:mod:`pkgutil` --- Package extension utility

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 4)

Unknown directive type "module".

```
.. module:: pkgutil
   :synopsis: Utilities for the import system.
```

Source code: :source:`Lib/pkgutil.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 7); *backlink*

Unknown interpreted text role "source".

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

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 18)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 20)

Unknown directive type "function".

```
.. function:: 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 :file:`__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 :data:`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 :file:`\*.pkg` files beginning where ``*`` matches the
   *name* argument.  This feature is similar to :file:`\*.pth` files (see the
   :mod:`site` module for more information), except that it doesn't special-case
   lines starting with ``import``.  A :file:`\*.pkg` file is trusted at face
   value: apart from checking for duplicates, all entries found in a
   :file:`\*.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 :data:`sys.path` is a sequence.  Items of :data:`sys.path`
   that are not strings referring to existing directories are ignored. Unicode
   items on :data:`sys.path` that cause errors when used as filenames may cause
   this function to raise an exception (in line with :func:`os.path.isdir`
   behavior).
```

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 :data:`sys.path`, plus any modules that are frozen or built-in.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 56); *backlink*

Unknown interpreted text role "data".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 61); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 61); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 64)

Unknown directive type "deprecated".

```
.. deprecated:: 3.3
   This emulation is no longer needed, as the standard import mechanism
   is now fully :pep:`302` compliant and available in :mod:`importlib`.
```

:term:`Loader <loader>` that wraps Python's "classic" import algorithm.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 71); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 73)

Unknown directive type "deprecated".

```
.. deprecated:: 3.3
   This emulation is no longer needed, as the standard import mechanism
   is now fully :pep:`302` compliant and available in :mod:`importlib`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 78)

Unknown directive type "function".

```
.. function:: find_loader(fullname)

   Retrieve a module :term:`loader` for the given *fullname*.

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

   .. versionchanged:: 3.3
      Updated to be based directly on :mod:`importlib` rather than relying
      on the package internal :pep:`302` import emulation.

   .. versionchanged:: 3.4
      Updated to be based on :pep:`451`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 94)

Unknown directive type "function".

```
.. function:: get_importer(path_item)

   Retrieve a :term:`finder` for the given *path_item*.

   The returned finder is cached in :data:`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
   :data:`sys.path_hooks` is necessary.

   .. versionchanged:: 3.3
      Updated to be based directly on :mod:`importlib` rather than relying
      on the package internal :pep:`302` import emulation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 109)

Unknown directive type "function".

```
.. function:: get_loader(module_or_name)

   Get a :term:`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__``.

   .. versionchanged:: 3.3
      Updated to be based directly on :mod:`importlib` rather than relying
      on the package internal :pep:`302` import emulation.

   .. versionchanged:: 3.4
      Updated to be based on :pep:`451`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 127)

Unknown directive type "function".

```
.. function:: iter_importers(fullname='')

   Yield :term:`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 :data:`sys.meta_path` and :data:`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.

   .. versionchanged:: 3.3
      Updated to be based directly on :mod:`importlib` rather than relying
      on the package internal :pep:`302` import emulation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 145)

Unknown directive type "function".

```
.. function:: iter_modules(path=None, prefix='')

   Yields :class:`ModuleInfo` for all submodules on *path*, or, if
   *path* is ``None``, all top-level modules on :data:`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 :term:`finder` which defines an ``iter_modules()``
      method. This interface is non-standard, so the module also provides
      implementations for :class:`importlib.machinery.FileFinder` and
      :class:`zipimport.zipimporter`.

   .. versionchanged:: 3.3
      Updated to be based directly on :mod:`importlib` rather than relying
      on the package internal :pep:`302` import emulation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 166)

Unknown directive type "function".

```
.. function:: walk_packages(path=None, prefix='', onerror=None)

   Yields :class:`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, :exc:`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 :term:`finder` which defines an ``iter_modules()``
      method. This interface is non-standard, so the module also provides
      implementations for :class:`importlib.machinery.FileFinder` and
      :class:`zipimport.zipimporter`.

   .. versionchanged:: 3.3
      Updated to be based directly on :mod:`importlib` rather than relying
      on the package internal :pep:`302` import emulation.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 205)

Unknown directive type "function".

```
.. function:: get_data(package, resource)

   Get a resource from a package.

   This is a wrapper for the :term:`loader`
   :meth:`get_data <importlib.abc.ResourceLoader.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 :term:`loader`
which does not support :meth:`get_data <importlib.abc.ResourceLoader.get_data>`,
then ``None`` is returned.  In particular, the :term:`loader` for
:term:`namespace packages <namespace package>` does not support
:meth:`get_data <importlib.abc.ResourceLoader.get_data>`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)pkgutil.rst, line 232)

Unknown directive type "function".

```
.. function:: resolve_name(name)

   Resolve a name to an object.

   This functionality is used in numerous places in the standard library (see
   :issue:`12915`) - and equivalent functionality is also in widely used
   third-party packages such as setuptools, Django and Pyramid.

   It is expected that *name* will be a string in one of the following
   formats, where W is shorthand for a valid Python identifier and dot stands
   for a literal period in these pseudo-regexes:

   * ``W(.W)*``
   * ``W(.W)*:(W(.W)*)?``

   The first form is intended for backward compatibility only. It assumes that
   some part of the dotted name is a package, and the rest is an object
   somewhere within that package, possibly nested inside other objects.
   Because the place where the package stops and the object hierarchy starts
   can't be inferred by inspection, repeated attempts to import must be done
   with this form.

   In the second form, the caller makes the division point clear through the
   provision of a single colon: the dotted name to the left of the colon is a
   package to be imported, and the dotted name to the right is the object
   hierarchy within that package. Only one import is needed in this form. If
   it ends with the colon, then a module object is returned.

   The function will return an object (which might be a module), or raise one
   of the following exceptions:

   :exc:`ValueError` -- if *name* isn't in a recognised format.

   :exc:`ImportError` -- if an import failed when it shouldn't have.

   :exc:`AttributeError` -- If a failure occurred when traversing the object
   hierarchy within the imported package to get to the desired object.

   .. versionadded:: 3.9
```