

Unknown interpreted text role "func".

The :keyword:`import` statement combines two operations; it searches for the named module, then it binds the results of that search to a name in the local scope. The search operation of the :keyword:`!import` statement is defined as a call to the :func:`__import__` function, with the appropriate arguments. The return value of :func:`__import__` is used to perform the name binding operation of the :keyword:`!import` statement. See the :keyword:`!import` statement for the exact details of that name binding operation.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 16); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 16); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 16); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 16); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 16); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 16); *backlink*

Unknown interpreted text role "keyword".

A direct call to :func:`__import__` performs only the module search and, if found, the module creation operation. While certain side-effects may occur, such as the importing of parent packages, and the updating of various caches (including :data:`sys.modules`), only the :keyword:`import` statement performs a name binding operation.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 25); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 25); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 25); *backlink*

Unknown interpreted text role "keyword".

When an :keyword:`import` statement is executed, the standard builtin :func:`__import__` function is called. Other mechanisms for invoking the import system (such as :func:`importlib.import_module`) may choose to bypass :func:`__import__` and use their own solutions to implement import semantics.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 31); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 31); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 31); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 31); *backlink*

Unknown interpreted text role "func".

When a module is first imported, Python searches for the module and if found, it creates a module object [1], initializing it. If the named module cannot be found, a :exc:`ModuleNotFoundError` is raised. Python implements various strategies to search for the named module when the import machinery is invoked. These strategies can be modified and extended by using various hooks described in the sections below.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 36); *backlink*

Unknown interpreted text role "exc".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 43)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   The import system has been updated to fully implement the second phase
   of :pep:`302`. There is no longer any implicit import machinery - the full
   import system is exposed through :data:`sys.meta_path`. In addition,
   native namespace package support has been implemented (see :pep:`420`).
```

:mod:`importlib`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 50); *backlink*

Unknown interpreted text role "mod".

The :mod:`importlib` module provides a rich API for interacting with the import system. For example :func:`importlib.import_module` provides a recommended, simpler API than built-in :func:`__import__` for invoking the import machinery. Refer to the :mod:`importlib` library documentation for additional detail.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 53); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 53); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 53); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 53); *backlink*

Unknown interpreted text role "mod".

Packages

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 64)

Unknown directive type "index".

```
.. index::
    single: package
```

Python has only one type of module object, and all modules are of this type, regardless of whether the module is implemented in Python, C, or something else. To help organize modules and provide a naming hierarchy, Python has a concept of :term:`packages <package>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 67); *backlink*

Unknown interpreted text role "term".

You can think of packages as the directories on a file system and modules as files within directories, but don't take this analogy too literally since packages and modules need not originate from the file system. For the purposes of this documentation, we'll use this convenient analogy of directories and files. Like file system directories, packages are organized hierarchically, and packages may themselves contain subpackages, as well as regular modules.

It's important to keep in mind that all packages are modules, but not all modules are packages. Or put another way, packages are just a special kind of module. Specifically, any module that contains a `__path__` attribute is considered a package.

All modules have a name. Subpackage names are separated from their parent package name by a dot, akin to Python's standard attribute access syntax. Thus you might have a package called :mod:`email`, which in turn has a subpackage called :mod:`email.mime` and a module within that subpackage called :mod:`email.mime.text`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 85); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 85); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 85); *backlink*

Unknown interpreted text role "mod".

Regular packages

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 95)

Unknown directive type "index".

```
.. index::
    pair: package; regular
```

Python defines two types of packages, :term:`regular packages <regular package>` and :term:`namespace packages <namespace package>`. Regular packages are traditional packages as they existed in Python 3.2 and earlier. A regular package is typically implemented as a directory containing an `__init__.py` file. When a regular package is imported, this `__init__.py` file is implicitly executed, and the objects it defines are bound to names in the package's namespace. The `__init__.py` file can contain the same Python code that any other module can contain, and Python will add some additional attributes to the module when it is imported.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 98); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 98); *backlink*

Unknown interpreted text role "term".

For example, the following file system layout defines a top level `parent` package with three subpackages:

```
parent/
    __init__.py
    one/
        __init__.py
    two/
        __init__.py
    three/
        __init__.py
```

Importing `parent.one` will implicitly execute `parent/__init__.py` and `parent/one/__init__.py`. Subsequent imports of `parent.two` or `parent.three` will execute `parent/two/__init__.py` and `parent/three/__init__.py` respectively.

Namespace packages

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 129)

Unknown directive type "index".

```
.. index::
    pair: package; namespace
    pair: package; portion
```

A namespace package is a composite of various :term:`portions <portion>`, where each portion contributes a subpackage to the parent package. Portions may reside in different locations on the file system. Portions may also be found in zip files, on the network, or anywhere else that Python searches during import. Namespace packages may or may not correspond directly to objects on the file system; they may be virtual modules that have no concrete representation.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 133); *backlink*

Unknown interpreted text role "term".

Namespace packages do not use an ordinary list for their `__path__` attribute. They instead use a custom iterable type which will automatically perform a new search for package portions on the next import attempt within that package if the path of their parent package (or :data:`sys.path` for a top level package) changes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 141); *backlink*

Unknown interpreted text role "data".

With namespace packages, there is no `parent/__init__.py` file. In fact, there may be multiple `parent` directories found during import search, where each one is provided by a different portion. Thus `parent/one` may not be physically located next to `parent/two`. In this case, Python will create a namespace package for the top-level `parent` package whenever it or one of its subpackages is imported.

See also PEP 420 for the namespace package specification.

Searching

To begin the search, Python needs the :term:`fully qualified <qualified name>` name of the module (or package, but for the purposes of this discussion, the difference is immaterial) being imported. This name may come from various arguments to the :keyword:`import` statement, or from the parameters to the :func:`importlib.import_module` or :func:`__import__` functions.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 160); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 160); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 160); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 160); *backlink*

Unknown interpreted text role "func".

This name will be used in various phases of the import search, and it may be the dotted path to a submodule, e.g. `foo.bar.baz`. In this case, Python first tries to import `foo`, then `foo.bar`, and finally `foo.bar.baz`. If any of the intermediate imports fail, a :exc:`ModuleNotFoundError` is raised.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 166); *backlink*

Unknown interpreted text role "exc".

The module cache

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 175)

Unknown directive type "index".

```
.. index::
   single: sys.modules
```

The first place checked during import search is :data:`sys.modules`. This mapping serves as a cache of all modules that have been previously imported, including the intermediate paths. So if `foo.bar.baz` was previously imported, :data:`sys.modules` will contain entries for `foo`, `foo.bar`, and `foo.bar.baz`. Each key will have as its value the corresponding module object.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 178); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 178); *backlink*

Unknown interpreted text role "data".

During import, the module name is looked up in :data:`sys.modules` and if present, the associated value is the module satisfying the import, and the process completes. However, if the value is `None`, then a :exc:`ModuleNotFoundError` is raised. If the module name is missing, Python will continue searching for the module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 185); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 185); *backlink*

Unknown interpreted text role "exc".

:data:`sys.modules` is writable. Deleting a key may not destroy the associated module (as other modules may hold references to it), but it will invalidate the cache entry for the named module, causing Python to search anew for the named module upon its next import. The key can also be assigned to `None`, forcing the next import of the module to result in a :exc:`ModuleNotFoundError`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 191); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 191); *backlink*

Unknown interpreted text role "exc".

Beware though, as if you keep a reference to the module object, invalidate its cache entry in :data:`sys.modules`, and then re-import the named module, the two module objects will *not* be the same. By contrast, :func:`importlib.reload` will reuse the *same* module object, and simply reinitialise the module contents by rerunning the module's code.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 198); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 198); *backlink*

Unknown interpreted text role "func".

Finders and loaders

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 210)

Unknown directive type "index".

```
.. index::
    single: finder
    single: loader
    single: module spec
```

If the named module is not found in :data:`sys.modules`, then Python's import protocol is invoked to find and load the module. This protocol consists of two conceptual objects, :term:`finders <finder>` and :term:`loaders <loader>`. A finder's job is to determine whether it can find the named module using whatever strategy it knows about. Objects that implement both of these interfaces are referred to as :term:`importers <importer>` - they return themselves when they find that they can load the requested module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 215); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 215); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 215); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 215); *backlink*

Unknown interpreted text role "term".

Python includes a number of default finders and importers. The first one knows how to locate built-in modules, and the second knows how to locate frozen modules. A third default finder searches an :term:`import path` for modules. The :term:`import path` is a list of locations that may name file system paths or zip files. It can also be extended to search for any locatable resource, such as those identified by URLs.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 223); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 223); *backlink*

Unknown interpreted text role "term".

The import machinery is extensible, so new finders can be added to extend the range and scope of module searching.

Finders do not actually load modules. If they can find the named module, they return a :dfn:`module spec`, an encapsulation of the module's import-related information, which the import machinery then uses when loading the module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 233); *backlink*

Unknown interpreted text role "dfn".

The following sections describe the protocol for finders and loaders in more detail, including how you can create and register new ones to extend the import machinery.

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 241)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   In previous versions of Python, finders returned :term:`loaders <loader>`
   directly, whereas now they return module specs which *contain* loaders.
   Loaders are still used during import but have fewer responsibilities.
```

Import hooks

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 249)

Unknown directive type "index".

```
.. index::
   single: import hooks
   single: meta hooks
   single: path hooks
   pair: hooks; import
   pair: hooks; meta
   pair: hooks; path
```

The import machinery is designed to be extensible; the primary mechanism for this are the *import hooks*. There are two types of import hooks: *meta hooks* and *import path hooks*.

Meta hooks are called at the start of import processing, before any other import processing has occurred, other than :data:`sys.modules` cache look up. This allows meta hooks to override :data:`sys.path` processing, frozen modules, or even built-in modules. Meta hooks are registered by adding new finder objects to :data:`sys.meta_path`, as described below.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 261); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 261); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 261); *backlink*

Unknown interpreted text role "data".

Import path hooks are called as part of :data:`sys.path` (or `package.__path__`) processing, at the point where their associated path item is encountered. Import path hooks are registered by adding new callables to :data:`sys.path_hooks` as described below.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 267); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 267); *backlink*

Unknown interpreted text role "data".

The meta path

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 276)

Unknown directive type "index".

```
.. index::
    single: sys.meta_path
```

```
        pair: finder; find_spec
```

When the named module is not found in :data:`sys.modules`, Python next searches :data:`sys.meta_path`, which contains a list of meta path finder objects. These finders are queried in order to see if they know how to handle the named module. Meta path finders must implement a method called :meth:`~importlib.abc.MetaPathFinder.find_spec()` which takes three arguments: a name, an import path, and (optionally) a target module. The meta path finder can use any strategy it wants to determine whether it can handle the named module or not.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 280); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 280); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 280); *backlink*

Unknown interpreted text role "meth".

If the meta path finder knows how to handle the named module, it returns a spec object. If it cannot handle the named module, it returns `None`. If :data:`sys.meta_path` processing reaches the end of its list without returning a spec, then a :exc:`ModuleNotFoundError` is raised. Any other exceptions raised are simply propagated up, aborting the import process.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 289); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 289); *backlink*

Unknown interpreted text role "exc".

The :meth:`~importlib.abc.MetaPathFinder.find_spec()` method of meta path finders is called with two or three arguments. The first is the fully qualified name of the module being imported, for example `foo.bar.baz`. The second argument is the path entries to use for the module search. For top-level modules, the second argument is `None`, but for submodules or subpackages, the second argument is the value of the parent package's `__path__` attribute. If the appropriate `__path__` attribute cannot be accessed, a :exc:`ModuleNotFoundError` is raised. The third argument is an existing module object that will be the target of loading later. The import system passes in a target module only during reload.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 295); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 295); *backlink*

Unknown interpreted text role "exc".

The meta path may be traversed multiple times for a single import request. For example, assuming none of the modules involved has already been cached, importing `foo.bar.baz` will first perform a top level import, calling `mpf.find_spec("foo", None, None)` on each meta path finder (`mpf`). After `foo` has been imported, `foo.bar` will be imported by traversing the meta path a second time, calling `mpf.find_spec("foo.bar", foo.__path__, None)`. Once `foo.bar` has been imported, the final traversal will call `mpf.find_spec("foo.bar.baz", foo.bar.__path__, None)`.

Some meta path finders only support top level imports. These importers will always return `None` when anything other than `None` is passed as the second argument.

Python's default :data:`sys.meta_path` has three meta path finders, one that knows how to import built-in modules, one that knows how to import frozen modules, and one that knows how to import modules from an :term:`import path` (i.e. the :term:`path based

finder`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 320); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 320); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 320); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 325)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   The :meth:`~importlib.abc.MetaPathFinder.find_spec` method of meta path
   finders replaced :meth:`~importlib.abc.MetaPathFinder.find_module`, which
   is now deprecated.  While it will continue to work without change, the
   import machinery will try it only if the finder does not implement
   ``find_spec()``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 332)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   Use of :meth:`~importlib.abc.MetaPathFinder.find_module` by the import system
   now raises :exc:`ImportWarning`.
```

Loading

If and when a module spec is found, the import machinery will use it (and the loader it contains) when loading the module. Here is an approximation of what happens during the loading portion of import:

```
module = None
if spec.loader is not None and hasattr(spec.loader, 'create_module'):
    # It is assumed 'exec_module' will also be defined on the loader.
    module = spec.loader.create_module(spec)
if module is None:
    module = ModuleType(spec.name)
# The import-related module attributes get set here:
_init_module_attrs(spec, module)

if spec.loader is None:
    # unsupported
    raise ImportError
if spec.origin is None and spec.submodule_search_locations is not None:
    # namespace package
    sys.modules[spec.name] = module
elif not hasattr(spec.loader, 'exec_module'):
    module = spec.loader.load_module(spec.name)
    # Set __loader__ and __package__ if missing.
else:
    sys.modules[spec.name] = module
    try:
        spec.loader.exec_module(module)
    except BaseException:
        try:
            del sys.modules[spec.name]
        except KeyError:
            pass
        raise
```

```
    return sys.modules[spec.name]
```

Note the following details:

- If there is an existing module object with the given name in :data:`sys.modules`, import will have already returned it.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 376); *backlink*

  Unknown interpreted text role "data".

- The module will exist in :data:`sys.modules` before the loader executes the module code. This is crucial because the module code may (directly or indirectly) import itself; adding it to :data:`sys.modules` beforehand prevents unbounded recursion in the worst case and multiple loading in the best.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 379); *backlink*

  Unknown interpreted text role "data".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 379); *backlink*

  Unknown interpreted text role "data".

- If loading fails, the failing module -- and only the failing module -- gets removed from :data:`sys.modules`. Any module already in the :data:`sys.modules` cache, and any module that was successfully loaded as a side-effect, must remain in the cache. This contrasts with reloading where even the failing module is left in :data:`sys.modules`.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 385); *backlink*

  Unknown interpreted text role "data".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 385); *backlink*

  Unknown interpreted text role "data".

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 385); *backlink*

  Unknown interpreted text role "data".

- After the module is created but before execution, the import machinery sets the import-related module attributes ("_init_module_attrs" in the pseudo-code example above), as summarized in a :ref:`later section <import-mod-attrs>`.

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 391); *backlink*

  Unknown interpreted text role "ref".

- Module execution is the key moment of loading in which the module's namespace gets populated. Execution is entirely delegated to the loader, which gets to decide what gets populated and how.
- The module created during loading and passed to exec_module() may not be the one returned at the end of import [2].

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 403)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   The import system has taken over the boilerplate responsibilities of
   loaders.  These were previously performed by the
   :meth:`importlib.abc.Loader.load_module` method.
```

Loaders

Module loaders provide the critical function of loading: module execution. The import machinery calls the :meth:`importlib.abc.Loader.exec_module` method with a single argument, the module object to execute. Any value returned from :meth:`~importlib.abc.Loader.exec_module` is ignored.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 411); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 411); *backlink*

Unknown interpreted text role "meth".

Loaders must satisfy the following requirements:

- If the module is a Python module (as opposed to a built-in module or a dynamically loaded extension), the loader should execute the module's code in the module's global name space (`module.__dict__`).
- If the loader cannot execute the module, it should raise an :exc:`ImportError`, although any other exception raised during :meth:`~importlib.abc.Loader.exec_module` will be propagated.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 422); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 422); *backlink*

Unknown interpreted text role "meth".

In many cases, the finder and loader can be the same object; in such cases the :meth:`~importlib.abc.MetaPathFinder.find_spec` method would just return a spec with the loader set to `self`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 426); *backlink*

Unknown interpreted text role "meth".

Module loaders may opt in to creating the module object during loading by implementing a :meth:`~importlib.abc.Loader.create_module` method. It takes one argument, the module spec, and returns the new module object to use during loading. `create_module()` does not need to set any attributes on the module object. If the method returns `None`, the import machinery will create the new module itself.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 430); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 437)

Unknown directive type "versionadded".

```
.. versionadded:: 3.4
   The :meth:`~importlib.abc.Loader.create_module` method of loaders.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 440)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   The :meth:`~importlib.abc.Loader.load_module` method was replaced by
   :meth:`~importlib.abc.Loader.exec_module` and the import
   machinery assumed all the boilerplate responsibilities of loading.

   For compatibility with existing loaders, the import machinery will use
   the ``load_module()`` method of loaders if it exists and the loader does
   not also implement ``exec_module()``.  However, ``load_module()`` has been
   deprecated and loaders should implement ``exec_module()`` instead.

   The ``load_module()`` method must implement all the boilerplate loading
   functionality described above in addition to executing the module.  All
   the same constraints apply, with some additional clarification:

    * If there is an existing module object with the given name in
      :data:`sys.modules`, the loader must use that existing module.
      (Otherwise, :func:`importlib.reload` will not work correctly.)  If the
      named module does not exist in :data:`sys.modules`, the loader
      must create a new module object and add it to :data:`sys.modules`.

    * The module *must* exist in :data:`sys.modules` before the loader
      executes the module code, to prevent unbounded recursion or multiple
      loading.

    * If loading fails, the loader must remove any modules it has inserted
      into :data:`sys.modules`, but it must remove **only** the failing
      module(s), and only if the loader itself has loaded the module(s)
      explicitly.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 469)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.5
   A :exc:`DeprecationWarning` is raised when ``exec_module()`` is defined but
   ``create_module()`` is not.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 473)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   An :exc:`ImportError` is raised when ``exec_module()`` is defined but
   ``create_module()`` is not.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 477)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   Use of ``load_module()`` will raise :exc:`ImportWarning`.
```

Submodules

When a submodule is loaded using any mechanism (e.g. `importlib` APIs, the `import` or `import-from` statements, or built-in `__import__()`) a binding is placed in the parent module's namespace to the submodule object. For example, if package `spam` has a submodule `foo`, after importing `spam.foo`, `spam` will have an attribute `foo` which is bound to the submodule. Let's say you have the following directory structure:

```
spam/
    __init__.py
    foo.py
    bar.py
```

and `spam/__init__.py` has the following lines in it:

```
from .foo import Foo
from .bar import Bar
```

then executing the following puts a name binding to `foo` and `bar` in the `spam` module:

```
>>> import spam
>>> spam.foo
<module 'spam.foo' from '/tmp/imports/spam/foo.py'>
>>> spam.bar
<module 'spam.bar' from '/tmp/imports/spam/bar.py'>
```

Given Python's familiar name binding rules this might seem surprising, but it's actually a fundamental feature of the import system. The invariant holding is that if you have `sys.modules['spam']` and `sys.modules['spam.foo']` (as you would after the above import), the latter must appear as the `foo` attribute of the former.

Module spec

The import machinery uses a variety of information about each module during import, especially before loading. Most of the information is common to all modules. The purpose of a module's spec is to encapsulate this import-related information on a per-module basis.

Using a spec during import allows state to be transferred between import system components, e.g. between the finder that creates the module spec and the loader that executes it. Most importantly, it allows the import machinery to perform the boilerplate operations of loading, whereas without a module spec the loader had that responsibility.

The module's spec is exposed as the `__spec__` attribute on a module object. See :class:`~importlib.machinery.ModuleSpec` for details on the contents of the module spec.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 529); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 533)

Unknown directive type "versionadded".

```
.. versionadded:: 3.4
```

Import-related module attributes

The import machinery fills in these attributes on each module object during loading, based on the module's spec, before the loader executes the module.

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

Unknown directive type "attribute".

```
.. attribute:: __name__

   The ``__name__`` attribute must be set to the fully-qualified name of
   the module.  This name is used to uniquely identify the module in
   the import system.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 550)

Unknown directive type "attribute".

```
.. attribute:: __loader__

   The ``__loader__`` attribute must be set to the loader object that
   the import machinery used when loading the module.  This is mostly
   for introspection, but can be used for additional loader-specific
```

```
   functionality, for example getting data associated with a loader.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 557)

Unknown directive type "attribute".

```
.. attribute:: __package__

   The module's ``__package__`` attribute must be set.  Its value must
   be a string, but it can be the same value as its ``__name__``.  When
   the module is a package, its ``__package__`` value should be set to
   its ``__name__``.  When the module is not a package, ``__package__``
   should be set to the empty string for top-level modules, or for
   submodules, to the parent package's name.  See :pep:`366` for further
   details.

   This attribute is used instead of ``__name__`` to calculate explicit
   relative imports for main modules, as defined in :pep:`366`. It is
   expected to have the same value as ``__spec__.parent``.

   .. versionchanged:: 3.6
      The value of ``__package__`` is expected to be the same as
      ``__spec__.parent``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 575)

Unknown directive type "attribute".

```
.. attribute:: __spec__

   The ``__spec__`` attribute must be set to the module spec that was
   used when importing the module. Setting ``__spec__``
   appropriately applies equally to :ref:`modules initialized during
   interpreter startup <programs>`.  The one exception is ``__main__``,
   where ``__spec__`` is :ref:`set to None in some cases <main_spec>`.

   When ``__package__`` is not defined, ``__spec__.parent`` is used as
   a fallback.

   .. versionadded:: 3.4

   .. versionchanged:: 3.6
      ``__spec__.parent`` is used as a fallback when ``__package__`` is
      not defined.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 592)

Unknown directive type "attribute".

```
.. attribute:: __path__

   If the module is a package (either regular or namespace), the module
   object's ``__path__`` attribute must be set.  The value must be
   iterable, but may be empty if ``__path__`` has no further significance.
   If ``__path__`` is not empty, it must produce strings when iterated
   over. More details on the semantics of ``__path__`` are given
   :ref:`below <package-path-rules>`.

   Non-package modules should not have a ``__path__`` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 603)

Unknown directive type "attribute".

```
.. attribute:: __file__
```

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

main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 604)

Unknown directive type "attribute".

```
.. attribute:: __cached__

   ``__file__`` is optional (if set, value must be a string). It indicates
   the pathname of the file from which the module was loaded (if
   loaded from a file), or the pathname of the shared library file
   for extension modules loaded dynamically from a shared library.
   It might be missing for certain types of modules, such as C
   modules that are statically linked into the interpreter, and the
   import system may opt to leave it unset if it has no semantic
   meaning (e.g. a module loaded from a database).

   If ``__file__`` is set, it may also be appropriate to set the
   ``__cached__`` attribute which is the path to any compiled version of
   the code (e.g. byte-compiled file). The file does not need to exist
   to set this attribute; the path can simply point to where the
   compiled file would exist (see :pep:`3147`).

   It is also appropriate to set ``__cached__`` when ``__file__`` is not
   set.  However, that scenario is quite atypical.  Ultimately, the
   loader is what makes use of ``__file__`` and/or ``__cached__``.  So
   if a loader can load from a cached module but otherwise does not load
   from a file, that atypical scenario may be appropriate.
```

module.__path__

By definition, if a module has a `__path__` attribute, it is a package.

A package's `__path__` attribute is used during imports of its subpackages. Within the import machinery, it functions much the same as :data:`sys.path`, i.e. providing a list of locations to search for modules during import. However, `__path__` is typically much more constrained than :data:`sys.path`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 634); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 634); *backlink*

Unknown interpreted text role "data".

`__path__` must be an iterable of strings, but it may be empty. The same rules used for :data:`sys.path` also apply to a package's `__path__`, and :data:`sys.path_hooks` (described below) are consulted when traversing a package's `__path__`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 640); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 640); *backlink*

Unknown interpreted text role "data".

A package's `__init__.py` file may set or alter the package's `__path__` attribute, and this was typically the way namespace packages were implemented prior to PEP 420. With the adoption of PEP 420, namespace packages no longer need to supply `__init__.py` files containing only `__path__` manipulation code; the import machinery automatically sets `__path__` correctly for the namespace package.

Module reprs

By default, all modules have a usable repr, however depending on the attributes set above, and in the module's spec, you can more explicitly control the repr of module objects.

If the module has a spec (`__spec__`), the import machinery will try to generate a repr from it. If that fails or there is no spec, the import system will craft a default repr using whatever information is available on the module. It will try to use the `module.__name__`, `module.__file__`, and `module.__loader__` as input into the repr, with defaults for whatever information is missing.

Here are the exact rules used:

- If the module has a `__spec__` attribute, the information in the spec is used to generate the repr. The "name", "loader", "origin", and "has_location" attributes are consulted.
- If the module has a `__file__` attribute, this is used as part of the module's repr.
- If the module has no `__file__` but does have a `__loader__` that is not `None`, then the loader's repr is used as part of the module's repr.
- Otherwise, just use the module's `__name__` in the repr.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 680)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   Use of :meth:`loader.module_repr() <importlib.abc.Loader.module_repr>`
   has been deprecated and the module spec is now used by the import
   machinery to generate a module repr.

   For backward compatibility with Python 3.3, the module repr will be
   generated by calling the loader's
   :meth:`~importlib.abc.Loader.module_repr` method, if defined, before
   trying either approach described above.  However, the method is deprecated.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 690)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10

   Calling :meth:`~importlib.abc.Loader.module_repr` now occurs after trying to
   use a module's ``__spec__`` attribute but before falling back on
   ``__file__``. Use of :meth:`~importlib.abc.Loader.module_repr` is slated to
   stop in Python 3.12.
```

Cached bytecode invalidation

Before Python loads cached bytecode from a `.pyc` file, it checks whether the cache is up-to-date with the source `.py` file. By default, Python does this by storing the source's last-modified timestamp and size in the cache file when writing it. At runtime, the import system then validates the cache file by checking the stored metadata in the cache file against the source's metadata.

Python also supports "hash-based" cache files, which store a hash of the source file's contents rather than its metadata. There are two variants of hash-based `.pyc` files: checked and unchecked. For checked hash-based `.pyc` files, Python validates the cache file by hashing the source file and comparing the resulting hash with the hash in the cache file. If a checked hash-based cache file is found to be invalid, Python regenerates it and writes a new checked hash-based cache file. For unchecked hash-based `.pyc` files, Python simply assumes the cache file is valid if it exists. Hash-based `.pyc` files validation behavior may be overridden with the :option:`--check-hash-based-pycs` flag.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 709); *backlink*

Unknown interpreted text role "option".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 720)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.7
   Added hash-based ``.pyc`` files. Previously, Python only supported
   timestamp-based invalidation of bytecode caches.
```

The Path Based Finder

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 728)

Unknown directive type "index".

```
.. index::
    single: path based finder
```

As mentioned previously, Python comes with several default meta path finders. One of these, called the :term:`path based finder` (:class:`~importlib.machinery.PathFinder`), searches an :term:`import path`, which contains a list of :term:`path entries <path entry>`. Each path entry names a location to search for modules.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 731); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 731); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 731); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 731); *backlink*

Unknown interpreted text role "term".

The path based finder itself doesn't know how to import anything. Instead, it traverses the individual path entries, associating each of them with a path entry finder that knows how to handle that particular kind of path.

The default set of path entry finders implement all the semantics for finding modules on the file system, handling special file types such as Python source code (`.py` files), Python byte code (`.pyc` files) and shared libraries (e.g. `.so` files). When supported by the :mod:`zipimport` module in the standard library, the default path entry finders also handle loading all of these file types (other than shared libraries) from zipfiles.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 741); *backlink*

Unknown interpreted text role "mod".

Path entries need not be limited to file system locations. They can refer to URLs, database queries, or any other location that can be specified as a string.

The path based finder provides additional hooks and protocols so that you can extend and customize the types of searchable path entries. For example, if you wanted to support path entries as network URLs, you could write a hook that implements HTTP semantics to find modules on the web. This hook (a callable) would return a :term:`path entry finder` supporting the protocol described below, which was then used to get a loader for the module from the web.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 752); *backlink*

Unknown interpreted text role "term".

A word of warning: this section and the previous both use the term *finder*, distinguishing between them by using the terms :term:`meta path finder` and :term:`path entry finder`. These two types of finders are very similar, support similar protocols, and function in similar ways during the import process, but it's important to keep in mind that they are subtly different. In particular, meta path finders operate at the beginning of the import process, as keyed off the :data:`sys.meta_path` traversal.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 760); *backlink*

Unknown interpreted text role "term".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 760); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 760); backlink

Unknown interpreted text role "data".

By contrast, path entry finders are in a sense an implementation detail of the path based finder, and in fact, if the path based finder were to be removed from :data:`sys.meta_path`, none of the path entry finder semantics would be invoked.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 768); backlink

Unknown interpreted text role "data".

Path entry finders

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 777)

Unknown directive type "index".

```
.. index::
   single: sys.path
   single: sys.path_hooks
   single: sys.path_importer_cache
   single: PYTHONPATH
```

The :term:`path based finder` is responsible for finding and loading Python modules and packages whose location is specified with a string :term:`path entry`. Most path entries name locations in the file system, but they need not be limited to this.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 783); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 783); backlink

Unknown interpreted text role "term".

As a meta path finder, the :term:`path based finder` implements the :meth:`~importlib.abc.MetaPathFinder.find_spec` protocol previously described, however it exposes additional hooks that can be used to customize how modules are found and loaded from the :term:`import path`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 788); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 788); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 788); backlink

Unknown interpreted text role "term".

Three variables are used by the :term:`path based finder`, :data:`sys.path`, :data:`sys.path_hooks` and :data:`sys.path_importer_cache`. The `__path__` attributes on package objects are also used. These provide additional ways that the import machinery can be customized.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 793); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 793); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 793); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 793); *backlink*

Unknown interpreted text role "data".

:data:`sys.path` contains a list of strings providing search locations for modules and packages. It is initialized from the :data:`PYTHONPATH` environment variable and various other installation- and implementation-specific defaults. Entries in :data:`sys.path` can name directories on the file system, zip files, and potentially other "locations" (see the :mod:`site` module) that should be searched for modules, such as URLs, or database queries. Only strings and bytes should be present on :data:`sys.path`; all other data types are ignored. The encoding of bytes entries is determined by the individual :term:`path entry finders <path entry finder>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 798); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 798); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 798); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 798); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 798); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 798); *backlink*

Unknown interpreted text role "term".

The :term:`path based finder` is a :term:`meta path finder`, so the import machinery begins the :term:`import path` search by calling the path based finder's :meth:`~importlib.machinery.PathFinder.find_spec` method as described previously. When the `path` argument to :meth:`~importlib.machinery.PathFinder.find_spec` is given, it will be a list of string paths to traverse - typically a package's `__path__` attribute for an import within that package. If the `path` argument is `None`, this indicates a top level import and :data:`sys.path` is used.

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 809); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 809); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 809); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 809); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 809); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 809); backlink

Unknown interpreted text role "data".

The path based finder iterates over every entry in the search path, and for each of these, looks for an appropriate :term:`path entry finder` (:class:`~importlib.abc.PathEntryFinder`) for the path entry. Because this can be an expensive operation (e.g. there may be *stat()* call overheads for this search), the path based finder maintains a cache mapping path entries to path entry finders. This cache is maintained in :data:`sys.path_importer_cache` (despite the name, this cache actually stores finder objects rather than being limited to :term:`importer` objects). In this way, the expensive search for a particular :term:`path entry` location's :term:`path entry finder` need only be done once. User code is free to remove cache entries from :data:`sys.path_importer_cache` forcing the path based finder to perform the path entry search again [3].

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); backlink

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 818); *backlink*

Unknown interpreted text role "data".

If the path entry is not present in the cache, the path based finder iterates over every callable in :data:`sys.path_hooks`. Each of the :term:`path entry hooks <path entry hook>` in this list is called with a single argument, the path entry to be searched. This callable may either return a :term:`path entry finder` that can handle the path entry, or it may raise :exc:`ImportError`. An :exc:`ImportError` is used by the path based finder to signal that the hook cannot find a :term:`path entry finder` for that :term:`path entry`. The exception is ignored and :term:`import path` iteration continues. The hook should expect either a string or bytes object; the encoding of bytes objects is up to the hook (e.g. it may be a file system encoding, UTF-8, or something else), and if the hook cannot decode the argument, it should raise :exc:`ImportError`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 831); *backlink*

Unknown interpreted text role "exc".

If :data:`sys.path_hooks` iteration ends with no :term:`path entry finder` being returned, then the path based finder's :meth:`~importlib.machinery.PathFinder.find_spec` method will store `None` in :data:`sys.path_importer_cache` (to indicate that there is no finder for this path entry) and return `None`, indicating that this :term:`meta path finder` could not find the module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 845); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 845); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 845); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 845); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 845); *backlink*

Unknown interpreted text role "term".

If a :term:`path entry finder` *is* returned by one of the :term:`path entry hook` callables on :data:`sys.path_hooks`, then the following protocol is used to ask the finder for a module spec, which is then used when loading the module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 852); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 852); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 852); *backlink*

Unknown interpreted text role "data".

The current working directory -- denoted by an empty string -- is handled slightly differently from other entries on :data:`sys.path`. First, if the current working directory is found to not exist, no value is stored in :data:`sys.path_importer_cache`. Second, the value for the current working directory is looked up fresh for each module lookup. Third, the path used for :data:`sys.path_importer_cache` and returned by :meth:`importlib.machinery.PathFinder.find_spec` will be the actual current working directory and not the empty string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 857); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 857); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 857); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 857); *backlink*

Unknown interpreted text role "meth".

Path entry finder protocol

In order to support imports of modules and initialized packages and also to contribute portions to namespace packages, path entry finders must implement the :meth:`~importlib.abc.PathEntryFinder.find_spec` method.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 869); *backlink*

Unknown interpreted text role "meth".

:meth:`~importlib.abc.PathEntryFinder.find_spec` takes two arguments: the fully qualified name of the module being imported, and the (optional) target module. `find_spec()` returns a fully populated spec for the module. This spec will always have "loader" set (with one exception).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 873); *backlink*

Unknown interpreted text role "meth".

To indicate to the import machinery that the spec represents a namespace :term:`portion`, the path entry finder sets "submodule_search_locations" to a list containing the portion.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 878); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 882)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   :meth:`~importlib.abc.PathEntryFinder.find_spec` replaced
   :meth:`~importlib.abc.PathEntryFinder.find_loader` and
   :meth:`~importlib.abc.PathEntryFinder.find_module`, both of which
   are now deprecated, but will be used if ``find_spec()`` is not defined.

   Older path entry finders may implement one of these two deprecated methods
   instead of ``find_spec()``.  The methods are still respected for the
   sake of backward compatibility.  However, if ``find_spec()`` is
   implemented on the path entry finder, the legacy methods are ignored.

   :meth:`~importlib.abc.PathEntryFinder.find_loader` takes one argument, the
   fully qualified name of the module being imported.  ``find_loader()``
   returns a 2-tuple where the first item is the loader and the second item
   is a namespace :term:`portion`.

   For backwards compatibility with other implementations of the import
   protocol, many path entry finders also support the same,
   traditional ``find_module()`` method that meta path finders support.
   However path entry finder ``find_module()`` methods are never called
   with a ``path`` argument (they are expected to record the appropriate
   path information from the initial call to the path hook).

   The ``find_module()`` method on path entry finders is deprecated,
   as it does not allow the path entry finder to contribute portions to
   namespace packages.  If both ``find_loader()`` and ``find_module()``
   exist on a path entry finder, the import system will always call
   ``find_loader()`` in preference to ``find_module()``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 911)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
    Calls to :meth:`~importlib.abc.PathEntryFinder.find_module` and
    :meth:`~importlib.abc.PathEntryFinder.find_loader` by the import
    system will raise :exc:`ImportWarning`.
```

Replacing the standard import system

The most reliable mechanism for replacing the entire import system is to delete the default contents of :data:`sys.meta_path`, replacing them entirely with a custom meta path hook.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 920); *backlink*

Unknown interpreted text role "data".

If it is acceptable to only alter the behaviour of import statements without affecting other APIs that access the import system, then replacing the builtin :func:`__import__` function may be sufficient. This technique may also be employed at the module level to only alter the behaviour of import statements within that module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 924); *backlink*

Unknown interpreted text role "func".

To selectively prevent the import of some modules from a hook early on the meta path (rather than disabling the standard import system entirely), it is sufficient to raise :exc:`ModuleNotFoundError` directly from :meth:`~importlib.abc.MetaPathFinder.find_spec` instead of returning `None`. The latter indicates that the meta path search should continue, while raising an exception terminates it immediately.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 930); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 930); *backlink*

Unknown interpreted text role "meth".

Package Relative Imports

Relative imports use leading dots. A single leading dot indicates a relative import, starting with the current package. Two or more leading dots indicate a relative import to the parent(s) of the current package, one level per dot after the first. For example, given the following package layout:

```
package/
    __init__.py
    subpackage1/
        __init__.py
        moduleX.py
        moduleY.py
    subpackage2/
        __init__.py
        moduleZ.py
    moduleA.py
```

In either `subpackage1/moduleX.py` or `subpackage1/__init__.py`, the following are valid relative imports:

```
from .moduleY import spam
from .moduleY import spam as ham
from . import moduleY
from ..subpackage1 import moduleY
from ..subpackage2.moduleZ import eggs
from ..moduleA import foo
```

Absolute imports may use either the `import <>` or `from <> import <>` syntax, but relative imports may only use the second form; the reason for this is that:

```
import XXX.YYY.ZZZ
```

should expose `XXX.YYY.ZZZ` as a usable expression, but .moduleY is not a valid expression.

Special considerations for __main__

The :mod:`__main__` module is a special case relative to Python's import system. As noted :ref:`elsewhere <programs>`, the `__main__` module is directly initialized at interpreter startup, much like :mod:`sys` and :mod:`builtins`. However, unlike those two, it doesn't strictly qualify as a built-in module. This is because the manner in which `__main__` is initialized depends on the flags and other options with which the interpreter is invoked.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 983); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 983); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 983); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 983); *backlink*

Unknown interpreted text role "mod".

__main__.__spec__

Depending on how :mod:`__main__` is initialized, `__main__.__spec__` gets set appropriately or to `None`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 996); *backlink*

Unknown interpreted text role "mod".

When Python is started with the :option:`-m` option, `__spec__` is set to the module spec of the corresponding module or package. `__spec__` is also populated when the `__main__` module is loaded as part of executing a directory, zipfile or other :data:`sys.path` entry.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 999); *backlink*

Unknown interpreted text role "option".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 999); *backlink*

Unknown interpreted text role "data".

In :ref:`the remaining cases <using-on-interface-options>` `__main__.__spec__` is set to `None`, as the code used to populate the :mod:`__main__` does not correspond directly with an importable module:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1004); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1004); *backlink*

Unknown interpreted text role "mod".

- interactive prompt
- :option:`-c` option

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1009); *backlink***
>
> Unknown interpreted text role "option".

- running from stdin
- running directly from a source or bytecode file

Note that `__main__.__spec__` is always `None` in the last case, *even if* the file could technically be imported directly as a module instead. Use the :option:`-m` switch if valid module metadata is desired in :mod:`__main__`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1013); *backlink***
>
> Unknown interpreted text role "option".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1013); *backlink***
>
> Unknown interpreted text role "mod".

Note also that even when `__main__` corresponds with an importable module and `__main__.__spec__` is set accordingly, they're still considered *distinct* modules. This is due to the fact that blocks guarded by `if __name__ == "__main__":` checks only execute when the module is used to populate the `__main__` namespace, and not during normal import.

Open issues

XXX It would be really nice to have a diagram.

XXX * (import_machinery.rst) how about a section devoted just to the attributes of modules and packages, perhaps expanding upon or supplanting the related entries in the data model reference page?

XXX runpy, pkgutil, et al in the library manual should all get "See Also" links at the top pointing to the new import system section.

XXX Add more explanation regarding the different ways in which `__main__` is initialized?

XXX Add more info on `__main__` quirks/pitfalls (i.e. copy from PEP 395).

References

The import machinery has evolved considerably since Python's early days. The original specification for packages is still available to read, although some details have changed since the writing of that document.

The original specification for :data:`sys.meta_path` was PEP 302, with subsequent extension in PEP 420.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1052); *backlink***
>
> Unknown interpreted text role "data".

PEP 420 introduced :term:`namespace packages <namespace package>` for Python 3.3. PEP 420 also introduced the :meth:`find_loader` protocol as an alternative to :meth:`find_module`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1055); *backlink***
>
> Unknown interpreted text role "term".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1055); *backlink***
>
> Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1055); *backlink*

Unknown interpreted text role "meth".

PEP 366 describes the addition of the `__package__` attribute for explicit relative imports in main modules.

PEP 328 introduced absolute and explicit relative imports and initially proposed `__name__` for semantics PEP 366 would eventually specify for `__package__`.

PEP 338 defines executing modules as scripts.

PEP 451 adds the encapsulation of per-module import state in spec objects. It also off-loads most of the boilerplate responsibilities of loaders back onto the import machinery. These changes allow the deprecation of several APIs in the import system and also addition of new methods to finders and loaders.

Footnotes

[1] See :class:`types.ModuleType`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1076); *backlink*

Unknown interpreted text role "class".

[2] The importlib implementation avoids using the return value directly. Instead, it gets the module object by looking the module name up in :data:`sys.modules`. The indirect effect of this is that an imported module may replace itself in :data:`sys.modules`. This is implementation-specific behavior that is not guaranteed to work in other Python implementations.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1078); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1078); *backlink*

Unknown interpreted text role "data".

[3] In legacy code, it is possible to find instances of :class:`imp.NullImporter` in the :data:`sys.path_importer_cache`. It is recommended that code be changed to use `None` instead. See :ref:`portingpythoncode` for more details.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1085); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1085); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\reference\[cpython-main][Doc][reference]import.rst, line 1085); *backlink*

Unknown interpreted text role "ref".