

# Woden Decision Making and Development Processes

## Table of contents

1 Introduction.....	2
2 Open Development.....	2
3 Project Status.....	2
4 Project Discussion and Communication.....	2
5 Development Process.....	3
6 Bug Tracking and Work Items.....	3
7 Source Code.....	3
8 Building.....	6
9 Testing.....	8
10 Release Process.....	11
11 Connection with W3C WSDL Working Group.....	13

## 1. Introduction

This document summarizes the Woden decision making and development processes. It is expected that this document will be useful to

1. Hold current committers accountable to the established Woden processes
2. Assist new contributors in getting up-to-speed with the Woden development processes in order to ensure all Woden contributors are doing things the Woden way and more generally the Apache way
3. Provide a clear view of the way in which Woden is developed for the Open Source community

## 2. Open Development

Apache Woden is an Open Source project and the development of Woden is therefore being conducted in an open fashion. This theme should be evident as you read through this guide of the Woden decision making and development processes. At a high level what Open development means is that the entire development process of Woden is open to the community. Nothing is kept behind closed doors and information is readily available via the Woden Web site, mailing list, and wiki. The key here is transparency. If you (that's right, YOU) think the Woden development team is not being transparent it is your right to identify in what way we are not being transparent and hold us accountable to [The Apache Way](#).

## 3. Project Status

Woden's status is currently available from a number of sources:

- [Apache Web Service Project board reports](#). Reports are currently produced every 3 months.
- [Apache Incubator board reports](#). Reports are currently produced every 3 months.
- [Woden Incubation Status File](#)
- [Woden Web site](#)
- [woden-dev mailing list posts](#)

## 4. Project Discussion and Communication

All project wide communication is done in the open. This is not to suggest that one-on-one conversations among Woden committers and contributors is not allowed. This type of conversation is extremely valuable to the project's development and is encouraged. However, when anything other than a trivial decision is to be made it must be done in the presence of the Woden community.

There are 3 methods of communicating openly with the Woden development team and community:

1. the woden-dev mailing list
2. the weekly public status telecons
3. IRC

## 5. Development Process

Woden follows a milestone development process. The key to milestone development is that there are a number of milestone defining criteria specified at the beginning of the milestone process. The milestone can only be declared once those milestone defining criteria are met. For example, in Woden M8 the priority 1 defining criteria are that Woden must pass all 'good' and 'bad' test cases in the WSDL 2.0 test suite.

Woden milestone plans are communicated through the woden-dev mailing list and posted to the [Woden Web site](#). In addition, all plan items are recorded in Jira, Woden's bug tracking system.

The Woden milestone planning process is an Open process. Woden takes into account the needs of adopter, like Apache Axis2, when planning. Please communicate requirements via Jira and the woden-dev list.

## 6. Bug Tracking and Work Items

All bugs and work items are stored in [Jira](#). It is a Woden best practice to associate a Jira entry with every code change.

## 7. Source Code

Woden uses Subversion (SVN) for source control and all Woden code is stored in the [Woden SVN repository](#). See the "SVN Client Configuration" section at this link for action required to handle EOL characters in a platform-neutral way.

All code developed as part of Woden is licensed under the Apache Software License v2.0 (ASL v2.0). All Woden source files must include the Apache boilerplate at the top of the file:

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more
 * contributor license agreements. See the NOTICE file
 * distributed with
 * this work for additional information regarding copyright
```

ownership.

```
* The ASF licenses this file to You under the Apache License,
Version 2.0
* (the "License"); you may not use this file except in
compliance with
* the License. You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied.
* See the License for the specific language governing
permissions and
* limitations under the License.
*/
```

### **Woden Java coding conventions**

Woden uses common Java coding conventions. The main reason Woden needs an agreed set of coding conventions is to avoid different auto-formatting results when using IDEs like Eclipse and IDEA. If different code formatting rules are applied to a file via auto-formatting, then reformatting changes may be made along with any functional changes. If both types of code change are committed to the repository at the same time, it can be difficult to identify the functional changes from a 'diff' of the file.

To avoid this problem, the Woden project has 2 rules about coding conventions:

1. Use the agreed set of code formatting conventions, described below.
2. Don't include auto-reformatting changes and other changes in the same commit. Commit them separately, but close together to avoid merge conflicts. If you need to auto-reformat and apply a lot of new function, warn other developers about the files you're working on, then auto-reformat and commit to establish the no-op baseline, then apply your functional changes and commit.

### **Eclipse code formatter default settings**

Use the default code formatting provided with Eclipse 3.3, with one exception - the max line length should be 100 instead of 80. A subset of the most notable conventions (i.e. the ones that have caused us the most auto-reformatting problems) are described below. (TODO -

check if there's a way to export these settings).

To access these settings in Eclipse 3.3 navigate to menu Window->Preferences->Java->Code Style->Formatter.

- click the 'Edit' button to see the settings on tabbed pages
- on the 'Line Wrapping' and 'Comments' tabs, change the Max Line Length from 80 to 100

### Indentation

Tab Policy - 4 spaces (not Tab character).

Indentation - 4 spaces for each indent.

What to indent;

- declarations within class body
- statements within method/constructor body
- statements within blocks (if/else, case, loops)

### Braces

Opening brace '{' on the same line and closing brace '}' with same indentation as the statement the block applies to.

Example:

```
while(loop_condition) {
    if(some_condition) {
        ...
    } else if(other_condition) {
        ...
    } else {
        break;
    }
    ...
}
```

### New Lines

Keep 'else if' on one line.

Example:

```
    } else if(other_condition) {
```

### Line Wrapping

Max line length (wrap after) 100 chars.

Indentation for wrapped lines is 2 indents (e.g. 8 spaces).

### Accessor method names

Use 'get' / 'set' prefix for public accessor methods (e.g. `getNamespace()`, `setNamespace(URI)`).

Use 'is' prefix for getters that return booleans (e.g. `isValid()`).

### Comments

Do not auto-reformat existing comments, but note the max line convention above (100 chars).

Use Javadoc comments for all public/protected API class, method and field declarations. For example;

- the class declaration for `org.apache.woden.wsdl20.Binding`
- the method declaration for `org.apache.woden.wsdl20.Binding.getInterface()`
- the static constant `org.apache.woden.WSDLReader.FEATURE_VERBOSE`

Use Javadoc comments for all public/protected implementation class, method and field declarations. For example;

- content of `.internal.` packages, such as `org.apache.woden.internal.wsdl20.*`

When implementing or overriding API methods, use the `@see` tag to specify the relevant method. For example, for the implementation method

`org.apache.woden.internal.wsdl20.BindingImpl.getInterface()` use:

- `@see org.apache.woden.wsdl20.Binding#getInterface()`

For all private/package private declarations, where comments are required use non-Javadoc comments.

## 8. Building

Woden defines an API and provides 2 implementations of it. One is based on the Document Object Model (DOM) and the other on the Axiom Object Model (OM).

The Woden build produces 3 binary jar files:

- `woden-api-[version].jar` - the WODEN API classes.
- `woden-impl-dom-[version].jar` - the DOM-specific classes and any common

implementation classes.

- `woden-impl-om-[version].jar` - the OM-specific classes and any common implementation classes.

These jar files are packaged into 2 binary distributions (as zip and tar archives):

- `apache-woden-dom-[version].zip` - contains the API and DOM jar files and the relevant jar file dependencies.
- `apache-woden-om-[version].zip` - contains the API and OM jar files and the relevant jar file dependencies.

The Woden build also produces a source code distribution (zip and tar):

- `apache-woden-src-[version].zip`

Woden has [Ant](#) and [Maven](#) build processes. The Ant build was created originally to produce the jar files and distribution archives. The Maven build was introduced later to integrate the Woden build process with other Apache projects that depend on Woden, like Axis2. We maintain both build processes so that developers can use the one they prefer. Note however, that the Maven build just produces the 3 jar files (it does not produce the distribution archives).

**Ant** uses a concept of targets to figure out what work to do when you want a certain task done, they are defined inside a `build.xml` which is in the root of our source directory. There are two main ways in which we run a target, from the command line or inside eclipse. If you want to use the command line you need to download and install Ant from their download site above, then you just use the command `ant [targetname]` to run a target called `targetname`. If you want to use Eclipse you need to make sure you have the Ant plug-in, which should come by default depending which package you downloaded, then to run a target you can use the "Run As" context menu on a `build.xml` file to run it as a "Ant Build" which will run the default ant task or as a "Ant Build..." to specify which target(s) you wish to run.

The top level ant targets we have and their uses are :-

- `buildAll` - Builds all Woden packages (API, DOM, and OM).
- `buildAndTestAll` - Builds all Woden packages (API, DOM, and OM) then runs all the tests as explained in the Testing section.
- `distBuild` - Builds all Woden packages (API, DOM, and OM) then creates the compressed archives for the DOM and OM distributions with dependencies.
- `buildAPI` - Builds the Woden API package.
- `buildDOM` - Builds the Woden DOM package.
- `buildOM` - Builds the Woden OM package.

These targets all use a common build directory to store their results in, its structure is :-

- `build/` - Root of the build directory.
- `build/classes/` - Classes from building the java source code.
- `build/dist/` - Zip and Tar files of complete distributions in the form `apache-woden-[src/dom/om]-[version].[zip/.tar.gz]`.
- `build/test/` - Test results from Junit tests.
- `build/jar/` - Jar files for each package in for form `woden-[api/dom/om]-[version].jar`.
- `build/JavaDoc/` - Java Doc pages for all the java classes.

**Maven** uses a set of conventions to simplify the build and deployment process and uses a concept of goals to figure out what needs to be done. The layout of the project is defined in `pom.xml` files, each of which specifies one distribution. In Woden we have three main distributors so have four `pom.xml` files, one in the root source directory and one in each of the `woden-api`, `woden-dom` and `woden-om` sub directories. To use Maven you need to [download](#) it, then run one of the maven goals with `mvn [goal]`. If you run these from the Woden root directory the goal will be applied all three distributions.

The main goals you will use are :-

- `compile` - Compiles all three distributions.
- `test` - Compiles then tests all three distributions.
- `package` - Compiles, tests then packages all three distributions.

Woden's Maven build uses the following directory structure :-

- `/` - Root source folder with the `pom.xml` linking all three distributions.
- `/woden-api` - Woden API distribution, runs Maven goals on just the API java source.
- `/woden-dom` - Woden DOM distribution, runs Maven goals on the DOM and Common java source.
- `/woden-om` - Woden OM distribution, runs Maven goals on the OM and Common java source.

## 9. Testing

There are 2 test suites for Woden. The **Woden JUnit Test Suite** provided with Woden tests the Woden API and functionality of the Woden framework (configuration, factory, reader, error handling, etc). The **W3C WSDL 2.0 Test Suite** tests Woden's compliance with the WSDL 2.0 Recommendation (i.e. its WSDL 2.0 spec compliance).

Both of these test suites *should* be run to test Woden thoroughly. It's not enough just to run the Woden JUnit tests, because while this suite will test some of the Woden framework

implementation and that Woden implements all of its API, it will not perform a complete WSDL 2.0 functional test. That's the job of the W3C WSDL 2.0 Test suite. Both test suites *must* be run *before* committing code to the repository. Code contributions submitted by non-committers as JIRA patches should also be tested against both test suites.

The Woden JUnit test suite should test the entire Woden API. It should invoke every API method at least once. So for any additions or changes to the API, the Woden JUnit test suite must be updated accordingly. For any non-WSDL functionality added or changed in the Woden implementation(s), appropriate test cases should also be included in the JUnit test suite where appropriate.

(TODO - maybe split the JUnit test suite so that an API-only suite exists for use by projects that want to create their own implementation of the Woden API).

The Woden JUnit test cases *must be written using the Woden API* (i.e. the correct Woden programming model), unless it's absolutely necessary to exercise the implementation class methods directly for a given test case. This will ensure that the Woden development team can change the implementation at any point without major impact on the test suite. On several occasions in the past, we have had to do significant, time-consuming rework on the test suite to fix compile breaks introduced when the implementation changed, because Woden developers had used the implementation classes directly when writing test cases. Our initial reasons for this were "we know the code, so it's okay", but as we've learnt, getting Woden client code to work is not the same as keeping it working. We expect Woden users to use the API only, so our test suite should set the right example.

The W3C WSDL 2.0 Test Suite is complete for the 'good' WSDL test cases. It should now have WSDL test cases that provide full coverage of what *is* allowed by the WSDL 2.0 specification. The test suite does not yet fully cover the 'bad' WSDL test cases. That is, it does not yet have WSDL test cases for every *assertion* specified by the WSDL 2.0 specification. All of Woden's WSDL 2.0 functionality (i.e. handling good WSDL and detecting bad WSDL) must be tested against the W3C WSDL 2.0 Test Suite. If a new WSDL 2.0 test case is needed to test some WSDL 2.0 functionality in Woden, Woden developers should create the required test case and then contribute it as a patch to the [W3C WSDL 2.0 CVS repository](#). The patch can be submitted via the [W3C Bugzilla system](#) (select the 'WSDL' category when creating a new bug report).

**Woden JUnit Tests** are located in the test/ directory which is structured to mirror the src/ directory for the parts of code tests are written for. There are two ways these can be run; either inside Eclipse or directly from Ant.

Inside Eclipse the JUnit test(s) can be run by selecting "JUnit Test" in the "Run As" context menu for one test, whole packages, or the entire test folder. Then the tests results with any error messages and stack traces will be displayed in the JUnit panel. You can also debug

JUnit tests from inside Eclipse by selecting the "JUnit Test" from the "Debug As" context menu.

You can also run the JUnit tests using Ant from the command line or inside Eclipse as described in the [building](#) section above. The Ant script is the build.xml file in the Woden project's root directory. This script has three test targets which each run a particular subset of the Woden JUnit tests.

- `runAllJUnitTests` - This runs all of the JUnit tests for both the DOM and OM implementations of Woden.
- `runOMTests` - This runs all the JUnit tests for the OM implementation.
- `runDOMTests` - This runs all of the JUnit tests for the DOM implementation.

These Ant targets produce an HTML report of the tests results in the file `build/test-results/(24 hour)_(minute)_(second)/junit-noframes.html`.

**W3C WSDL 2.0 Test Suite** is a collection of WSDL 2.0 documents used to test that a WSDL 2.0 processor complies with the W3C WSDL 2.0 Recommendation. The processor must successfully process all of the documents in the test suite to demonstrate its compliance. Each WSDL document represents a test case for some aspect of the WSDL 2.0 spec. These include *good* test cases, containing valid WSDL which collectively demonstrate all of the types of WSDL 2.0 syntax permitted by the spec. They also include *bad* test cases, containing invalid WSDL which violates one of the WSDL 2.0 assertions defined in the spec. There should be at least one *bad* test case for each assertion, but the *bad* test suite is not yet complete. The Woden project will contribute further assertion test cases to the W3C as development of Woden's WSDL validation continues.

To demonstrate its spec-compliance, a WSDL 2.0 processor should correctly parse all *good* WSDL documents and should detect the correct assertion violation for all *bad* WSDL documents. The W3C WSDL 2.0 test framework can check that the processor complies in this way. The WSDL 2.0 processor simply needs to capture its results from processing the test suite in some XML formats that the W3C WSDL 2.0 test framework will evaluate against baseline XML results. It will then generate a compliance report based on this comparison.

The Woden code tree is setup to facilitate using the W3C WSDL 2.0 test suite in this way. It uses ANT scripts to download the W3C WSDL 2.0 test suite (the good and bad WSDL documents), run Woden against this test suite and generate the Woden result files, then copy the result files to your local copy of the W3C project that contains the WSDL 2.0 test framework. This W3C project is a CVS project called 'desc', short for Description - more on this later. The last step is then to run an ANT script in the 'desc' project to evaluate Woden's results and generate the HTML reports which compare Woden's results to the W3C baseline.

The official versions of these compliance reports are published on the [WSDL 2.0 Interop Dashboard](#), where the *Component Model Test Results* show the results for the *good* WSDL test suite and the *Validation Test Results* show the results for the *bad* WSDL test suite.

Here are the steps to run Woden against the W3C WSDL 2.0 Test suite:

1. You need to have run Ant to build Woden as described in the [building](#) section above. This will not only build the Woden jar's to test against, but it will also ensure that the W3C WSDL 2.0 test suite has been downloaded by the `getW3cWsd120` target in the main `build.xml` script.
2. Run the default target in `ant-test/build.xml`. This will generate the xml files containing the Woden results. The *good* test suite result can be found in the `ant-test/results/` directory or zipped in the `ant-test/test-suite-results.zip` file and the *bad* test suite results are in the `ant-test/validation-results.xml` file.
3. Checkout the 'desc' project from the W3C WSDL 2.0 CVS repository to your local development environment (`:pserver:anonymous@dev.w3.org:/sources/public/2002/ws/desc`). Ensure the 'desc' project is in the same local directory as the woden project (e.g. `/workspace/woden` and `/workspace/desc`). The WSDL 2.0 test suite and test framework are located in `/desc/test-suite` directory.
4. Run the "copyResultsToW3C" target in Woden's `ant-test/build.xml` script. This will copy the `ant-test/test-suite-results.zip` and `ant-test/validation-results.xml` files from the woden project to the `test-suite/results/downloads/Woden` directory in the 'desc' project, then extract these files to the `test-suite/results/Woden` directory.
5. Run the targets "canonicalize-Woden", "evaluate-Woden", "Interchange.html" and "Validation.html" in the `test-suite/results/build.xml` script in the 'desc' project. These will prepare the Woden results for baseline comparison, do the baseline comparison, then generate the *good* and *bad* test suite reports.
6. View the reports in `test-suite/results/Interchange.html` and `test-suite/results/Validation.html` and check that no regressions have been introduced by the latest Woden build being tested.

Periodically the W3C WSDL 2.0 Interop Dashboard is republished, using the Woden result files in the `ant-test/` directory in the Woden repository. Therefore, these files should be committed periodically to reflect the up-to-date Woden results. (these `ant-test/` files include the `test-suite-results.zip` and `validation-results.xml` files mentioned above).

## 10. Release Process

The Woden release process involves many steps and checks. To keep compliant with Apache process requirements of WS and incubator projects it is important that it is followed.

1. Build and test the current Woden release candidate. The 'buildDist' ANT target will

create the binary and source archives (.zip, .tar.gz, .tar.bz2) and the hash digests (md5, sha1) for each archive file.

2. Sign the binary and source archives, which will create a .asc signature file for each archive file.

e.g.

```
gpg --armor --output apache-woden-incubating-1.0M7a.zip.asc
--detach-sig apache-woden-incubating-1.0M7a.zip
gpg --verify apache-woden-incubating-1.0M7a.zip.asc
apache-woden-incubating-1.0M7a.zip
```

3. Upload the binary and source archives and their hash digests and signature files to people.apache.org into some directory path under your public\_html directory so that you can include a link to the files in the [VOTE] request email. Also upload the KEYS file and release-notes.html from [woden root] and junit-noframes.html from the [woden root]/build/test-results/html directory. Make sure you chmod the file permissions so others can read them (e.g. 744).

e.g.

```
[ jkaputin
home ] /public_html/woden/milestones/1.0M7a-incubating
...is accessible at url...
```

<http://people.apache.org/~jkaputin/woden/milestones/1.0M7a-incubating/>

Note, because Woden is in incubation you must not upload these files to the Woden project on the file server until the Incubator PMC vote has passed....so you upload to your own space, then move to Woden space after voting.

4. Check that you can download/unzip the files.  
Create hash digests of the downloaded archives and check them against the downloaded hash files.

e.g.

```
$ dir
apache-woden-incubating-1.0M7a.zip
apache-woden-incubating-1.0M7a.zip.MD5
$ cat apache-woden-incubating-1.0M7a.zip.MD5
3009d6f6fea14b7536c22028944bb03a
$ md5sum apache-woden-incubating-1.0M7a.zip
3009d6f6fea14b7536c22028944bb03a
*apache-woden-incubating-1.0M7a.zip
```

5. Post a vote request email to woden-dev asking devs to vote on the proposed M7b release. Post the voting results.  
When posting a vote request to any mailing list, start the subject line with the eye-catcher [VOTE].
6. If woden-dev vote successful, post to general@ws.apache.org asking the WSPMC to

approve a Woden release. Post the voting results.

When posting a vote request to any mailing list, start the subject line with the eye-catcher [VOTE].

7. If WSPMC vote successful, post to IPMC at general@incubator. Be specific about timeframe (usually 3 days). Post the results afterwards. Success criteria is at least 3 binding IPMC votes (i.e. 3 x +1 from IPMC members). Remember, Dims, Sanjiva and Paul F are IPMC members as well as WSPMC.

When posting a vote request to any mailing list, start the subject line with the eye-catcher [VOTE].

8. If IPMC vote successful, move all the release files from your public\_html directory to the Woden file space on people.apache.org.

```
cd /www/people.apache.org/dist/ws/woden
cd milestones
```

Create a new directory for the release (e.g. /1.0M7a-incubating)

Move the release files to this new directory.

Copy the file release-notes.html to a new file called README.html in this new directory (this will ensure the release notes are displayed after the list of files, when this directory is accessed via the web).

e.g.

```
/www/people.apache.org/dist/ws/woden/milestones/1.0M7a-incubating
...will be accessible via url...
```

```
http://people.apache.org/dist/ws/woden/milestones/1.0M7a-incubating/
```

9. Once again, check that:

- the file permissions are set correctly
- you can download/unzip the files
- the downloaded hash digests are correct

10. Update the Woden web site (add the release download to the Builds page and add a News item announcing the release to the Woden home page).

11. Post a release announcement to woden-dev, general@ws and general@incubator.

12. Final step, which Axis2 folks will do, it to upload Woden release binary jar to a maven repository...I think Dims can upload to ws.zones.

Some example mailing list posts for reference:

[\[VOTE\] woden-dev and WSPMC](#)

[\[RESULT\]](#)

[\[VOTE\] IPMC](#)

[\[ANNOUNCE\]](#)

## 11. Connection with W3C WSDL Working Group

Apache Woden was started to answer the W3C WSDL working group's [call for implementations](#). As such, Woden has been in close communication with the WSDL working group providing feedback on the WSDL 2.0 specification, requesting clarification, and contributing test cases to the WSDL 2.0 test suite.

WSDL 2.0 has been promoted to recommendation status and the WSDL working group has completed its work and therefore closed on June 29, 2007. With the working group closed it is expected that a maintenance group will be set up. Woden will maintain close ties with the maintenance group as there may still be issues with the WSDL 2.0 specification and the test suite still requires work.

Any issues with the WSDL 2.0 specification or test suite should be reported via the WSDL mailing list [www-ws-desc@w3.org](mailto:www-ws-desc@w3.org) and issue tracking repository <http://www.w3.org/Bugs/Public/>.