Authoring with Eclipse

Chris Aniszczyk, IBM Corporation <zx@us.ibm.com>
Lawrence Mandel, IBM Corporation <lmandel@ca.ibm.com>

$Id: article.xml 84 2005-12-13 16:25:20Z lmandel $

Abstract

The topic of technical publishing is relatively new to the world of Eclipse. One can make the argument that technical publishing is just another collaborative development process involving several people with different backgrounds and skills. This article will show that the Eclipse platform is a viable platform for technical publishing by discussing how to write documents such as an article or a book within Eclipse. In fact, this article was written using Eclipse.

Environment

The examples in this article were built and tested with:

- Eclipse 3.1 [http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.1-200506271435/eclipse-SDK-3.1-win32.zip]
- Eclipse Web Tools Platform (WTP) 1.0 [http://download.eclipse.org/webtools/downloads/drops/R-1.0-/]
- Orangevolt XSLT 1.0.4 [http://eclipsexslt.sourceforge.net/]

Introduction

The authors of this document view technical documentation as another development process that shares the same characteristics as a software process. In technical publishing, you have writers, editors, typesetters, QA reviewers, and so on. Technical publishing is a collaborative process that currently lacks the tools to facilitate collaboration. The goal of this article is two-fold: give an introduction to technical doc-

umentation and show, through an example, how Eclipse can help make technical documentation a collaborative process.

Technical Documentation

In the open source world, technical documentation is primarily accomplished using two popular formats: DocBook and the Darwin Information Typing Architecture (DITA). These two formats share two important characteristics: they are both systems for creating structured documents using XML and both focus on content that is written in plain text (or in an editor such as OpenOffice). In this article we focus on DocBook because of our familiarity with the format. However, we will also provide complementary DITA information where appropriate.

If you're unfamiliar with DocBook, there's an article [http://www-128.ibm.com/developerworks/library/l-docbk.html] on the IBM® developerWorks® site by Joe Brockmeier that can serve as a gentle introduction. There is also an introduction [http://www-128.ibm.com/developerworks/xml/library/x-dita1/index.html] to DITA on developerWorks.

The technical documentation process can be broken down into three broad stages: creation, review, and publication.

Creation simply refers to populating a document with content that adheres to whatever format you choose to write against. This document is usually edited using an XML editor of choice, although it can be edited with any text editor as well. In this article we will use WTP's XML editor to edit documents.

Once an initial version of the content has been written, it is typically handed off to one or more trusted colleagues for review. The role of these reviewers is to ensure technical accuracy and improve the quality of the writing. The comments and suggestions gathered from the review stage are then used by the document's authors to create a final revision of the document.

The final revision of a document involves making it is ready for publication. When authoring in an XML format, you must eventually transform the document must be transformed from XML to a human-readable format (that is, one that has both style and formatting applied) such as HTML or PDF. Once in a human-readable format, the document is ready to be published by a selected publisher.

Advantages of an XML format

Before diving into an actual example of the technical documentation process using Eclipse, let's take a look at some of the benefits of authoring in an XML format.

There are four advantages to authoring in XML that really show the benefit of this format: modularity, version control, consistent formatting, and publishing to multiple formats.

Modularity

- XML formats such as DocBook and DITA are modular. This allows you to break up your documents into multiple sections, which can be automatically combined into one document using transformation during the publication stage. Modularizing your documents can be very beneficial when working on large documents, such as a book, or when working with multiple authors. As an example, the book *Java Web Application Development with Eclipse* (set to be published in time for EclipseCon 2006), was written in DocBook. The book was structured with one table of contents XML file and a separate file for each chapter. Of course, it's up to you to determine the structure that works best for your project. The key is that by using an XML format, you have the freedom to configure your document's structure.

Version Control

- Version control is very useful and has become a staple in most development processes. Why is it, then, that a system that allows you to maintain the complete history of your files, allowing you to revert to a previous version at any time, is not part of the authoring process? One reason is that many documents are authored using word-processing tools that mix formatting information and content, such as Microsoft® Word or Corel WordPerfect. This mix results in files containing many changes between revisions, reducing the usefulness of version control because it is difficult to view the relevant changes between versions of the document. XML formats do not suffer from this problem as they are content-specific. Authoring in an XML format allows you to use a version control system and reap the benefits that go with it.

Consistent Formatting

- Ensuring that your document is consistently formatted is a time-consuming aspect of the authoring process. This task can be further aggravated when the authoring format mixes content and formatting in one document or when working with multiple authors or files. One of the typical final steps in the authoring process is ensuring that your document uses consistent formatting. Using an XML format solves the consistent formatting problem by separating your document's content from its formatting. In the XML case, formatting can be applied uniformly to your entire document using a style sheet. An XML format saves you time and guarantees consistent formatting.

Publishing to Multiple Formats

- By separating your document's formatting from its content, you gain an enormous freedom. Documents authored in the XML format are not bound by one set of formatting rules. This means that you can author an article, such as this one, and create an HTML version, a PDF version, and even an Eclipse help system version simply by transforming your document with different style sheets. In fact, DocBook includes all three of these stylesheets allowing you to easily publish to any of the formats listed above.

Examples

To show the authoring tool chain in Eclipse, this article will use a sample book document from the DocBook XSL project. The XML version of the document can be seen here [files/book.xml]. This DocBook source for this article is also available and can be seen here [AuthoringWithEclipse.xml].

Tool Chain

A tool chain is a set of tools that are used to create a more complex tool or product. The tools may be used in a chain, so the output of each tool becomes the input of the next [toolchain] . This concept should be very familiar to those who work on the UNIX®, Linux®, and AIX® platforms, for example, where the output of one command line tool is typically piped to the next tool, allowing complex operations to be performed using several simple tools.

The beginning of our technical publishing tool chain is the WTP XML editor, which we use to edit our content. After we have finished editing the content, we will feed the output of what we edited into OrangeVolt, an XSLT transformation engine, which will use style sheets to publish the content into a human-readable format.

The limitation we put on these examples is that our tool chain, including all three stages, creation, review, and publication, will be built with tools available within Eclipse. From our experience, Eclipse has enormous potential as an integrated documentation development environment. In the following sections, we'll discuss how you can make this a reality with current Eclipse tooling and where the tooling falls

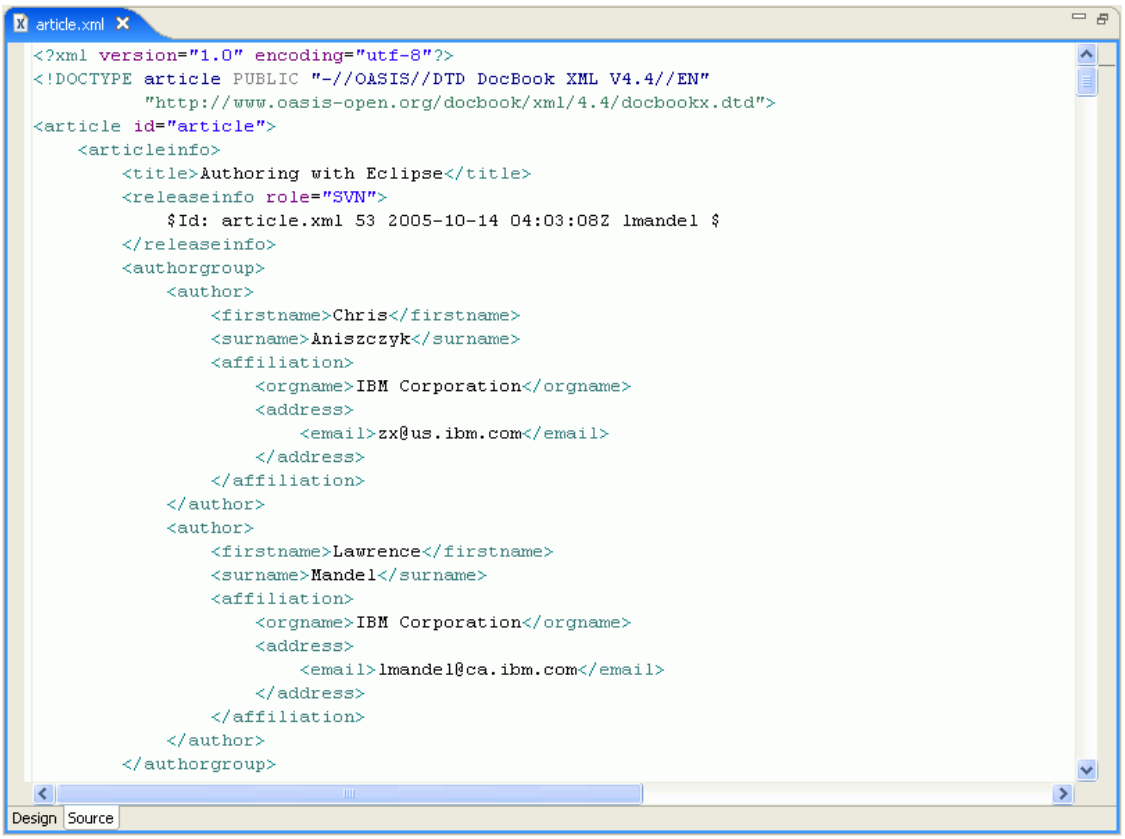
short.

Creation and Review

Although creation and review are two separate parts of the technical documentation process, the same tools are required and therefore will be discussed together.

As you may already know, the Eclipse project is composed of serveral top-level projects including Eclipse itself (known as the Eclipse base) and the WTP project. WTP adds many tools to the Eclipse base including an XML editor with graphical and source representations of the content. Although the graphical editor is useful for viewing your document, we've found that the source editor, shown in Figure 1, “The XML Source Editor” , is more useful when authoring in XML.

Figure 1. The XML Source Editor



The XML editor provides many of the Eclipse franchise functions that Java™ developers have become accustomed to with the Java editor.

Content Assistance

- Gives you a list of valid XML elements constrained by an associated grammar.

Syntax Highlighting

- Gives you improved code readability by making certain errors instantly visible.

Validation

- Ensures validity of an XML document based on an associated grammar.

Outline View

- Assists you in editing and viewing the content of your document.

XML Catalog

- Allows you to register Document Type Definitions (DTD) and XML Schema grammars associated with your document with your workspace so you can work with the benefits of validation while disconnected from the Internet.

Aside from the benefits of the XML editor, working in Eclipse provides other benefits. Eclipse includes integrated version control for CVS. (There is also a freely available plug-in for subversion [subversion], another version control system.) Integrated version control allows you to check your changes into, and view others' changes in, your version control system from within Eclipse. These tools are also useful for your reviewers, who, if you give them permission, can add comments and suggestions to your document and check their changes in. Giving your reviewers permission to make these changes allows you to avoid the need to use e-mail or some other communication mechanism.

Publication

The DocBook XSL [docbookxsl] and DITA [dita] projects offer numerous transformations, including HTML and PDF formats. The most common transformation technique is to use an Ant file with the appropriate tasks for the various transformations. In this article we use the Orangevolt XSLT tool to simplify this task. Orangevolt XSLT integrates into the familiar Eclipse launcher framework. This integration allows you to select the style sheet and pass in necessary parameters for the transformation.

The DITA Open Toolkit (DITA-OT) includes a DITA to DocBook transformation [http://dita-ot.sourceforge.net/doc/DITA-antscript.html].

Along with the description in this article, we have provided Flash movies that demonstrate how to perform each transformation. Transformations for both DocBook and DITA will be provided where appropriate.

HTML

Of all the available transformations, transforming your document into HTML is the easiest to use. All that you need to do is create a proper transformation launch configuration and run the transformation. Specifically, you need to specify the correct style sheet:

- DocBook
  - html/docbook.xsl

- DITA
  - xsl/dita2html.xsl

Figure 2, “Sample HTML Transformation Configuration for book.xml” shows a sample transformation configuration that will transform our DocBook sample document [files/book.xml] into HTML.

Figure 2. Sample HTML Transformation Configuration for book.xml

tip

You can augment the transformation by passing parameters to the style sheet. There's a full listing of DocBook XSL parameters that can be used to configure the transformation located here [http://docbook.sourceforge.net/release/xsl/current/doc/html/].

A Flash movie that shows how to run the transformation can be seen here [files/DocBook-HTML.htm] .

PDF

Transforming a DocBook XML file to PDF format is more involved than the transformation to HTML but it is still possible using a style sheet. The difference lies in a task that must be performed before the actual transformation. So, the transformation from XML to PDF is a two-step process.

Step one is to generate an XSL formatting objects (XSL-FO) document. This document will then be transformed into a PDF. In order to generate an XSL-FO document, you need to use the following stylesheet: *fo/docbook.xsl*. Figure 3, “Sample XSL-FO Transformation Configuration for book.xml” shows a sample transformation configuration used to generate an XSL-FO document from book.xml.

Figure 3. Sample XSL-FO Transformation Configuration for book.xml

Step two is to use a Formatting Objects Processor (FOP) to transform your XSL-FO document into a PDF. One of the more popular open source FOPs is the Apache FOP [http://xmlgraphics.apache.org/fop/] . We'll use a third-party plug-in from Ahmadsoft [http://www.ahmadsoft.org/fopbridge.html] that integrates Apache FOP into Eclipse. After installing this plug-in, all that you need to do to render the XSL-FO document is run the FOP transformation. Figure 4, “Sample FOP Transformation” shows an example of running the FOP transformation.

Figure 4. Sample FOP Transformation

note

The example includes a sample Ant file [files/pdf/build.xml] that performs the same transformation as running the FOP transformation using the plug-in from Ahmadsoft [http://www.ahmadsoft.org/fopbridge.html]. An Ant script is a popular method of performing the publishing stage, and this example should give you a good starting point if you'd prefer to go this route. The DITA project already includes an Ant script (found in ant/sample_pdf.xml in DITA-OT) to perform this exact task on DITA source files.

As before, a Flash movie that shows the transformation is available here [files/DocBook-PDF.htm] .

Eclipse Infocenter

In our opinion, one of the coolest features of the DocBook and DITA projects is the generation of an Eclipse help plug-in (information center) from your source XML file. In order to perform this transformation in DocBook, you need to specify a few parameters and use the following style sheet: *eclipse/eclipse.xsl*. Figure 5, “Sample Eclipse Infocenter Transformation Configuration” shows a sample transformation configuration along with the correct parameters. To perform this transformation using a DITA source file, use the *ant/sample_eclipsehelp.xml* Ant file.

Figure 5. Sample Eclipse Infocenter Transformation Configuration

The complete list of DocBook XSL parameters for the Eclipse Infocenter transformation is located here [http://docbook.sourceforge.net/release/xsl/current/doc/html/rn22.html] .

The Flash movie that shows the Eclipse Infocenter DocBook transformation can be found here [files/DocBook-InfoCenter.htm] .

Current Limitations

Although we have shown that Eclipse's current XML authoring support is pretty good, there are a two noteworthy limitations.

The first is grammar and spell-checking. While these tools are commonplace in word-processing software, they do not yet exist for WTP's XML editor.

The second is a WYSIWYG editor for XML documentation formats such as DITA and DocBook (or a preview window). The lack of a sophisticated editor or a way to preview what you've written requires that you stop authoring and transform your document in order to view the results of your changes.

While neither of these limitations has been a show-stopper in our authoring process, our hope is that as Eclipse is recognized as an integrated documentation development environment, these limitations will be addressed.

Summary

In this article, we introduced the technical documentation process and showed that technical documentation development is possible in Eclipse. We worked through examples showing how to use Eclipse to aid the different phases of the technical documentation process. Although there is still a lot of room for improvement in this area we hope we've convinced you that technical documentation in Eclipse is both possible and already viable. It's now up to you in the technical documentation community to speak up, make it clear that Eclipse is being used for the authoring process, and push to get the current limitations addressed.

Acknowledgements

We'd like to thank:

- Sushma Patel and Anne James for correcting our horrible grammar.

- Don Day and Michael Priestly for their DITA expertise.

About the Authors

Chris Aniszczyk is a software developer at the IBM Austin Labs and works in Tivoli Security. He's a developer on the Gentoo Linux [http://www.gentoo.org] distribution and also a committer on the Eclipse Modeling Framework Technology (EMFT) [http://www.eclipse.org/emft] project.

Lawrence Mandel, a software developer at the IBM Toronto Laboratory, is the documentation lead and a committer for the Eclipse Web Tools Platform (WTP) [http://www.eclipse.org/webtools] project. He is also authoring a book with Arthur Ryman and Naci Dai about Java Web application development with Eclipse.

Resources

[docbookxsl] DocBook XSL Style Sheets [http://docbook.sourceforge.net/projects/xsl/] .

[dita] DITA Open Toolkit [http://dita-ot.sourceforge.net/] .

[toolchain] Wikipedia: Toolchain [http://en.wikipedia.org/wiki/Toolchain] .

[vex] Vex [http://vex.sf.net] .

[openoffice] OpenOffice [http://www.openoffice.org] .

[subversion] Subversion [http://subversion.tigris.org/] .

A. Appendix

The appendix contains a discussion about how this article was written (including the HTML style sheet so you can write your own eclipse.org article in DocBook). The appendix also reviews a couple of other editors out there for technical documentation in case WTP's XML editor doesn't suit your fancy.

The Article

This article was written in DocBook using WTP's XML editor. To transform the article into the correct format for eclipse.org, a style sheet was developed that extends the transformation included in the DocBook XSL project. The eclipse.org article style sheet can be downloaded here [files/article.xsl].

Eclipse.org is in the process of moving to a data driven format for articles. You can find more information about this process by following bug #115473 [https://bugs.eclipse.org/bugs/show_bug.cgi?id=115473]. We will be contributing our stylesheets for DocBook and DITA to this bug.

Editors

Part of the creation process involves editing the content of your XML document in an editor. The editor you use is a preference that is usually precious to the content creator (think EMACS versus VI). We decided to use the WTP XML editor as the editor for this article because of our familiarity with it and be-

cause both of us like working within Eclipse. However, we realize that there are other options for creating and editing content so we'll discuss of a couple of those options in the following sections.

Vex

Vex [vex] is an open source project that lets you edit XML files visually. Vex uses standard Document Type Definition (DTD) files to define document types and Cascading Style Sheets (CSS) to define document layout. In essence, Vex only requires that you have knowledge of CSS and DTDs in order to contribute a visual editor for XML files. The Vex editor can be seen in Figure A.1, “Vex DocBook editor screenshot” .

Figure A.1. Vex DocBook editor screenshot

OpenOffice

OpenOffice [openoffice] is a multi-platform open source office suite that is capable of visually editing DocBook and various other formats. OpenOffice is a popular editing choice because of its ability to open multiple document formats, including Microsoft Word, and then export the documents to DocBook.

OpenOffice doesn't fully support DocBook. An updated list of what portions of DocBook OpenOffice supports can be found on the OpenOffice site here [http://xml.openoffice.org/xmerge/docbook/DocBookTags.html] . The site also contains a getting started guide [http://xml.openoffice.org/xmerge/docbook/UserGuide.html] that will get you started with DocBook in OpenOffice.