Documentation Development Guide

A quick start template

Workgroup Notes

Revision 1.2 (November 16, 2018)

www.openpowerfoundation.org

Documentation Development Guide: A quick start template

System Software Work Group <syssw-chair@openpowerfoundation.org>
OpenPower Foundation

Revision 1.2 (November 16, 2018)

Abstract

The purpose of this document is to provide a guide for OpenPOWER documentation writers. As such, it provides directions, policies, references, and examples of the XML Docbook environment. It is intended to be used both in final product form (PDF and html) as a document and in source form as a template for new documents.

This document is a Non-standard Track, Work Group Note work product owned by the System Software Workgroup and handled in compliance with the requirements outlined in the *OpenPOWER Foundation Work Group (WG) Process* document. It was created using the *Document Development Guide* version 1.1.0. Comments, questions, etc. can be submitted to the public mailing list for this document at <syssw-doc_devel_guide@mailinglist.openpowerfoundation.org>. Additionally, the #doc-devel channel in the OpenPOWER Foundation Slack room (openpowerfoundation.slack.com) can be used to answer more interactive questions.

Table of Contents

List of Figures

List of Tables

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- *New text will appear like this.* Text marked in this way is completely new.
- ~~Deleted text will appear like this.~~ Text marked in this way was removed from the previous version and will not appear in the final, published document.
- Changed text will appear like this. Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [vii], references the Preface chapter on page vii.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
August 27, 2018	Version 1.2 additional updates: • Add a section on circumventing Java AWT exception. • Add information on key document tags which need update for new documents. • Extend information on modifying an existing document to include a step-by-step description of how to get started. • Rename the `template` directory to `doc_dev_guide`.
April 11, 2018	Version 1.2 updates: • Extend the Getting Started section to include a first document build. • Add a section on document packaging for publish in the Publishing OpenPOWER Documents section. • Add examples for background color in tables, and variablelists.
February 17, 2017	Version 1.1.0 updates: • Enhancements document creation to address project creation and update process. • Add "git" error to troubleshooting sections until JAR dependency removed. • Add optional font installation step to getting started. • Provide samples of how to access symbols by value, including extension for new symbol library. • Provide example usage of OPF Docbook extensions -- hard page breaks, soft line breaks, font-size changes, setting text color, and explicitly using symbol library. • Extend explanation of versioning policy. • Correct, improve miscellenous wording and grammar.
September 13, 2016	• Version 1.0.1: Minor updates to guide naming.
August 25, 2016	• Version 1.0.0: Document approval for publish
April 28, 2016	• Version 0.9.5: Removal of confidentiality and preview of change notations for final review by TSC.
March 21, 2016	• Version 0.9.4: Review version for TSC.
February 25, 2016	• Version 0.9.3: Technical and process updates. Addition of documentation lifecycle and git command cheat sheets.
February 25, 2016	• Version 0.9.2: Technical and process updates. Explanation of project structure.
January 25, 2016	• Version 0.9.1: Technical and process updates.
August 20, 2015	• Version 0.9: Draft for format review with TSC.
September 3, 2014	• Creation based on OpenStack documentation

1. Document development overview

The *OpenPOWER Foundation Documentation Development Guide* provides a framework for OpenPOWER public and private documentation. The goal of the document is to describe the documentation development process, to promote community contributions to OpenPOWER documenation and to enable new contributions with a common look and feel.

The major sections of this document addresses the following topics:

- Getting started [1]: This section details tools and commands used to contribute to OpenPOWER documentation. All users who are new to OpenPOWER Foundation documentation should start here.
- Understanding the project structure [5]: This section provides detailed descriptions of the various project components and their roles in the documentation creation process.
- Creating a new document [7]: This section provides step-by-step instructions on how to create a new document from scratch. Use this section to start a new document.
- Modifying an existing document [14]: This section highlights common steps in editing an existing OpenPOWER Foundation document. Use this section as a guideline for contributing to an existing document.
- Debugging build failures [16]: This section provides examples of the two most common types of build failures and helps users find the relevant failure information.
- Publishing OpenPOWER Documents [21]: This section explains key document types and the appropriate work flow for publishing OpenPOWER Foundation documents.
- Policies and conventions [35]: This section contains the generally accepted guidelines for creating OpenPOWER documentation. Use this section as a reference for documentation source contruction and community process.
- Frequently asked questions [36]: This section answers common questions. Use this section when the other sections do not answer your questions.
- Common git commands [36]: This section contains examples of commonly used git commands. Reference this section to find information on a specific git operation.
- Finding more information [39]: This section provides pointers to on-line information about XML, Docbook, Maven, and other relevant references.

In addition to OpenPOWER Foundation specific topics, Chapter 2, “Documentation examples” [41] provides examples of common documenation constructs in XML.

1.1. Getting started

To begin contributing to the OpenPOWER Foundation documentation, the following steps must be completed:

1. Installing tools [2]

2. Installing fonts [3]

3. Creating accounts [3]

4. Cloning master document information [3]

5. Building the first document [4]

Once complete, you can proceed to either Section 1.3, “Creating a new document” [7] or Section 1.4, “Modifying an existing document” [14] as needed.

1.1.1. Installing tools

Only two tools are required to update documentation, git and maven. Git manages the documentation source and maven provides the build framework to create the published content in PDF and html form. Installation steps for these tools varies by operating system.

On Debian-based Linux operating systems (Ubuntu and Debian), install maven and git as follows:

```
# apt-get install git
# apt-get install maven
```

On RPM-based Linux operating systems (Fedora, RHEL, openSUSE, SLES), install maven and git as follows:

```
# yum install git
# yum install maven
```

On Mac OS X, use Macports to install maven and git as follows:

```
# port install git
# port install maven3
```

or use Homebrew to install maven and git as follows:

```
$ brew install git
$ brew install maven
```

For information on how to setup the environment on Windows, see the following websites:

- git for Windows - http://msysgit.github.io/
- Maven on Windows - http://maven.apache.org/guides/getting-started/windows-prerequisites.html

Note

Modification of documentation source files requires a text editor. While standard editors like vim, emacs, or gedit can be used, it is highly recommended that an editor be used which highlights XML or Docbook syntax. If your favorite editor does not include an extension or plugin to accomplish this, you might consider using Bluefish to edit your docbook files. Details on this editor can be found at http://bluefish.openoffice.nl/index.html.

1.1.2. Installing fonts

The OpenPOWER Foundation documentation utilizes opensource fonts known as the **Chrome OS core fonts** or **Croscore fonts**. The three TrueType fonts (TTFs) in this family Arimo (sans-serif), Tinos (serif), and Cousine (monospace). While not strictly required to have these fonts on your system, it can be helpful when designing graphics and other images to have them installed on your development system.

Only two tools are required to update documentation, git and maven. Git manages the documentation source and maven provides the build framework to create the published content in PDF and html form. Installation steps for these tools varies by operating system.

On Debian-based Linux operating systems (Ubuntu and Debian), install Croscore fonts as follows:

```
# apt-get install fonts-croscore
```

On RPM-based Linux operating systems (Fedora, RHEL, openSUSE, SLES), install Croscore fonts as follows:

```
# yum install google-croscore-fonts
```

On Mac OS X and Windows systems, use a font website to download and install the Croscore fonts individually. Most of these sites provide directions for Mac OS and Windows.

1.1.3. Creating accounts

All OpenPOWER project documentation is maintained in GitHub trees, public and private. The first step to creating documentation will be joining the GitHub community.

To join the GitHub community, apply at https://github.com/join.

The OpenPOWER Foundation documentation trees are grouped in the OpenPOWER Foundation project at https://github.com/OpenPOWERFoundation. Everyone should be able to see and access public trees like Docs-Master. However, if you will be participating in private OpenPOWER Foundation trees, you will need to request access from the Technical Steering Committee Chair, `<tsc-chair@openpowerfoundation.org>`.

To learn more about using git, see the online article in GitHub Help, "Good Resources for Learning Git and GitHub." at https://help.github.com/articles/good-resources-for-learning-git-and-github/.

1.1.4. Cloning master document information

To successfully build OpenPOWER Foundation documents, common document files must be in place in addition to the specific document files. These common files are obtained by cloning the OpenPOWER Foundation public project `Docs-Master`.

To clone the OpenPOWER Foundation master document framework, use the clone git command:

```
$ git clone https://github.com/OpenPOWERFoundation/Docs-Master.git
Cloning into 'Docs-Master'...
remote: Counting objects: 24, done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 24 (delta 6), reused 20 (delta 5), pack-reused 0
```

```
Unpacking objects: 100% (24/24), done.
Checking connectivity... done.
$
```

More information can be found about the Docs-Master project online at https://github.com/OpenPOWERFoundation/Docs-Master. Additional details about the OpenPOWER Foundation documentation structure are explained in Section 1.2, “Understanding the project structure” [5] of this document.

1.1.5. Building the first document

The final step of setting up your environment to perform the first build. The following steps are recommended:

1. Clone the *Documentation Development Guide* (this document) as source from which to build. To accomplish this, issue the following command in the same directory as as the master document clone from Section 1.1.4, “Cloning master document information” [3].

   ```
   $ git clone https://github.com/OpenPOWERFoundation/Docs-Template.git
   Cloning into 'Docs-Template'...
   remote: Counting objects: 253, done.
   remote: Total 253 (delta 0), reused 0 (delta 0), pack-reused 253
   Receiving objects: 100% (253/253), 468.94 KiB | 0 bytes/s, done.
   Resolving deltas: 100% (151/151), done.
   Checking connectivity... done.
   $
   ```

2. Change the working directory into the source directory for the *Documentation Development Guide*.

   ```
   $ cd Docs-Template/doc_dev_guide
   Docs-Template/doc_dev_guide$
   ```

3. Build the document in Maven.

   ```
   Docs-Template/doc_dev_guide$ mvn generate-sources
   [INFO] Scanning for projects...
   [INFO]
   [INFO] ------------------------------------------------------------------------
   [INFO] Building Documentation Development Guide 1.0.0-SNAPSHOT
   [INFO] ------------------------------------------------------------------------
   [INFO]
   [INFO] --- openpowerdocs-maven-plugin:1.1.0:generate-webhelp (generate-webhelp) @
    openpower-template-guide ---
   [INFO] Processing input file: bk_main.xml
   [WARNING] Property not found in com.agilejava.docbkx.maven.DocbkxWebhelpMojo
   [WARNING] Property not found in com.agilejava.docbkx.maven.DocbkxWebhelpMojo
   [WARNING] Property not found in com.agilejava.docbkx.maven.DocbkxWebhelpMojo
   [WARNING] Property not found in com.agilejava.docbkx.maven.DocbkxWebhelpMojo
   Feb 27, 2018 11:43:28 AM org.apache.fop.apps.FopFactoryConfigurator configure
   INFO: Default page-height set to: 11in
   Feb 27, 2018 11:43:28 AM org.apache.fop.apps.FopFactoryConfigurator configure
    ...snip...
   [INFO] Applying customization parameters

   <!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
   xhtml1-transitional.dtd">
   ```

```
Parsing: /home/scheel/mydocs/Docs-Template/doc_dev_guide/target/docbkx/webhelp/doc-
devel-guide/content/section_cloning_project.html
 ...snip...
The created index files are located in /home/scheel/mydocs/Docs-Template/
doc_dev_guide/target/docbkx/webhelp/doc-devel-guide/content/search/.js
[INFO] See /home/scheel/mydocs/Docs-Template/doc_dev_guide/target/docbkx/webhelp/
bk_main for generated file(s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 31.547 s
[INFO] Finished at: 2018-02-27T11:43:45-06:00
[INFO] Final Memory: 83M/729M
[INFO] ------------------------------------------------------------------------
Docs-Template/doc_dev_guide$
```

Note

The first time one builds in a Maven environment, the build time will be noticeably long due to JAR file downloads associated with the new Maven project types. In future builds, these JAR files will only be downloaded when they are updated. As such, one should both allow for this extra time and not be discouraged by the duration of the first build.

Once complete, there should be a single directory in the `target/docbkx/webhelp/` directory. For the Docs-Template project, that directory is `doc-devel-guide`. Inside this directory will be both the PDF file and the `index.html` file for the HTML document.

To verify this for the *Documentation Development Guide*, perform these commands:

```
Docs-Template/doc_dev_guide$ cd target/docbkx/webhelp/
Docs-Template/doc_dev_guide/target/docbkx/webhelp$ ls
doc-devel-guide
Docs-Template/doc_dev_guide/target/docbkx/webhelp$ cd doc-devel-guide
Docs-Template/doc_dev_guide/target/docbkx/webhelp/doc-devel-guide$ ls
bookinfo.xml  common  content  doc-devel-guide-20180227.pdf  favicon.ico  index.html
webapp
```

Now, you are ready to begin working on your own document. Useful information on how to proceed can be found in Section 1.3, “Creating a new document” [7] and Section 1.4, “Modifying an existing document” [14].

1.2. Understanding the project structure

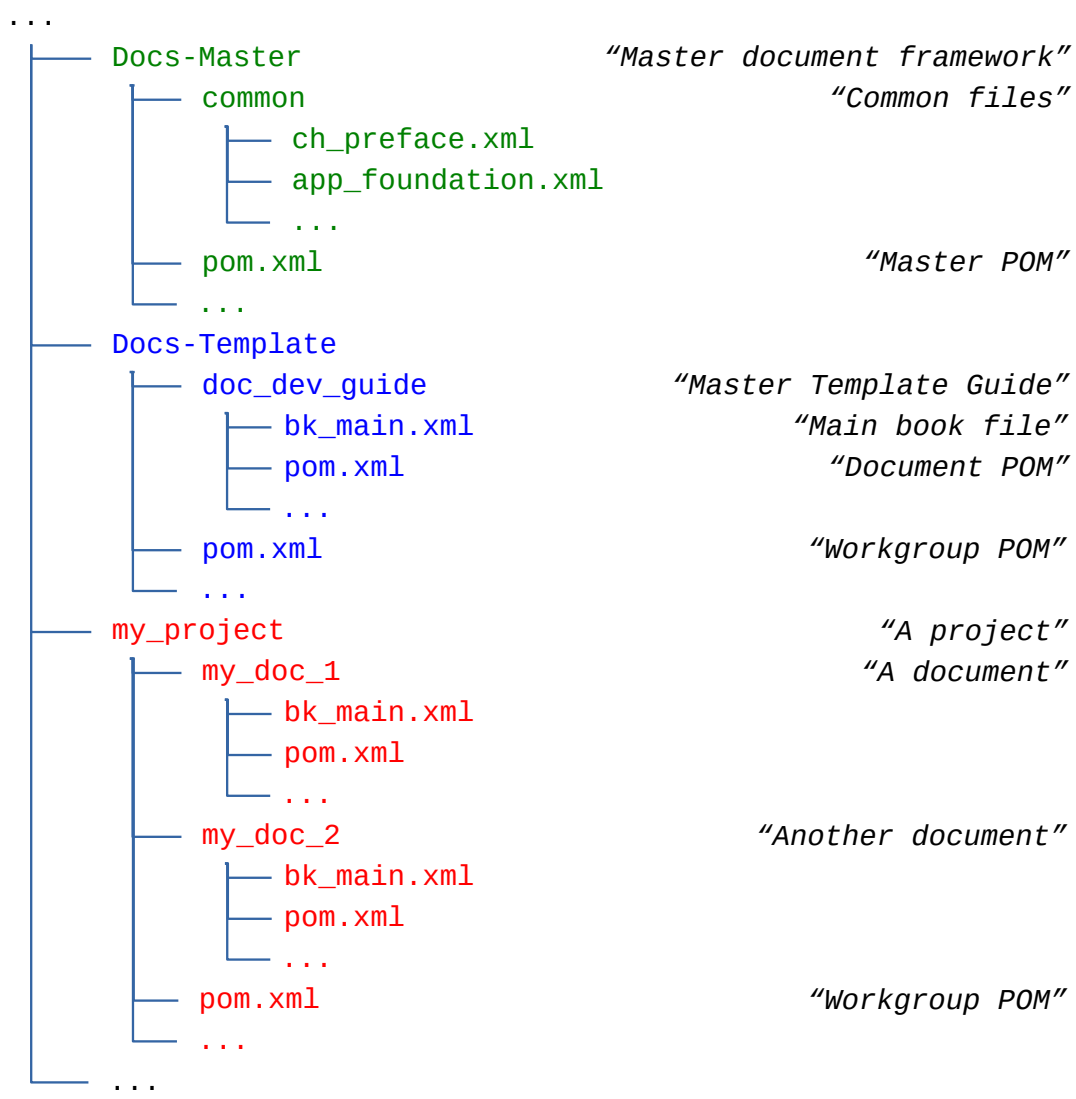
The OpenPOWER Foundation documentation build process involves dependency on a common framework and shared files. As such, a deeper explanation about the relationships of key projects and their components may be helpful to prevent and diagnose documentation build problems. This section provides a pictorial layout of key files and explains their roles and relationships.

As mentioned multiple times throughout this guide, successful build of any OpenPOWER Foundation document requires two things:

1. The cloning of the `Docs-Master` project.
2. The cloning of the specific documentation project into the same parent directory as the `Docs-Master` project.

To begin to understand why, let us use a picture. The following graphic illustrates the directory structure of three projects: two existing OpenPOWER Foundation GitHub projects, `Docs-Master` and `Docs-Template`, and a hypothetical new project named `my_project`.

Figure 1.1. Directory structure and key files in the OpenPOWER Foundation Docbook projects



To create this structure, one would clone the `Docs-Master` project to get the `Docs-Master` directory and all its contents (shown above in green), clone the `Docs-Template` project to get the `Docs-Template` directory and all its contents (shown in blue), and clone `my_project` project to get the `my_project` directory and all its contents (shown in red).

Among these projects, the most important directory and project is `Docs-Master`. Without this project and associated directory, no document will build. As depicted in the above figure, the `Docs-`

`Master` directory must sit at a level equal to all other project directories. Details on how to install this project can be found in the Section 1.1.4, “Cloning master document information” [3] section.

Inside the `Docs-Master`project directory, the two most important pieces are a `commmon` directory and a `pom.xml` file. The directory contains common files used by all projects such as the common preface (`ch_preface.xml`) and the common appendix (`app_foundation.xml`). The `pom.xml` file in this directory serves as the "Master POM" (POM stands for Program Object Model and serves as the main configuration file) for all builds. This file references the OpenPOWER Maven Plugin JAR (found in the OpenPOWER Foundation Repository at http://openpowerfoundation.org/repo.openpowerfoundation.org/) used to control the OpenPOWER Foundation document builds where all other dependencies, supporting tools, and document build rules are defined.

The `Docs-Template` project and directory are depicted in blue in the above figure. The top level of the project `Docs-Template` must be cloned into the same parent directory as the `Docs-Master` for Maven builds to complete successfully. At the top level of the `Docs-Template` project are a `pom.xml` referred to as the "Workgroup POM" and a single document directory (`template`). The "Workgroup POM" is a minimal POM file that locates the parent, "Master POM" in the `Docs-Master`project directory with a `<relativePath>` definition of `../Docs-Master/pom.xml`. The document directory contains the unique files used to create the document. The two most important files in the `Docs-Template/doc_dev_guide` directory (and in any project) are the `pom.xml` or "Document POM" which describes how to build the document and which points to the main document file, the `bk_main.xml` file. This book file contains all the Docbook source, directly or through include statements (`<xi:include href="..."`), to build the document.

For completeness of understanding, a hypothetical project `my_project` is also depicted in red. Like all OpenPOWER Foundation projects, it is cloned at the correct level, equal to `Docs-Master`. Like the `Docs-Template` project, it has a "Workgroup POM" which will differ only in the `<modules>` section where it will describe two document projects, `my_doc_1` and `my_doc_2`. But, each document directory has similar contents to `Docs-Template/template` -- a "Document POM" (`pom.xml`) and a "Main book file" (`bk_main.xml`).

1.3. Creating a new document

Creating a new document from scratch follows four simple steps:

1. Cloning a project [7]
2. Finding a document framework [9]
3. Modifying core project files [11]
4. Adding new content [13]

Before undertaking one of these activities, it may be helpful to read the Understanding the project structure [5] section to learn the basics about the documentation project structure.

1.3.1. Cloning a project

All documentation projects reside in a Git project directory, either locally or in the cloud at GitHub. As described in Section 1.1.4, “Cloning master document information” [3], your document project directory must reside locally in the same directory as the `Docs-Master` framework.

To clone a project in which to work, select from one of the two approaches below:

- Cloning an existing project [8]
- Creating a new project locally [8]

Complete the project cloning and then continue with the next step in Creating a new document [7].

1.3.1.1. Cloning an existing project

To work in an existing OpenPOWER Foundation project like the *Documentation Development Guide* (`Docs-Template`), use the following command in the same directory that contains `Docs-Master`:

```
$ git clone https://github.com/OpenPOWERFoundation/Docs-Template.git
Cloning into 'Docs-Template'...
remote: Counting objects: 163, done.
remote: Total 163 (delta 0), reused 0 (delta 0), pack-reused 163
Receiving objects: 100% (163/163), 275.60 KiB | 494.00 KiB/s, done.
Resolving deltas: 100% (96/96), done.
Checking connectivity... done.
$
```

The results should look roughly something like above with actual numbers of objects, files, etc. varying for different projects.

Note

Private projects prompt for a GitHub userid and and password immediately following the "Cloning into..." message. When cloning public projects such as `Docs-Template`, these prompts are skipped.

A list of additional OpenPOWER Foundation projects can be found at https://github.com/OpenPOWERFoundation/. To work on an existing project, note its name it the list and apply the above steps replacing `Docs-Template` with your preferred project from the list.

Note

If you do not see the project for which you are looking, you may not be authorized to it. See Section 1.1.3, “Creating accounts” [3] for details about joining the OpenPOWER Foundation private projects. If you feel that you need a new GitHub project, work with the Technical Steering Committee Chair, `<tsc-chair@openpowerfoundation.org>`, to request and get this setup.

The existing project should now be cloned. Continue with the next step in Creating a new document [7].

1.3.1.2. Creating a new project locally

To create a new project locally, the simplest way is to clone the *Documentation Development Guide* (`Docs-Template`) into a new project. In our directions, `my_project` will be our new project name. Use the following command in the same directory that contains `Docs-Master`:

```
$ git clone https://github.com/OpenPOWERFoundation/Docs-Template.git my_project
Cloning into 'my_project'...
remote: Counting objects: 163, done.
remote: Total 163 (delta 0), reused 0 (delta 0), pack-reused 163
```

```
Receiving objects: 100% (163/163), 275.60 KiB | 494.00 KiB/s, done.
Resolving deltas: 100% (96/96), done.
Checking connectivity... done.
$
```

The results should look roughly something like above with actual numbers of objects, files, etc. varying for different projects.

The new project should now be generally setup. Continue with the next step in Creating a new document [7].

1.3.2. Finding a document framework

When creating a new document, the simplest way to start is to use an existing document. This ensures that you have a basic document structure and allows you to start with a working document from which to make changes. Select from one of the following scenarios for detailed directions on creating your document framework:

- If your project exits on GitHub in the OpenPOWER Foundation area and it contains a `doc_template` directory, then follow the directions in Moving the template document into your new document framework [9] to use this document as a base.
- If you have an existing document in your project that you want to use as a base for your new document, then follow the directions in Copying an existing document as a new document framework [10] to establish your base document.
- Otherwise, the instructions in Copying the Document Development Guide as a new document framework [10] will clone and copy this document as a base.

1.3.2.1. Moving the template document into your new document framework

If this is your first document, in a brand new OpenPOWER Foundation project (on GitHub), you have the fewest number of steps to perform because your project should have been primed with a single project based on `Docs-Template`. You can verify this by inspecting the files in your project directory. A new project will contain a `doc_template` directory, a `pom.xml` file, a `LICENSE` file, and a `README.md` file. If this is the case, you simply need to perform the following three steps:

1. Navigate down to your project directory, called `my_project` for this example. This can be achieved using the `cd` command:

   ```
   $ cd ~/my_project
   $
   ```

   This directory should contain the `doc_template` folder used to prime the project.

2. Rename the `doc_template` document directory to something new like `my_doc`. To accomplish this, use the `mv` command::

   ```
   $ mv doc_template/ my_doc
   ```

3. Change the project name in the Workgroup POM file (`my_project/pom.xml`). Using your editor, change this line between the <modules> and the </modules> tags near the top of the file:

   ```
   <module>template</module>
   ```

to read like this:

```
<module>my_doc</module>
```

Your new document frameword has been copied from the *Document Development Guide*. Continue with the next step in Creating a new document [7].

1.3.2.2. Copying an existing document as a new document framework

If you have another document within your project that would serve as a good base for your new one, you can copy the existing document as the source for your new document. Follow these steps:

1. Navigate down to your project directory, called `my_project` for this example. This can be achieved using the `cd` command:

   ```
   $ cd ~/my_project
   $
   ```

   This directory should contain the folder name of the document wishing to be copied, called `source_doc` for clarity in these directions.

2. To create a new document directory, simply create a new directory and copy the contents of the `source_doc` directory. If creating a new directory named `my_doc` via a command line, the command sequence would look like this:

   ```
   $ mkdir my_doc
   $ cp -r source_doc/*.* my_doc
   $
   ```

3. Add the new project to the Workgroup POM file (`my_project/pom.xml`). Using your editor, add the following lines between the <modules> and the </modules> tags near the top of the file:

   ```
   <module>my_doc</module>
   ```

Note

Before committing the project back to git, you will need to add the new directory to the git repository. This can be performed using the `git add my_doc/` command on the whole directory.

You are now ready to begin making updates to your new document. Continue with the next step in Creating a new document [7].

1.3.2.3. Copying the Document Development Guide as a new document framework

Instead of copying an existing document, you may want to start with the *Document Development Guide* (`Doces-Template`) source. The steps to do this are similar to those above, but with a few more commands. The following commands will create a new document based on this guide:

1. Navigate down to your project directory, called `my_project` for this example. This can be achieved using the `cd` command:

   ```
   $ cd ~/my_project
   ```

```
$
```

This directory should contain any existing document folders along with at least a `pom.xml` file, a `LICENSE` file, and a `README.md` file.

2. Clone the the *Documentation Development Guide* (`Docs-Template`) project into your working directory with this command:

```
$ git clone https://github.com/OpenPOWERFoundation/Docs-Template.git
Cloning into 'Docs-Template'...
remote: Counting objects: 163, done.
remote: Total 163 (delta 0), reused 0 (delta 0), pack-reused 163
Receiving objects: 100% (163/163), 275.60 KiB | 0 bytes/s, done.
Resolving deltas: 100% (96/96), done.
Checking connectivity... done.
$
```

3. To create a new project directory, simply create a new directory and copy the contents of the `Docs-Template/doc_dev_guide` directory. If creating a new project named `my_doc` via a command line, the command sequence would look like this:

```
$ mkdir my_doc
$ cp -r Docs-Template/doc_dev_guide/* my_doc
$
```

4. Once copied, the Docs-Template directory and all its contents should be removed from your project so that it does not accidentally get included in your project. The command **`rm -rf Docs-Template`**

5. Finally, add the new project to the Workgroup POM file (`my_project/pom.xml`). Using your editor, add the following lines between the <modules> and the </modules> tags near the top of the file:

```
<module>my_doc</module>
```

Note

Before committing the project back to git, you will need to add the new directory to the git repository. This can be performed using the `git add my_doc/` command on the whole directory.

You are now ready to begin making updates to your new document. Continue with the next step in Creating a new document [7].

1.3.3. Modifying core project files

The first step to customizing a new project is to modify two core project files--`pom.xml` and `bk_main.xml`. Within these two files are XML comment tags that begin "`<!-- TODO:`" to identify places which need customization. The surrounding comments will provide guidance on what needs to change and how it may be changed. Simply work through each item, making updates as requested.

In the `pom.xml` file, pick your settings for document work product type (`<workProduct>`, work flow status (`<documentStatus>`), and security (`<security>`) carefully. Section 1.6, “Publish-

ing OpenPOWER Documents" [21] provides an overview of the process and details the various settings needed in the document core project files. If you still have questions after reading this section, consult with your Technical Steering Committee representative.

Note

In addition to the document settings, be sure to remember two key values you used in the `pom.xml` file, `<webhelpDirname>` and `<pdfFilenameBase>`, as these will be used to locate your generated document.

In the `book.xml` file, you will find the following document unique values which you most likely want to change:

`<title>` The main title of the document. This appears in the largest font at the top of the title page.

`<subtitle>` The second title of the document. This title appears in a smaller font below the `<title>` on the title page.

`<realeaseinfo>` The document version value. This value should take the form of "Revision V.R.M" as described in Section 1.7, "Policies and conventions" [35] recommendation 2.

When ready, build your new document using standard maven commands like this:

```
$ cd my_project/my_doc
$ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenPOWER Template Guide 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ openpower-template-guide ---
[INFO] Deleting ~/my_doc/my_proj/target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.353s
[INFO] Finished at: Wed Feb 25 12:54:47 CST 2015
[INFO] Final Memory: 3M/7M
[INFO] ------------------------------------------------------------------------
$ mvn generate-sources
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenPOWER Template Guide 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- openpowerdocs-maven-plugin:1.0.0:generate-webhelp (generate-webhelp) @
 openpower-template-guide ---
[INFO] Processing input file: bk_main.xml
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 20.361s
```

```
[INFO] Finished at: Wed Feb 25 12:55:15 CDT 2015
[INFO] Final Memory: 30M/390M
[INFO] ------------------------------------------------------------------------
$
```

If all goes well, the new generated pdf should be available in `target/docbkx/webhelp/<webhelpDirname>/<pdfFilenameBase>.pdf`.

For assistance correcting commmon build failures, see Section 1.5, “Debugging build failures” [16].

Note

The permutations of Maven invocations may be combined into one operation where the parameters are specified in the order in which one wishes to execute them. Thus, the command `mvn clean generate-sources` would accomplish the same thing as the above sequence of commands.

You have completed updates to core project files for your new document. Continue with the next step in Creating a new document [7].

1.3.4. Adding new content

The starting point for book content is the `bk_main.xml` file (or whatever to which it was renamed in the previous step). Removal and additions of the main chapter files will be controlled by entries near the end of that file which appear as follows:

```
 <!-- The ch_preface.xml file is required by all documents -->
 <xi:include href="../../Docs-Master/common/ch_preface.xml"/>

 <!-- TODO: Add your chapter heading files here.  Remove both files and insert your
own. -->
 <!--       See the template document for naming conventions and location of files.
  -->
 <xi:include href="ch_template_overview.xml"/>
 <xi:include href="ch_example.xml"/>

 <!-- The app_foundation.xml appendix file is required by all documents. -->
 <xi:include href="../../Docs-Master/common/app_foundation.xml"/>

 <!-- TODO: The following template document may be modified to create additional
appendices as needed. -->
 <xi:include href="app_template.xml"/>
```

Copying and modifying existing files from the template or other documents is a great way to get started. When creating whole new chapter or appendix files from scratch, the `ch_example.xml` and `app_template.xml` files may serve as excellent starting points. For XML examples of various document structures, please see Chapter 2, “Documentation examples” [41] and its supporting source files in this document. Online resources such as those listed in Section 1.10, “Finding more information” [39] may also be helpful.

Note

When creating new files for the project, remember to use the `git add <file name>` command to add new files to the git tree.

1.4. Modifying an existing document

To begin editing an existing document, you may need to clone up to two projects -- the specific document project, and if not already cloned, the master document framework project too. If needed, clone the master document as described in Section 1.1.4, “Cloning master document information” [3].

To obtain a copy of the desired document source, clone its project. For example, to clone this document, *Documentation Development Guide*, from the public OpenPOWER Foundation git repository, use this command:

```
$ git clone https://github.com/OpenPOWERFoundation/Docs-Template.git
Cloning into 'Docs-Template'...
Username for 'https://github.com': my_userid
Password for 'https://my_userid@github.com': my_password
remote: Counting objects: 62, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 62 (delta 2), reused 0 (delta 0), pack-reused 52
Unpacking objects: 100% (62/62), done.
Checking connectivity... done.
$
```

To build a specific document such as this guide, follow these steps from the directory where you just cloned:

```
$ cd Docs-Template/doc_dev_guide
$ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenPOWER Template Guide 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ openpower-template-guide ---
[INFO] Deleting ~/Docs-Template/doc_dev_guide/target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.353s
[INFO] Finished at: Wed Feb 25 12:54:47 CST 2015
[INFO] Final Memory: 3M/7M
[INFO] ------------------------------------------------------------------------
$ mvn generate-sources
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building OpenPOWER Template Guide 1.0.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- openpowerdocs-maven-plugin:1.0.0:generate-webhelp (generate-webhelp) @
 openpower-template-guide ---
[INFO] Processing input file: bk_main.xml
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 20.361s
[INFO] Finished at: Wed Feb 25 12:55:15 CDT 2015
```

```
[INFO] Final Memory: 30M/390M
[INFO] ------------------------------------------------------------------------
$
```

Note

The permutations of Maven invocations may be combined into one operation where the parameters are specified in the order in which one wishes to execute them. Thus, the command `mvn clean generate-sources` would accomplish the same thing as the above sequence of commands.

If all goes well, the generated pdf should be available in `~/Docs-Template/doc_dev_guide/target/docbkx/webhelp/template-guide/`.

For assistance correcting commmon build failures, see Section 1.5, “Debugging build failures” [16].

Note

Projects may contain multiple documents. While specific documents can be built by executing a `mvn clean generate-sources` in the specific document directory, executing this command in the base project directory will build all projects identified in the `<module>` list in the top-level `pom.xml` file, known as the "Workgroup POM".

Before diving deeply into text updates, you should consider the following items for your project and document:

- Ensure that the previous version of the tree is tagged.

  The command `git tag` may be used to see existing tree tags and set new ones. See Section 1.9, “Common git commands” [36] for more specifics on `git tag` commands.

- Reset the document status.

  The `pom.xml` file contains the `<documentStatus>` field which generally needs to be reset to the `draft` value. In addition, for non-public work groups, the `<security>` field should be returned to `workgroupConfidential` or `foundationConfidential` values during the document update process. More information on document development process can be found in Section 1.6, “Publishing OpenPOWER Documents” [21]. Detailed information on key document settings can be found in Section 1.6.1, “Understanding document marking variables in the pom.xml file” [24] and Section 1.6.2, “Navigating the OpenPOWER Foundation documentation publishing process” [25].

- Incement the new document release information.

  The `bk_main.xml` file contains the `<releaseinfo>` field which contains the Versions, Release, and Modification values. Typically, new documents when first being edited will increment the correct value, reset sub-values to zero, and append a "_preN" tag. During the development process, you will likely increment the "N-value" in your "pre" release information. Then, at publish, you can remove the "_preN" suffix.

  More details on the release information can be found in Section 1.7, “Policies and conventions” [35] recommendation 2.

- Create a new entry in the revision history.

  The `bk_main.xml` file contains the revision history in `<revhistory>` table. To start a new entry, add a new `<revision>` entry with `<date>` and `<revdescription>` fields at the top of the list of revisions.

You are now ready to make textual updates.

1.5. Debugging build failures

Maven/docbkx failures generally fall into these categories:

- Project structure errors [16]
- Docbook validation errors [17]
- Build failures [17]
- FO validation failures [18]

Correcting the document errors starts with understanding which type of failure has occurred and understanding where to look in your document source.

1.5.1. Project structure errors

Because the OpenPOWER Foundation documentation projects are not self-contained in the GitHub repositories, forgetting to clone the `Docs-Master` project in addition to the document project or cloning it in the wrong location is a common problem. Failures of this kind produce the following error:

```
...
[INFO] Scanning for projects...
[ERROR] The build could not read 1 project -> [Help 1]
[ERROR]
[ERROR]   The project org.openpowerfoundation.docs:workgroup-pom:1.0.0-SNAPSHOT (/home/
scheel/Docs-Template/pom.xml) has 1 error
[ERROR]     Non-resolvable parent POM: Could not find artifact org.openpowerfoundation.
docs:master-pom:pom:1.0.0-SNAPSHOT and 'parent.relativePath' points at wrong local POM
 @ line 6, column 11 -> [Help 2]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the
 following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/
ProjectBuildingException
[ERROR] [Help 2] http://cwiki.apache.org/confluence/display/MAVEN/
UnresolvableModelException
...
```

The identifying characteristic of this error type is the message, "Non-resolvable parent POM". This occurs because the `pom.xml` file in the documentation project, called the "workgroup-pom" because of a project `<artifactId>workgroup-pom</artifactId>` declaration, expects its parent

pom file to be in the location defined by the `<relativePath>../Docs-Master/pom.xml</relativePath>`, up one directory and then in the `Docs-Master` director.

So, if you see the message "Non-resolvable parent POM", ensure that the `Docs-Master` project and your project have been cloned into the same parent directory. See Section 1.1.4, “Cloning master document information” [3] for detailed directions on how to do this.

1.5.2. Docbook validation errors

Validation errors are generally indicated in the build output with text like the following:

```
...
@@@@@@@@@@@@@@@@@@@@@@
!!!VALIDATION ERRORS!!
!!!!!!!!!!!!!!!!!!!!

Note: Open the temporary file:

file:/home/user1/Docs-Template/doc_dev_guide/target//bk_main.xml-invalid.xml

to see all the errors in context.
You must correct the errors in the original
source DocBook or wadl files however.

You can control whether build fails or not by
setting failOnValidationError to no in your pom.

lineNumber:  272; columnNumber: 70; text not allowed here; expected element "address",
 ...
```

This error message contains three key pieces of information:

1. The full path and filename that contains the context for the failure. In the message above, this is `/home/user1/Docs-Template/doc_dev_guide/target//bk_main.xml-invalid.xml`.

2. The location within the file of the syntax error. For the above example, the key information is "`lineNumber: 272; columnNumber: 70`.

Note

In some XML validation failure scenarios, the `lineNumber` or `colNumber` values are not specified or are `-1`. If you encounter such a situation, please post to the Documentation Development mailing list at `<syssw-doc_devel_guide@mailinglist.openpowerfoundation.org>` so they can assist in identifying the exact location of the failure.

3. An explanation of the failure. This information in the above error reads, "text not allowed here; expected element "address", ...".

1.5.3. Build failures

Build errors are easily identified as well. Below is the most common example:

```
...
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.827s
[INFO] Finished at: Wed Jul 29 14:55:33 CDT 2015
[INFO] Final Memory: 17M/171M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.openpowerfoundation.docs:openpowerdocs-
maven-plugin:1.0.0:generate-webhelp (generate-webhelp) on project openpower-
template-guide: Execution generate-webhelp of goal org.openpowerfoundation.
docs:openpowerdocs-maven-plugin:1.0.0:generate-webhelp failed: XInclude resource error
 (sec_template_new_document.xml) and no fallback provided. XProc error err:XD0011:
 org.xml.sax.SAXParseException; systemId: file:/home/user1/openpower-foundation-
docbkx-framework/doc/doc_dev_guide/sec_template_new_document.xml; lineNumber: 55;
 columnNumber: 17; The element type "para" must be terminated by the matching end-tag
 "</para>". -> [Help 1]
...
```

Like validation errors, three key pieces of information are again provided:

1. The full path and filename of our failure is `/home/user1/Docs-Template/doc_dev_guide/sec_template_new_document.xml`.

2. The location within the file of the error is "`lineNumber: 55; columnNumber: 17`.

3. An explanation of the failure begins with the text, "The element type "para" must be terminated by the matching end-tag "</para>."

With these details in hand for either error, one simply locates the offending syntax and makes the appropriate correction. Online resources such as those listed in Section 1.10, “Finding more information” [39] may be helpful.

When creating new documentation projects, you may encounter the following error during your first build:

```
...
[ERROR] Failed to execute goal org.openpowerfoundation.docs:openpowerdocs-maven-
plugin:1.0.5:generate-webhelp (generate-webhelp) on project openpower-vector-
programming-guide: Execution generate-webhelp of goal org.openpowerfoundation.
docs:openpowerdocs-maven-plugin:1.0.5:generate-webhelp failed: One of setGitDir or
 setWorkTree must be called. -> [Help 1]
...
```

This error results from interactions of the maven build process and git. It may be circumvented by issuing the **`git init`** command in your directory.

1.5.4. FO validation failures

FO (formatting objects) validation failures are a slight bit more difficult to identify and require more effort to correct. A sample appears as follows:

```
...
Error
  SXCH0003: org.apache.fop.fo.ValidationException:
  "{http://www.w3.org/1999/XSL/Format}block" is not a valid child of "fo:list-block"!
  (See position 70:-1): null:70:-1: "{http://www.w3.org/1999/XSL/Format}block" is not
```

```
  a valid   child of "fo:list-block"! (See position 70:-1)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 35.900s
[INFO] Finished at: Sat Mar 19 15:54:34 CDT 2016
[INFO] Final Memory: 107M/256M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.openpowerfoundation.docs:openpowerdocs-maven-
plugin:1.0.3:generate-webhelp (generate-webhelp) on project hwarch-caia-spec: Failed
 to transform to PDF: org.apache.fop.fo.ValidationException: "{http://www.w3.org/
1999/XSL/Format}block" is not a valid child of "fo:list-block"! (See position 70:-1):
 null:70:-1: "{http://www.w3.org/1999/XSL/Format}block" is not a valid child of
 "fo:list-block"! (See position 70:-1) -> [Help 1]
...
```

The "`org.apache.fop.fo.ValidationException`" text indicates that this error was during FO validation. The key pieces of information are as follows:

1. The error type is indicated in the text following the exception indictor. In our case, the error statement is: `"{http://www.w3.org/1999/XSL/Format}block" is not a valid child of "fo:list-block"!`. This error clearly has something to do with the nesting of a "fo:block" statement in a "fo:list-block" statement.

2. The location of the validation error is given in the statement "`See position 70:-1`". These two values are the line number and character number of the error. So, our sample error occurs on line 70, but the character number of `-1` is an indication that the line is too long to effectively point.

What this information fails to detail is which file has the problem. To find the particular offending file, one must understand the Docbook build process. This process begins by collecting all XML into a working copy of the main book file. The build failure error in Section 1.5.2, “Docbook validation errors” [17] includes a reference to this file which will be found in the `.../target/` directory. It generally has the same name as the main book file of the document, which if copied from the *Documentation Development Guide* project, will be `bk_main.xml`. When in doubt about this file name, you will find it in the `<includes>` tag in the `pom.xml` file.

Once all information has been pulled into the working XML file, the XML statements are transformed into FO statements in preparation for building the PDF. This step generates a `.fo` file which can be found in the `.../target/docbkx/autopdf/` directory and typically has the same base file name as the target PDF file. Again, the `pom.xml` file will clarify this name with the `<pdfFilenameBase>` variable.

If one locates and opens the .fo file, it becomes obvious that it was intended as a working file and is not readily readable. Therefore, the first step to understanding this error is to make the FO file more readable. The `xmllint` tool can be used to create a more readable FO file. Assuming you have been working in the document directory, the follow steps can be used to produce a more readable XML file:

```
$ cd target/docbkx/autopdf
$ xmllint --nonet --noent --nowarning --version --timing --format -o outfile infile
xmllint: using libxml version 20901
   compiled with: Threads Tree Output Push Reader Patterns Writer SAXv1 FTP HTTP
 DTDValid HTML Legacy C14N Catalog XPath XPointer XInclude Iconv ISO8859X Unicode
 Regexps Automata Expr Schemas Schematron Modules Debug Zlib Lzma
Parsing took 63 ms
```

```
Saving took 39 ms
Freeing took 9 ms
$
```

For your invocation of `xmllint`, substitute `infile` with the name of the Maven-generated .fo file for your new project and pick a new `outfile` for the new .fo file.

Note

The `xmllint` utility may need to be loaded on your system. On an Ubuntu Linux system, this utility is provided in the `libxml2-utils` package. To locate the proper package for your system, you may need to reference Google.

Now, with a nicely formatted FO file, we can re-invoke the FO Processor (FOP) directly and achieve a more readable error. To do this, invoke `fop` as follows:

```
$ fop -fo fofile and -pdf pdffile
Rendered page #1.
Rendered page #2.
Rendered page #3.
Rendered page #4.
Rendered page #5.
Rendered page #6.
Rendered page #7.
Exception
javax.xml.transform.TransformerException: org.apache.fop.fo.ValidationException:
 "fo:block" is not a valid child of "fo:list-block"! (See position 7830:112)
$
```

As expected, the FOP again reports an exception. However, this time the position information appears more complete. With this new information and a nicely formatted .fo file, one can find the format statements in error, find the context for the error, and then locate the correct source DocBook (XML) file. With this information, one can inspect the document source to decide if the error is bad DocBook syntax or a tooling bug. If the latter, please save the newly formatted .fo file and include it in the bug writeup.

Note

This error generally indicates a problem with documentation tooling. If you encounter such a situation, please post to the Documentation Development mailing list at `<syssw-doc_devel_guide@mailinglist.openpowerfoundation.org>` so they can assist in identifying the exact cause of the failure.

If you wish to fully understanding the error, you may require knowing more about XSL FO syntax. Many such web sites exist for this, but the *XSL Formatting Objects Summary* from W3C (World Wide Web Consortium) provides a good starting reference online at https://www.w3.org/2002/08/XSLFOsummary.html.

1.5.5. Java AWT exception

Use of Maven in headless environments from Mac OS has uncovered an intermittent exception in the AWT libraries. This error looks like the following:

```
...
```

```
-----------------------------------------------
Exception in thread "main" java.awt.AWTError: Can't connect to X11 window server using
 'localhost:11.0' as the value of the DISPLAY variable.
 at sun.awt.X11GraphicsEnvironment.initDisplay(Native Method)
 at sun.awt.X11GraphicsEnvironment.access$200(X11GraphicsEnvironment.java:65)
 at sun.awt.X11GraphicsEnvironment$1.run(X11GraphicsEnvironment.java:115)
 at java.security.AccessController.doPrivileged(Native Method)
 at sun.awt.X11GraphicsEnvironment.<clinit>(X11GraphicsEnvironment.java:74)
 at java.lang.Class.forName0(Native Method)
 at java.lang.Class.forName(Class.java:264)
 at java.awt.GraphicsEnvironment.createGE(GraphicsEnvironment.java:103)
 at java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment(GraphicsEnvironment.
java:82)
 at sun.awt.X11.XToolkit.<clinit>(XToolkit.java:126)
...
```

The circumvention for this error, is force AWT to run headless. This can be accomplished by adding the `-Djava.awt.headless=true` parameter to the maven invocation such that it looks like this:

```
$ mvn clean generate-sources -Djava.awt.headless=true
```

1.6. Publishing OpenPOWER Documents

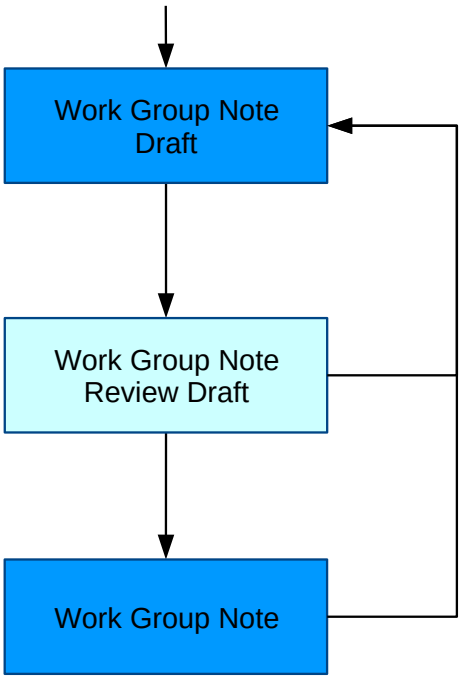
The *OpenPOWER Foundation Work Group (WG) Process* document found in the OpenPOWER Foundation Members Community documents is the definitive guide for understanding OpenPOWER Foundation documents and their work flow. Details such as the duration and types of reviews as well as approval voting specifics are found in this document.

This section of the guide does not attempt to provide process details, but instead strives to provide an overview to help writers understand enough of the basics to know how to prepare their document and what to expect as they proceed through various stages of document development from first draft to publication.

The first key concept to understand about OpenPOWER Foundation documents and the first decision to make when creating a new document is available document types or "Work Products". These fall into one of two categories -- Standards Track or Non-standards Track -- with the simple distinguishing factor being use. If the purpose of a document is to define a specification or standard for hardware or software, then the document is "Standards Track". Everything else is "Non-standards Track." For example, this document is a Non-Stardard Work Product as noted on the title page and the lower right corner of every subsequent page.

Non-standard Track Work Products exist simply as Work Group Notes. Their document lifecycle follows this simplified workflow:

Figure 1.2. Overview of Non-standard Track Work Products

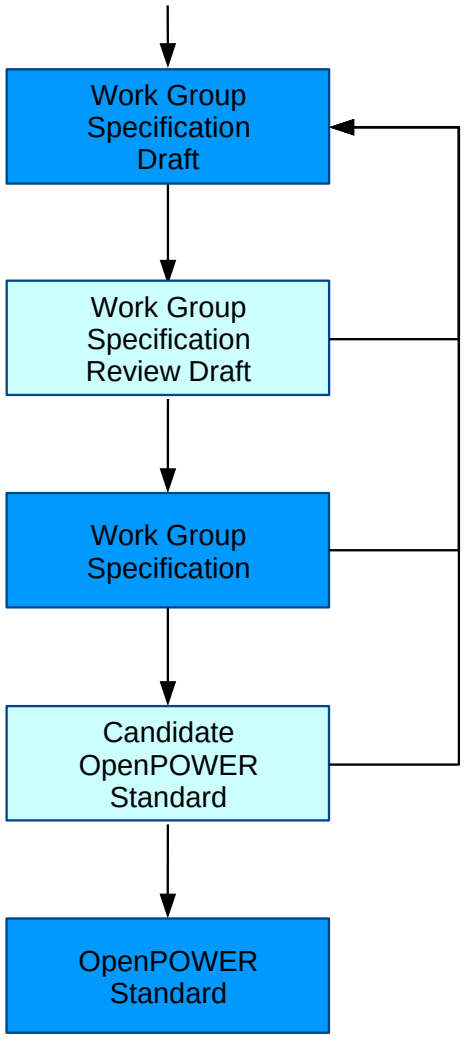


Non-standard Track, Work Group Notes begin as Drafts and drop the "Draft" annotation once reviewed. As shown in the figure, the document lifecycle always returns to a "Draft" form for updates and new versions as needed.

At any step in cycle, these documents may have security classifications as Public (non-confidential), Members-only (OpenPOWER Foundation Confidential), or Work-Group only (OpenPOWER Work Group Confidential) which will in turn dictate the review context (public or private).

Standards Track Work Products begin their life as Work Group Specification and may ultimately become an OpenPOWER Standard. Their document lifecycle is defined in the following illustration:

Figure 1.3. Overview of Standard Track Work Products



Standard Track Work Products begin their lives as Work Group Specifications and have security classifications of Public (non-confidental), Members-only (OpenPOWER Foundation Confidental), or Work Group-only (OpenPOWER Work Group Confidential). The security classification impacts the review type -- either public or internal to the Foundation -- as appropriate. Only Work Group Specifications classified as Public may proceed into OpenPOWER Standard Documents. Confidential documents will remain Work Group Specifications.

The following sections will provide additional details about how to control the document markings and what the process that dictates those markings:

- Understanding document marking variables in the pom.xml file [24]

- Navigating the OpenPOWER Foundation documentation publishing process [25]
- Understanding the specific steps of Standard Work Product documents [30]

1.6.1. Understanding document marking variables in the pom.xml file

Once the document type decision has been made (Work Group Note or Work Group Specification), two additional markings must be considered during the documentation process: the document confidentiality and the document status. The next section, Navigating the OpenPOWER Foundation documentation publishing process [25], details how these values will change during the publishing process. But, before diving into the process, let us see what values in the document `pom.xml` file play a role in the document development process.

The document Work Product categorization, security classification, and document status are reflected in the following ways:

1. The document Work Product type is defined in the document `pom.xml` file with the `<workProduct>` variable. Valid settings are `workgroupNotes`, `workgroupSpecification`, `candidateStandard`, and `openpowerStandard`. Select the appropriate setting in the following section:

```
<!-- TODO: Define the appropriate work product type.  These values are defined by
 the
           IPR Policy. Consult with the Work Group Chair or a Technical Steering
           Committee member if you have questions about which value to select.

          If no value is provided below, the document will default to "Work Group
          Notes".-->
<workProduct>workgroupNotes</workProduct>
<!-- workProduct>workgroupSpecification</workProduct -->
<!-- workProduct>candidateStandard</workProduct -->
<!-- workProduct>openpowerStandard</workProduct -->
```

2. The document security is set in the document `pom.xml` file with the `<security>` variable. Valid settings are `public`, `foundationConfidential`, and `workgroupConfidential`. Select the appropriate setting in the following section:

```
<!-- TODO: Set the appropriate security policy for the document.  For documents
           which are not "public" this will affect the document title page and
           create a vertical running ribbon on the internal margin of the
           security status in all CAPS.  Values and definitions are formally
           defined by the IPR policy.  A layman's definition follows:

           public =                this document may be shared outside the
                                   foundation and thus this setting must be
                                   used only when completely sure it allowed
           foundationConfidential = this document may be shared freely with
                                   OpenPOWER Foundation members but may not be
                                   shared publicly
           workgroupConfidential =  this document may only be shared within the
                                   work group and should not be shared with
                                   other Foundation members or the public

           The appropriate starting security for a new document is
 "workgroupConfidential". -->
```

```
<security>workgroupConfidential</security>
<!-- security>foundationConfidential</security -->
<!-- security>public</security -->
```

3. The document work flow status is set in the document `pom.xml` file with the `<documentStatus>` variable. Valid settings are `draft`, `review`, and `published`. Select the appropriate setting in the following section:

```
<!-- TODO: Set the appropriate work flow status for the document.  For documents
           which are not "published" this will affect the document title page
           and create a vertical running ribbon on the internal margin of the
           security status in all CAPS.  Values and definitions are formally
           defined by the IPR policy.  A layman's definition follows:

               published = this document has completed all reviews and has
                           been published
               draft =     this document is actively being updated and has
                           not yet been reviewed
               review =    this document is presently being reviewed

           The appropriate starting security for a new document is "draft". -->
<documentStatus>draft</documentStatus>
<!-- documentStatus>review</documentStatus -->
<!-- documentStatus>publish</documentStatus -->
```

4. The final place to make updates to a new document is in the `<abstract>` section of the `bk_main.xml` file for the document. This section needs to be updated with the appropriate work group information and document information. Typical text appears as follows:

```
<!-- TODO: Update the following text with the correct document description (first
           paragraph), Work Group name, and Work Product track (both in second
           paragraph). -->
<abstract>
    <para>The purpose of this document is to provide a guide for OpenPOWER
    documentation writers.  As such, it provides directions, policies,
    references, and examples of the XML Docbook environment.  It is intended to be
    used both in final product form (PDF and html) as a document and in source form
    as a template for new documents.</para>
    <para>This document is a Non-standard Track, Work Group Note work product
    owned by the System Software Workgroup and handled in compliance with the
    requirements outlined in the <citetitle>OpenPOWER Foundation Work Group (WG)
    Process</citetitle> document.</para>
</abstract>
```

As stated in the comment text of the book file, the first paragraph provides a typical abstract statement about your particular document. The second paragraph provides more structured text which should be updated with the appropriate Work Group name, Work Product type, and Work Product process. The rest of the information in this paragraph should remain as-is.

1.6.2. Navigating the OpenPOWER Foundation documentation publishing process

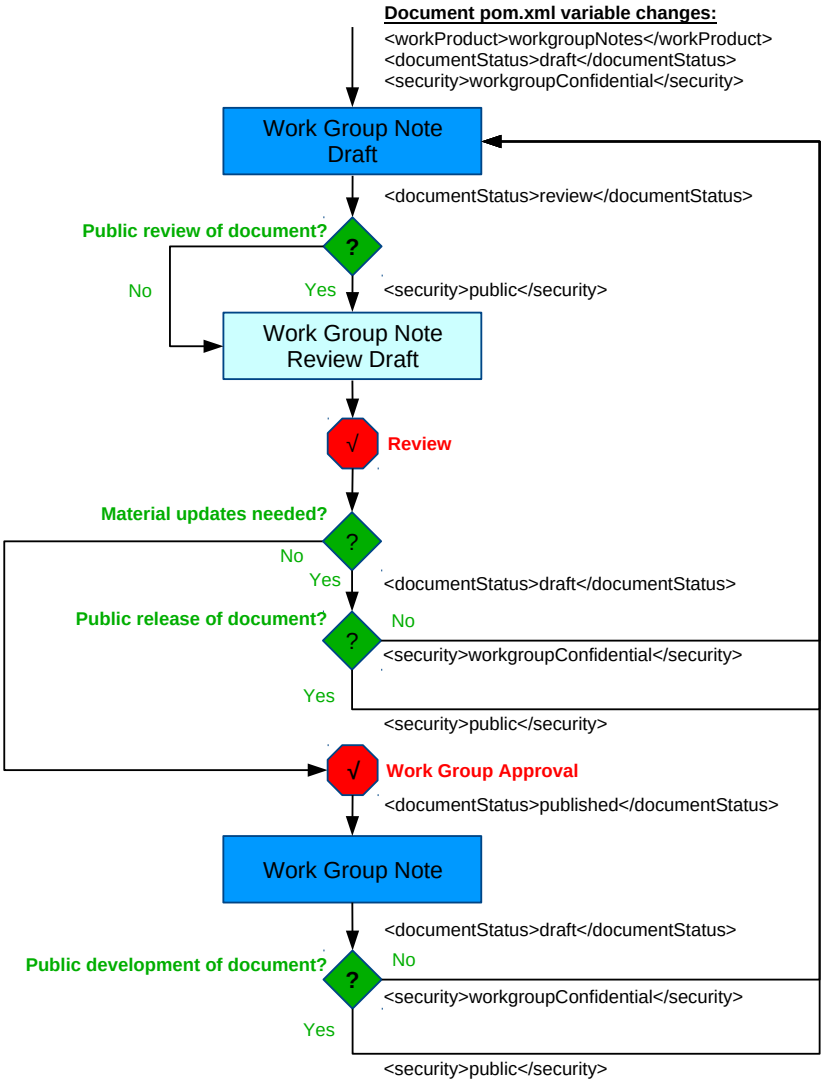
As described in the previous section, Section 1.6.1, “Understanding document marking variables in the pom.xml file” [24], document markings for work product, document confidentiality, and document status are set by the `<workProduct>`, `<security>`, and `<documentStatus>` variables respectively. Selecting the appropriate value for each variable, however, generally depends on the status of the document in the development process.

The following figures and sub-sections provide detailed information about variable settings and process steps. For these figures, the following standards are used:

- Rectangle boxes in various shades of blue represent the work product states previous introduced in Section 1.6, “Publishing OpenPOWER Documents” [21].
- Green diamonds containing question marks, represent decision points with their key questions in bold green and the answers in standard green text.
- Red octagons represent actions required in the process such as reviews or approvals. Specific descriptions are noted in bold red text beside the octagon.
- Black text along the right side of the connecting lines, indicates changes to the various variables in the document `pom.xml` file.

This flowchart expands upon the Non-Standard Track Work Product lifecycle first introduced in Figure 1.2, “Overview of Non-standard Track Work Products” [22]. Document markings and key process decisions and approvals occur as shown.

Figure 1.4. Document work flow for Non-Standard Track Work Products



The only Non-Standard Track Work Product `<workProduct>` setting is `workgroupNotes`. Documents in this track have this value set and never changed.

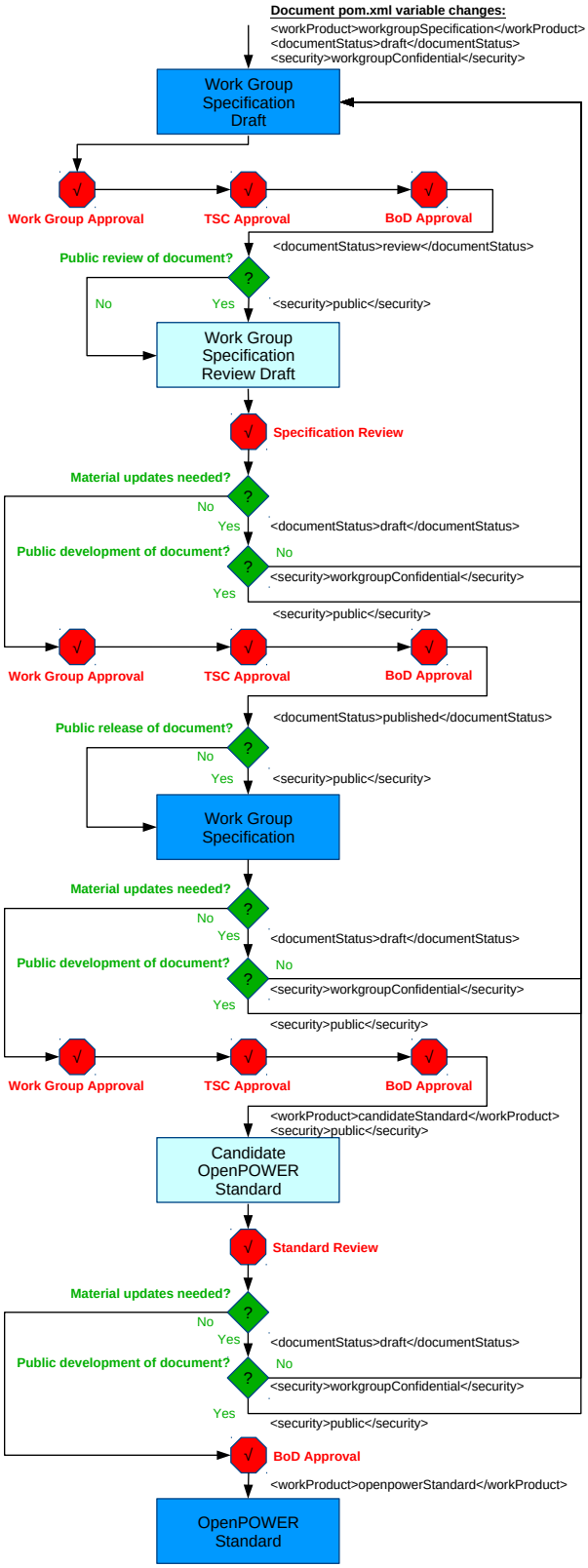
During the work flow progression of the document, a common decision point for the Non-Standard Track Work Product centers on `<security>` settings. Documents may be marked as `public` just prior to review or prior to approval. Each work group will need to review their charter and determine whether public release of their work products is expected or allowed.

The `<documentStatus>` variable tracks quite simply through the work flow, beginning as `draft`, transitioning to `review`, and finishing as `published` when finished.

A feature which makes a Non-Standard Track document unique is that the Work Group is the only approver prior to publish as a Work Group Note. As will be seen in the next figure, Standard Track Work Products often require multiple reviews.

The following flowchart expands upon the Standard Track Work Product lifecycle first introduced in Figure 1.3, “Overview of Standard Track Work Products” [23]. Document markings and key process decisions and approvals reflect a more complex process than the previous one for Non-Standard Work Products.

Figure 1.5. Document work flow for Standard Track Work Products



Like Non-Standard Track Work Products, Standard Track documents frequently evaluate the appropriate security setting. Unlike them, Standard Track Work Products involve many more steps, require numerous approval cycles, and ultimately create a public document (`<security>public</security>`) when they become a Candidate OpenPOWER Standard Work Product.

While the `<workProduct>` type has a value of `workgroupSpecification`, the `<documentStatus>` variable progress as expected -- beginning as `draft`, transitioning to `review`, and finishing as `published`.

Unlike the Non-Standard Work Product, the `<workProduct>` variable begins as `workgroupSpecification`, but may transition to `candidateStandard` as it is proposed to be a Candidate OpenPOWER Standard Work Product and ultimately becomes `openpowerStandard` if the document is approved as an OpenPOWER Standard Work Product. In these latter work flow stages, the `<documentStatus>` and `<security>`remain as `published` and `public` respectively and never change. However, it is work noting that a document may simply exist as a Work Group Specification Work Product for its whole lifecycle. Progression through Candidate OpenPOWER Standard to OpenPOWER Standard is an optional step.

For a deeper look at the process, see the next section, Understanding the specific steps of Standard Work Product documents [30], for step-by-step descriptions of the Standard Product work flow.

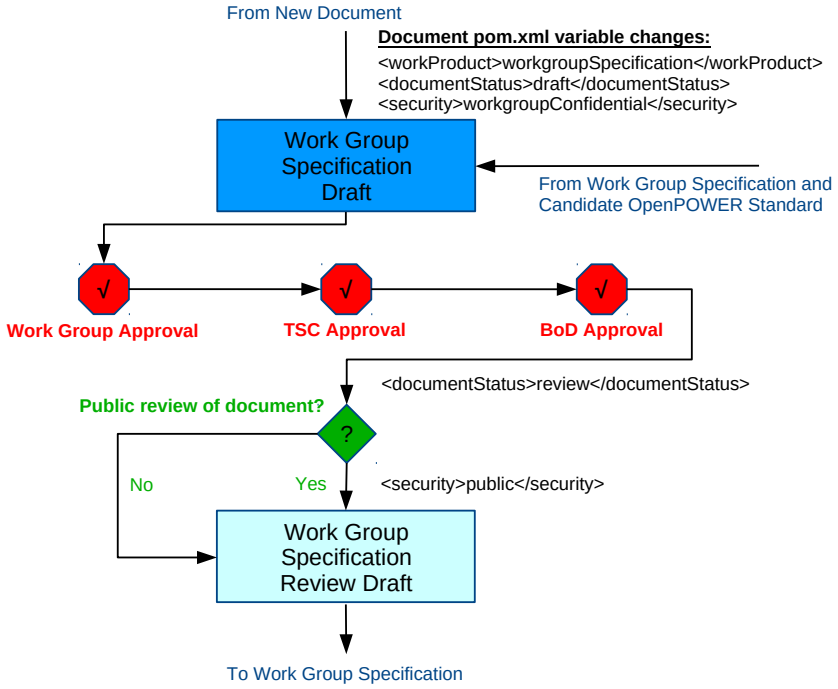
1.6.3. Understanding the specific steps of Standard Work Product documents

Section 1.6.2, “Navigating the OpenPOWER Foundation documentation publishing process” [25] provides an overview of the work flow of both Non-Standard and Standard Work Products. While Figure 1.4, “Document work flow for Non-Standard Track Work Products” [27] is rather straightforward, Figure 1.5, “Document work flow for Standard Track Work Products” [29] is larger and more complex. In an attempt to simplify the process, the following figures decompose each state into just the actions needed to progress to the next step for Standard Track Work Products.

For detailed assistance with the development of Standard Track Work Products, select the figure which reflects your current document state. Then, follow the work flow to understand both the document settings and actions needed to progress to the next document state.

For documents either getting started as Work Group Specification Draft or having returned to this state for updates, reference the following figure. Documents in this state will have `<workProduct>workgroupSpecification</workProduct>` and `<documentStatus>draft</documentStatus>` in their document POM (`pom.xml`).

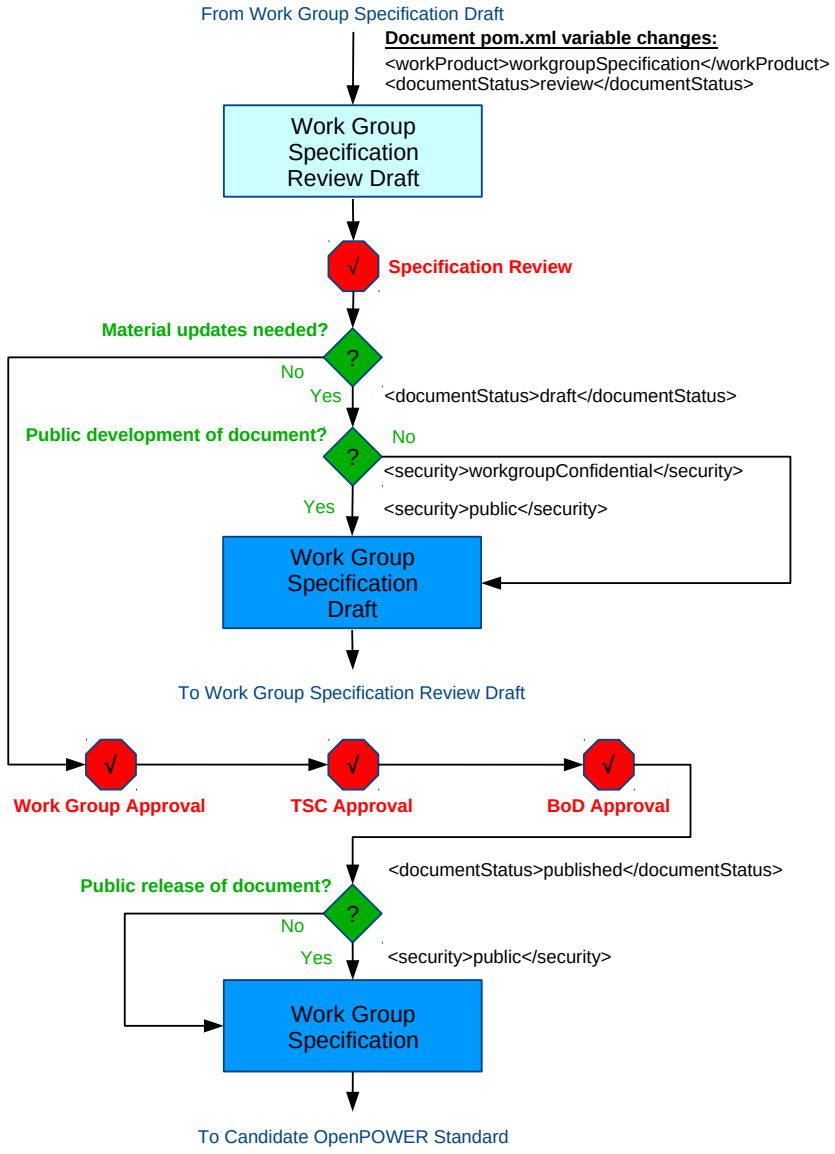
Figure 1.6. Document work flow for Standard Track Work Products in the Specification Draft State



To proceed from a Work Group Specification Draft to a Work Group Specification Review Draft, a document requires 3 approvals, in this order: sponsoring Work Group, Technical Steering Committee, and Board of Directors. Following these three approvals, the document POM (`pom.xml`) variable `<documentStatus>` should be set to `review`. In addition, the `<security>` variable may be set to `public` if the review is targeted to be public.

For documents currently in Work Group Specification Review Draft state (`<workProduct>workgroupSpecification</workProduct>` and `<documentStatus>review</documentStatus>`), consult this figure.

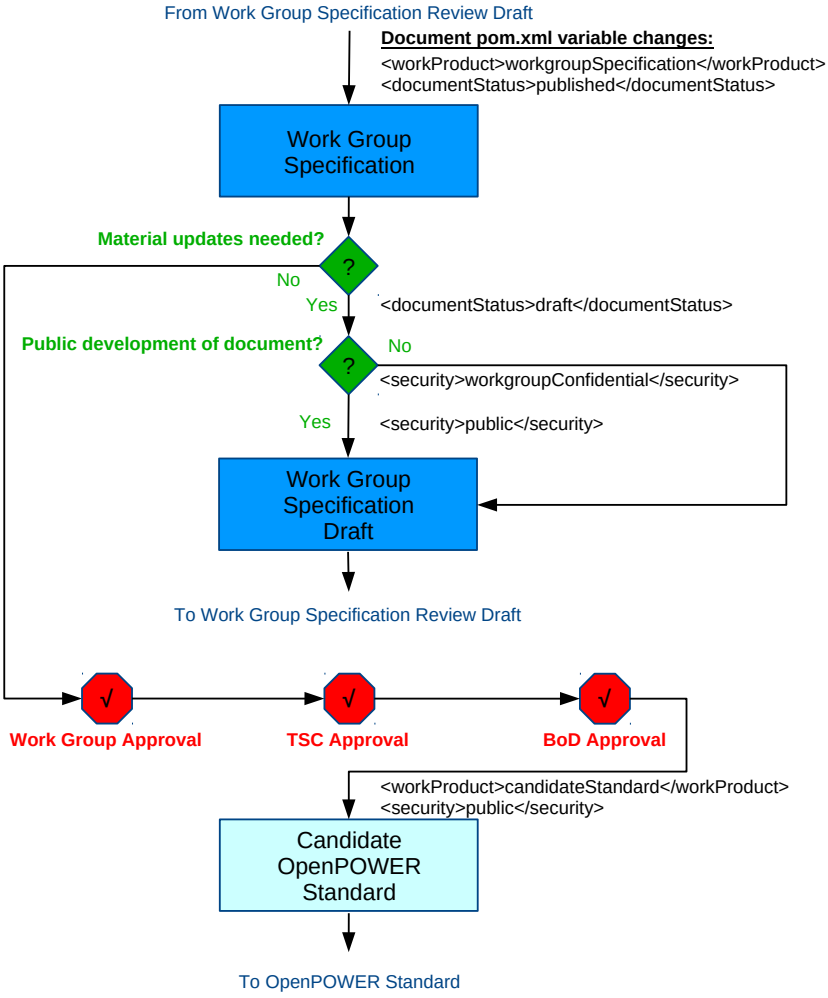
Figure 1.7. Document work flow for Standard Track Work Products in the Specification Review Draft State



To proceed from a Work Group Specification Review Draft to a Work Group Specification, a document requires a successful review and 3 approvals in this order: sponsoring Work Group, Technical Steering Committee, and Board of Directors. Following these three approvals, the document POM (`pom.xml`) variable `<documentStatus>` should be set to `published`. In addition, the `<security>` variable should be set to `public` if for public specifications.

For Work Group Specifications marked `<workProduct>workgroupSpecification</workProduct>` and `<documentStatus>published</documentStatus>`, see the next figure.

Figure 1.8. Document work flow for Standard Track Work Products in the Specification State



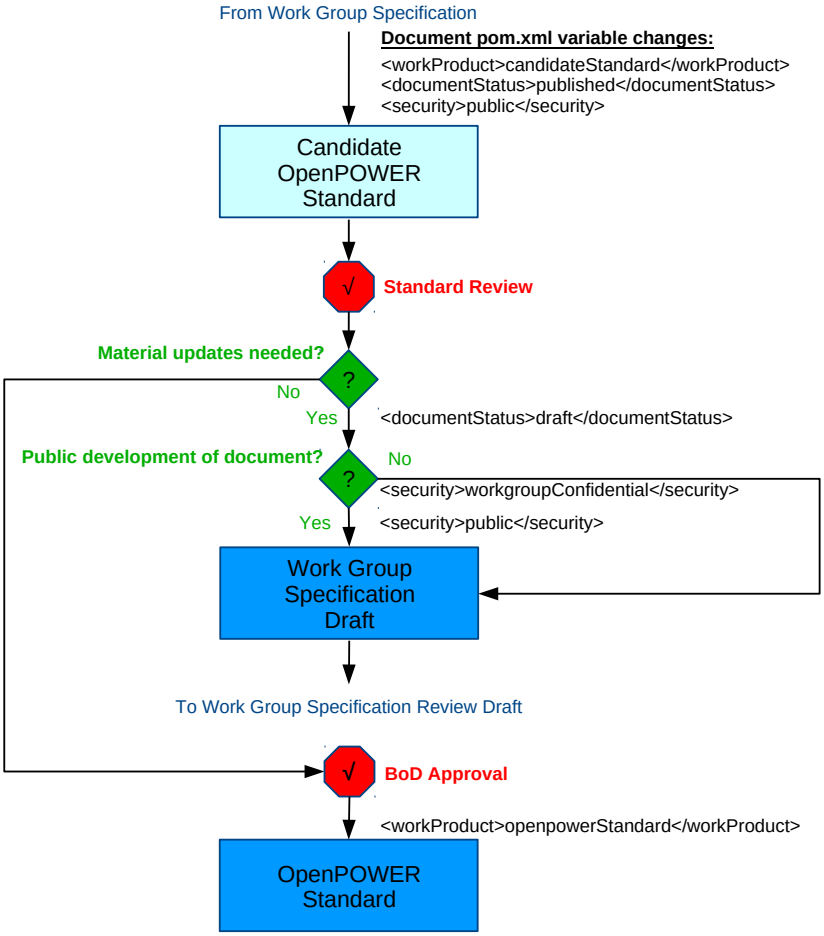
A document in the Work Group Specification state may return to a Work Group Specification Draft or proceed as a Candidate OpenPOWER Standard.

To make updates, the document returns to the Work Group Specification Draft state. To accomplish this, the `<documentStatus>` variable should be set to `draft` and `<security>` should be set to either `public` or `workgroupConfidential`.

To proceed to a Candidate OpenPOWER Standard, a document requires 3 approvals, in this order: sponsoring Work Group, Technical Steering Committee, and Board of Directors. Following these three approvals, the `<workProduct>` variable should be set to `candidateStandard` and `<security>` should be set to `public`.

For documents currently in Work Group Candidate OpenPOWER Standard state (`<workProduct>candidateStandard</workProduct>` and `<documentStatus>published</documentStatus>`), reference the following figure.

Figure 1.9. Document work flow for Standard Track Work Products in the Candidate OpenPOWER Standard State



A document in the Work Group Candidate OpenPOWER Standard state may proceed in two directions, back to a Work Group Specification Draft or on to a Candidate OpenPOWER Standard.

To make updates to a Work Group Candidate OpenPOWER Standard document, the document returns to the Work Group Specification Draft state. To accomplish this, the `<documentStatus>` variable should be set to `draft` and `<security>` should be set to either `public` or `workgroup-Confidential` depending on how the Work Group handles document drafts.

To proceed to an OpenPOWER Standard, a document requires a successful review and a single approval from the Board of Directors. Following this approval, the document POM (`pom.xml`) variable `<workProduct>` should be set to `openpowerStandard`.

1.6.4. Packaging the document for publish

The OpenPOWER Foundation process for publishing documents from WordPress in the Resource Catalog on openpowerfoundatoin.org website has the following requirements:

- The PDF and all HTML source must be bundled in a self-contained zip file.
- The zip file is expected to contain a single directory in which the document PDF and `index.html` file are found.
- The filename of the zip file must be the same name as the contained directory.

To create this package for the *Documentation Development Guide*, one would perform the following commands in Linux from the document source directory (`.../Docs-Template/doc_dev_guide/`):

```
Docs-Template/doc_dev_guide$ cd target/docbkx/webhelp/
Docs-Template/doc_dev_guide/target/docbkx/webhelp$ ls
doc-devel-guide
Docs-Template/doc_dev_guide/target/docbkx/webhelp$ zip -rv doc-devel-guide.zip doc-
devel-guide/
  adding: doc-devel-guide/ (in=0) (out=0) (stored 0%)
  adding: doc-devel-guide/favicon.ico (in=806) (out=806) (stored 0%)
  adding: doc-devel-guide/index.html (in=654) (out=385) (deflated 41%)
  ...snip...
  adding: doc-devel-guide/doc-devel-guide-20180406.pdf  (in=413655) (out=305492)
 (deflated 26%)
  ...snip...
  adding: doc-devel-guide/common/ (in=0) (out=0) (stored 0%)
  adding: doc-devel-guide/common/main.js (in=5674) (out=2119) (deflated 63%)
  ...snip...
  adding: doc-devel-guide/common/jquery/jquery-ui-1.8.2.custom.min.js  (in=87032) (out=
22729) (deflated 74%)
total bytes=3342807, compressed=1332882 -> 60% savings
Docs-Template/doc_dev_guide/target/docbkx/webhelp/doc-devel-guide$ ls
doc-devel-guide  doc-devel-guide.zip
```

For MacOS and Windows, the steps will be similar with slight variations on the command to create the zip file.

This zip file can be sent to the person managing the documents in the OpenPOWER Resource Catalog.

1.7. Policies and conventions

Most document style policies are established simply by using the provided documentation framework. However, by applying some conventions to the document source structure, community members will be able to work across more d ocumentation projects.

The recommended documentation structure guidelines are as follows:

1. The head book file should be named with the prefix "bk_".
2. The document versioning as defined by the `<releaseinfo>` tag in the main book file `bk_xxx` should be named "Revision V.R.M", not "Version V.R.M" or simply "V.R.M" where:
    - Significant updates increment the `V` (Version) value while reseting the `R` and `M` values to `0`,

- Material, but small, updates increment the `R` (Release) value and reset the `M` to `0`, and
- Trivial updates (such as typos and grammatical changes) only need to increment the `M` (Modifier) value.

Note

Numbering of "pre-release" versions or draft versions of a document may be handled in multiple ways such as incrementing the previous modifier level until publication and then updating appropriately, setting the releases to the anticipated level and then appending a "_preN" suffix where "N" can be incremented during drafting. Each Work Group may set their own policy here.

3. Chapters files should be named with the prefix "ch_".
4. Section and sub-section files should be named with the prefix "sec_".
5. Appendix files should be named with the prefix "app_".
6. Figures source and images should be placed in the `figures` sub-directory for the document.
7. Releases of the same document sound be contained in the same tree, but tagged at levels of interest using the `git tag` command. See the Section 1.9, “Common git commands” [36] for more specifics on `git` commands.

In addition to documentation structure, general community/project guidelines are as follows:

1. Contributions to the documentation projects should conform to the *Developer Certificate Of Origin* as defined at http://elinux.org/Developer_Certificate_Of_Origin. Commits to the GitHub project need to contain the following line to indicate the submitter accepts the DCO:

```
Signed-off-by: Your name <your_email@domain.com>
```

1.8. Frequently asked questions

The list of questions and answers may be helpful to first time document writers:

Q: Do I have to follow the guidelines in Section 1.7, “Policies and conventions” [35] of this guide?

A: No. *HOWEVER*, doing so makes it simpler for all community members to participate in maintaining your document.

1.9. Common git commands

This section provides a list of commonly used git command invocations. All commands shown, except the first one (`git clone` must be issued from within the project directory.

- To clone a git tree for first time or temporary use via http, use:

```
$ git clone <URL>
```

The <URL> value for OpenPOWER Foundation GitHub projects can be found on the project web pages. They generally take the form of `https://github.com/OpenPOWERFounda-`

tion/project_name where the project_name can be found on the OpenPOWER Foundation Git Hub community page at https://github.com/OpenPOWERFoundation. The result of this command will be a new directory with the same name as the project and in which will be the project files.

Note

Trees can only be cloned once. To update a tree, use a `git pull` or `git merge` command.

Note

When cloning from a private tree, you will be prompted for your GitHub userid and password.

- To update a git tree with new files from the remote repository, use:

```
$ git pull
```

This command assumes that the local tree has not been updated since the clone or last pull. If updates have been made to the local tree, the command will fail. Use the `git status` command to see what has changed in a local tree.

Note

When pulling from a private tree, you will be prompted for your GitHub userid and password.

- To see the status of the local repository, use:

```
$ git status
```

This command identifies files which have changed in the local repository and provides suggestions on how to handle.

Note

Adding the `-s` parameter to the end of the command will provide a simplified view in which changed files are listed with flags such as `M` for modified files, `A` for newly added files, and ?? for new or unknown files. This parameter also suppresses suggested action information for the files.

- To add a new file or directory to a git tree, use:

```
$ git add <new_file>
```

The <new_file> value can be either a file or a whole directory and may include the path to the target file or directory. This command will convert the status of file in the `git status -s` command from ?? to `A` or move it from the "Untracked files" section to the "Changes to be committed" section of the `git status` command.

- To remove a file from a git tree, use:

```
$ git rm <file>
```

The <file> value must be a file and may include wildcard characters or the path to the target file. This command will both remove the file(s) from the directory and the git tree. Removed files will show in a status modifier of D in the `git status -s` command and be reflected in the "Changes not staged for commit" section of the`git status` command with a "deleted:" status.

- To remove a directory from a git tree, use:

```
$ git rm -rf <directory>
```

The <directory> value must be a directory name and may include wildcard characters or the path to the target directory. This command will remove all files in the directory from the git tree, but will not remove the directory locally. Standard operating system commands such as the Linux `rmdir <directory>` command must be issued separately to remove the local directory. All removed files will show in a status modifier of D in the `git status -s` command and be reflected in the "Changes not staged for commit" section of the`git status` command with a "deleted:" status. Because git does not track directories, they will not be reflected in status.

- To move or rename a file or directory in a git tree, use:

```
$ git mv <source> <destination>
```

The <source> value must be a file or directory and may include the path to the target file. The <destination> value may be a file (if renaming a file) or a directory if moving a file or directory. This command will move or rename the file(s) in both the local and remote the git trees.

- To commit all local changes to the staging area for a git tree, use:

```
$ git commit -a
```

This command will invoke an editor for a commit message. A well-formatted commit message includes a title on the first line, a blank line, one or more lines of details describing the changes, and a Developer's Certificate of Orig (DCO) Sign-off statement at the end.

```
Signed-off-by: Your name <your_email@domain.com>
```

For information on the DCO, see *Developer Certificate Of Origin* at http://elinux.org/Developer_Certificate_Of_Origin.

- To push all locally staged changes to the remote git tree, use:

```
$ git push
```

Note

When pushing to a private tree, you will be prompted for your GitHub userid and password.

- To see what tags exist in a git tree, use:

```
$ git tag
```

- To create a new tag locally, use:

```
$ git tag -a <tag_name> -m"text"
```

The `tag_name` represents the simple value of the tag. The `text` string provides more description of the tag for readibility.

Note

This command simply tags locally. See the next command for how to push the tag to the remote repository.

- To push a new tag from the local tree to the remote tree, use:

```
$ git push origin <tag_name>
```

  This commands assumes the `git tag` command has been run on the local tree.

- To discard changes from a locally changed file and return to the last copy, use:

```
$ git checkout -- <file>
```

  The `<file>` value must be a file and may include wildcard characters or the path to the target file.

- To identify what changes have been made locally to a file use:

```
$ git diff <file>
```

  The `<file>` value must be a file and may include wildcard characters or the path to the target file. The output will be in format similar to the standalone `diff` command.

Additional resources about git can be found online at the following locations:

- The *GitHub Glossary* at https://help.github.com/articles/github-glossary/. This site provides a list of common terms associated with git and GitHub.
- The GitHub *Git Cheat Sheet* at https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf. This two page pdf provides a quick summary of many common commands.
- The *Git Reference* at http://gitref.org/. This is a deeper and more comprehensive reference of important commands.
- The git-scm.com *Documentation* library at http://git-scm.com/doc. This site provides education in the form of books, videos, and other tutorials for common git activities.

1.10. Finding more information

The following lists of references may be helpful in learning about XML, Docbook, and/or Maven:

- *XML In a Nutshell* by Elliotte Rusy Harold and W. Scott Means. Online at http://docstore.mik.ua/orelly/xml/xmlnut/index.htm.
- *DocBook 5: The Definitive Guide* by Normal Walsh. Online at https://www.safaribooksonline.com/library/view/docbook-5-the/9781449380243/.
- *DocBook XSL: The Complete Guide* by Bob Stayton. Online at http://www.sagehill.net/docbookxsl/.

- *Maven: The Complete Reference* by Tim O'Brien, Manfred Moser, John Casey, Brian Fox, Jason Van Zyl, Eric Redmond, and Larry Shatzer. Online at http://books.sonatype.com/mvnref-book/reference/index.html.

2. Documentation examples

2.1. Section Title goes here

This Section covers something of interest to a limited number of people and shows a 1st level section

2.1.1. Example Itemized List

Here is an example of an itemized list

A list title is completely optional

- Item you don't care about
  - Perhaps you'd like a sub-list
    - Oooh, here's about another
- Item you might care about
- Item you do care about

2.1.2. Example ordered list

All good documents need ordered lists.

Another purely optional title

1. First item
2. Second item
   a. first indented item
   b. second indented item
3. Third item

2.1.3. Example variable list

One of my favorite list types for formating items with definitions is called a variablelist. Here is an example with an embedded variablelist.

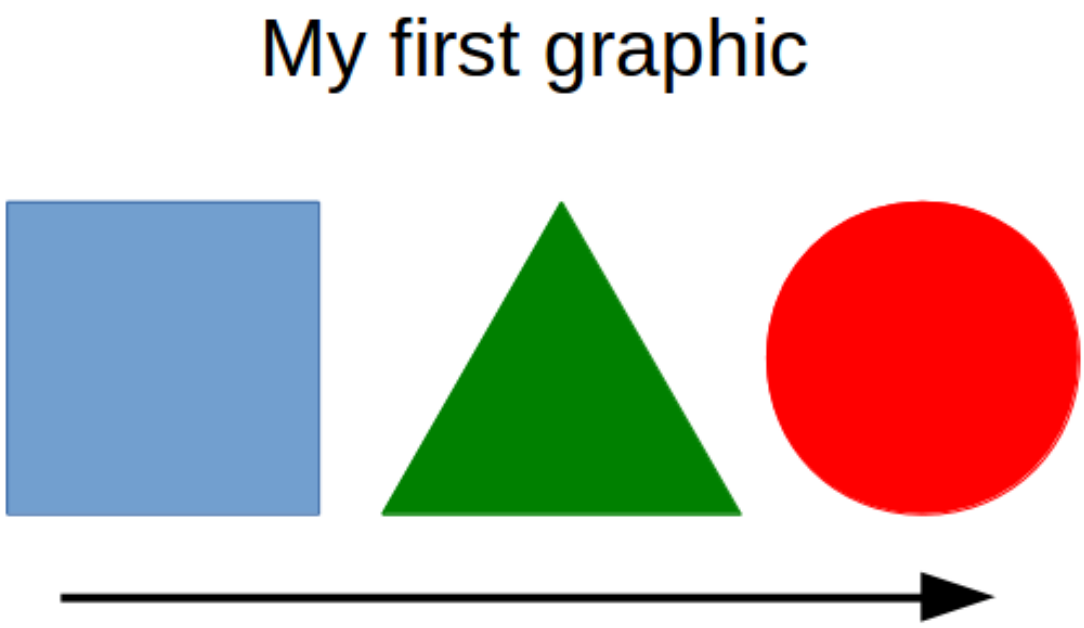
Kirk Captain

Crew Members
Scotty Engineering
McCoy Doctor
Spock Science Officer

2.1.4. Example figure with embedded graphic

Here is how you embed a graphic.

Figure 2.1. Example figure



Note

Raw images such as the bitmap (bmp) file above may become blurry as they are scaled. Scalable graphic formats like SVG (Scalable Vector Graphics) embed and scale the best.

2.1.5. Example table

Of course all good documents need tables. Here's how you build a basic table.

Table 2.1. Example Table Title

1st Column Heading	2nd Column Heading	3rd Column Heading	4th Column Heading
Yes	Red Green Blue Custom (Amber)	MAIN_Junk	More_Junk
merged cells horizontal			cell_stuff

1st Column Heading	2nd Column Heading	3rd Column Heading	4th Column Heading
Merge cells vertical	filler	merge cells both ways	
	filler 2		
How about we put a list in the table cell • item 1 • item 2 • item 2	Another Cell	Yet Another Cell	Finally the last cell
This Row	Has	background	color
Eenie	Meenie	Meinie	Entry with background color

2.1.6. Example of crossreferences and footnotes

To reference another section or table is pretty easy. For example: see Table 2.1, “Example Table Title” [42] for how tables look.

Lists are shown in Section 2.1.1, “Example Itemized List” [41] and if you need to make a footnote [1] For clarification that is easy. Of course you might want an additional reference to the footnote [1] which can also be done easily.

Lastly you probably want to mark text by making it *italic text example* or **Bold Text Example**.

2.1.7. Example of code citations and user input

When showing user input, you want a nice sceen-looking layout, a prompt, monospace text, and a way to differentiate input from output. Here's an example:

```
$ echo "Hello world"
Hello world
$
```

Docbook also allows for formatting and display of common languages, allowing for whitespace and line returns just as they are written. Here's a sample snippet of C code with line numbering enabled:

```
  1 #include<stdio.h>
    main()
    {
        printf("Hello world\n");
  5 }
```

If code formatting is not quite what you need, simply displaying text "literally" may suffice as follows:

```
This is my literal text. It ignores whitespace.
```

2.1.8. Example of special characters in text

Sometimes in text you need special characters. These can be provided using their UNICODE values such as ≠ (&#8800), Ω (&#x2126), and Δ (∆). These can be "coded" using the form &#*ddddd*; where *ddddd* is the up to five digit decimal representation of the character. The form &#x*hhhh*; where *hhhh* is the up to 4 digit hexidecimal representation of the character.

[1] The footnote text goes here and can reference something like Figure 2.1, “Example figure” [42] for additional explanation.

This formatting works well as long as the symbol to which you are referring is contained in the font set used for the document -- Arimo for standard text and Cousine for monospace. If when building a document, you see a message like "WARNING, Glyph...not available in font 'Arimo'," see the section called “Using additional symbols” [46] in Section 2.1.10, “Examples of OpenPOWER Foundation Docbook extensions” [44] for details on using the provided symbol fonts explicitly.

2.1.9. Sample section include

This section was developed in a separate file but included in the document by using the following text:

```
  <xi:include href="sec_example.xml"/>
```

where `sec_example.xml` is the source file name.

2.1.10. Examples of OpenPOWER Foundation Docbook extensions

The OpenPOWER Foundation Maven Plugin supports a number of extensions that are not pure Docbook. These are:

Setting text color explicitly

Text color can be controlled using `<phrase role="color:`*color_name*`">` tag where *color_name* contains the color setting. For example, this text:

```
<para role="color:red">A red sentence contains a <phrase role="color:blue">blue</
phrase> word.</para>
```

produces this sentence:
A red sentence contains a blue word.

Valid colors include either a keyword color name or a numerical RGB specification. Keyword names are common with the HTML 4 specificiation: `aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `purple`, `red`, `silver`, `teal`, `white`, and `yellow`. Additionally, RGB values can be `#nnnnnn` where `nnnnnn` is a hexidecimal color value or `rgb(n1, n2, n3)` where `n1`, `n2`, and `n3` are integers 0-255.

This tag has also been implemented on the following tags: `<thead>`, `<tbody>`, and `<tfoot>`.

Warning

This parameter should only be used for tags listed above.

Inserting line breaks

Line breaks can be introduced using `<?linebreak?>` tags. For example, this text:

```
<para>A line break <?linebreak?> in the middle of text</para>
```

produces this sentence:

A line break

in the middle of text

This tag becomes useful in table text spacing.

Inserting page breaks

Page breaks can be introduced using <?hard-pagebreak?> tags. For example, this text:

```
<para>A page break</para> <?hard-pagebreak?> <para>Between two paragraphs</para>
```

produces this output:

A page break

Between two paragraphs

This tag becomes useful in placing tables on page. Placing this statement before a large table may prevent it from spanning a page.

Warning

Because the XSL template behind the Processing Instruction generates a

```
<fo:block break-after='page'/>
```

in the book FO output, this instruction should be used in the outer most blocks of a section to work effectively. Use inside lists and other structural components may result in the text after the break being dropped. **User beware!**.

Varying the font size

Font sizes can also be set using the `<phrase role="font-size:`*size*`">` tag where *size* contains a size value such as "6pt" or "50%" or "1.5em".

For example, a paragraph can be made to be 6 point as follows:

```
<para>A sentence that contains some <phrase role="font-size:6pt">6pt font</phrase>,
<phrase role="font-size:50%">50% font</phrase>, and
<phrase role="font-size:1.5em">1.5em font</phrase> in it.</para>
```

produces this output:

A sentence that contains some 6pt font, 50% font, and 1.5em font in it.

This tag has also been implemented on the following tags: `<para>`, `<thead>`, `<tbody>`, and `<tfoot>`.

Warning

This parameter should only be used for tags listed above.

Using additional symbols

If you find that the Arimo and Cousine fonts do not contain the special symbol you need for your document, you may use the additional symbol font provided for document (STIX Two Math). Due to an unimplemented feature in the Apach FO Processor, selection of this font needs to be explicitly performed using the `<symbol role="symbolfont">` wrapper around your symbol value.

For example, the symbol coding of

```
&#x2A01;
```

should produce a circle with a cross in here "#", but instead creates a "Glyph...not available in font 'Arimo'" error on document build and the PDF renders as a "#".

Re-coding this to use

```
<symbol role="symbolfont">&#x2A01;</symbol>
```

produces the correct symbole here "⨁".

If this still does not provide the symbol you expected, double check the code and the font maps found at http://www.stixfonts.org/charactertable.html.

Appendix A. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

A.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

A.2. Technical resources

Development resouces fall into the following general categories:

- Foundation work groups
- Remote development environments (VMs)
- Development systems
- Technical specifications
- Software
- Developer tools

The complete list of technical resources are maintained on the foundation Technical Resources web page.

A.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- <info@openpowerfoundation.org>
- Membership -- <membership@openpowerfoundation.org>
- Technical Work Groups and projects -- <tsc-chair@openpowerfoundation.org>
- Events and other activities -- <admin@openpowerfoundation.org>
- Press/Analysts -- <press@openpowerfoundation.org>

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.

Appendix B. Appendix template

This is the first paragraph of a new appendix...

B.1. Section title

Section text...