



Lifetime of documents

1	Contents	
2	Document types	3
3	Architecture	3
4	Concept Designs	3
5	Guides	3
6	Policies	3
7	Reference hardware	3
8	Releases	4
9	Default review period	4
10	Metadata	4
11	Dates	4
12	Status	5
13	Tags	5
14	Process	7
15	Creation	7
16	Evolution	7
17	End-of-life	7
18	Examples	8
19	Statistics	9
20	Changes and contributors	9
21	Visits	9
22	Surveys	9
23	Dashboard	10
24	Automated tasks	10
25	Apertis is an already mature distribution which fully supports the development	
26	of systems for embedded devices as well as a whole infrastructure to make that	
27	possible. During its evolution changes are necessary and updating documenta-	
28	tion is a key point to making it user friendly.	
29	With the goal of having documentation that really expresses the current state	
30	of the distribution, having a process and procedure to manage the lifetime of	
31	documents is crucial since each new release introduces changes than can lead to	
32	documentation to become outdated. This document describes the process and	
33	procedure to follow from creation until the end-of-life of documents to achieve	
34	the desired goal.	

35 Document types

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

39 Architecture

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

43 Concept Designs

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

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

53 Guides

54 Guides are meant to be more practical documents describing the steps for exe-
55 cuting a procedure. These documents generally include command line examples,
56 sample sources and logs.

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

61 Policies

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

64 Reference hardware

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

68 Releases

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

72 Default review period

73 Based on the above comments the following table shows the default review
74 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

75 Metadata

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

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

81 Dates

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

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

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

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

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

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

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

113 Status

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

116 • `status`, a string which gives a clear idea of the validity of a document, Al-
117 lowed values are: “”, “Requires Review”, “Requires Update”, “Deprecated”
118 and “Not Applicable”. A value of “” or empty indicates that the docu-
119 ment is valid. The value “Not Applicable” can be used to mark a page as
120 invalid for e.g. a specific release. Further information shall be given in
121 the `statusDescription` field.

122 • `statusDescription`, a short sentence giving additional information about
123 the status

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

125 As an example:

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

128 Tags

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

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

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

- 136 • Tags should be meaningful, the name should condense an important concept
137 leading to significant search results. For instance a tag like `Apertis`
138 will have little value, since it will bring thousands of results. On the other
139 hand, a tag like `licensing` will probably be useful.
- 140 • In order to enforce the previous comment, the tags used in a document
141 should be checked against a list of valid tags. This list should be part of
142 the documentation so new tags can be added by developers in the same
143 way documentation is updated.
- 144 • Tags can be implemented gradually, as tagging will require a manual work
145 in order to highlight the relevant information. Initially tags can appear at
146 the top of the page to summarize the topics the document covers. Once
147 the tagging process is complete, additional pages to list all the available
148 tags can be added to improve the user experience, by creating a link to
149 <https://www.apertis.org/tags>. However, this should be done only when
150 all the documents have a valid tag to avoid showing misleading results.
- 151 • Order tagged documents. When searching using tags it will be useful to
152 have the list ordered by relevance if the number of matches is higher than
153 5, to highlight the most important documents. This is true only when
154 searching for tags, other searching and listing should follow the current
155 approach of sorting by title.

156 As an example this is a list of proposed tags, the list is not meant to be exhaus-
157 tive:

- 158 • Application Framework
- 159 • Connectivity
- 160 • Licensing
- 161 • Packaging
- 162 • Security
- 163 • Upgrade
- 164 • Infrastructure

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

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

168 Process

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

171 Creation

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

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

186 Evolution

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

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

- 191 • `lastmod` should be updated in case the document has been modified with
192 non-trivial changes as the committer considers it still has value
- 193 • `lastreview` should be updated in case the whole document has been re-
194 viewed during the modification, as the committer considers it is still valid
195 and up to date
- 196 • As needed, update the tags used in the document in case of a review.
- 197 • `reviewperiod` should be added/updated if for some reason the default value
198 does not fit
- 199 • `status` and `statusDescription` should be updated to reflect important as-
200 pects of a document, for instance that the document requires a review or
201 update

202 End-of-life

203 Following the description in the section above, at some point in time a document
204 probably becomes obsolete and should be dropped. Sometimes this is straight
205 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, `lastmodified` 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

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

- 244 • The document has value, is accurate and updated, in which case the de-
245 veloper is confident to approve it and updates `lastreview` to the current
246 date.
- 247 • The document has value but some minor updates are needed. In this case,
248 in order to avoid any overhead, the developer updates the document as
249 required and sets `lastmod` and `lastreview` accordingly.
- 250 • The document has value but some major updates are needed. In this case,
251 the `status` should be updated to “Requires update”and `statusDescription`
252 should be used to describe the situation.
- 253 • The document has no longer value, in which case the developer should
254 submit a MR to drop the document.

255 Statistics

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

262 Changes and contributors

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

267 Visits

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

275 Surveys

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

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

281 Suggested questions:

- 282 • Did you find the information useful?
- 283 • Did you find the information up to date?

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

286 Dashboard

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

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

- 292 • Lines changed in the last week, month, year
- 293 • Top contributor in the last week, month, year
- 294 • Visits in the last week, month, year
- 295 • Surveys results in the last week, month, year

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

- 297 • Documents that need to be updated
- 298 • Documents that need to be reworked
- 299 • Topics where documentation is more consumed
- 300 • Possible reviewers

301 The recommendation is to include these reports in a `dashboard` similar to current
302 `package dashboard`²

303 Automated tasks

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

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

¹https://ec.europa.eu/info/law/law-topic/data-protection_en

²<https://infrastructure.pages.apertis.org/dashboard/>

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

321 Any issue with the performed checks should make the pipeline fail prevent-
322 ing a merge request to be merged and reported in the pipeline log to help
323 the submitter to fix it.

324 • Scheduled pipeline: a pipeline should be triggered on a weekly basis to
325 ensure:

326 – Documents provide valid external links. A task should be automati-
327 cally created to address any update requirement
328 – Documents which have reached their lifetime period since the `las-`
329 `treview` are highlighted. A MR should be automatically created to
330 change the `status` to “Requires review” for each document in this sit-
331 uation. A separated task should be created to review each of these
332 documents
333 – Documents with `status` “Requires review” or “Requires update” have a
334 task associated to address the issue on them
335 – A task is created to ensure documents that have reached their lifetime
336 period since the `lastreview` will be reviewed
337 – Update statistics reports available through the **dashboard** based on:
338 * Changes and contributors
339 * Visits
340 * Surveys