31.2. pkgutil — Package extension utility

Source code: Lib/pkgutil.py

This module provides utilities for the import system, in particular package support.

class pkgutil.**ModuleInfo**(*module_finder*, *name*, *ispkg*)

A namedtuple that holds a brief summary of a module's info.

New in version 3.6.

pkgutil.**extend_path**(*path*, *name*)

Extend the search path for the modules which comprise a package. Intended use is to place the following code in a package's `__init__.py`:

```
from pkgutil import extend_path
__path__ = extend_path(__path__, __name__)
```

This will add to the package's `__path__` all subdirectories of directories on `sys.path` named after the package. This is useful if one wants to distribute different parts of a single logical package as multiple directories.

It also looks for `*.pkg` files beginning where `*` matches the *name* argument. This feature is similar to `*.pth` files (see the `site` module for more information), except that it doesn't special-case lines starting with `import`. A `*.pkg` file is trusted at face value: apart from checking for duplicates, all entries found in a `*.pkg` file are added to the path, regardless of whether they exist on the filesystem. (This is a feature.)

If the input path is not a list (as is the case for frozen packages) it is returned unchanged. The input path is not modified; an extended copy is returned. Items are only appended to the copy at the end.

It is assumed that `sys.path` is a sequence. Items of `sys.path` that are not strings referring to existing directories are ignored. Unicode items on `sys.path` that cause errors when used as filenames may cause this function to raise an exception (in line with `os.path.isdir()` behavior).

class pkgutil.**ImpImporter**(*dirname=None*)

PEP 302 Finder that wraps Python's "classic" import algorithm.

If *dirname* is a string, a **PEP 302** finder is created that searches that directory. If *dirname* is `None`, a **PEP 302** finder is created that searches the current `sys.path`, plus any modules that are frozen or built-in.

Note that `ImpImporter` does not currently support being used by placement on `sys.meta_path`.

Deprecated since version 3.3: This emulation is no longer needed, as the standard import mechanism is now fully PEP 302 compliant and available in `importlib`.

class `pkgutil.`**`ImpLoader`**(*fullname*, *file*, *filename*, *etc*)

Loader that wraps Python's "classic" import algorithm.

Deprecated since version 3.3: This emulation is no longer needed, as the standard import mechanism is now fully PEP 302 compliant and available in `importlib`.

`pkgutil.`**`find_loader`**(*fullname*)

Retrieve a module loader for the given *fullname*.

This is a backwards compatibility wrapper around `importlib.util.find_spec()` that converts most failures to `ImportError` and only returns the loader rather than the full `ModuleSpec`.

Changed in version 3.3: Updated to be based directly on `importlib` rather than relying on the package internal PEP 302 import emulation.

Changed in version 3.4: Updated to be based on **PEP 451**

`pkgutil.`**`get_importer`**(*path_item*)

Retrieve a finder for the given *path_item*.

The returned finder is cached in `sys.path_importer_cache` if it was newly created by a path hook.

The cache (or part of it) can be cleared manually if a rescan of `sys.path_hooks` is necessary.

Changed in version 3.3: Updated to be based directly on `importlib` rather than relying on the package internal PEP 302 import emulation.

`pkgutil.`**`get_loader`**(*module_or_name*)

Get a loader object for *module_or_name*.

If the module or package is accessible via the normal import mechanism, a wrapper around the relevant part of that machinery is returned. Returns `None` if the module cannot be found or imported. If the named module is not already imported, its containing package (if any) is imported, in order to establish the package `__path__`.

Changed in version 3.3: Updated to be based directly on `importlib` rather than relying on the package internal PEP 302 import emulation.

Changed in version 3.4: Updated to be based on **PEP 451**

`pkgutil.`**`iter_importers`**(*fullname=''*)

Yield finder objects for the given module name.

If fullname contains a '.', the finders will be for the package containing fullname, otherwise they will be all registered top level finders (i.e. those on both sys.meta_path and sys.path_hooks).

If the named module is in a package, that package is imported as a side effect of invoking this function.

If no module name is specified, all top level finders are produced.

Changed in version 3.3: Updated to be based directly on `importlib` rather than relying on the package internal PEP 302 import emulation.

`pkgutil.`**`iter_modules`**(*path=None*, *prefix=''*)

Yields `ModuleInfo` for all submodules on *path*, or, if *path* is `None`, all top-level modules on `sys.path`.

path should be either `None` or a list of paths to look for modules in.

prefix is a string to output on the front of every module name on output.

> **Note:** Only works for a finder which defines an `iter_modules()` method. This interface is non-standard, so the module also provides implementations for `importlib.machinery.FileFinder` and `zipimport.zipimporter`.

Changed in version 3.3: Updated to be based directly on `importlib` rather than relying on the package internal PEP 302 import emulation.

`pkgutil.`**`walk_packages`**(*path=None*, *prefix=''*, *onerror=None*)

Yields `ModuleInfo` for all modules recursively on *path*, or, if *path* is `None`, all accessible modules.

path should be either `None` or a list of paths to look for modules in.

prefix is a string to output on the front of every module name on output.

Note that this function must import all *packages* (*not* all modules!) on the given *path*, in order to access the `__path__` attribute to find submodules.

onerror is a function which gets called with one argument (the name of the package which was being imported) if any exception occurs while trying to import a package. If no *onerror* function is supplied, `ImportError`s are caught and ignored, while all other exceptions are propagated, terminating the search.

Examples:

```
# list all modules python can access
walk_packages()

# list all submodules of ctypes
walk_packages(ctypes.__path__, ctypes.__name__ + '.')
```

Note: Only works for a finder which defines an `iter_modules()` method. This interface is non-standard, so the module also provides implementations for `importlib.machinery.FileFinder` and `zipimport.zipimporter`.

Changed in version 3.3: Updated to be based directly on `importlib` rather than relying on the package internal PEP 302 import emulation.

`pkgutil.`**`get_data`**(*package*, *resource*)

Get a resource from a package.

This is a wrapper for the loader `get_data` API. The *package* argument should be the name of a package, in standard module format (`foo.bar`). The *resource* argument should be in the form of a relative filename, using `/` as the path separator. The parent directory name `..` is not allowed, and nor is a rooted name (starting with a `/`).

The function returns a binary string that is the contents of the specified resource.

For packages located in the filesystem, which have already been imported, this is the rough equivalent of:

```
d = os.path.dirname(sys.modules[package].__file__)
data = open(os.path.join(d, resource), 'rb').read()
```

If the package cannot be located or loaded, or it uses a loader which does not support `get_data`, then `None` is returned. In particular, the loader for namespace packages does not support `get_data`.