Lifetime of documents

Contents

Apertis is an already mature distribution which fully supports the development of systems for embedded devices as well as a whole infrastructure to make that possible. During its evolution changes are necessary and updating documentation is a key point to making it user friendly.

With the goal of having documentation that really expresses the current state of the distribution, having a process and procedure to manage the lifetime of documents is crucial since each new release introduces changes than can lead to documentation to become outdated. This document describes the process and procedure to follow from creation until the end-of-life of documents to achieve the desired goal.

Document types

After the migration to the new website a massive restructuring of the documentation has been made. As a result of this process, a clear organization has been implemented, leading to different document types being identified.

Architecture

Documents of this type are meant to show the current state of the distribution, from a high level point of view. They are the result of design decision that were adopted in the past to provide users with the basis to build their solutions.

Concept Designs

Information under this category express a concept that has been analyzed to improve Apertis in some way. They are an essential part of the development process since they set the basis for future work.

Since these documents reflect an implementation plan, their validity should match the one of the plan they describe and reflect the adaptations that may have chosen during the implementation to provide better results. It is expected that once the plan has been executed the document should be refactored and moved to more suitable sections, since it should describe the current state of one aspect of Apertis.

Guides

Guides are meant to be more practical documents describing the steps for executing a procedure. These documents generally include command line examples, sample sources and logs.

The information validity for this type of document is expected to be short since new application releases might introduce changes or even simplify the described workflow. Therefore, the default time limit for a guide should be shorter than for other types of documents.

Policies

This category encompasses documents that cover the rules or principles that are the basis for Apertis as a distribution.

Reference hardware

Under this section is the list of devices currently supported by Apertis, as well as small guides, which are very hardware specific. The information here is meant to help users bring up Apertis on those devices.

Releases

Documents under this section are meant to be kept as a historical reference, to show how Apertis has been evolving across time. Therefore they should be preserved without any modification, except for links updates.

Default review period

Based on the above comments the following table shows the default review period for each type of document expressed in months.

Type	Default review period	Notes
Architecture	24 months	Based on Apertis release support period
Concept Designs	12 months	
Guides	6 months	
Policies	24 months	
Reference hardware	12 months	Based on Apertis release cadence
Releases	-	Not meant to have an end of life

Metadata

Based on the concepts and categories already described a process needs to be created to reduce the gap in-between current status and what the documentation reflects.

To make this possible, documents should include metadata which will make it easier to determine the moment an update or review is required.

Dates

Currently each document provides its creation date, which is useful to provide information about its age. However, this does not express if the document is still valid. In order to provide additional information on this topic some additional fields should be added, leading to the following list:

- `date`, which indicates the date the document was published. This date is important since it brings information about how old the document is and the type of review it will require. Most probably, older documents should require a deeper review than a newer one.
- `lastmod`, date which advertises when the document was last updated with non-trivial changes. This modification implies that the document is still valid as the committer needed to check the information on it. However this does not imply that all the information on it is valid. This date is important as some documents could be updated with minor changes, like URL updates, which does not say anything about the validity of the page.

- `lastreview`, date which shows when the document was last reviewed to ensure that it is still valid and the information in it is still accurate. This implies that the whole document has been reviewed and the committer is confident about its state.

- `reviewperiod`, which shows the number of months after a document requires a review to confirm it is still valid if the default value from the table is not adequate. This could be helpful to highlight documents that might need to be reviewed sooner than the default as they could be affected by the natural evolution of the project.

All the dates from the previous list should match the format "YYYY-MM-DD" which is currently widely used in the documentation, while the period should use an integer to match the default review period.

The following example shows a header in a document tracking different stages in its lifetime:

```
date = "2020-06-09"
lastmod = "2021-02-03"
lastreview =  "2021-05-05"
```

Status

Some fields can be used to advertise the status of a document, allowing readers to quickly spot important information about a document:

- `status`, a string which gives a clear idea of the validity of a document, Allowed values are: "", "Requires Review", "Requires Update", "Deprecated" and "Not Applicable". A value of "" or empty indicates that the document is valid. The value "Not Applicable"can be used to mark a page as invalid for e.g. a specific release. Further information shall be given in the `statusDescription` field.

- `statusDescription`, a short sentence giving additional information about the status

The use of these fields is recommended to support a healthy documentation.

As an example:

```
status = "Deprecated"
statusDescription = "This document covers frameworks that have been removed in v2022dev2."
```

Tags

Tags are a good way to connect different documents that are related. Based on them it is easy to perform a quick search on a specific topic and retrieve the most significant results. While searching by keywords instead of tags is useful,

sometimes it results in a long list of documents where the keyword appears but it is only referenced, giving it little value.

Tags are useful as they can efficiently connect documents and ideas, but to make this true they should follow these guidelines:

- Tags should be meaningful, the name should condense an important concept leading to significant search results. For instance a tag like `Apertis` will have little value, since it will bring thousands of results. On the other hand, a tag like `licensing` will probably be useful.
- In order to enforce the previous comment, the tags used in a document should be checked against a list of valid tags. This list should be part of the documentation so new tags can be added by developers in the same way documentation is updated.
- Tags can be implemented gradually, as tagging will require a manual work in order to highlight the relevant information. Initially tags can appear at the top of the page to summarize the topics the document covers. Once the tagging process is complete, additional pages to list all the available tags can be added to improve the user experience, by creating a link to https://www.apertis.org/tags. However, this should be done only when all the documents have a valid tag to avoid showing misleading results.
- Order tagged documents. When searching using tags it will be useful to have the list ordered by relevance if the number of matches is higher than 5, to highlight the most important documents. This is true only when searching for tags, other searching and listing should follow the current approach of sorting by title.

As an example this is a list of proposed tags, the list is not meant to be exhaustive:

- Application Framework
- Connectivity
- Licensing
- Packaging
- Security
- Upgrade
- Infrastructure

The inclusion of tags is done by adding a list of strings in the header of the document, as in the example above:

```
tags = ["licensing"]
```

Process

This section describes the process that should be put in place to keep track of each document's life, from its creation until its end-of-life.

Creation

When a new document needs to be created to help users of Apertis take full advantage of it, a set of steps should be followed to ensure that it will be supported over time:

- Choose the type of document as described in document types
- Choose a title which describes the document
- Add `date` metadata as creation date
- Add `lastmod` metadata as a placeholder with empty value
- Add `lastreview` metadata as a placeholder with empty value
- Add tags to link the document to other (when tags are widely available) documents
- Use neutral language, focus on simplicity, since many Apertis users are not native English speakers
- Use links as references to reduce the number of changes if the referenced URL is updated

Evolution

As time passes, Apertis evolves adding, updating and dropping packages and functionalities. Documentation should evolve as well, continuously reflecting these changes as part of the standard development process.

During the documentation update it is important to follow some guidelines:

- `lastmod` should be updated in case the document has been modified with non-trivial changes as the committer considers it still has value
- `lastreview` should be updated in case the whole document has been reviewed during the modification, as the committer considers it is still valid and up to date
- As needed, update the tags used in the document in case of a review.
- `reviewperiod` should be added/updated if for some reason the default value does not fit
- `status` and `statusDescription` should be updated to reflect important aspects of a document, for instance that the document requires a review or update

End-of-life

Following the description in the section above, at some point in time a document probably becomes obsolete and should be dropped. Sometimes this is straight forward, since during the update process it can be easily spotted. However, in

most cases documentation that references dropped packages or functionalities are kept unnoticed.

In order to minimize the amount of documents in this state a task should be created to review any document where `reviewperiod` from `lastreview` has expired, in order to confirm that it still has value and the information it provides is accurate and up to date.

In case the document is obsolete but need to be published for different reasons, such as historical reason, this fact should be reflected with the `status` and `statusDescription` fields.

Examples

In this section some examples will be introduced to described how the workflow is implemented with some real scenarios.

Case 1: Developer edits a document to update an URL

Since the problem is minimal and does not invalidate the usefulness of the document the metadata does not need to be updated. However, if during this process the developer performs a review of it, he should update the `lastreview` date accordingly.

Case 2: Developer needs to update a section of a document

In this case, a minimal understanding of the document is required, which should be sufficient to find the value of keeping it updated. As consequence, `lastmod` date should be updated to reflect that this document is alive and that it is worth to keep it updated. As in the previous case, if during this process the developer performs a review of it, he should update the `lastreview` date accordingly.

However, if during this process the developer understands that the document is outdated or with no value for the project he should rise a warning. To do it he should change the `status` to document following:

- Requires Review: The developer suspects that some information in the document is not accurate or updated as there were changes in the project after the last document update.
- Requires Update: The developer is confident that the document requires an update to show the current state of the project.

In both cases the field `statusDescription` should be updated to provide additional information.

Case 3: Developer reviews a document

The purpose of reviewing a document is to validate that it still has value, it is accurate and updated. During this process the developer should read the

whole document and make an statement, according to the best of his knowledge. There can be different situations:

- The document has value, is accurate and updated, in which case the developer is confident to approve it and updates `lastreview` to the current date.
- The document has value but some minor updates are needed. In this case, in order to avoid any overhead, the developer updates the document as required and sets `lastmod` and `lastreview` accordingly.
- The document has value but some major updates are needed. In this case, the `status` should be updated to "Requires update"and `statusDescription` should be used to describe the situation.
- The document has no longer value, in which case the developer should submit a MR to drop the document.

Statistics

The main purpose of documentation is to be consumed by users. In this regard, having feedback on different aspects of its usage is a key element to design a documentation strategy. Feedback can be collected from different sources, such as analyzing visits or performing simple pools. The results of this feedback should be summarized in a web interface in order to provide information to help improve documentation.

Changes and contributors

Apertis website is kept under revision control using `git`, which makes it easy to keep track of changes and contributors. Having this information recorded allows to have an overall idea of how documents are changing. Based on this information it is possible to customize the `reviewperiod` and to suggest a reviewer.

Visits

The first and easiest statistics that can be included is the number of visits for each document. This information should help to put the focus on those documents that bring more attention, in order to provide a better user experience. For this reason, documents with a high number of visits should be analyzed and set a `reviewperiod` accordingly, and given high priority when the time for a review comes. It can also help to spot on which topic additional documentation should be created.

Surveys

Besides just analyzing visits, additional information can be obtained by simple surveys to visitors. The key point on the survey should be its simplicity in

order to avoid discouraging visitors from using them. A suggested approach is to use simple questions that can be answered by just selecting an option, such as yes/no questions.

Suggested questions:

- Did you find the information useful?
- Did you find the information up to date?

It is important to note that any survey sent, should comply with legal aspects such as GDPR[1] to ensure personal data protection.

Dashboard

The information generated based on the previous comments should be available through a web page with administrator restrictions, to have a single point to generate queries.

As an initial approach, the report should present a table with the following information:

- Lines changed in the last week, month, year
- Top contributor in the last week, month, year
- Visits in the last week, month, year
- Surveys results in the last week, month, year

The result obtained from statistics should provide the basis to infer:

- Documents that need to be updated
- Documents that need to be reworked
- Topics where documentation is more consumed
- Possible reviewers

The recommendation is to include these reports in a `dashboard` similar to current package dashboard[2]

Automated tasks

Keeping documentation reliable is impossible without setting an automated task. There are two kinds of automated tasks that can help to address potential issues which can be implemented with CI pipelines.

- Pipeline for work in progress branches: on every branch change, several checks should be performed to ensure the health of documentation:
  - Documents updated should provide `title` and `date`

[1] https://ec.europa.eu/info/law/law-topic/data-protection_en
[2] https://infrastructure.pages.apertis.org/dashboard/

  - Documents updated with non-trivial changes should have a valid `lastmod`, in a time window of one month. Since it is not easy to decide whether a change is trivial or not, as a default rule, this check should be performed for every document that has been updated where the diff shows a difference of more than 20 lines
  - Documents should be checked to provide valid internal and external links
  - Documents should provide tags to link them to other documents (once tags are enabled)
  - Ensure formatting follows a standard
  - Test code blocks for syntax errors

  Any issue with the performed checks should make the pipeline fail preventing a merge request to be merged and reported in the pipeline log to help the submitter to fix it.

- Scheduled pipeline: a pipeline should be triggered on a weekly basis to ensure:

  - Documents provide valid external links. A task should be automatically created to address any update requirement
  - Documents which have reached their lifetime period since the `lastreview` are highlighted. A MR should be automatically created to change the `status` to "Requires review"for each document in this situation. A separated task should be created to review each of these documents
  - Documents with `status` "Requires review"or "Requires update"have a task associated to address the issue on them
  - A task is created to ensure documents that have reached their lifetime period since the `lastreview` will be reviewed
  - Update statistics reports available trough the dashboard based on:
    * Changes and contributors
    * Visits
    * Surveys