# Contributing To AAI Best Practise

**Attention To**

- AAI PTL James Forsyth
- Coding Style Initiative Pamela Dragosh
- Integration PTL Yang Xu (oparent module owner)

**Navigation**



## Contribution Type New API Version

Approximately once or twice per ONAP release, the AAI API version is incremented, e.g. in Casablanca the version was v14 and in Dublin the versions were v15 and v16.

New API version number should be accompanied by Updating AAI charts for MSB discovery config

New API version number should be accompanied by updates to AAI R4 Integration Sanity Test Plans

Updated procedures should be published in setup wiki pages like AAI Developer Environment Setup - Dublin

Updated procedures should be published in tutorial wiki pages like AAI Tutorial-Making and Testing a Schema Change - Dublin

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

tbc

## Contribution Type New Schema or EdgeRule

Approximately one or more times per ONAP release, new schema or EdgeRule updates are done within an API version, e.g. the CCVPN use case was introduced in Casablanca (v14) .

New schema elements are done in the OXM file. Relationships between schema elements are done in the EdgeRule file.

New schema or EdgeRule changes should be following AAI Data Model Principles

New schema or EdgeRule changes should be accompanied by new wiki pages under AAI Design Documentation

New schema or EdgeRule changes should be accompanied by updates to AAI R4 Integration Sanity Test Plans

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

tbc

## Contribution Type Breaking Change to Schema or EdgeRule

Less than once per several ONAP releases, breaking changes to schema or EdgeRule may be contemplated, e.g. the pnf unique key was changed in Dublin (v16).

Breaking changes are those which break compatibility with the API, thus forcing clients of AAI to adapt both code and data to use the updated API.

New schema or EdgeRule changes should be following AAI Data Model Principles

New schema or EdgeRule changes should be accompanied by new wiki pages under AAI Design Documentation

New schema or EdgeRule changes should be accompanied by updates to AAI R4 Integration Sanity Test Plans

Migration plan should be published in new wiki pages under AAI Design Documentation

Updated procedures should be published in tutorial wiki pages like AAI Tutorial-Making and Testing a Schema Change - Dublin

Breaking changes should be accompanied by new documentation for readthedocs

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

Bring discussion to PTL Meeting

tbc

# Contribution Type New Microservice

New microservice is assumed to be within existing code repositories.

New microservice should be accompanied by new wiki pages under AAI Design Documentation

New microservice should be accompanied by updates to AAI R4 Integration Sanity Test Plans

Updated procedures should be published in setup wiki pages like AAI Developer Environment Setup - Dublin

Updated procedures should be published in tutorial wiki pages like AAI Tutorial-Making and Testing a Schema Change - Dublin

New microservice should be accompanied by Updating AAI charts for MSB discovery config

New microservice should be accompanied by new test coverage measured by Sonar

New microservice should be accompanied by new documentation for readthedocs

New microservice should be accompanied by new Helm charts for OOM deployment

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

tbc

# Contribution Type New Code Repository

New code repository usually means new microservice(s) as well (include the actions from section "Contribution Type New Microservice").

New code repository should be accompanied by new Jenkins jobs

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

tbc

# Contribution Type New REST URL API within Microservice

New REST URL API should be accompanied by new wiki pages under AAI Design Documentation

New REST URL API should be accompanied by updates to AAI R4 Integration Sanity Test Plans

New REST URL API should be accompanied by Updating AAI charts for MSB discovery config

New REST URL API should be accompanied by new test coverage measured by Sonar

New REST URL API should be accompanied by new documentation for readthedocs

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

tbc

# Contribution Type Decommissioning API version, Schema, EdgeRule, Microservice or REST URL API

Decommissioning existing functions hasn't really happened yet, but it is definitely a "breaking change" (include the actions from section "Contribution Type Breaking Change to Schema or EdgeRule").

Bring discussion to AAI Weekly Status Meeting (Cancelled)

Bring discussion to AAI Developers Meeting

tbc

# Other Contributions

Contributing To AAI By Assisting with Sonar Reports

Contributing To AAI By Assisting with Maven Warnings

# Contributions Coding Style

See also:

- Setting Up Your Development Environment#ONAPEclipseJavaFormatter
- Policy Framework Project: Software Development Best Practices
- **AAI-2198** - Getting issue details... STATUS
- **TSC-71** - Getting issue details... STATUS

ONAP projects are built using maven and include the "checkstyle" plugin that is configured by the "oparent" module.

Developers can see the output of the "checkstyle" audit by explicitly triggering the phase with "mvn process-sources".

Most of the warnings from the "checkstyle" audit are considered to be "Low" priority (as per the JIRA case definitions Tracking Issues with JIRA#JIRAPriorityDefinitionforBugs) and many PTLs do not want to be flooded with these reviews while there are more urgent and important cases to finish.

However, the volume of the warnings can mask real problems, so it is still useful to resolve the underlying issues. Using the tools mentioned below could be a way to resolve many "checkstyle" warnings quickly, efficiently and consistently across the ONAP projects.

Referring to aai-common/pom.xml as configuration for the sub-components:

**aai-common/pom.xml**

```
<!--
Using https://code.revelc.net/formatter-maven-plugin/ for Eclipse formatter
Using https://github.com/diffplug/spotless/tree/master/plugin-maven for import order
Use in combination to rewrite code and imports, then checkstyle

mvn formatter:format spotless:apply process-sources
-->
<plugin>
     <groupId>net.revelc.code.formatter</groupId>
     <artifactId>formatter-maven-plugin</artifactId>
     <version>2.8.1</version>
     <configuration>
        <configFile>${project.parent.basedir}/onap-java-formatter.xml</configFile>
     </configuration>
     <!-- https://code.revelc.net/formatter-maven-plugin/
          use mvn formatter:format to rewrite source files
          use mvn formatter:validate to validate source files -->
 </plugin>
<plugin>
  <groupId>com.diffplug.spotless</groupId>
  <artifactId>spotless-maven-plugin</artifactId>
  <version>1.18.0</version>
  <configuration>
    <java>
     <importOrder>
       <order>com,java,javax,org</order>
     </importOrder>
    </java>
  </configuration>
<!-- https://github.com/diffplug/spotless/tree/master/plugin-maven
     use mvn spotless:apply to rewrite source files
     use mvn spotless:check to validate source files -->
</plugin>
```

The "onap-java-formatter.xml" file contains an Eclipse formatter configuration, which can be imported and used in the Eclipse IDE by developers writing new code.

The "net.revelc.code.formatter" plugin uses the Eclipse formatter configuration to re-format the Java source code.

The "com.diffplug.spotless" plugin is used to rewrite the "import" statements in the Java source code. Technically, "com.diffplug.spotless" plugin can also use Eclipse formatter configuration to rewrite the Java source code, but it seemed to throw exceptions on some scenarios, making it a less reliable way than the "net.revelc.code.formatter" plugin.

The "com.diffplug.spotless" plugin also has other language capabilities, which could be useful as well, e.g. Javascript, JSON, YAML, etc, as well as a modular way to add more FormatterSteps.

After using the plugins to rewrite the source files, developers should choose which ones to commit to the repository. There will be several "checkstyle" warnings remaining that cannot be automatically rewritten, so the developers can focus on fixing those few items manually.

Other projects will be able to use this solution by:

- copying the pom.xml plugin configuration into their own pom.xml (or refer to it from a common module like oparent)
- copying the "onap-java-formatter.xml" file into their own code repositories (or refer to it from a common module like oparent)
- training developers to make use of the plugins