

# Coding+Guide

These guidelines are meant to encourage consistency and best practices among people working on DistributedLog code base. They should be observed unless there is compelling reason to ignore them.

## Basics

### Java

DistributedLog code should follow the [Sun Java Coding Convention](#), with the following additions.

- Lines can be up to 120 characters long.
- Indentation should be 4 spaces. Tabs should never be used.
- Use curly braces even for single-line ifs and elses.
- No @author tags in any javadoc

## Logging

- Logging should be taken seriously. Please take the time to access the logs when making a change to ensure that the important things are getting logged and there is no junk there.
- Logging statements should be complete sentences with proper capitalization that are written to be read by a person not necessarily familiar with the source code.
- All loggings should be done with SLF4j, never System.out or System.err.
- Logging levels:
  - *INFO* is the level you should assume the software will be run in. *INFO* messages are things which are not bad but which the user will definitely want to know about every time they occur.
  - *TRACE* and *DEBUG* are both things you turn on when something is wrong and you want to figure out what is going on. *DEBUG* should not be so fine grained that it will seriously affect performance of the program. *TRACE* can be anything. Both *DEBUG* and *TRACE* statements should be considered to be wrapped in an *if (logger.isDebugEnabled)* or *if (logger.isTraceEnabled)* check to avoid performance degradation.
  - *WARN* and *ERROR* indicate something that is **BAD**. Use *WARN* if you aren't totally sure it is bad, and *ERROR* if you are.

## Monitoring

- DistributedLog uses a pluggable [StatsProvider](#) on exporting metrics
- Any new features should come with appropriate metrics for monitoring the feature is working correctly.
- Those metrics should be taken seriously and only export useful metrics that would be used on production on monitoring/alerting healthy of the system, or troubleshooting problems.

## Unit Tests

- New changes should come with unit tests that verify the functionality being added
- Unit tests should test the least amount of code possible. Don't start the whole server unless there is no other way to test a single class or small group of classes in isolation.
- Tests should not depend on any external resources. They need to setup and teardown their own stuff.
- It is okay to use the filesystem and network in tests since that's our business but you need to clean up them after yourself.
- Do not use sleep or other timing assumptions in tests. It is always, always, wrong and will fail intermittently on any test server with other things going on that causes delays.

## Configuration

- Names should be thought through from the point of view of the person using the config.
- DistributedLog uses CamelCase naming convention on naming configuration settings.
- The default values should be thought as best value for people who runs the program without tuning parameters.
- All configuration settings should be added to default configuration file and documented.

## Concurrency

- DistributedLog is a purely asynchronous program. It is latency sensitive. All synchronization and locking should be in a fine granularity way.
- All threads should have proper meaningful name.

## Backwards Compatibility

- Wire protocol should support backwards compatibility to enable no-downtime upgrades. This means the servers (e.g bookies or write proxies) **MU** **ST** be able to support requests from both old and new clients simultaneously.
- Metadata formats and data formats should support backwards compatibility.

