FOP Development: Coding Conventions

$Revision: 627324 $

Table of contents

Acknowledgement: Some content in this guide was adapted from other Apache projects such as Avalon, Cactus, Turbine and Velocity.

1. Subversion Repository

Conventions in this section apply to Repository content, regardless of type:

- Files checked in must conform to the code conventions for that type of file (java files must conform to java requirements, xml to xml requirements, etc.). If a submitted patch does not conform, it is the responsibility of the committer to bring it into conformance before checking it in. Developers submitting patches are encouraged to follow the code conventions to reduce the work load on the committers.
- To reduce the amount of spurious deltas, all text (non-binary) files checked into SVN must have Unix-style line endings (LF only). Many IDEs and editors (even on non-Unix platforms) have settings that can facilitate this convention.
- In order to be able to discern commits from a committer from those where a committer applied a patch from a contributor, the commit message must contain a separate line following this pattern: **"Submitted by: [contributor's name] <[contributor's obfuscated e-mail address]>"**. This also helps doing audits on the repository.

2. Java

2.1. Java Style

In order to facilitate the human reading of FOP source code, reduce churning in code, and prevent disputes, the FOP developers have agreed on a set of coding conventions. The basis of these coding conventions is documented in the Apache XML Project Guidelines, which requires that **all Java Language source code in the repository must be written in conformance to Sun's** Code Conventions for the Java Programming Language. In addition, the FOP developers have agreed to other conventions, which are summarized in the following table:

Convention	Rationale	Enforced By
Every Java source file starts with the Apache licence header.	Required by Apache.	checkstyle
No tabs in content.	Programmers should not have to adjust the tab settings in their editor to be able to read the source code.	checkstyle
Indentation of 4 spaces per level.	Maximize readability.	Not enforced
Comments, identifiers, and project documentation must be in English. In general, other	To avoid the need for everyone to learn all languages, English has become the standard	Not enforced

languages must not be used, except in translated documentation and language-specific i10n files.	language for many technology projects, and is the only human language that all FOP developers are expected to know.	
American English spelling should be used. Alternative spelling and idioms are tolerated, but may be changed by anyone to American.	Some standard is useful, and American English is widely used and accepted for technology standards and projects.	Not enforced.
Fully qualify all import statements (no "import java.util.*")	Clarity	checkstyle
No underscores in variable names except for static finals.	Upper/lower case distinctions can be made in all other variable names, eliminating the need for artificial word boundaries.	checkstyle
Opening brace for a block should be on the same line as its control statement (if, while, etc.).	Standardization, general preference.	checkstyle
Write appropriate javadoc entries for all public and protected classes, methods, and variables.	Basic API documentation is needed.	checkstyle
Personal attribution in the source code, such as @author tags and attribution comments should not be used. Excepted from this general rule are potentially confusing or wide-ranging changes. If such changes prove useful over time, the related comments should be removed.	Personal attribution tends to clutter the code. The relevant historical information that might be useful for problem-solving is tracked in the code repository.	Not enforced. Anyone is free to remove such comments.

For developers that dislike these conventions, one workaround is to develop using their own style, then use a formatting tool like astyle (Artistic Style) before committing.

2.2. Checkstyle

The java syntax checker "Checkstyle" is used to enforce many of the FOP coding standards. The standards enforced through Checkstyle are documented in its configuration file (xml-fop/checkstyle.cfg). The conventions defined in the configuration file are an integral part of FOP's coding conventions, and should not be changed without common consent. In other words, the configuration file contains additional conventions that are not documented on this page, but are

generally accepted as good style within the java community (i.e. they are the default behavior of checkstyle, which the FOP developers have decided to adopt *de facto*). Any apparent contradiction between the configuration file and this document should be raised on the fop-dev mailing list so that it can be clarified.

To use the "checkstyle" target in FOP's build process, download the source from the Checkstyle web site, place checkstyle-all-*.jar in the lib directory and call "build checkstyle". Output (in the build directory) includes checkstyle_report.txt and checkstyle_report.xml. If you copy the file contrib/checkstyle-noframes.xsl from Checkstyle into FOP's root directory, you will also get an HTML report.

Checkstyle is probably most useful when integrated into your IDE. See the Checkstyle web site for more information about IDE plugins.

2.3. Java Best Practices

The following general principles are a distillation of best practice expectations on the FOP project.

- Apply common sense when coding. When coding keep in mind that others will read your code and have to understand it.
- Readability comes before performance, at least initially.
- If you can refactor some code to make it more understandable, please do so.
- Properly document code, especially where it's important.
- Use interfaces instead of implementations where possible. This favors a clearer design and makes switching between implementations easier (Examples: List instead of ArrayList/Vector, Map instead of HashMap/Hashtable).
- Avoid using exceptions for flow control.
- Try to catch exceptions as much as possible and rethrow higher level exceptions (meaning hiding the low level detailed and putting a message that is more related to the function of your code).
- It is important not to lose the stack trace which contains important information. Use chained exception for that. Avalon Framework provides CascadingException (and similar) for this. Exception class names and stack traces must be treated like gold. Do whatever is required so that this information is not lost. Printing error messages to System.err or System.out is useless in a server-side environment where this info is usually lost.
- Always log the exception at the higher level (i.e. where it is handled and not rethrown).
- Try to avoid catching Throwable or Exception and catch specific exceptions instead.

2.4. Resources

- [book on code style] Code Complete by Steve McConnell.
- [code formatting software] JRefactory.

2.5. Related Links

- Apache XML Graphics Code Repositories
- Jakarta Code Conventions and Standards (see Coding Conventions and Standards section)

3. XML

Convention	Rationale	Enforced By
XML files must always be well-formed. Validation is optional.	Document integrity	Not enforced
No tabs in content.	Users should not have to adjust tab settings in their editor to be able to read the content.	Not enforced
Indentation of 2 spaces per level	Maximize readability	Not enforced