Publiccode.yml Standard

italia

Feb 13, 2020

Contents

`publiccode.yml` is a metadata standard for repositories containing software developed or acquired by the Public Administration, aimed at making them easily discoverabile and thus reusable by other entities.

By including a `publiccode.yml` file in the root of a repository, and populating it with information about the software, technicians and civil servants can evaluate it. Automatic indexing tools can also be built, since the format is easily readable by both humans and machines.

`publiccode.yml` is mandatory for all public software developed in Italy, according to the national guidelines[1]: this enables the Developers Italia crawler to build the national software catalog[2]. The standard is designed to be interoperable internationally, thus the country-specific keys are separated by the core part and are defined in specific sections that each government can rule.

Details carried by a `publiccode.yml` file include:

- title and description of the project or product (in one or more languages);
- development state (e.g., `concept`, `development`, `beta`, `stable`, `obsolete`;
- contacts of the entity who published the codebase;
- contacts of the maintainer, if any, including the expire date of the maintenance contract;
- information about the legal context for which the project or product was designed;
- dependencies.

and much more.

[1] https://docs.italia.it/AgID/linee-guida-riuso-software/lg-acquisizione-e-riuso-software-per-pa-docs/
[2] https://developers.italia.it/

CHAPTER 1

See also

- More information about software reuse[3]
- publiccode.yml web editor[4]

[3] https://developers.italia.it/en/reuse
[4] https://publiccode-editor.developers.italia.it/

CHAPTER 2

Table of contents

2.1 The Standard (core)

This document represents the description of the `publiccode.yml` schema.

2.1.1 Top-Level Keys and Sections

Key `publiccodeYmlVersion`

- Type: string
- Presence: mandatory
- Example: `"0.1"`

This key specifies the version to which the current `publiccode.yml` adheres to, for forward compatibility.

Key `name`

- Type: string
- Presence: mandatory
- Example: `"Medusa"`

This key contains the name of the software. It contains the (short) public name of the product, which can be localised in the specific `localisation` section. It should be the name most people usually refer to the software. In case the software has both an internal "code" name and a commercial name, use the commercial name.

Key `applicationSuite`

- Type: string

- Presence: optional
- Example: `"MegaProductivitySuite"`

This key contains the name of the "suite" to which the software belongs.

Key `url`

- Type: string (URL)
- Presence: mandatory
- Example: `"https://example.com/italia/medusa.git"`

A unique identifier for this software. This string must be a URL to the source code repository (git, svn, ...) in which the software is published. If the repository is available under multiple protocols, prefer HTTP/HTTPS URLs which don't require user authentication.

Forks created for the purpose of contributing upstream should not modify this file; this helps software parsing `publiccode.yml` to immediately skip technical forks. On the contrary, a complete fork that is meant to be maintained separately from the original software should modify this line, to give themselves the status of a different project.

See *Forks and variants* (page 20) for a complete description of what is a software variant and how to handle forked software as a parser or an author.

Key `landingURL`

- Type: string (URL)
- Presence: optional
- Example: `"https://example.com/italia/medusa"`

If the `url` parameter does not serve a human readable or browsable page, but only serves source code to a source control client, with this key you have an option to specify a landing page. This page, ideally, is where your users will land when they will click a button labeled something like "Go to the application source code". In case the product provides an automated graphical installer, this URL can point to a page which contains a reference to the source code but also offers the download of such an installer.

Key `isBasedOn`

- Type: string or array of strings
- Presence: optional
- Example: `"https://github.com/italia/otello.git"`

In case this software is a variant or a fork of another software, which might or might not contain a `publiccode.yml` file, this key will contain the `url` of the original project(s).

The existence of this key identifies the fork as a software variant, descending from the specified repositories.

Key `softwareVersion`

- Type: string
- Presence: optional
- Example: `"1.0"`, `"dev"`

This key contains the latest stable version number of the software. The version number is a string that is not meant to be interpreted and parsed but just displayed; parsers should not assume semantic versioning or any other specific version format.

The key can be omitted if the software is currently in initial development and has never been released yet.

Key `releaseDate`

- Type: string (date)
- Presence: mandatory
- Example: `"2017-04-15"`

This key contains the date at which the latest version was released. This date is mandatory if the software has been released at least once and thus the version number is present.

Key `logo`

- Type: string (relative path to file or absolute URL)
- Presence: optional
- Acceptable formats: SVG, SVGZ, PNG
- Example: `"img/logo.svg"`

This key contains the path to the logo of the software. Logos should be in vector format; raster formats are only allowed as a fallback. In this case, they should be transparent PNGs, minimum 1000px of width. The key value can be the relative path to the file starting from the root of the repository, or it can be an absolute URL pointing to the logo in raw version. In both cases, the file must reside inside the same repository where the `publiccode.yml` file is stored.

Key `monochromeLogo`

- Type: string (path to file)
- Presence: optional
- Acceptable formats: SVG, SVGZ, PNG
- Example: `"img/logo-mono.svg"`

A monochromatic (black) logo. The logo should be in vector format; raster formats are only allowed as a fallback. In this case, they should be transparent PNGs, minimum 1000px of width. The key value can be the relative path to the file starting from the root of the repository, or it can be an absolute URL pointing to the logo in raw version. In both cases, the file must reside inside the same repository where the `publiccode.yml` file is stored.

Key `inputTypes`

- Type: array of enumerated strings
- Presence: optional
- Values: as per RFC 6838
- Example: `"text/plain"`

A list of Media Types (MIME Types) as mandated in RFC 6838[5] which the application can handle as input.

In case the software does not support any input, you can skip this field or use `application/x.empty`.

Key `outputTypes`

- Type: array of enumerated strings
- Presence: optional
- Values: as per RFC 6838
- Example: `"text/plain"`

A list of Media Types (MIME Types) as mandated in RFC 6838[6] which the application can handle as output.

In case the software does not support any output, you can skip this field or use `application/x.empty`.

Key `platforms`

- Type: enumerated string or array of strings
- Presence: mandatory
- Values: `web`, `windows`, `mac`, `linux`, `ios`, `android`. Human readable values outside this list are allowed.
- Example: `web`

This key specifies which platform the software runs on. It is meant to describe the platforms that users will use to access and operate the software, rather than the platform the software itself runs on.

Use the predefined values if possible. If the software runs on a platform for which a predefined value is not available, a different value can be used.

Key `categories`

- Type: array of strings
- Presence: mandatory
- Acceptable values: see *List of software categories* (page 22)

A list of words that can be used to describe the software and can help building catalogs of open software.

The controlled vocabulary *List of software categories* (page 22) contains the list of allowed values.

Key `usedBy`

- Type: array of strings
- Presence: optional

A list of the names of prominent public administrations (that will serve as "testimonials") that are currently known to the software maintainer to be using this software.

Parsers are encouraged to enhance this list also with other information that can obtain independently; for instance, a fork of a software, owned by an administration, could be used as a signal of usage of the software.

[5] https://tools.ietf.org/html/rfc6838
[6] https://tools.ietf.org/html/rfc6838

Key `roadmap`

- Type: string
- Presence: optional

A link to a public roadmap of the software.

Key `developmentStatus`

- Type: enumerated string
- Presence: mandatory
- Allowed values: `concept`, `development`, `beta`, `stable`, `obsolete`

The keys are:

- `concept` - The software is just a "concept". No actual code may have been produced, and the repository could simply be a placeholder.
- `development` - Some effort has gone into the development of the software, but the code is not ready for the end user, even in a preliminary version (beta or alpha) to be tested by end users.
- `beta` - The software is in the testing phase (alpha or beta). At this stage, the software might or might not have had a preliminary public release.
- `stable` - The software has seen a first public release and is ready to be used in a production environment.
- `obsolete` - The software is no longer maintained or kept up to date. All of the source code is archived and kept for historical reasons.

Key `softwareType`

- Type: enumerated string
- Presence: mandatory
- Allowed values: `"standalone/mobile"`, `"standalone/iot"`, `"standalone/desktop"`, `"standalone/web"`, `"standalone/backend"`, `"standalone/other"`, `"addon"`, `"library"`, `"configurationFiles"`

The keys are:

- `standalone/mobile` - The software is a standalone, self-contained The software is a native mobile app.
- `standalone/iot` - The software is suitable for an IoT context.
- `standalone/desktop` - The software is typically installed and run in a a desktop operating system environment.
- `standalone/web` - The software represents a web application usable by means of a browser.
- `standalone/backend` - The software is a backend application.
- `standalone/other` - The software has a different nature from the once listed above.
- `softwareAddon` - The software is an addon, such as a plugin or a theme, for a more complex software (e.g. a CMS or an office suite).
- `library` - The software contains a library or an SDK to make it easier to third party developers to create new products.

- `configurationFiles` - The software does not contain executable script but a set of configuration files. They may document how to obtain a certain deployment. They could be in the form of plain configuration files, bash scripts, ansible playbooks, Dockerfiles, or other instruction sets.

Section `intendedAudience`

Key `intendedAudience/countries`

- Type: array of strings
- Presence: optional

This key explicitly includes certain countries in the intended audience, i.e. the software explicitly claims compliance with specific processes, technologies or laws. All countries are specified using lowercase ISO 3166-1 alpha-2 two-letter country codes.

Key `intendedAudience/unsupportedCountries`

- Type: array of strings
- Presence: optional

This key explicitly marks countries as NOT supported. This might be the case if there is a conflict between how software is working and a specific law, process or technology. All countries are specified using lowercase ISO 3166-1 alpha-2 two-letter country codes.

Key `intendedAudience/scope`

- Type: array of strings
- Presence: optional
- Acceptable values: see *Scope list* (page 25)

This key contains a list of tags related to the field of application of the software.

Section `description`

This section contains a general description of the software. Parsers can use this section for instance to create a web page describing the software.

Note: since all the strings contained in this section are user-visible and written in a specific language, you **must** specify the language you are editing the text in (using the IETF BCP 47[7] specifications) by creating a sub-section with that name. The primary language subtag cannot be omitted, as mandated by the BCP 47.

An example for English:

```
description:
  en:
    shortDescription: ...
    longDescription: ...
```

[7] https://tools.ietf.org/html/bcp47

In the following part of the document, all keys are assumed to be in a sub-section with the name of the language (we will note this with `[lang]`).

Note: it is mandatory to have *at least* one language in this section. All other languages are optional.

Key `description/[lang]/localisedName`

- Type: string
- Presence: optional
- Example: `"Medusa"`

This key is an opportunity to localise the name in a specific language. It contains the (short) public name of the product. It should be the name most people usually refer to the software. In case the software has both an internal "code" name and a commercial name, use the commercial name.

Key `description/[lang]/genericName`

- Type: string (max 35 chars)
- Presence: mandatory
- Example: `"Text Editor"`

This key is the "Generic name", which refers to the specific category to which the software belongs. You can usually find the generic name in the presentation of the software, when you write: "Software xxx is a yyy". Notable examples include "Text Editor", "Word Processor", "Web Browser", "Chat" and so on… The generic name can be up to 35 characters long.

Key `description/[lang]/shortDescription`

- Type: string (max 150 chars)
- Presence: mandatory
- Example: `"Advanced booking system for hospitals"`

This key contains a short description of the software. It should be a single line containing a single sentence. Maximum 150 characters are allowed.

Key `description/[lang]/longDescription`

- Type: string (min 500 chars, max 10000 chars)
- Presence: mandatory (for at least one language)

This key contains a longer description of the software, between 500 and 10000 chars. It is meant to provide an overview of the capabilities of the software for a potential user. The audience for this text should be that of users of the software, not developers. You can think of this text as the description of the software that would be in its website (if the software had one).

This description can contain some basic markdown: `*italic*`, `**bold**`, bullet points and `[links](#)`.

Key `description/[lang]/documentation`

- Type: URL
- Presence: optional

This key contains a reference to the user-level (not developer-level) documentation of the software. The value must be a URL to a hosted version of the documentation.

It is suggested that the URL points to a hosted version of the documentation that is immediately readable through a common web browser in both desktop and mobile format. The documentation should be rendered in HTML and browsable like a website (with a navigation index, a search bar, etc.).

If the documentation is instead available only as a document, put a direct view/download link as URL in this key. You should commit the document as part of the source code repository, and then link to it using the code hosting source browser URL (e.g.: GitHub URL to the file). Prefer using open formats like PDF or ODT for maximum interoperability.

Whichever the format for the documentation, remember to make its source files available under an open license, possibly by committing them as part of the repository itself.

Key `description/[lang]/apiDocumentation`

- Type: URL
- Presence: optional

This key contains a reference to the API documentation of the software. The value must be a URL to a hosted version of the documentation.

It is suggested that the URL points to a hosted version of the documentation that is immediately readable through a common web browser. The documentation should be rendered in HTML and browsable like a website (with a navigation index, a search bar, etc.), and if there is a reference or test deployment, possibly offer an interactive interface (e.g. Swagger).

If the documentation is instead available only as a document, put a direct view/download link as URL in this key. You should commit the document as part of the source code repository, and then link to it using the code hosting source browser URL (e.g.: GitHub URL to the file). Prefer using open formats like PDF or ODT for maximum interoperability.

Whichever the format for the documentation, remember to make its source files available under an open license, possibly by committing them as part of the repository itself.

Key `description/[lang]/features`

- Type: array of strings
- Presence: mandatory (for at least one language)

This key contains a list of software features, describing what capabilities the software allows to do. The audience for this text should be that of public decision makers who will be commissioning the software. The features should thus not target developers; instead of listing technical features referring to implementation details, prefer listing user-visible functionalities of the software.

While the key is mandatory, there is no mandatory minimum or maximum number of features that should be listed in this key. Each feature must use a maximum of 100 characters.

The suggested number of features to list is between 5 and 20, depending on the software size and complexity. There is no need for exhaustiveness, as users can always read the documentation for additional information.

Key `description/[lang]/screenshots`

- Type: array of strings (paths)
- Presence: optional
- Formats: PNG, JPG
- Example: `"data/screenshots/configuration.png"`

This key contains one or multiple paths to files showing screenshots of the software. They are meant to give a quick idea on how the software looks like and how it works. The key value can be the relative path to the file starting from the root of the repository, or it can be an absolute URL pointing to the screenshot in raw version. In both cases, the file must reside inside the same repository where the `publiccode.yml` file is stored.

Screenshots can be of any shape and size; the suggested formats are:

- Desktop: 1280x800 @1x
- Tablet: 1024x768 @2x
- Mobile: 375x667 @2x

Key `description/[lang]/videos`

- Type: array of strings (URLs)
- Presence: optional
- Example: `"https://youtube.com/xxxxxxxx"`

This key contains one or multiple URLs of videos showing how the software works. Like screenshots, videos should be used to give a quick overview on how the software looks like and how it works. Videos must be hosted on a video sharing website that supports the oEmbed[8] standard; popular options are YouTube and Vimeo.

Since videos are an integral part of the documentation, it is recommended to publish them with an open license.

Key `description/[lang]/awards`

- Type: array of strings
- Presence: optional

A list of awards won by the software.

Section `legal`

Key `legal/license`

- Type: string
- Presence: mandatory
- Example: `"AGPL-3.0-or-later"`

[8] https://oembed.com

This string describes the license under which the software is distributed. The string must contain a valid SPDX expression, referring to one (or multiple) open-source license. Please refer to the SPDX documentation[9] for further information.

Key `legal/mainCopyrightOwner`

- Type: string
- Presence: optional
- Example: `"City of Amsterdam"`

This string describes the entity that owns the copyright on "most" of the code in the repository. Normally, this is the line that is reported with the copyright symbol at the top of most files in the repo.

It is possible to list multiple owners if required so, using an English sentence. It is also possible to informally refer to a community of group of people like "Linus Torvalds and all Linux contributors".

In case it is not possible to name a main copyright owner, it is possible to omit this key; in those cases, if the repo has a authors file, you can point to it through `legal/authorsFile`.

Key `legal/repoOwner`

- Type: string
- Presence: optional
- Example: `"City of Amsterdam"`

This string describes the entity that owns this repository; this might or might not be the same entity who owns the copyright on the code itself. For instance, in case of a fork of the original software, the `repoOwner` is probably different from the `mainCopyrightOwner`.

Key `legal/authorsFile`

- Type: string (path to file)
- Presence: optional
- Example: `"doc/AUTHORS.txt"`

Some open-source software adopt a convention of identify the copyright holders through a file that lists all the entities that own the copyright. This is common in projects strongly backed by a community where there are many external contributors and no clear single/main copyright owner. In such cases, this key can be used to refer to the authors file, using a path relative to the root of the repository.

Section `maintenance`

This section provides information on the maintenance status of the software, useful to evaluate whether the software is actively developed or not.

[9] https://spdx.org/licenses/

Key `maintenance/type`

- Type: enumerate
- Presence: mandatory
- Values: `"internal"`, `"contract"`, `"community"`, `"none"`

This key describes how the software is currently maintained.

- `internal` - means that the software is internally maintained by the repository owner;
- `contract` - means that there is a commercial contract that binds an entity to the maintenance of the software;
- `community` - means that the software is currently maintained by one or more people that donate their time to the project;
- `none` - means that the software is not actively maintained.

Key `maintenance/contractors`

- Type: array of Contractor (see below)
- Presence: mandatory (if `maintenance/type` **is** `contract`)

This key describes the entity or entities, if any, that are currently contracted for maintaining the software. They can be companies, organizations, or other collective names.

Key `maintenance/contacts`

- Type: List of Contacts (see below)
- Presence: mandatory (if `maintenance/type` **is** `internal` or `community`)

One or more contacts maintaining this software.

This key describes the technical people currently responsible for maintaining the software. All contacts need to be a physical person, not a company or an organisation. If somebody is acting as a representative of an institution, it must be listed within the `affiliation` of the contact.

In case of a commercial agreement (or a chain of such agreements), specify the final entities actually contracted to deliver the maintenance. Do not specify the software owner unless it is technically involved with the maintenance of the product as well.

Section `localisation`

This section provides an overview of the localization features of the software.

Key `localisation/localisationReady`

- Type: boolean
- Presence: mandatory

If `yes`, the software has infrastructure in place or is otherwise designed to be multilingual. It does not need to be available in more than one language.

Key `localisation/availableLanguages`

- Type: list of IETF BCP 47 language tags
- Presence: mandatory
- Example: `"it"`, `"en"`, `"sl-IT-nedis"`

If present, this is the list of languages in which the software is available. Of course, this list will contain at least one language. The primary language subtag cannot be omitted, as mandated by the BCP 47[10].

Section `dependsOn`

This section provides an overview on the system-level dependencies required to install and use this software.

NOTE: do not list dependencies at the source code level (e.g.: software libraries being used), and focus only on runtime and/or system-level dependencies that must be installed and maintained separately. For instance, a database is a good example of such dependencies.

Key `dependsOn/open`

- Type: array of `dependency` (see below)
- Presence: optional

This key contains a list of runtime dependencies that are distributed under an open-source license.

Key `dependsOn/proprietary`

- Type: array of `dependency` (see below)
- Presence: optional

This key contains a list of runtime dependencies that are distributed under a proprietary license.

Key `dependsOn/hardware`

- Type: array of `dependency` (see below)
- Presence: optional

This key contains a list of hardware dependencies that must be owned to use the software.

2.1.2 Special data formats

Dependency

A `dependency` is a complex object. The properties are the following:

- `name` - **mandatory** - The name of the dependency (e.g. MySQL, NFC Reader)
- `versionMin` - the first compatible version
- `versionMax` - the latest compatible version

[10] https://tools.ietf.org/html/bcp47

- `version` - the only major version for which the software is compatible. It assumes compatibility with all patches and bugfixes later applied to this version.
- `optional` - whether the dependency is optional or mandatory

Complex versioning

It is of course possible to use the various keys to specify a complex compatibility matrix.

Ex. 1

```
- name: PostgreSQL
  version: "3.2"
  optional: yes
```

This snippet marks an optional dependency on PostgreSQL exactly version 3.2.

Ex. 2

```
- name: MySQL
  versionMin: "1.1"
  versionMax: "1.3"
```

This snippet marks a mandatory dependency on MySQL, allowing any version between 1.1 and 1.3.

Contact

A Contact is an object with the following properties:

- `name` - **mandatory** - This key contains the full name of one of the technical contacts. It must be a real person; do NOT populate this key with generic contact information, company departments, associations, etc.
- `email` - This key contains the e-mail address of the technical contact. It must be an email address of where the technical contact can be directly reached; do NOT populate this key with mailing-lists or generic contact points like "info@acme.inc". The e-mail address must not be obfuscated. To improve resistance against e-mail collection, use `\x64` to replace `@`, as allowed by the YAML specification.
- `phone` - phone number (with international prefix). This has to be a string.
- `affiliation` - This key contains an explicit affiliation information for the technical contact. In case of multiple maintainers, this can be used to create a relation between each technical contact and each maintainer entity. It can contain for instance a company name, an association name, etc.

Contractor

A Contractor is an object with the following properties:

- `name` - **mandatory** - The name of the contractor, whether it's a company or a physical person.
- `until` - **mandatory** - This is a date (YYYY-MM-DD). This key must contain the date at which the maintenance is going to end. In case of community maintenance, the value should not be more than 2 years in the future, and thus will need to be regularly updated as the community continues working on the project.
- `email` - This key contains the e-mail address of the technical contact. It must be an email address of where the technical contact can be directly reached; do NOT populate this key with mailing-lists or generic contact points like "info@acme.inc". The e-mail address must not be obfuscated. To improve resistance against e-mail collection, use `\x64` to replace `@`, as allowed by the YAML specification.

- `website` - This key points to the maintainer website. It can either point to the main institutional website, or to a more project-specific page or website.

Dates

All dates in `publiccode.yml` must follow the format "YYYY-MM-DD", which is one of the ISO8601 allowed formats. This is the only allowed format though, so not the full ISO8601 is allowed for the date keys.

Encoding

publiccode.yml **MUST** be UTF-8 encoded.

2.2 Country-Specific Extensions

While the standard is structured to be meaningful on an international level, there are additional information that can be added that makes sense in specific countries, such as declaring compliance with local laws or regulations. The provided extension mechanism is the usage of country-specific sections.

All country-specific extensions are contained in a section named with the two-letter lowercase ISO 3166-1 alpha-2 country code[11]. For instance `spid` is a property for Italian software declaring whether the software is integrated with the Italian Public Identification System.

If a software is compliant I will find:

```
it:
  countryExtensionVersion: "0.2"
  piattaforme:
   - spid: yes
```

Notice that country-specific extensions within international sections are not allowed. Countries that want to extend the format should add a country-specific section instead.

2.2.1 Italy

All the extensions listed below are specific for Italy and, as such, they must be inserted in a section named with the `it` code. Every Country is specified using a two letters *country code* following the ISO 3166-1 alpha-2 standard.

Key `countryExtensionVersion`

- Type: string
- Presence: mandatory
- Example: `"1.0"`

This key specifies the version to which the current extension schema adheres to, for forward compatibility.

Please note how the value of this key is independent from the top-level `publiccodeYmlVersion` one (see *The Standard (core)* (page 5)). In such a way, the extensions schema versioning is independent both from the core version of the schema and from every other Country.

[11] https://it.wikipedia.org/wiki/ISO_3166-1_alpha-2

Key `conforme`

This section contains the keys for auto-declaring the compliance with the current legislation, with respect to the following sections. Not including these keys implies that the compliance is not known or not declared.

Key `conforme/lineeGuidaDesign`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software is compliant with the Italian accessibility laws (L. 4/2004), as further explained in the linee guida di design[12] (Italian language).

Key `conforme/modelloInteroperatibilita`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software is compliant with the linee guida sull'interoperabilità[13].

Regulatory reference: Art. 73 del CAD[14] (Italian language).

Key `conforme/misureMinimeSicurezza`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software is compliant with the Misure minime di sicurezza ICT per le Pubbliche amministrazioni[15] (Italian language).

Key `conforme/gdpr`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software respects the GDPR.

Section `piattaforme`

Key `piattaforme/spid`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software interfaces with SPID - il Sistema Pubblico di Identità Digitale[16].

[12] https://docs.italia.it/italia/designers-italia/design-linee-guida-docs
[13] https://docs.italia.it/italia/piano-triennale-ict/lg-modellointeroperabilita-docs
[14] https://docs.italia.it/italia/piano-triennale-ict/codice-amministrazione-digitale-docs/it/v2017-12-13/_rst/capo8_art73.html
[15] http://www.agid.gov.it/sites/default/files/documentazione/misure_minime_di_sicurezza_v.1.0.pdf
[16] https://developers.italia.it/it/spid

Key `piattaforme/cie`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software interfaces with the Italian electronic ID card (`Carta di Identità Elettronica`).

Key `piattaforme/anpr`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software interfaces with ANPR.

Key `piattafome/pagopa`

- Type: boolean
- Presence: optional

If present and set to `yes`, the software interfaces with pagoPA.

Section `riuso`

This section contains a set of keys related to the publication of the software inside the reuse catalog of Developers Italia[17].

Chiave `riuso/codiceIPA`

- Type: string (iPA code)
- Presence: mandatory if `repoOwner` is a Public Administration
- Example: `c_h501`

This key represents the administration code inside the Public Administration index (codice IPA).

2.3 Forks and variants

As already cited before, a fork may have two different forms based on the final aim. As such, in order to make it clear how to handle the *publiccode.yml* in both cases, below we define two different semantics: technical forks and software variants.

[17] https://developers.italia.it

2.3.1 Technical forks (i.e. to publish patches)

A technical fork is a fork made by a developer for the purpose of working on the original code base or sending improvements to the original authors, without any explicit goal of creating and publishing an alternative variant of the original software.

In the context of distributed control systems and collaborative code hosting platforms like GitHub, forking is almost always used by developers as a step to work on a contribution on an existing codebase, by sending "pull requests".

Because of the way forking works on GitHub and other platforms, developers publish their forks as perfect copies of the original software, thus including also `publiccode.yml`. However, parsers need to be able to distinguish such technical forks from the original codebase.

Parsers

Parsers **SHOULD** identify a technical fork by noticing that the top-level `url` key does not point to the repository in which the `publiccode.yml` is found.

Parsers **MIGHT** identify a technical fork also through metadata that might be exposed by the code hosting platform (eg: GitHub marks forks explicitly as forks)

Authors

Authors of technical forks **SHOULD NOT** modify `publiccode.yml` in any way. Specifically, they **MUST NOT** modify the top-level `url` key that **MUST** continue pointing to the original repository.

There is no explicit key to mark a fork as a technical fork. This is a conscious design decision because we do not want authors of technical forks to be aware of `publiccode.yml` and necessarily be aware of how to modify it. The current design does not require authors to do anything.

2.3.2 Software variants

A software variant is a fork that is offered as an alternative to the original upstream software.

It contains modifications that are still not part of the upstream version, like more features, different dependencies, optimizations, etc.

By marking a fork as a variant, the author indicates that they believe that the variant includes a complete and working set of modifications that might be useful to other people.

Marking a fork as a variant does **not** relate to the willingness of contributing upstream; the author might still plan to contribute the modifications upstream, or even being in the process of doing so. Thus, even if the fork will eventually be merged upstream, it might make sense to mark it as a variant during the process, so that others might discover it and benefit from it.

Parsers

Parsers **SHOULD** identify a variant by noticing that the top-level `url` key matches to the repository in which the `publiccode.yml` is found, **AND** a top-level `isBasedOn` exists and points to a different repository.

Parsers should expect and analyze other differences in `publiccode.yml` between variants of the software. Specifically `description/features` is designed to be compared across variants to identify and show user-visible differences.

Authors

Authors that are willing to publish a fork as a variant **MUST** at least:

- add a key `isBasedOn` pointing to one or more upstream repositories from which this variant is derived.
- Change the value for `url` to point to the repository holding the variant.
- Change the value for `legal/repoOwner` to refer to the themselves (the authors of the variant).
- Revisit the `maintenance` section to refer to the maintenance status of the variant.

Moreover, authors **SHOULD** evaluate the following changes:

- add the features that differentiate the variant to the `description/features` key. Existing features **SHOULD NOT** be edited or removed from this list unless they have been removed from the variant, to allow parsers to easily compare feature lists.

2.4 List of software categories

Here follows a controlled vocabulary of useful tags for categorizing the software.

2.4.1 Valid Tags

- accounting
- agile-project-management
- applicant-tracking
- application-development
- appointment-scheduling
- backup
- billing-and-invoicing
- blog
- budgeting
- business-intelligence
- business-process-management
- cad
- call-center-management
- cloud-management
- collaboration
- communications
- compliance-management
- contact-management
- content-management
- crm

- customer-service-and-support
- data-analytics
- data-collection
- data-visualization
- digital-asset-management
- digital-citizenship
- document-management
- donor-management
- e-commerce
- e-signature
- email-management
- email-marketing
- employee-management
- enterprise-project-management
- enterprise-social-networking
- erp
- event-management
- facility-management
- feedback-and-reviews-management
- financial-reporting
- fleet-management
- fundraising
- gamification
- geographic-information-systems
- grant-management
- graphic-design
- help-desk
- hr
- ide
- identity-management
- instant-messaging
- inventory-management
- it-asset-management
- it-development
- it-management
- it-security

- it-service-management
- knowledge-management
- learning-management-system
- marketing
- mind-mapping
- mobile-marketing
- mobile-payment
- network-management
- office
- online-booking
- online-community
- payment-gateway
- payroll
- predictive-analysis
- procurement
- productivity-suite
- project-collaboration
- project-management
- property-management
- real-estate-management
- remote-support
- resource-management
- sales-management
- seo
- service-desk
- social-media-management
- survey
- talent-management
- task-management
- taxes-management
- test-management
- time-management
- time-tracking
- translation
- video-conferencing
- video-editing

- visitor-management
- voip
- warehouse-management
- web-collaboration
- web-conferencing
- website-builder
- workflow-management

2.5 Scope list

Here follows a controlled vocabulary of useful tags for categorizing the software.

2.5.1 Valid Tags

- agriculture
- culture
- defence
- education
- emergency-services
- employment
- energy
- environment
- finance-and-economic-development
- foreign-affairs
- government
- healthcare
- infrastructures
- justice
- local-authorities
- manufacturing
- research
- science-and-technology
- security
- society
- sport
- tourism
- transportation

- welfare

2.6 Esempi

Below there are two examples of *publiccode.yml* files, the first one represents the minimum configuration, which contains the mandatory fields, whilst the second one represents a more extended version.

2.6.1 Minimum Version

```yaml
publiccodeYmlVersion: "0.2"

name: Medusa
url: "https://example.com/italia/medusa.git"
softwareVersion: "dev"    # Optional
releaseDate: "2017-04-15"
platforms:
  - web

categories:
  - financial-reporting

developmentStatus: development

softwareType: "standalone/desktop"

description:
  en:
    localisedName: medusa    # Optional
    genericName: Text Editor
    shortDescription: >
          A rather short description which
          is probably useless

    longDescription: >
          Very long description of this software, also split
          on multiple rows. You should note what the software
          is and why one should need it. We can potentially
          have many pages of text here.

    features:
       - Just one feature

legal:
  license: AGPL-3.0-or-later

maintenance:
  type: "community"

  contacts:
    - name: Francesco Rossi

localisation:
  localisationReady: yes
  availableLanguages:
    - en
```

2.6.2 Extended Version

```yaml
publiccodeYmlVersion: "0.2"

name: Medusa
applicationSuite: MegaProductivitySuite
url: "https://example.com/italia/medusa.git"
landingURL: "https://example.com/italia/medusa"
isBasedOn: "https://github.com/italia/otello.git"
softwareVersion: "1.0"
releaseDate: "2017-04-15"
logo: img/logo.svg
monochromeLogo: img/logo-mono.svg

inputTypes:
  - text/plain
outputTypes:
  - text/plain

platforms:
  - android
  - ios

categories:
  - content-management
  - office

usedBy:
  - Comune di Firenze
  - Comune di Roma

roadmap: "https://example.com/italia/medusa/roadmap"

developmentStatus: development

softwareType: "standalone/desktop"

intendedAudience:
  scope:
    - science-and-technology
  countries:
    - it
    - de
  unsupportedCountries:
    - us

description:
  en:
    localisedName: Medusa
    genericName: Text Editor
    shortDescription: >
          This description can have a maximum 150
          characters long. We should not fill the
          remaining space with "Lorem Ipsum". End

    longDescription: >
          Very long description of this software, also split
```

(continues on next page)

(continued from previous page)

```
          on multiple rows. You should note what the software
          is and why one should need it.

    documentation: "https://read.the.documentation/medusa/v1.0"
    apiDocumentation: "https://read.the.api.doc/medusa/v1.0"

    features:
       - Very important feature
       - Will run without a problem
       - Has zero bugs
       - Solves all the problems of the world
    screenshots:
       - img/sshot1.jpg
       - img/sshot2.jpg
       - img/sshot3.jpg
    videos:
       - https://youtube.com/xxxxxxxx
    awards:
       - 1st Price Software of the year

legal:
  license: AGPL-3.0-or-later
  mainCopyrightOwner: City of Chicago
  repoOwner: City of Chicago
  authorsFile: AUTHORS

maintenance:
  type: "contract"

  contractors:
    - name: "Fornitore Privato SPA"
      email: "dario.bianchi@fornitore.it"
      website: "https://privatecompany.com"
      until: "2019-01-01"

  contacts:
    - name: Francesco Rossi
      email: "francesco.rossi@comune.reggioemilia.it"
      affiliation: Comune di Reggio Emilia
      phone: "+3923113215112"

localisation:
  localisationReady: yes
  availableLanguages:
    - en
    - it
    - fr
    - de

dependsOn:
  open:
    - name: MySQL
      versionMin: "1.1"
      versionMax: "1.3"
      optional: yes
    - name: PostgreSQL
      version: "3.2"
```

(continues on next page)

(continued from previous page)

```
      optional: yes
  proprietary:
    - name: Oracle
      versionMin: "11.4"
    - name: IBM SoftLayer
  hardware:
    - name: NFC Reader
      optional: yes

it:
  countryExtensionVersion: "0.2"

  conforme:
    lineeGuidaDesign: yes
    modelloInteroperabilita: yes
    misureMinimeSicurezza: yes
    gdpr: yes

  piattaforme:
    spid: yes
    cie: yes
    anpr: yes
    pagopa: yes

  riuso:
    codiceIPA: c_h501
```