KEY TO DIGITAL OBJECTS

Kitodo developer guidelines

In accordance with section 11 of the articles of association, this document came into force following a resolution passed by the general meeting of the association Kitodo. Key to digital objects e.V. on 01/06/2017 and on this effective date replaced the previously valid version adopted by the general meeting on 17/09/2012 and amended on 31/07/2014.

This document sets out the principles of collaborative software development with Kitodo. It describes the general development methodology, binding standards such as coding guidelines, and the tools and means of communication to be applied. The guidelines are intended for use by software developers who wish to work on the Kitodo source code and decision-makers who want to place development requests and ensure that the work they contract is compatible with the open source community product Kitodo.[1]

The requirements apply to all components collaboratively developed within the Kitodo environment (Kitodo.Production, Kitodo.Presentation, etc.), to the extent that they can be used for the respective product. Source code that is not the result of collaborative development and contributes to a Kitodo product must at minimum comply with the specified standards and conditions on using source code.

Decentralised, function-driven development

Software development in the Kitodo environment is generally a decentralised and function-driven process. Only those release managers responsible for the product are given write access to the central trunk. All further development and debugging takes place in source code branches that must be generated from the present trunk. **Development branches not generated from the trunk cannot be merged.** A development branch should always serve to deal with a single, self-contained work package (e.g. fixing one specific bug or enabling one specific function).

On completion of the programming and successful testing of the changes, the branch is then released for integration in the trunk (pull request). The release managers subject the source code to a final audit within a maximum of three months, primarily to review compliance with formal criteria and perform a superficial function test. **The audit does not substitute for proper function and performance testing, which is the responsibility of the developers.** If the audit is positive, the release managers merge the source code in the trunk; if negative, they return the development branch to the developers for reworking.

[1] In accordance with section 3.5 of the articles of association, members of the association *Kitodo. Key to digital objects e. V.* are obliged to ensure compliance with the rules set out in this document. This also applies to development requests placed with third parties.

In general, the developer decides when to approve the source code for integration, allowing, for example, commercial developments to be withheld for a certain period of time. As, however, integration of the source code increases in complexity with the passing of time, the code should be released no later than three months following completion of development.[2] **One exception are simple bug fixes, which must be released immediately on completion if they concern a bug in a public development branch.**

Coordination of development projects

All development projects should be publicly documented and must be communicated to a release manager. On request, release managers will treat these communications confidentially. This aims to prevent parallel development and thus promote a stronger concentration of development resources within the community and reliable release scheduling.

As release management resources are required for the audit of code releases and integration in the trunk and generally need to be made available at short notice, the development schedule on larger projects should be coordinated with release management at an early stage. Only then can release management dedicate the relevant resources at the time they are required, thereby helping to ensure the project is not delayed.

Coding guidelines

Development branches should only contain the changes needed to attain the development objective. Purely cosmetic changes (e.g. formatting) or non-functional additions (e.g. to the documentation) to otherwise unaltered sections of source code should therefore be undertaken in a separate development branch and not be accompanied by functional changes. This ensures it is easy to identify sections of code with modified functions when integrating the code in the trunk. Several development objectives should also not be realised within the same development branch. This facilitates function and performance tests, improves the transparency of the development process, and simplifies auditing by the release managers.

The following coding guidelines must be adhered to depending on the technology and product used. This standardisation aims to make it easier for developers to find their way around third-party source code while ensuring a consistently high code quality.

General standards

These stipulations apply irrespective of the programming language used and to all components of the Kitodo suite in equal measure.

- The working language is generally English, in particular regarding the choice of variable, class, and method names and the source code documentation.
- The relevant rules concerning the choice of meaningful identifiers and their notation (`camelCase` or `CamelCase`) must be observed. One exception are constants, which are always written in capitals.
- Object-oriented programming is generally to be preferred. Large classes and methods must be avoided. Each file should contain only one class and be named accordingly.

[2] See also section 3.5 of the articles of association, according to which this deadline particularly applies to association members.

- The character encoding is generally UTF-8; line breaks are coded in Unix format; indents are made with spaces (4 spaces per indent). Each file must end with a line break.
- Line lengths must not exceed 120 characters. Comments may contain a maximum 80 characters per line.
- Each class, each public method, and each class variable must be documented using the PHPDoc or JavaDoc standard. The source code should be largely self-explanatory thanks to the use of meaningful identifiers and small methods. Single-line comments should be used to explain specific functionalities in the source code. Non-public methods should also be documented in this way unless they are trivial.
- Getter and setter methods should always be used in place of public variables. Setters are always `void` methods; getters do not have any parameters. Within a class, access to private variables should always be direct.
- Loops (`for`, `while`), conditions (`if`, `else`), and comparable code blocks must always be enclosed in braces.
- Exceptions (`error`) must always be treated in an appropriate way and, where necessary, passed on to a point at which this is possible. They must always be registered in the log. The latter also applies to non-critical programming errors (`warning`) and notes (`notice`).
- All changes are briefly and meaningfully documented in the commit messages. By contrast, no change log is kept within the source code files.

Kitodo.Presentation

The presentation component of the Kitodo digitisation suite comprises an extension for the open source CMS TYPO3. This extension contains all the key function modules of a modern digital library, and at the same time also provides an extensive API for developing add-on modules (in the form of separate extensions). The TYPO3 project[3] coding guidelines are binding on development work. The following stipulations, which may contain divergent rules, also apply:

- New frontend plug-ins and backend modules should typically be implemented in the form of separate extensions, thus enabling the modular combination of required components. Only basic functions may form part of the `dlf` extension.
- The `dlf` extension forms the core of the presentation layer and, in separate extensions, must therefore always be named as a dependency in ext_emconf.php. Wherever possible, the APIs from TYPO3 and Kitodo must be used.
- The dlf extension and all other extensions must at minimum be operable with the oldest LTS version of TYPO3 that is still supported.
- Frontend plug-ins should always be derived from the class `tx_dlf_plugin`, backend modules from the class `tx_dlf_module`, both of which are themselves derived from the relevant TYPO3 standard classes.

[3] https://docs.typo3.org/typo3cms/CodingGuidelinesReference/

- The class variable `$prefixId` must always be `tx_dlf` to ensure a common namespace for all input variables. This allows any plug-in to access the input of any other Kitodo plug-in as required.
- The release managers must be contacted if intervention in the core components' processes is required. They will then endeavour to modify the core components or implement a hook. Core component classes may only be overwritten as XCLASSES in substantiated exceptions that have been agreed with the release managers.

Kitodo.Production

The production component of the Kitodo digitisation suite comprises a web application for an application server such as Tomcat or Jetty. In general, the coding standards of Scott Ambler[4] are considered binding. The following stipulations, which may contain divergent rules, also apply:

- Class files must be saved in the directory structure in accordance with the package to which they belong (e.g. class Foo, package `org.Kitodo.bar:/src/org/Kitodo/bar/Foo.java`).
- The source code must be compatible with the Java8 standard.
- Names of databases, tables, and columns must be written entirely in small letters.
- Type-safe lists must always be used (`ArrayList<Element> list; HashMap<Integer, String> map;`).
- Models for type-safe lists should be used for loops.
- If possible, an empty list should be returned instead of null.
- The source code must be formatted using the included code formatter.

Tools and communication

GitHub[5] is the central development platform. It contains the public source code repository, a bug and feature tracker, and a Wiki for discussing and developing new concepts. **This is the basis of the Git version control system**. It is also possible to contact the release managers via GitHub. A free account needs to be set up to use GitHub.

In general, all program errors must be reported using the GitHub bug tracker. Security-critical bugs can also be confidentially disclosed to the release management to prevent publication prior to their elimination. Wherever possible, development projects should also be documented in the GitHub bug tracker, and at minimum confidentially disclosed to the relevant release managers to prevent the unwitting duplication of work.

Source code released for integration in the trunk must always be provided on GitHub in the form of a git branch based on the present trunk and announced to the release managers as a pull request using the function available there. Source code delivered in other ways cannot be accepted.[6] Pro-

[4] http://www.ambysoft.com/downloads/javaCodingStandards.pdf
[5] https://GitHub.com/Kitodo
[6] See also section 3.5 of the articles of association, according to which association members commit themselves to actively supporting the release management.

vided the requirements of a decentralised, function-driven development are met, the actual development may also take place in a privately created local branch, which is not uploaded to GitHub for integration in the trunk until the programming has been completed.

Last update: June 2017