Using :mod:`!importlib.metadata`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 4); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 7)

Unknown directive type "module".

```
.. module:: importlib.metadata
   :synopsis: The implementation of the importlib metadata.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 10)

Unknown directive type "versionadded".

```
.. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 11)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   ``importlib.metadata`` is no longer provisional.
```

Source code: :source:`Lib/importlib/metadata/__init__.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 14); *backlink*

Unknown interpreted text role "source".

`importlib.metadata` is a library that provides for access to installed package metadata. Built in part on Python's import system, this library intends to replace similar functionality in the entry point API and metadata API of `pkg_resources`. Along with :mod:`importlib.resources` in Python 3.7 and newer (backported as importlib_resources for older versions of Python), this can eliminate the need to use the older and less efficient `pkg_resources` package.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 16); *backlink*

Unknown interpreted text role "mod".

By "installed package" we generally mean a third-party package installed into Python's `site-packages` directory via tools such as pip. Specifically, it means a package with either a discoverable `dist-info` or `egg-info` directory, and metadata defined by PEP 566 or its older specifications. By default, package metadata can live on the file system or in zip archives on :data:`sys.path`. Through an extension mechanism, the metadata can live almost anywhere.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 25); *backlink*

Unknown interpreted text role "data".

Overview

Let's say you wanted to get the version string for a package you've installed using `pip`. We start by creating a virtual environment and installing something into it:

```
$ python3 -m venv example
$ source example/bin/activate
(example) $ pip install wheel
```

You can get the version string for `wheel` by running the following:

```
(example) $ python
>>> from importlib.metadata import version  # doctest: +SKIP
>>> version('wheel')  # doctest: +SKIP
'0.32.3'
```

You can also get the set of entry points keyed by group, such as `console_scripts`, `distutils.commands` and others. Each group contains a sequence of :ref:`EntryPoint <entry-points>` objects.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 57); *backlink*

Unknown interpreted text role "ref".

You can get the :ref:`metadata for a distribution <metadata>`:

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

Unknown interpreted text role "ref".

```
>>> list(metadata('wheel'))  # doctest: +SKIP
['Metadata-Version', 'Name', 'Version', 'Summary', 'Home-page', 'Author', 'Author-email', 'Maintainer', 'Maintainer-email', 'License', '
```

You can also get a :ref:`distribution's version number <version>`, list its :ref:`constituent files <files>`, and get a list of the distribution's :ref:`requirements`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 66); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 66); *backlink*

Unknown interpreted text role "ref".

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 66); backlink

Unknown interpreted text role "ref".

Functional API

This package provides the following functionality via its public API.

Entry points

The `entry_points()` function returns a collection of entry points. Entry points are represented by `EntryPoint` instances; each `EntryPoint` has a `.name`, `.group`, and `.value` attributes and a `.load()` method to resolve the value. There are also `.module`, `.attr`, and `.extras` attributes for getting the components of the `.value` attribute.

Query all entry points:

```
>>> eps = entry_points()  # doctest: +SKIP
```

The `entry_points()` function returns an `EntryPoints` object, a sequence of all `EntryPoint` objects with `names` and `groups` attributes for convenience:

```
>>> sorted(eps.groups)  # doctest: +SKIP
['console_scripts', 'distutils.commands', 'distutils.setup_keywords', 'egg_info.writers', 'setuptools.installation']
```

`EntryPoints` has a `select` method to select entry points matching specific properties. Select entry points in the `console_scripts` group:

```
>>> scripts = eps.select(group='console_scripts')  # doctest: +SKIP
```

Equivalently, since `entry_points` passes keyword arguments through to select:

```
>>> scripts = entry_points(group='console_scripts')  # doctest: +SKIP
```

Pick out a specific script named "wheel" (found in the wheel project):

```
>>> 'wheel' in scripts.names  # doctest: +SKIP
True
>>> wheel = scripts['wheel']  # doctest: +SKIP
```

Equivalently, query for that entry point during selection:

```
>>> (wheel,) = entry_points(group='console_scripts', name='wheel')  # doctest: +SKIP
>>> (wheel,) = entry_points().select(group='console_scripts', name='wheel')  # doctest: +SKIP
```

Inspect the resolved entry point:

```
>>> wheel  # doctest: +SKIP
EntryPoint(name='wheel', value='wheel.cli:main', group='console_scripts')
>>> wheel.module  # doctest: +SKIP
'wheel.cli'
>>> wheel.attr  # doctest: +SKIP
'main'
>>> wheel.extras  # doctest: +SKIP
[]
>>> main = wheel.load()  # doctest: +SKIP
>>> main  # doctest: +SKIP
<function main at 0x103528488>
```

The `group` and `name` are arbitrary values defined by the package author and usually a client will wish to resolve all entry points for a particular group. Read the setuptools docs for more information on entry points, their definition, and usage.

Compatibility Note

The "selectable" entry points were introduced in `importlib_metadata` 3.6 and Python 3.10. Prior to those changes, `entry_points` accepted no parameters and always returned a dictionary of entry points, keyed by group. For compatibility, if no parameters are passed to entry_points, a `SelectableGroups` object is returned, implementing that dict interface. In the future, calling `entry_points` with no parameters will return an `EntryPoints` object. Users should rely on the selection interface to retrieve entry points by group.

Distribution metadata

Every distribution includes some metadata, which you can extract using the `metadata()` function:

```
>>> wheel_metadata = metadata('wheel')  # doctest: +SKIP
```

The keys of the returned data structure, a `PackageMetadata`, name the metadata keywords, and the values are returned unparsed from the distribution metadata:

```
>>> wheel_metadata['Requires-Python']  # doctest: +SKIP
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
```

`PackageMetadata` also presents a `json` attribute that returns all the metadata in a JSON-compatible form per PEP 566:

```
>>> wheel_metadata.json['requires_python']
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 177)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   The ``Description`` is now included in the metadata when presented
   through the payload. Line continuation characters have been removed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 181)

Unknown directive type "versionadded".

```
.. versionadded:: 3.10
   The ``json`` attribute was added.
```

Distribution versions

The `version()` function is the quickest way to get a distribution's version number, as a string:

```
>>> version('wheel')  # doctest: +SKIP
'0.32.3'
```

Distribution files

You can also get the full set of files contained within a distribution. The `files()` function takes a distribution package name and returns all of the files installed by this distribution. Each file object returned is a `PackagePath`, a :class:`pathlib.PurePath` derived object with additional `dist`, `size`, and `hash` properties as indicated by the metadata. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 202); backlink

Unknown interpreted text role "class".

```
>>> util = [p for p in files('wheel') if 'util.py' in str(p)][0]  # doctest: +SKIP
>>> util  # doctest: +SKIP
PackagePath('wheel/util.py')
>>> util.size  # doctest: +SKIP
859
>>> util.dist  # doctest: +SKIP
<importlib.metadata._hooks.PathDistribution object at 0x101e0cef0>
>>> util.hash  # doctest: +SKIP
<FileHash mode: sha256 value: bYkw5oMccfazVCoYQwKkkemoVyMAFoR34mmKBx8R1NI>
```

Once you have the file, you can also read its contents:

```
>>> print(util.read_text())  # doctest: +SKIP
import base64
import sys
...
def as_bytes(s):
    if isinstance(s, text_type):
        return s.encode('utf-8')
    return s
```

You can also use the `locate` method to get a the absolute path to the file:

```
>>> util.locate()  # doctest: +SKIP
PosixPath('/home/gustav/example/lib/site-packages/wheel/util.py')
```

In the case where the metadata file listing files (RECORD or SOURCES.txt) is missing, `files()` will return `None`. The caller may wish to wrap calls to `files()` in always_iterable or otherwise guard against this condition if the target distribution is not known to have the metadata present.

Distribution requirements

To get the full set of requirements for a distribution, use the `requires()` function:

```
>>> requires('wheel')  # doctest: +SKIP
["pytest (>=3.0.0) ; extra == 'test'", "pytest-cov ; extra == 'test'"]
```

Package distributions

A convenience method to resolve the distribution or distributions (in the case of a namespace package) for top-level Python packages or modules:

```
>>> packages_distributions()
{'importlib_metadata': ['importlib-metadata'], 'yaml': ['PyYAML'], 'jaraco': ['jaraco.classes', 'jaraco.functools'], ...}
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 265)

Unknown directive type "versionadded".

```
.. versionadded:: 3.10
```

Distributions

While the above API is the most common and convenient usage, you can get all of that information from the `Distribution` class. A `Distribution` is an abstract object that represents the metadata for a Python package. You can get the `Distribution` instance:

```
>>> from importlib.metadata import distribution  # doctest: +SKIP
>>> dist = distribution('wheel')  # doctest: +SKIP
```

Thus, an alternative way to get the version number is through the `Distribution` instance:

```
>>> dist.version  # doctest: +SKIP
'0.32.3'
```

There are all kinds of additional metadata available on the `Distribution` instance:

```
>>> dist.metadata['Requires-Python']  # doctest: +SKIP
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
>>> dist.metadata['License']  # doctest: +SKIP
'MIT'
```

The full set of available metadata is not described here. See PEP 566 for additional details.

Extending the search algorithm

Because package metadata is not available through :data:`sys.path` searches, or package loaders directly, the metadata for a package is found through import system :ref:`finders <finders-and-loaders>`. To find a distribution package's metadata, `importlib.metadata` queries the list of :term:`meta path finders <meta path finder>` on :data:`sys.meta_path`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 301); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 301); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 301); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 301); *backlink*

Unknown interpreted text role "data".

The default `PathFinder` for Python includes a hook that calls into `importlib.metadata.MetadataPathFinder` for finding distributions loaded from typical file-system-based paths.

The abstract class :py:class:`importlib.abc.MetaPathFinder` defines the interface expected of finders by Python's import system. `importlib.metadata` extends this protocol by looking for an optional `find_distributions` callable on the finders from :data:`sys.meta_path` and presents this extended interface as the `DistributionFinder` abstract base class, which defines this abstract method:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 311); *backlink*

Unknown interpreted text role "py:class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)importlib.metadata.rst, line 311); *backlink*

Unknown interpreted text role "data".

```
@abc.abstractmethod
def find_distributions(context=DistributionFinder.Context()):
    """Return an iterable of all Distribution instances capable of
    loading the metadata for packages for the indicated ``context``.
    """
```

The `DistributionFinder.Context` object provides `.path` and `.name` properties indicating the path to search and name to match and may supply other relevant context.

What this means in practice is that to support finding distribution package metadata in locations other than the file system, subclass `Distribution` and implement the abstract methods. Then from a custom finder, return instances of this derived `Distribution` in the `find_distributions()` method.