CodingConventions

(Draft) Coding conventions

⚠️ THESE CONVENTIONS ARE NOT OFFICIAL IN ANY WAY. PLEASE IGNORE THEM FOR NOW ⚠️

The conventions are marked with the following symbols

Symbol	Meaning
1	Must have convention. A failure to comply with the requirement may lead to the rejection of a patch
2	Should have convention. Recommended but not mandatory
3	Unapproved convention. Use when adding a new convention before it is officially signed off. Developers please ignore these conventions for now

Eclipse IDE

If you are using the Eclipse IDE for development, then it is recommended to download the following:

- Code formatter configuration (TODO link)
- Cleanup configuration (TODO link)
- TODO add this? .project and .classpath and .settings folders
- Try and fix all the code warnings that Eclipse flags in your code (the warnings are there for a reason)
- Don't supress any types of warnings (with the execption of generics warnings when using 3rd party code that you cannot modify)

Style

- **1** Ensure the Apache copyright header is at the top of every class
- **3** Imports should be organized into blocks starting with "java", "javax", "org", "com" in that order, and within each block should be sorted alphabetically (Note that Eclipse IDE does this automatically for you)
- **1** Use camelCase naming convention for classes/methods/fields/variables
- **1** All indentation should be done with 4 spaces; tabs should never be used
- **3** Lines should be wrapped if it improves readability
- **3** The name of an "abstract" class should generally be prefixed with Abstract..., especially if it is the base implementation of an interface
- **3** Use a single blank line to separate code when appropriate; there shouldn't be any more than one consecutive blank line in any part of the code
- **3** Classes that are not explicitly designed for extension should be made final
- **3** Put braces around if blocks in any part of the if/else sections have multiple lines

Javadoc & comments

- **3** Write Javadoc at the class level
- **3** Write Javadoc on public methods when appropriate, and only if the Javadoc will add more than the obvious (for example getters and setters are self-explanitory)
- **3** Use @since markers on classes / methods to show the version where they were introduced
- **3** Use the @deprecated marker to show in which version the item was first deprecated as well as the replacement to be used: e.g. @deprecated (4.2) use {@link URIBuilder}.
- **3** Javadoc should only precede class, field, constructor or method declarations and should not appear anywhere else
- **3** Javadoc should wrap at 80 chars
- **3** Comments in the code are also encouraged.
- **3** Don't rely on users being able to read the SVN log comments. Log comments are not permanent and are not provided to end users so should only be used to explain the commit message. Additional comments should be added to the code as necessary so that it is clear without needing to refer to the log comments.
- **3** Use "//" for all non-Javadoc comments, including multi-line comments - avoid the use of /**/ comment blocks.
- **3** Mark empty blocks of code with a "//NOP" comment to document in was intentially left blank (although a more useful comment is also welcome)
- **3** Use "//TODO" before the comment to mark possible improvements or remaining tasks (Eclipse IDE will flag these comments)
- **3** Use "//HACK" before the comment to mark incomplete patches, fragile code or poor solutions (Eclipse IDE will flag these comments)
- **3** Use "//NOTE" before the comment to emphasize important comments, for example something counter-intuitive or unexpected like a for loop going backwards (Eclipse IDE will flag these comments)

Fields

- **1** Fields should appear at the top of the class, not declared half-way down the class
- **3** Fields should appear in the following order: static, then instance. Group fields with the same qualifiers together if possible.
- **3** (TODO the following rule is not adhered to anywhere, but I would recommend it rather than using the this. prefix) All instance fields should start with the m prefix (this is also setup in the standard preferences). Don't use m prefix on local variables.
- **3** All static final fields should be capitalized, static non-final fields should have an s prefix (but avoid these as they are inherently thread-hostile)
- **3** Make fields final when possible
- **3** Don't use public or protected fields (unless they are immutable/final)

- **3** All fields in an enum should be final (i.e. all enum instances should be immutable)

Constructors

- **3** Constructors should be declared at the top of the class (below the fields)
- **3** Singleton constructors should be made private
- **3** The constructor should not pass "this" to any other method, as the object is not fully initialized yet (TODO link to corresponding Josh Bloch Effective Java Item number). For the same reason: ctors should not start a thread, nor should they invoke overrideable methods.

Exceptions

- **3** Use exceptions for exceptional circumstances
- **3** Never simply print an exception to stderr or stdout, as the exception will effectively just be lost.
- **3** In very rare cases it may be approriate to ignore an exception, in which case document it clearly.
- **3** TODO is there a custom HTTP client exception that is used?
- **3** Always try and add a useful message to the exception in case any one encounters it, containing for example any runtime data that can be used to diagnose the exception
- **3** Document the exception and what causes it in the Javadoc (using @throws)
- **3** Handle exceptional conditions by throwing an exception and not, for example, returning null from a method.

- **3** Checked exceptions should represent potentially recoverable exceptions; runtime exceptions should represent non-recoverable errors / unexpected errors / programming errors, as per [http://www.oracle.com/technology/pub/articles/dev2arch/2006/11/effective-exceptions.html].

Collections

- **3** Use classes from the Collections API such as ArrayList or HashMap over legacy classes such as Vector or Hashtable.
- **3** For situations that do not require synchronization, ArrayList is significantly faster than Vector.

- **3** For situations that do require synchronisation, use classes from the Concurrent API if possible. Vector is not enough on its own as the following link explains [http://www.ibm.com/developerworks/java/library/j-jtp09263.html]

JIRA integration

- **3** Comment a block of code with the JIRA reference if appropriate (especially for bug fixes), for example HTTPCLIENT-1
- **3** When committing a patch, add the JIRA reference in the commit comment (JIRA can then list the file under the "Subversion Commits" tab)

General

- **3** Prioritize readability of code over speed/cunning code - flag with comments when being cunning, so others don't inadvertently break the code by not having spotted the subtlety. "Java files are for humans, class files are for the JVM"
- **3** Use assert to check assumptions where appropriate, no need to assert the obvious though as too many asserts impact readability
- **3** Override both hashCode() and equals() when your object will be used in a Set/Map, and toString() is also useful
- **3** TODO remove this one? The "final" keyword should be whenever a field/parameter/variable does not change during its scope, to document the intent that it should remain unchanged, and simply the lifecycle of the class/method for other readers of the code
- **3** Don't deprecate code unless there is an alternative method to use
- **3** Consistency, standardization and simplicity are useful rules of thumb...
- **3** Where a method returns a collection or array, if there are no valid entries return an empty collection or array rather than returning null. Use exceptions for errors. This simplifies code, as there is no need to check for null before iterating over the response.

Annotations

TODO use of jcip annotations like ThreadSafe etc.

- **3** Don't use the following SVN tags: @author (not useful for collaborative code), @date (causes problems when comparing SVN with source archive). @version or @id is sufficient.