importlib_metadata Documentation

Release 0.1

Brett Cannon, Barry Warsaw

Jul 13, 2019

CONTENTS:

`importlib_metadata` is a library which provides an API for accessing an installed package's metadata, such as its entry points or its top-level name. This functionality intends to replace most uses of `pkg_resources` entry point API and metadata API. Along with `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.

`importlib_metadata` is a backport of Python 3.8's standard library importlib.metadata module for Python 2.7, and 3.4 through 3.7. Users of Python 3.8 and beyond are encouraged to use the standard library module, and in fact for these versions, `importlib_metadata` just shadows that module. Developers looking for detailed API descriptions should refer to the Python 3.8 standard library documentation.

The documentation here includes a general *usage* guide.

CHAPTER

ONE

USING IMPORTLIB_METADATA

`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 `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.

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 `sys.path`. Through an extension mechanism, the metadata can live almost anywhere.

1.1 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 importlib_metadata
(example) $ pip install wheel
```

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

```
(example) $ python
>>> from importlib_metadata import version
>>> version('wheel')
'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 *EntryPoint* objects.

You can get the *metadata for a distribution*:

```
>>> list(metadata('wheel'))
['Metadata-Version', 'Name', 'Version', 'Summary', 'Home-page', 'Author', 'Author-
↪email', 'Maintainer', 'Maintainer-email', 'License', 'Project-URL', 'Project-URL',
↪'Project-URL', 'Keywords', 'Platform', 'Classifier', 'Classifier', 'Classifier',
↪'Classifier', 'Classifier', 'Classifier', 'Classifier', 'Classifier', 'Classifier',
↪'Classifier', 'Classifier', 'Classifier', 'Requires-Python', 'Provides-Extra',
↪'Requires-Dist', 'Requires-Dist']
```

You can also get a *distribution's version number*, list its *constituent files*, and get a list of the distribution's *Distribution requirements*.

1.2 Functional API

This package provides the following functionality via its public API.

1.2.1 Entry points

The `entry_points()` function returns a dictionary of all entry points, keyed by group. Entry points are represented by `EntryPoint` instances; each `EntryPoint` has a `.name`, `.group`, and `.value` attributes and a `.load()` method to resolve the value:

```
>>> eps = entry_points()
>>> list(eps)
['console_scripts', 'distutils.commands', 'distutils.setup_keywords', 'egg_info.
↪writers', 'setuptools.installation']
>>> scripts = eps['console_scripts']
>>> wheel = [ep for ep in scripts if ep.name == 'wheel'][0]
>>> wheel
EntryPoint(name='wheel', value='wheel.cli:main', group='console_scripts')
>>> main = wheel.load()
>>> main
<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 entrypoints, their definition, and usage.

1.2.2 Distribution metadata

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

```
>>> wheel_metadata = metadata('wheel')
```

The keys of the returned data structure[1] name the metadata keywords, and their values are returned unparsed from the distribution metadata:

```
>>> wheel_metadata['Requires-Python']
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
```

1.2.3 Distribution versions

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

```
>>> version('wheel')
'0.32.3'
```

[1] Technically, the returned distribution metadata object is an email.message.Message instance, but this is an implementation detail, and not part of the stable API. You should only use dictionary-like methods and syntax to access the metadata contents.

1.2.4 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 pathlib.Path derived object with additional `dist`, `size`, and `hash` properties as indicated by the metadata. For example:

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

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

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

1.2.5 Distribution requirements

To get the full set of requirements for a distribution, use the `requires()` function. Note that this returns an iterator:

```
>>> list(requires('wheel'))
["pytest (>=3.0.0) ; extra == 'test'"]
```

1.3 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
>>> dist = distribution('wheel')
```

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

```
>>> dist.version
'0.32.3'
```

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

```
>>> d.metadata['Requires-Python']
'>=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*'
>>> d.metadata['License']
'MIT'
```

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

1.4 Extending the search algorithm

Because package metadata is not available through `sys.path` searches, or package loaders directly, the metadata for a package is found through import system finders. To find a distribution package's metadata, `importlib_metadata` queries the list of meta path finders on sys.meta_path.

By default `importlib_metadata` installs a finder for distribution packages found on the file system. This finder doesn't actually find any *packages*, but it can find the packages' metadata.

The abstract 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 `sys.meta_path`. If the finder has this method, it must return an iterator over instances of the `Distribution` abstract class. This method must have the signature:

```
def find_distributions(name=None, path=None):
    """Return an iterable of all Distribution instances capable of
    loading the metadata for packages matching the name
    (or all names if not supplied) along the paths in the list
    of directories ``path`` (defaults to sys.path).
    """
```

What this means in practice is that to support finding distribution package metadata in locations other than the file system, you should derive from `Distribution` and implement the `load_metadata()` method. This takes a single argument which is the name of the package whose metadata is being found. This instance of the `Distribution` base abstract class is what your finder's `find_distributions()` method should return.

CHAPTER

TWO

IMPORTLIB_METADATA NEWS

2.1 0.19

- Restrain over-eager egg metadata resolution.

2.2 0.18

2019-06-09

- Parse entry points case sensitively. Closes #68
- Add a version constraint on the backport configparser package. Closes #66

2.3 0.17

2019-05-29

- Fix a permission problem in the tests on Windows.

2.4 0.16

2019-05-29

- Don't crash if there exists an EGG-INFO directory on sys.path.

2.5 0.15

2019-05-24

- Fix documentation.

2.6 0.14

2019-05-24

- Removed `local_distribution` function from the API. **This backward-incompatible change removes this behavior summarily**. Projects should remove their reliance on this behavior. A replacement behavior is under review in the pep517 project. Closes #42.

2.7 0.13

2019-05-18

- Update docstrings to match PEP 8. Closes #63.
- Merged modules into one module. Closes #62.

2.8 0.12

2019-05-14

- Add support for eggs. !65; Closes #19.

2.9 0.11

2019-05-09

- Support generic zip files (not just wheels). Closes #59
- Support zip files with multiple distributions in them. Closes #60
- Fully expose the public API in `importlib_metadata.__all__`.

2.10 0.10

2019-05-07

- The `Distribution` ABC is now officially part of the public API. Closes #37.
- Fixed support for older single file egg-info formats. Closes #43.
- Fixed a testing bug when `$CWD` has spaces in the path. Closes #50.
- Add Python 3.8 to the `tox` testing matrix.

2.11 0.9

2019-03-25

- Fixed issue where entry points without an attribute would raise an Exception. Closes #40.
- Removed unused `name` parameter from `entry_points()`. Closes #44.
- `DistributionFinder` classes must now be instantiated before being placed on `sys.meta_path`.

2.12 0.8

2019-01-01

- This library can now discover/enumerate all installed packages. **This backward-incompatible change alters the protocol finders must implement to support distribution package discovery.** Closes #24.
- The signature of `find_distributions()` on custom installer finders should now accept two parameters, `name` and `path` and these parameters must supply defaults.
- The `entry_points()` method no longer accepts a package name but instead returns all entry points in a dictionary keyed by the `EntryPoint.group`. The `resolve` method has been removed. Instead, call `EntryPoint.load()`, which has the same semantics as `pkg_resources` and `entrypoints`. **This is a backward incompatible change.**
- Metadata is now always returned as Unicode text regardless of Python version. Closes #29.
- This library can now discover metadata for a 'local' package (found in the current-working directory). Closes #27.
- Added `files()` function for resolving files from a distribution.
- Added a new `requires()` function, which returns the requirements for a package suitable for parsing by `packaging.requirements.Requirement`. Closes #18.
- The top-level `read_text()` function has been removed. Use `PackagePath.read_text()` on instances returned by the `files()` function. **This is a backward incompatible change.**
- Release dates are now automatically injected into the changelog based on SCM tags.

2.13 0.7

2018-11-27

- Fixed issue where packages with dashes in their names would not be discovered. Closes #21.
- Distribution lookup is now case-insensitive. Closes #20.
- Wheel distributions can no longer be discovered by their module name. Like Path distributions, they must be indicated by their distribution package name.

2.14 0.6

2018-10-07

- Removed `importlib_metadata.distribution` function. Now the public interface is primarily the utility functions exposed in `importlib_metadata.__all__`. Closes #14.
- Added two new utility functions `read_text` and `metadata`.

2.15 0.5

2018-09-18

- Updated README and removed details about Distribution class, now considered private. Closes #15.

- Added test suite support for Python 3.4+.
- Fixed SyntaxErrors on Python 3.4 and 3.5. !12
- Fixed errors on Windows joining Path elements. !15

2.16 0.4

2018-09-14

- Housekeeping.

2.17 0.3

2018-09-14

- Added usage documentation. Closes #8
- Add support for getting metadata from wheels on `sys.path`. Closes #9

2.18 0.2

2018-09-11

- Added `importlib_metadata.entry_points()`. Closes #1
- Added `importlib_metadata.resolve()`. Closes #12
- Add support for Python 2.7. Closes #4

2.19 0.1

2018-09-09

- Initial release.

CHAPTER

THREE

PROJECT DETAILS

- Project home: https://gitlab.com/python-devs/importlib_metadata
- Report bugs at: https://gitlab.com/python-devs/importlib_metadata/issues
- Code hosting: https://gitlab.com/python-devs/importlib_metadata.git
- Documentation: http://importlib_metadata.readthedocs.io/

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search