PyPI to 0install Documentation

Release 0.1.0

Tim Diels

Mar 14, 2017

Contents

Contents:

CHAPTER 1

User documentation

Installation

To install:

```
git clone https://github.com/timdiels/pypi-to-0install.git
cd pypi-to-0install
python3 -m venv venv && . venv/bin/activate
pip install -r requirements.txt
wget https://downloads.sf.net/project/zero-install/0install/2.3.4/0install-2.3.4.tar.
↪bz2
tar -xaf 0install-2.3.4.tar.bz2
mv 0install-2.3.4/zeroinstall .
rm -rf 0install-2.3.4*  # cleanup, optional
```

python3 should be at least python 3.4.

Running

To run:

```
. $repo_root/venv/bin/activate
export PYTHONPATH="$repo_root"
python3 $repo_root/pypi_to_0install/main.py
```

CHAPTER 2

Developer documentation

For full testing, set up a local mirror of PyPI and use that instead. This way you do not download all of PyPI more than once. You can set it up with bandersnatch.

For tests, set PYTHONPATH as in the regular run instructions, then run `pytest`.

CHAPTER 3

Design

PyPI XMLRPC interface

This document clarifies some aspects of PyPI's XMLRPC interface, which is used for the conversion.

While PyPI's metadata is structured, little input validation is performed. E.g. some fields may be `None`, `''` or something bogus such as `UNKNOWN` (analyzing pypi metadata). E.g. author_email isn't required to be an email adress.

The following is a non-exhaustive list of descriptions of the output of some of the interface's commands:

- release_data:
  - name: the package name. This is not the canonical name. You are required to use this name when requesting info through the interface, not the canonical name.
  - home_page: a URL.
  - license: a string such as `GPL`, potentially has variations such as `General Public License` (and bogus values such as `LICENSE.txt`).
  - summary: short description string
  - description: long description string in reStructuredText format
  - keywords: whitespace separated list of keywords as string
  - classifiers: list of Trove classifiers, list of str.
  - release_url: the PyPI page corresponding to this version
  - package_url: the PyPI page of the latest version of this package
  - docs_url: if the package has hosted its documentation at PyPI, this URL points to it. Submitting documentation to PyPI has been deprecated (in favor of Read the Docs).
  - platform: do not use. It is tied to a version, but not to a download url (release_urls), so it can't be meaningful. E.g. for numpy it returns `Windows` while numpy is supported on Linux as well.
  - stable_version: always empty string, useless.

  - requires, requires_dist, provides, provides_dist: seems these are not returned or are always empty

- release_urls:
  - packagetype:

    Meaning of the most common values:

    * sdist: source distribution
    * bdist_wheel: Python wheel
    * bdist_egg: Python egg, can be converted to a wheel
    * bdist_wininst: `...  bdist --format=wininst` output, a self-extracting ZIP for Windows; but it can be converted to a wheel
  - python_version: unknown. Examples: `source`, `any`, `py3`, ...
  - url: the download url
  - filename: file name of the download. For a wheel, this follows the wheel file name convention. Eggs also follow a file name convention. Metadata such as which platform the download is for is missing, instead one has to derive it from the filename or download and inspect the binary.

Conversion: General

This details how PyPI packages are converted to ZI feeds. Parts specific to the *packagetype* (sdist, wheel, ...) are detailed in the other conversion pages. I will use shorthands such as `release_data['summary']` throughout the text (instead of `release_data(...)['summary']`) to refer to the PyPI XMLRPC interface.

We will refer to a PyPI project as a package (e.g. numpy; this follows PyPI's terminology) and its downloads as distributions (e.g. an sdist/wheel of numpy).

Overview

This pseudo-feed gives an overview of the conversion (end tags omitted):

```
<interface>
  <name>{canonical_name(release_data['name'])}
  <summary>{release_data['summary']}
  <homepage>{release_data['home_page']}
  <description>{pandoc(release_data['description'], from=rst, to=txt)}
  <category type={uri_to_trove_namespace}>{release_data['classifiers'][i]}
    ...
  <needs-terminal/> iff ``Environment :: Console`` in classifiers

  <implementation
    id={release_urls['path']}
    version={converted_version}
    released={format(release_urls['upload_time'], 'YYYY-MM-DD')}
    stability={stability}
    langs={langs}
    license={license}
    ...
  >
    <requires interface='https://pypi_to_zi_feeds.github.io/...' importance='
↪{importance}' />
    ...
```

Where:

```
def canonical_name(pypi_name):
    re.sub(r"[-_.]+", "-", pypi_name).lower()
```

Here, `release_data` refers to the release data of the newest release/version of the package.

The description is converted from reST to plain text.

Categories are Trove classifiers.

TODO What's the format of the xml file describing the categories? Need more info before I can convert Trove database into what's expected by ZI (or find something existing).

For the meaning of `{converted_version}`, see the *Version conversion* section below.

`{stability}` is `developer` if Python version has a `.dev` segment. Else, if the version contains a prerelease segment (`.a|b|rc`), stability is `testing`. Otherwise, stability is `stable`.

`{langs}` is derived from `Natural Language ::` classifiers.

`{license}` is a Trove classifier. If `License ::` is in classifiers, it is used. If there are multiple, pick one in a deterministic fashion. If none, try to derive it from `release_data['license']`. If none or its value is not understood, try to derive it from a `LICENSE.txt`. If no such file, omit the license attribute.

For `<requires ...>...`, see *dependency conversion* below.

Additional attributes and content of each `<implementation>` depends on the *packagetype* of the corresponding *release_url*.

Version conversion

As Python and ZI versioning schemes differ, conversion is required. Given a Python conversion, we convert it to a normalised Python version (via packaging.version.parse), which gives us:

```
{epoch}!{release}[{prerelease_type}{prerelease_number}][.post{post_number}][.dev{dev_
↪number}]
```

Where:

- `[]` denotes optional part
- `release := N(.N)*`, with `N` an integer
- `prerelease_type := a|b|rc`
- `epoch, prerelease_number, post_number, dev_number` are non-negative numbers

This is converted to the ZI version:

```
{epoch}-{stripped_release}-{modifiers}
```

Where:

- `stripped_release` is `release` with trailing `.0` components trimmed off. This is necessary due to `1 < 1.0` in ZI, while `1 == 1.0` in Python.
- `modifiers` is a list of up to 3 modifiers where prereleases, post and dev segments are considered modifiers. Modifiers are joined by `-`, e.g. `{modifiers[0]}-{modifier[1]}`. A modifier is formatted as:

```
{type}.{number}
```

where:

- `type` is a number derived from this mapping:

```
types = {
  'dev': 0,
  'a': 1,
  'b': 2,
  'rc': 3,
  'post': 5,
}
```

- `number` is one of `prerelease_number`, `post_number`, `dev_number`, depending on the modifier type.

When a version has less than the maximum amount of modifiers, i.e. less than 3, an empty modifier (`-4`) is appended to the list. This ensures correct version ordering.

Some examples of modifier conversion:

```
a10.post20.dev30 -> 1.10-5.20-0.30
b10.dev30 -> 2.10-0.30-4
post20.dev30 -> 5.20-0.30-4
dev30 -> 0.30-4
rc10 -> 3.10-4
```

For examples of the whole conversion, see test_convert_version.

This conversion does not change version ordering.

Dependency conversion

Dependencies are derived from the the distribution (`egg_info`: `requires.txt` and `depends.txt`) as this information is not available through PyPI's metadata (e.g. `release_data['requires']` is missing). `{importance}` is `essential` if the dependency is in `install_requires` and `recommended` otherwise (`extras_require`).

Python packages allow for optional named groups of dependencies called extras. Further, Python dependencies can be conditional (by using environment markers). If a dependency is either conditional or appears in extras_require, it is added as a recommended dependencies in the converted feed, else it is added as a required dependency. Note that Zero Install tries to select all recommended dependencies, but does not fail to select the depending interface when one of its recommended dependencies cannot be selected.

For example:

```
install_requires = ['dep1 ; python_version<2.7', 'dep2==3.*']
extras_require = {
    ':python_version<2.7': ['install_requires_dep'],
    'test:platform_system=="Windows"': ['pywin32'],  # only on windows
    'test': ['somepkg'], # regardless of platform
    'special_feature': ['dep2>=3.3,<4'], # regardless of platform
}
```

is converted to:

```
<implementation ...>
  <requires interface='.../feeds/dep1.xml' importance='recommended' />
  <requires interface='.../feeds/dep2.xml' importance='required' version='
↪{constraints}' />
  <requires interface='.../feeds/install_requires_dep.xml' importance='recommended' />
  <requires interface='.../feeds/pywin32.xml' importance='recommended' />
  <requires interface='.../feeds/somepkg.xml' importance='recommended' />
```

where `{constraints}` are all Python version specifiers converted to a ZI version expression.

Conversion: Packagetype specifics

This documents the parts of the conversion that depend on the *packagetype* of each download (from `release_urls`). These only affect `<implementation>`. There can be multiple download urls for the same version, each can have a different *packagetype*.

Currently, only source distributions are supported.

Generally, a `<manifest-digest>` requires downloading and unpacking the archive. In doing so, the download's md5sum is compared to `release_urls['md5_digest']`.

Python distributions, installation

Generally, a Python distribution (the download from `release_urls`) is an archive/executable which installs:

- Platform independent Python code into a location in `PYTHONPATH`.
- Platform dependent libraries, such as extension modules, into `PYTHONPATH`.
- Python scripts (according to distutils). These are added to `PATH`. Some of these are stored as files in the distribution, others are generated from `entry_points` metadata.

  Upon build (`python setup.py build_scripts`), the stored scripts are copied and their shebang is edited to point to the python interpreter used for the build (this is an absolute path).

  Only upon installation, are scripts generated from `entry_points`.
- Data files as specified by `data_files` in `setup.py`. This does not include `package_data` files, those are placed next to the Python source files. `data_files` can have both absolute and relative destination paths.

  Files with a relative destination path can end up being installed anywhere and the application/library has no way of finding out where these data files have been installed; as such we can safely ignore these files in converting to ZI.

  Files with an absolute destination path will be installed to a predictable location and so the application/library can depend on them. However, making this possible in ZI would require a layered file system to make the file appear installed (e.g. a destination in /etc) without modifying global state. This is not currently supported. I expect there are few popular packages, if any, which use this.

  Bottom line: the conversion drops `data_files`. (`package_data` is still included!)
- C/C++ header files.

pyc files

Normally, when installed, py files are compiled to pyc files. These are specific to the Python version and implementation (e.g. CPython 3.6). Having pyc files in our binary ZI implementation would restrict its reusability to

`os-cpu-python_implementation-python_implementation_version`, i.e. it kills reuse. So, pyc files are not included in implementations.

When Python imports a package, it tries to write a pyc file if missing. This pyc file is written (in a *__pycache__* directory) near the py file. There is no way of writing pyc files to a different location. All these pyc writes result in permission errors as the 0store cache is read-only.

This means we either generate highly platform-specific ZI implementations or have no pyc files. According to #python, the lack of pyc files results in an unnoticeable performance hit on startup time.

The permission errors can be avoided by setting the environment var `PYTHONDONTWRITEBYTECODE=true`.

As such, we disable pyc file generation on installation and set `PYTHONDONTWRITEBYTECODE`.

Source distribution

A source distribution (`release_urls['packagetype'] == 'sdist'`) is a tgz/zip containing at least a `setup.py`. The preferred way to install these is with pip.

After unpacking the distribution, it can be installed without affecting global state like so:

```
pip install \
  --install-option="--install-purelib=/lib" \
  --install-option="--install-platlib=/lib" \
  --install-option="--install-headers=/headers" \
  --install-option="--install-scripts=/scripts" \
  --install-option="--install-data=/data" \
  --root "$PWD/install" \
  --no-deps .
```

`--root` prevents installing outside the install directory; this mainly counters counter `data_files` with absolute paths.

The resulting dir contains:

lib

- Cross platform 'libraries': Python source and pyc files, egg-info directories, package_data files, ...
- Platform specific libraries such as Python extension modules.

scripts Python scripts with a shebang that points by absolute path to the python used by pip. This includes generated scripts.

headers C/C++ headers. Unused.

data Data files from `data_files` with relative destination paths. Unused.

***** Data files from `data_files` with absolute destination paths. Unused.

The source implementation as pseudo-code (extends the `<implementation>` from Conversion: general):

```
<implementation arch='*-src'>
  <archive href='{release_urls['url']}' size='{release_urls['size']}' />
  <command name='compile' ...>
    ...
    <compile:implementation arch='*-*'>
      <archive href='{release_urls['url']}' size='{release_urls['size']}' />
      <environment name='PYTHONPATH' insert='{lib}' />
      <environment name='PATH' insert='{scripts}' />
      <environment name='PYTHONDONTWRITEBYTECODE' value='true' mode='replace' />
```

For now, some requirements are omitted from the compiled implementation (it may be easier to tackle them when real life cases arise where this forms a problem):

- For example, the NumPy package does not work on PyPy. One way to add this constraint is `<restricts interface=PyPy version='0'>` where version 0 does not exist.
- script generation depends on `os.name=posix|java|nt` and `sys.platform.startswith('java')`. It appears it is not possible to express this in ZI currently. Though, instead of expressing it in ZI, we should instead generate our own cross-platform scripts.
- The Python code itself could be platform dependent. This could be derived from classifiers; but these are often omitted and one can doubt the correctness of those that do list it. In this case, it may be better to be too lenient rather than too restrictive.
- extension modules require a certain os-cpu architecture (and perhaps an ABI unless that's standardised by a PEP). When these are present, `os-cpu` should be set

Wheel

Not supported.

Notes:

- `release_urls['packagetype'] == 'bdist_wheel'`
- can derive *arch* from `release_urls['filename']`. See the PyPI XMLRPC interface notes.
- bdist_egg and bdist_wininst can be converted to a wheel
- Wheels cannot be used as binary ZI implementation as scripts need to be generated for `entry_points`.
- `release_urls['python_version']` should be used to restrict which python interpreters and versions may be used; if it's not already mentioned in the wheel name.

Egg

Not supported.

Notes:

- `release_urls['packagetype'] == 'bdist_egg'`
- can derive *arch* from `release_urls['filename']`. See the PyPI XMLRPC interface notes (follow the link to the egg file name convention and search it for "Filename-Embedded Metadata").
- for an example of eggs, see the pymongo project on PyPI
- Eggs cannot be used as binary ZI implementation as scripts need to be generated for `entry_points`.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search