Core metadata specifications

Fields defined in the following specification should be considered valid, complete and not subject to change. The required fields are:

- `Metadata-Version`
- `Name`
- `Version`

All the other fields are optional.

The standard file format for metadata (including in wheels and installed projects) is based on the format of email headers. However, email formats have been revised several times, and exactly which email RFC applies to packaging metadata is not specified. In the absence of a precise definition, the practical standard is set by what the standard library `email.parser` module can parse using the `compat32` policy.

Whenever metadata is serialised to a byte stream (for example, to save to a file), strings must be serialised using the UTF-8 encoding.

Although **PEP 566** defined a way to transform metadata into a JSON-compatible dictionary, this is not yet used as a standard interchange format. The need for tools to work with years worth of existing packages makes it difficult to shift to a new format.

Note: *Interpreting old metadata:* In **PEP 566**, the version specifier field format specification was relaxed to accept the syntax used by popular publishing tools (namely to remove the requirement that version specifiers must be surrounded by parentheses). Metadata consumers may want to use the more relaxed formatting rules even for metadata files that are nominally less than version 2.1.

Contents

Metadata-Version

New in version 1.0.

Version of the file format; legal values are “1.0”, “1.1”, “1.2”, “2.1” and “2.2”.

Automated tools consuming metadata SHOULD warn if `metadata_version` is greater than the highest version they support, and MUST fail if `metadata_version` has a greater major version than the highest version they support (as described in **PEP 440**, the major version is the value before the first dot).

For broader compatibility, build tools MAY choose to produce distribution metadata using the lowest metadata version that includes all of the needed fields.

Example:

```
Metadata-Version: 2.2
```

Name

New in version 1.0.

Changed in version 2.1: Added additional restrictions on format from **PEP 508**

The name of the distribution. The name field is the primary identifier for a distribution. A valid name consists only of ASCII letters and numbers, period, underscore and hyphen. It must start and end with a letter or number. Distribution names are limited to those which match the following regex (run with `re.IGNORECASE`):

```
^([A-Z0-9]|[A-Z0-9][A-Z0-9._-]*[A-Z0-9])$
```

Example:

```
Name: BeagleVote
```

Version

New in version 1.0.

A string containing the distribution’s version number. This field must be in the format specified in **PI** v: latest

Example:

Dynamic (multiple use)

New in version 2.2.

A string containing the name of another core metadata field. The field names `Name` and `Version` may not be specified in this field.

When found in the metadata of a source distribution, the following rules apply:

1. If a field is *not* marked as `Dynamic`, then the value of the field in any wheel built from the sdist MUST match the value in the sdist. If the field is not in the sdist, and not marked as `Dynamic`, then it MUST NOT be present in the wheel.
2. If a field is marked as `Dynamic`, it may contain any valid value in a wheel built from the sdist (including not being present at all).

If the sdist metadata version is older than version 2.2, then all fields should be treated as if they were specified with `Dynamic` (i.e. there are no special restrictions on the metadata of wheels built from the sdist).

In any context other than a source distribution, `Dynamic` is for information only, and indicates that the field value was calculated at wheel build time, and may not be the same as the value in the sdist or in other wheels for the project.

Full details of the semantics of `Dynamic` are described in **PEP 643**.

Platform (multiple use)

New in version 1.0.

A Platform specification describing an operating system supported by the distribution which is not listed in the “Operating System” Trove classifiers. See “Classifier” below.

Examples:

```
Platform: ObscureUnix
Platform: RareDOS
```

Supported-Platform (multiple use)

New in version 1.1.

Binary distributions containing a PKG-INFO file will use the Supported-Platform field in their metadata to specify the OS and CPU for which the binary distribution was compiled. The semantics of the Supported-Platform field are not specified in this PEP.

Example:

```
Supported-Platform: RedHat 7.2
Supported-Platform: i386-win32-2791
```

New in version 1.0.

A one-line summary of what the distribution does.

Example:

```
Summary: A module for collecting votes from beagles.
```

Description

New in version 1.0.

Changed in version 2.1: This field may be specified in the message body instead.

A longer description of the distribution that can run to several paragraphs. Software that deals with metadata should not assume any maximum size for this field, though people shouldn't include their instruction manual as the description.

The contents of this field can be written using reStructuredText markup [1]. For programs that work with the metadata, supporting markup is optional; programs can also display the contents of the field as-is. This means that authors should be conservative in the markup they use.

To support empty lines and lines with indentation with respect to the RFC 822 format, any CRLF character has to be suffixed by 7 spaces followed by a pipe ("|") char. As a result, the Description field is encoded into a folded field that can be interpreted by RFC822 parser [2].

Example:

```
Description: This project provides powerful math functions
        |For example, you can use `sum()` to sum numbers:
        |
        |Example::
        |
        |    >>> sum(1, 2)
        |    3
        |
```

This encoding implies that any occurrences of a CRLF followed by 7 spaces and a pipe char have to be replaced by a single CRLF when the field is unfolded using a RFC822 reader.

Alternatively, the distribution's description may instead be provided in the message body (i.e., after a completely blank line following the headers, with no indentation or other special formatting necessary).

Description-Content-Type

New in version 2.1.

A string stating the markup syntax (if any) used in the distribution's description, so that tools can v: latest - der the description.

code hosting sites render Markdown READMEs, and authors would reuse the file for the description. PyPI didn't recognize the format and so could not render the description correctly. This resulted in many packages on PyPI with poorly-rendered descriptions when Markdown is left as plain text, or worse, was attempted to be rendered as reST. This field allows the distribution author to specify the format of their description, opening up the possibility for PyPI and other tools to be able to render Markdown and other formats.

The format of this field is the same as the `Content-Type` header in HTTP (i.e.: RFC 1341). Briefly, this means that it has a `type/subtype` part and then it can optionally have a number of parameters:

Format:

```
Description-Content-Type: <type>/<subtype>; charset=<charset>[; <param_name>=<param value> ...]
```

The `type/subtype` part has only a few legal values:

- `text/plain`
- `text/x-rst`
- `text/markdown`

The `charset` parameter can be used to specify the character encoding of the description. The only legal value is `UTF-8`. If omitted, it is assumed to be `UTF-8`.

Other parameters might be specific to the chosen subtype. For example, for the `markdown` subtype, there is an optional `variant` parameter that allows specifying the variant of Markdown in use (defaults to `GFM` if not specified). Currently, two variants are recognized:

- `GFM` for **Github-flavored Markdown**
- `CommonMark` for **CommonMark**

Example:

```
Description-Content-Type: text/plain; charset=UTF-8
```

Example:

```
Description-Content-Type: text/x-rst; charset=UTF-8
```

Example:

```
Description-Content-Type: text/markdown; charset=UTF-8; variant=GFM
```

Example:

```
Description-Content-Type: text/markdown
```

If a `Description-Content-Type` is not specified, then applications should attempt to render it as `text/x-rst; charset=UTF-8` and fall back to `text/plain` if it is not valid rst.

If a `Description-Content-Type` is an unrecognized value, then the assumed content type is `text/plain` (Although PyPI will probably reject anything with an unrecognized value).

So for the last example above, the `charset` defaults to `UTF-8` and the `variant` defaults to `GFM` and thus it is equivalent to the example before it.

Keywords

New in version 1.0.

A list of additional keywords, separated by commas, to be used to assist searching for the distribution in a larger catalog.

Example:

```
Keywords: dog,puppy,voting,election
```

Note: The specification previously showed keywords separated by spaces, but distutils and setuptools implemented it with commas. These tools have been very widely used for many years, so it was easier to update the specification to match the de facto standard.

Home-page

New in version 1.0.

A string containing the URL for the distribution's home page.

Example:

```
Home-page: http://www.example.com/~cschultz/bvote/
```

Download-URL

New in version 1.1.

A string containing the URL from which this version of the distribution can be downloaded. (This means that the URL can't be something like "…/BeagleVote-latest.tgz", but instead must be "…/BeagleVote-0.45.tgz".)

Author

New in version 1.0.

A string containing the author's name at a minimum; additional contact information may be provided.

Example:

```
Author: C. Schultz, Universal Features Syndicate,
        Los Angeles, CA <cschultz@peanuts.example.com>
```


Author-email

A string containing the author's e-mail address. It can contain a name and e-mail address in the legal forms for a RFC-822 `From:` header.

Example:

```
Author-email: "C. Schultz" <cschultz@example.com>
```

Per RFC-822, this field may contain multiple comma-separated e-mail addresses:

```
Author-email: cschultz@example.com, snoopy@peanuts.com
```

Maintainer

New in version 1.2.

A string containing the maintainer's name at a minimum; additional contact information may be provided.

Note that this field is intended for use when a project is being maintained by someone other than the original author: it should be omitted if it is identical to `Author`.

Example:

```
Maintainer: C. Schultz, Universal Features Syndicate,
        Los Angeles, CA <cschultz@peanuts.example.com>
```

Maintainer-email

New in version 1.2.

A string containing the maintainer's e-mail address. It can contain a name and e-mail address in the legal forms for a RFC-822 `From:` header.

Note that this field is intended for use when a project is being maintained by someone other than the original author: it should be omitted if it is identical to `Author-email`.

Example:

```
Maintainer-email: "C. Schultz" <cschultz@example.com>
```

Per RFC-822, this field may contain multiple comma-separated e-mail addresses:

```
Maintainer-email: cschultz@example.com, snoopy@peanuts.com
```

License

New in version 1.0.

Text indicating the license covering the distribution where the license is not a selection from the "License" Trove classifiers. See "Classifier" below. This field may also be used to specify a particular version of a license which is

Examples:

```
License: This software may only be obtained by sending the
        author a postcard, and then the user promises not
        to redistribute it.

License: GPL version 3, excluding DRM provisions
```

Classifier (multiple use)

New in version 1.1.

Each entry is a string giving a single classification value for the distribution. Classifiers are described in **PEP 301**, and the Python Package Index publishes a dynamic list of currently defined classifiers.

This field may be followed by an environment marker after a semicolon.

Examples:

```
Classifier: Development Status :: 4 - Beta
Classifier: Environment :: Console (Text Based)
```

Requires-Dist (multiple use)

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string naming some other distutils project required by this distribution.

The format of a requirement string contains from one to four parts:

- A project name, in the same format as the `Name:` field. The only mandatory part.
- A comma-separated list of 'extra' names. These are defined by the required project, referring to specific features which may need extra dependencies.
- A version specifier. Tools parsing the format should accept optional parentheses around this, but tools generating it should not use parentheses.
- An environment marker after a semicolon. This means that the requirement is only needed in the specified conditions.

See **PEP 508** for full details of the allowed format.

The project names should correspond to names as found on the Python Package Index.

Version specifiers must follow the rules described in Version specifiers.

Examples:


```
Requires-Dist: pkginfo
Requires-Dist: PasteDeploy
```

Requires-Python

New in version 1.2.

This field specifies the Python version(s) that the distribution is guaranteed to be compatible with. Installation tools may look at this when picking which version of a project to install.

The value must be in the format specified in Version specifiers.

This field cannot be followed by an environment marker.

Examples:

```
Requires-Python: >=3
Requires-Python: >2.6,!=3.0.*,!=3.1.*
Requires-Python: ~=2.6
```

Requires-External (multiple use)

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string describing some dependency in the system that the distribution is to be used. This field is intended to serve as a hint to downstream project maintainers, and has no semantics which are meaningful to the `distutils` distribution.

The format of a requirement string is a name of an external dependency, optionally followed by a version declaration within parentheses.

This field may be followed by an environment marker after a semicolon.

Because they refer to non-Python software releases, version numbers for this field are **not** required to conform to the format specified in **PEP 440**: they should correspond to the version scheme used by the external dependency.

Notice that there is no particular rule on the strings to be used.

Examples:

```
Requires-External: C
Requires-External: libpng (>=1.5)
Requires-External: make; sys_platform != "win32"
```

Project-URL (multiple-use)

New in version 1.2.

A string containing a browsable URL for the project and a label for it, separated by a comma.

```
Project-URL: Bug Tracker, http://bitbucket.org/tarek/distribute/issues/
```

The label is free text limited to 32 characters.

Provides-Extra (multiple use)

New in version 2.1.

A string containing the name of an optional feature. Must be a valid Python identifier. May be used to make a dependency conditional on whether the optional feature has been requested.

Example:

```
Provides-Extra: pdf
Requires-Dist: reportlab; extra == 'pdf'
```

A second distribution requires an optional dependency by placing it inside square brackets, and can request multiple features by separating them with a comma (,). The requirements are evaluated for each requested feature and added to the set of requirements for the distribution.

Example:

```
Requires-Dist: beaglevote[pdf]
Requires-Dist: libexample[test, doc]
```

Two feature names `test` and `doc` are reserved to mark dependencies that are needed for running automated tests and generating documentation, respectively.

It is legal to specify `Provides-Extra:` without referencing it in any `Requires-Dist:`.

Rarely Used Fields

The fields in this section are currently rarely used, as their design was inspired by comparable mechanisms in Linux package management systems, and it isn't at all clear how tools should interpret them in the context of an open index server such as PyPI.

As a result, popular installation tools ignore them completely, which in turn means there is little incentive for package publishers to set them appropriately. However, they're retained in the metadata specification, as they're still potentially useful for informational purposes, and can also be used for their originally intended purpose in combination with a curated package repository.

Provides-Dist (multiple use)

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string naming a Distutils project which is contained within this distribution. This field *must* include the project identified in the `Name` field, followed by the version : Name (Version).

now available as a separate distribution. Installing such a source distribution satisfies requirements for both `ZODB` and `transaction`.

A distribution may also provide a “virtual” project name, which does not correspond to any separately-distributed project: such a name might be used to indicate an abstract capability which could be supplied by one of multiple projects. E.g., multiple projects might supply RDBMS bindings for use by a given ORM: each project might declare that it provides `ORM-bindings`, allowing other projects to depend only on having at most one of them installed.

A version declaration may be supplied and must follow the rules described in Version specifiers. The distribution’s version number will be implied if none is specified.

This field may be followed by an environment marker after a semicolon.

Examples:

```
Provides-Dist: OtherProject
Provides-Dist: AnotherProject (3.4)
Provides-Dist: virtual_package; python_version >= "3.4"
```

Obsoletes-Dist (multiple use)

New in version 1.2.

Changed in version 2.1: The field format specification was relaxed to accept the syntax used by popular publishing tools.

Each entry contains a string describing a distutils project’s distribution which this distribution renders obsolete, meaning that the two projects should not be installed at the same time.

Version declarations can be supplied. Version numbers must be in the format specified in Version specifiers.

This field may be followed by an environment marker after a semicolon.

The most common use of this field will be in case a project name changes, e.g. Gorgon 2.3 gets subsumed into Torqued Python 1.0. When you install Torqued Python, the Gorgon distribution should be removed.

Examples:

```
Obsoletes-Dist: Gorgon
Obsoletes-Dist: OtherProject (<3.0)
Obsoletes-Dist: Foo; os_name == "posix"
```

[1] reStructuredText markup: https://docutils.sourceforge.io/

[2] RFC 822 Long Header Fields: **RFC 822#section-3.1.1**