Annex F

Package URL specification v1 (Normative)

25 Introduction

The Package URL core specification defines a versioned and formalized format, syntax, and rules used to represent and validate package URLs.

A package URL or *purl* is an attempt to standardize existing approaches to reliably identify the location of software packages.

A *purl* is a URL string used to identify the location of a software package in a mostly universal and uniform way across programming languages, package managers, packaging conventions, tools, APIs and databases.

Such a package URL is useful to reliably reference the same software package using a simple and expressive syntax and conventions based on familiar URLs.

26 Syntax definition

purl stands for **package URL**.

A *purl* is a URL composed of seven components:

```
scheme:type/namespace/name@version?qualifiers#subpath
```

Components are separated by a specific character for unambiguous parsing.

The definition for each components is:

- **scheme**: this is the URL scheme with the constant value of "`pkg`". One of the primary reason for this single scheme is to facilitate the future official registration of the "`pkg`" scheme for package URLs. Required.
- **type**: the package type or package protocol such as maven, npm, nuget, gem, pypi, etc. Required.
- **namespace**: some name prefix such as a Maven groupid, a Docker image owner, a GitHub user or organization. Optional and type-specific.
- **name**: the name of the package. Required.
- **version**: the version of the package. Optional.
- **qualifiers**: extra qualifying data for a package such as an OS, architecture, a distribution, etc. Optional and type-specific.
- **subpath**: extra subpath within a package, relative to the package root. Optional.

Components are designed such that they form a hierarchy from the most significant on the left to the least significant components on the right.

A *purl* is a valid URL and URI that conforms to the URL definitions and specifications in RFC 3986 https://datatracker.ietf.org/doc/html/rfc3986.

A *purl* must not contain a URL Authority i.e. there is no support for username, password, host and port components. A `namespace` segment may sometimes look like a host but its interpretation is specific to a type.

The *purl* components are mapped to the following URL components:

- *purl* scheme: this is a URL scheme with a constant value: `pkg`
- *purl* type, namespace, name and version components: these are collectively mapped to a URL path
- *purl* qualifiers: this maps to a URL query
- *purl* subpath: this is a URL fragment

27 Character encoding

For clarity and simplicity a *purl* is always an ASCII string. To ensure that there is no ambiguity when parsing a *purl*, separator characters and non-ASCII characters must be encoded in UTF-8, and then percent-encoded as defined in RFC 3986 https://datatracker.ietf.org/doc/html/rfc3986.

Use these rules for percent-encoding and decoding *purl* components:

- the type must NOT be encoded and must NOT contain separators
- the `#`, `?`, `@` and `:` characters must NOT be encoded when used as separators. They may need to be encoded elsewhere
- the `:` scheme and type separator does not need to and must NOT be encoded. It is unambiguous unencoded everywhere
- the `/` used as type/namespace/name and subpath segments separator does not need to and must NOT be percent-encoded. It is unambiguous unencoded everywhere
- the `@` version separator must be encoded as `%40` elsewhere
- the `?` qualifiers separator must be encoded as `%3F` elsewhere
- the `=` qualifiers key/value separator must NOT be encoded
- the `#` subpath separator must be encoded as `%23` elsewhere
- All non-ASCII characters must be encoded as UTF-8 and then percent-encoded

It is OK to percent-encode any *purl* components, except for the type. Producers and consumers of *purl* data must always percent-decode and percent-encode components and component segments as explained in the "How to produce and consume *purl* data" section.

28 Rules for each component

A *purl* string is an ASCII URL string composed of seven components.

Some components are allowed to use other characters beyond ASCII: these components must then be UTF-8-encoded strings and percent-encoded as defined in the "Character encoding" section.

The rules for each component are:

28.1 Rules for scheme

- The scheme is a constant with the value "`pkg`"
- Since a *purl* never contains a URL Authority, its scheme must not be suffixed with double slash as in `pkg://` and should use instead `pkg:`.
- *purl* parsers must accept URLs such as 'pkg://' and must ignore the '//'.
- *purl* builders must not create invalid URLs with such double slash '//'.
- The scheme is followed by a ':' separator.
- For example, the two purls `pkg:gem/ruby-advisory-db-check@0.12.4` and `pkg://gem/ruby-advisory-db-chec` are strictly equivalent. The first is in canonical form while the second is an acceptable *purl* but is an invalid URI/URL per RFC3986.

28.2 Rules for type

- The package type is composed only of ASCII letters and numbers, `.`, + and − (period, plus, and dash).
- The type cannot start with a number.
- The type cannot contain spaces.
- The type must not be percent-encoded.
- The type is case insensitive, with the canonical form being lowercase.

28.3 Rules for namespace

- The optional namespace contains zero or more segments, separated by slash /.
- Leading and trailing slashes / are not significant and should be stripped in the canonical form. They are not part of the namespace.
- Each namespace segment must be a percent-encoded string.
- When percent-decoded, a segment must not contain a slash / and must not be empty.
- A URL host or Authority must NOT be used as a namespace. Use instead a `repository_url` qualifier. Note however that for some types, the namespace may look like a host.

28.4 Rules for name

- The name is prefixed by a slash / separator when the namespace is not empty.
- This slash / is not part of the name.
- A name must be a percent-encoded string.

28.5 Rules for version

- The version is prefixed by a at-sign `@` separator when not empty.
- This at-sign `@` is not part of the version.
- A version must be a percent-encoded string.
- A version is a plain and opaque string. Some package types use versioning conventions such as semver for NPMs or nevra conventions for RPMS. A type may define a procedure to compare and sort versions, but there is no reliable and uniform way to do such comparison consistently.

28.6 Rules for qualifiers

- The qualifiers string is prefixed by a `?` separator when not empty.
- This `?` is not part of the qualifiers.
- This is a string composed of zero or more key=value pairs each separated by an ampersand &. A key and value are separated by an equal = character.
- These & are not part of the key=value pairs.
- Each key must be unique within the keys of the qualifiers string.
- A value cannot be an empty string; a key=value pair with an empty value is the same as no key/value at all for this key.
- Each key must be composed only of ASCII letters and numbers, `.`, − and `\_` (period, dash and underscore).
- A key cannot start with a number.
- A key must NOT be percent-encoded.
- A key is case insensitive, with the canonical form being lowercase.
- A key cannot contain spaces.
- A value must be a percent-encoded string.
- The = separator is neither part of the key nor of the value.

28.7 Rules for subpath

- The subpath string is prefixed by a # separator when not empty.
- This # is not part of the subpath.
- The subpath contains zero or more segments, separated by slash /.

- Leading and trailing slashes / are not significant and should be stripped in the canonical form.
- Each subpath segment must be a percent-encoded string.
- When percent-decoded, a segment must not contain a /, must not be any of `..` or `.`, and must not be empty.
- The subpath must be interpreted as relative to the root of the package.

29 Known types

There are several known *purl* package type definitions. The current list of known types is: `alpm`, `apk`, `bitbucket`, `bitnami`, `cargo`, `cocoapods`, `composer`, `conan`, `conda`, `cpan`, `cran`, `deb`, `docker`, `gem`, `generic`, `github`, `golang`, `hackage`, `hex`, `huggingface`, `luarocks`, `maven`, `mlflow`, `npm`, `nuget`, `oci`, `pub`, `pypi`, `qpkg`, `rpm`, `swid`, and `swift`.

The list, with definitions for each type, is maintained in the file named `PURL-TYPES.rst` in the online repository https://github.com/package-url/purl-spec.

30 Known qualifiers key/value pairs

Qualifiers should be limited to the bare minimum for proper package identification, to ensure that a *purl* stays compact and readable in most cases. Separate external attributes stored outside of a *purl* are the preferred mechanism to convey extra long and optional information. API, database or web form.

The following keys are valid for use in all package types:

- `repository_url` is an extra URL for an alternative, non-default package repository or registry. The default repository or registry of each type is documented in the "Known types" section.
- `download_url` is an extra URL for a direct package web download URL.
- `vcs_url` is an extra URL for a package version control system URL.
- `file_name` is an extra file name of a package archive.
- `checksum` is a qualifier for one or more checksums stored as a comma-separated list. Each item in the list is in form of algorithm:hex_value (all lowercase), such as `sha1:ad9503c3e994a4f611a4892f2e67ac82df727086`.

31 How to produce and consume *purl* data

The following provides rules to be followed when building or deconstructing *purl* instances.

31.1 How to build *purl* string from its components

Building a *purl* ASCII string works from left to right, from type to subpath.

To build a *purl* string from its components:

1. Start a *purl* string with the "`pkg:`" scheme as a lowercase ASCII string

2. Append the type string to the *purl* as a lowercase ASCII string

3. Append / to the *purl*

4. If the namespace is not empty:

   1. Strip the namespace from leading and trailing /
   2. Split on / as segments
   3. Apply type-specific normalization to each segment, if needed
   4. Encode each segment in UTF-8-encoding
   5. Percent-encode each segment
   6. Join the segments with /
   7. Append this to the *purl*

   8. Append / to the *purl*

5. Strip the name from leading and trailing /
6. Apply type-specific normalization to the name, if needed
7. Encode the name in UTF-8-encoding
8. Percent-encode the name
9. Append the percent-encoded name to the *purl*
10. If the version is not empty:
    1. Append `@` to the *purl*
    2. Encode the version in UTF-8-encoding
    3. Percent-encode the version
    4. Append the percent-encoded version to the *purl*
11. If the qualifiers are not empty and not composed only of key/value pairs where the value is empty:
    1. Append `?` to the *purl*
    2. Discard any pair where the value is empty
    3. Encode each value in UTF-8-encoding
    4. If the key is `checksum` and there are more than one checksums, join the list with `,` to create the qualifier value
    5. Create each qualifier string by joining the lowercased key, the equal = sign, and the percent-encoded value
    6. Sort this list of qualifier strings lexicographically
    7. Join this list of sorted qualifier strings with `&`
    8. Append this string to the *purl*
12. If the subpath is not empty and not composed only of empty, `.`, and `..` segments:
    1. Append # to the *purl*
    2. Strip the subpath from leading and trailing /
    3. Split the subpath on / as a list of segments
    4. Discard empty, `.`, and `..` segments
    5. Encode each segment in UTF-8-encoding
    6. Percent-encode each segment
    7. Join the segments with /
    8. Append this string to the *purl*

31.2 How to parse a *purl* string to its components

Parsing a *purl* ASCII string into its components works by splitting the string on different characters.

To parse a *purl* string in its components:

1. Split the *purl* string once from right on #, if present; the left side is the remainder.
2. If the right side is not empty, it contains subpath information:
    1. Strip it from leading and trailing /.
    2. Split this on / in a list of segments.
    3. Discard empty, `.`, and `..` segments.
    4. Percent-decode each segment.

   5. UTF-8-decode each of these.
   6. Join segments with `/`.
   7. This is the subpath.

3. Split the remainder once from right on `?`, if present; the left side is the remainder.

4. If the right side is not empty, it contains qualifiers information:

   1. Split it on `&` in a list of key=value pairs.
   2. Split each pair once from left on = in key and value parts.
   3. The key is the lowercase left side.
   4. Percent-decode the right side.
   5. UTF-8-decode this to get the value.
   6. Discard any key/value pairs where the value is empty.
   7. If the key is `checksum`, split the value on `,` to create a list of checksums.
   8. This list of keys/values is the qualifiers.

5. Split the remainder once from left on `:`; the right side is the remainder.

6. The left side lowercased is the scheme. It should be exactly “`pkg:`”.

7. Strip the remainder from leading and trailing `/`.

8. Split this once from left on `/`; the right side is the remainder.

9. The left side lowercased is the type.

10. Split the remainder once from right on `@`, if present; the left side is the remainder.

11. If the right side is not empty, it contains version information:

    1. Percent-decode the string.
    2. UTF-8-decode this.
    3. This is the version.

12. Split the remainder once from right on `/`, if present; the left side is the remainder.

13. The right side contains name information.

14. Percent-decode the name string.

15. UTF-8-decode this.

16. Apply type-specific normalization, if needed.

17. This is the name.

18. If the remainder is not empty, it contains namespace information:

    1. Split the remainder on `/` to a list of segments.
    2. Discard any empty segment.
    3. Percent-decode each segment.
    4. UTF-8-decode each of these.
    5. Apply type-specific normalization to each segment, if needed.
    6. Join segments with `/`.
    7. This is the namespace.

32 Examples

The following list includes some valid *purl* examples:

- `pkg:bitbucket/birkenfeld/pygments-main@244fd47e07d1014f0aed9c`
- `pkg:deb/debian/curl@7.50.3-1?arch=i386&distro=jessie`
- `pkg:gem/ruby-advisory-db-check@0.12.4`
- `pkg:github/package-url/purl-spec@244fd47e07d1004f0aed9c`
- `pkg:golang/google.golang.org/genproto#googleapis/api/annotations`
- `pkg:maven/org.apache.xmlgraphics/batik-anim@1.9.1?packaging=sources`
- `pkg:npm/foobar@12.3.1`
- `pkg:nuget/EnterpriseLibrary.Common@6.0.1304`
- `pkg:pypi/django@1.11.1`
- `pkg:rpm/fedora/curl@7.50.3-1.fc25?arch=i386&distro=fedora-25`

33 Original license

This specification is based on the texts published in the https://github.com/package-url/purl-spec online repository. The original license and attribution are reproduced below: