IBM

developerWorks.

Automation for the people: Pushbutton documentation

Automate the generation of your developer and user documentation

Skill Level: Introductory

Paul Duvall (pduvall@gmail.com)
CTO
Stelligent Incorporated

10 Jun 2008

Project documentation is often one of the necessary evils in delivering a software product. But imagine being able to generate your documentation at the click of a button. In this installment of *Automation for the people*, automation expert Paul Duvall explains how you can use open source tools to automate the generation of Unified Modeling Language (UML) diagrams, build figures, entity-relationship diagrams (ERDs), and even user documentation.

I don't often run across developers who tell me they like writing documentation for their software-development projects. Unless you never expect to leave your project, will always be the lone developer, or have no users — probably not good signs for a project — you need a semipermanent way to communicate the software's purpose to others. Some developers misconstrue the Agile Manifesto's "Working software *over* comprehensive documentation" statement to mean that *no* documentation is necessary (see Resources). On the other hand, you don't need to burden users or other developers with unnecessary documentation. I typically look for the happy medium. You guessed it: This article shows you how *automation* can streamline the process (and reduce the misery) of generating project documentation.

About this series

As developers, we work to automate processes for end-users; yet, many of us overlook opportunities to automate our own development processes. To that end, *Automation for the people* is a

© Copyright IBM Corporation 1994, 2008. All rights reserved.

series of articles dedicated to exploring the practical uses of automating software development processes and teaching you *when* and *how* to apply automation successfully.

In my experience, two key problems plague documentation in software development. The first is the likelihood that no one is reading it. A second common problem is that almost the moment documentation is written, it is outdated. The two problems are causally connected. If documentation were current, people would be more likely to read it. Automating your documentation's generation will help you keep it up to date and thereby make it more useful for your software's users.

Other types of documentation could probably benefit from automation, but I'll focus on automating the documentation tasks that have often caused me anguish (see Resources for links to the tools in this list):

- Generating **UML** diagrams based on the current source code, using UMLGraph.
- Creating **entity-relationship diagrams (ERDs)** to document the tables and relationships in a database, using SchemaSpy.
- Producing **build diagrams** of Ant build targets and their relationships, using Grand.
- Spawning **source-code documentation**, using Doxygen.
- Making the **user documentation**, using DocBook.

I'll use a general format in which I:

1. Describe the issues involved in performing each task manually.
2. Present an automation example that uses Apache Ant in conjunction with the relevant document- or diagram-generating tool.
3. Show an image of the script-generated documentation that's based on the code examples.

As usual in this series, all of the examples use freely available and open source tools that you can use in your projects. Some of the tools (for example, UMLGraph and Grand) use an accompanying tool called GraphViz, which consumes a .dot file generated by the particular tool.

Reverse-engineering code into UML

© Copyright IBM Corporation 1994, 2008. All rights reserved.

I've worked on projects that had some of the most wonderfully designed UML diagrams you've ever seen — at the beginning of the project. Trouble was, in one particular case, the technical lead didn't always keep the model up to date with the source code. Or he spent valuable time *manually* reverse-engineering the source into the model. Neither approach was an optimal solution. A beautifully constructed UML diagram means nothing if the code that's checked into the version-control repository is not accurately represented in the model. If the decisions you make aren't based on the actual code, you're likely to experience problems downstream.

You can make diagrams meaningful for decision making, easier to create, and up to date by including diagram generation in the build process and setting up a Continuous Integration (CI) environment to create the diagrams with every change (or periodically).

Listing 1 uses Ant, UMLGraph, and Graphviz to document source code:

Listing 1. Ant script using UMLGraph documentation tool

```
<property name="reports.dir" value="${basedir}/reports"/>
  <mkdir dir="${reports.dir}"/>
  <path id="project.path">
    <pathelement path="${basedir}/src"/>
    <pathelement path="${basedir}/lib"/>
  </path>
  <javadoc sourcepath="${basedir}/src" destdir="${reports.dir}"
    classpathref="project.path" access="private">
    <doclet name="org.umlgraph.doclet.UmlGraphDoc"
          path="UMLGraph.jar">
        <param name="-attributes" />
        <param name="-enumerations" />
        <param name="-enumconstants" />
        <param name="-operations" />
        <param name="-qualify" />
        <param name="-types" />
        <param name="-visibility" />
    </doclet>
  </javadoc>
  <apply executable="dot" dest="${reports.dir}" parallel="false">
    <arg value="-Tpng"/>
    <arg value="-o"/>
     <targetfile/>
     <srcfile/>
     <fileset dir="${reports.dir}" includes="*.dot"/>
     <mapper type="glob" from="*.dot" to="*.png"/>
  </apply>
```

In Listing 1, I use UMLGraph in combination with Javadoc to generate some basic UML class diagrams within a Javadoc HTML report. When calling `UmlGraphDoc`, I pass in the following attributes to customize the information that's displayed in each class diagram:

> **Graphing with GraphViz**
> For UMLGraph to work properly, you must have installed the Graphviz tool (see Resources), and the Graphviz .dot file must be

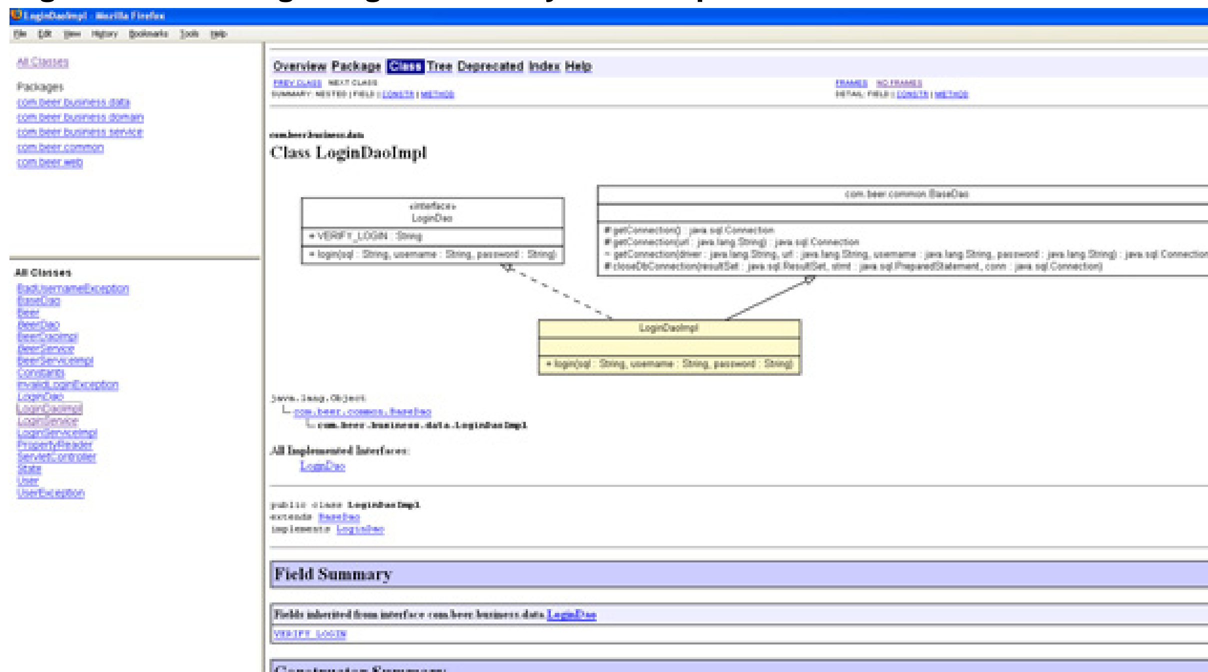
© Copyright IBM Corporation 1994, 2008. All rights reserved.

available in your machine's system path.

- **`-attributes`** displays fields for a class.
- **`-enumerations`** shows different enumerations.
- **`-enumconstants`** shows possible values for enumerations.
- **`-operations`** displays the Java methods for a class.
- **`-qualify`** shows fully qualified class names.
- **`-types`** displays data types for arguments and return types.
- **`-visibility`** shows field and method modifiers: `public`, `protected`, `private`, or `default`.

Figure 1 shows an example UML diagram for the `LoginDaoImpl` class and its relationships, as generated to HTML by UMLGraph:

Figure 1. UML diagram generated by UMLGraph



UMLGraph can generate more complex relationships and additional detail. But even the rather rudimentary UML class diagram in this simple example provides a lot of information. It also serves the purpose of providing a quick visual representation of the software based on the current code base. It prevents "Well, that's what the code is *supposed* to look like" types of statements and can enable opportunities for better decision making (see Resources for a link describing the many attributes that can be used to customize UMLGraph's output).

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Documenting the database

Just as automating the creation of the latest UML diagrams is beneficial, so too is automating a visual representation of your database. The entity-relationship diagram (ERD) is the most popular diagram type for visualizing databases. Most tools that create ERDs (ERWin, for example, among many others) require manual generation of the ERD. Although the tool I'll demonstrate, SchemaSpy, is no match for some of the more sophisticated tools available, it can give you a high-level ERD view of your database — along with constraints, relationships, and so on. What's more, by running it with an automated build, you can painlessly check the latest representation of the Data Definition Language (DDL) from your version-control repository.

The Ant script in Listing 2 uses the SchemaSpy tool to create a file formatted for Javadoc:

Listing 2. Using SchemaSpy with Ant and Javadoc

```
<property name="reports.dir" value="${basedir}"/>
<java jar="schemaSpy_3.1.1.jar"
  output="${reports.dir}/output.log"
  error="${reports.dir}/error.log"
  fork="true">
  <arg line="-t mysql"/>
  <arg line="-host localhost"/>
  <arg line="-port 3306"/>
  <arg line="-db brewery"/>
  <arg line="-u root"/>
  <arg line="-p sa"/>
  <arg line="-cp mysql-connector-java-5.0.5-bin.jar"/>
  <arg line="-o ${reports.dir}"/>
</java>
```

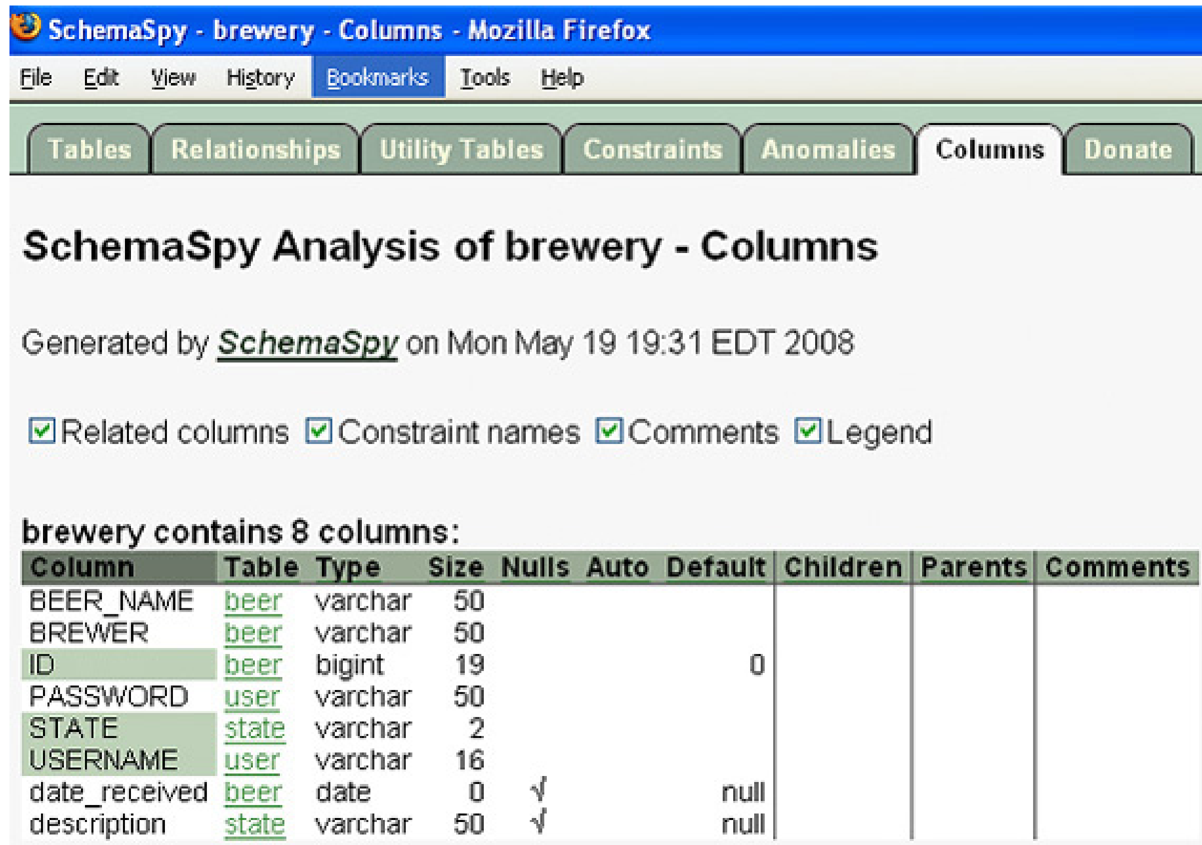
Listing 2 uses the `java` Ant task to call SchemaSpy, passing a number of attributes:

- **`-t`** is the type of database. (Valid values are `mysql`, `ora`, `db2`, and the like.)
- **`-host`** is name of the machine where the database is hosted.
- **`-port`** is the port number for the database URL.
- **`-u`** is the database username.
- **`-p`** is the database password.
- **`-cp`** is the classpath (used to indicate the location of the database driver JAR file).
- **`-o`** is the output directory location.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

These command-line attributes for SchemaSpy are used to generate the HTML file that displays the ERD, as shown in Figure 2:

Figure 2. ERD created with SchemaSpy and Ant



Column	Table	Type	Size	Nulls	Auto	Default	Children	Parents	Comments
BEER_NAME	beer	varchar	50						
BREWER	beer	varchar	50						
ID	beer	bigint	19			0			
PASSWORD	user	varchar	50						
STATE	state	varchar	2						
USERNAME	user	varchar	16						
date_received	beer	date	0	√		null			
description	state	varchar	50	√		null			

By using a combination of tools, executing ERD-generation scripts against the database as part of a build, and scheduling ERD generation, you gain a quick and easy way to make many database decisions during development.

Diagramming your builds

Often, build scripts have complex relationships among dependent targets. I've seen many build scripts of more than 1,000 lines containing unnecessary duplication. It can be difficult to get a handle on them without a high-level view of the automated build. An effective way to determine quickly what a build is doing or not doing is to create a diagram of its targets and relationships. A visual model of your Ant targets can help you make more effective and expedient decisions to improve your build scripts' designs.

A combination of Ant, Grand, and Graphviz lets you create an accurate visual representation of your build targets. A number of tools purport to visualize Ant targets. An advantage of Grand is that it uses the Ant API, so it creates a diagram

© Copyright IBM Corporation 1994, 2008. All rights reserved.

whether or not the Ant script is fully working.

The Ant script in Listing 3 demonstrates the use of the Grand tool to create a visual representation of an Ant script's targets, including the dependencies among the targets:

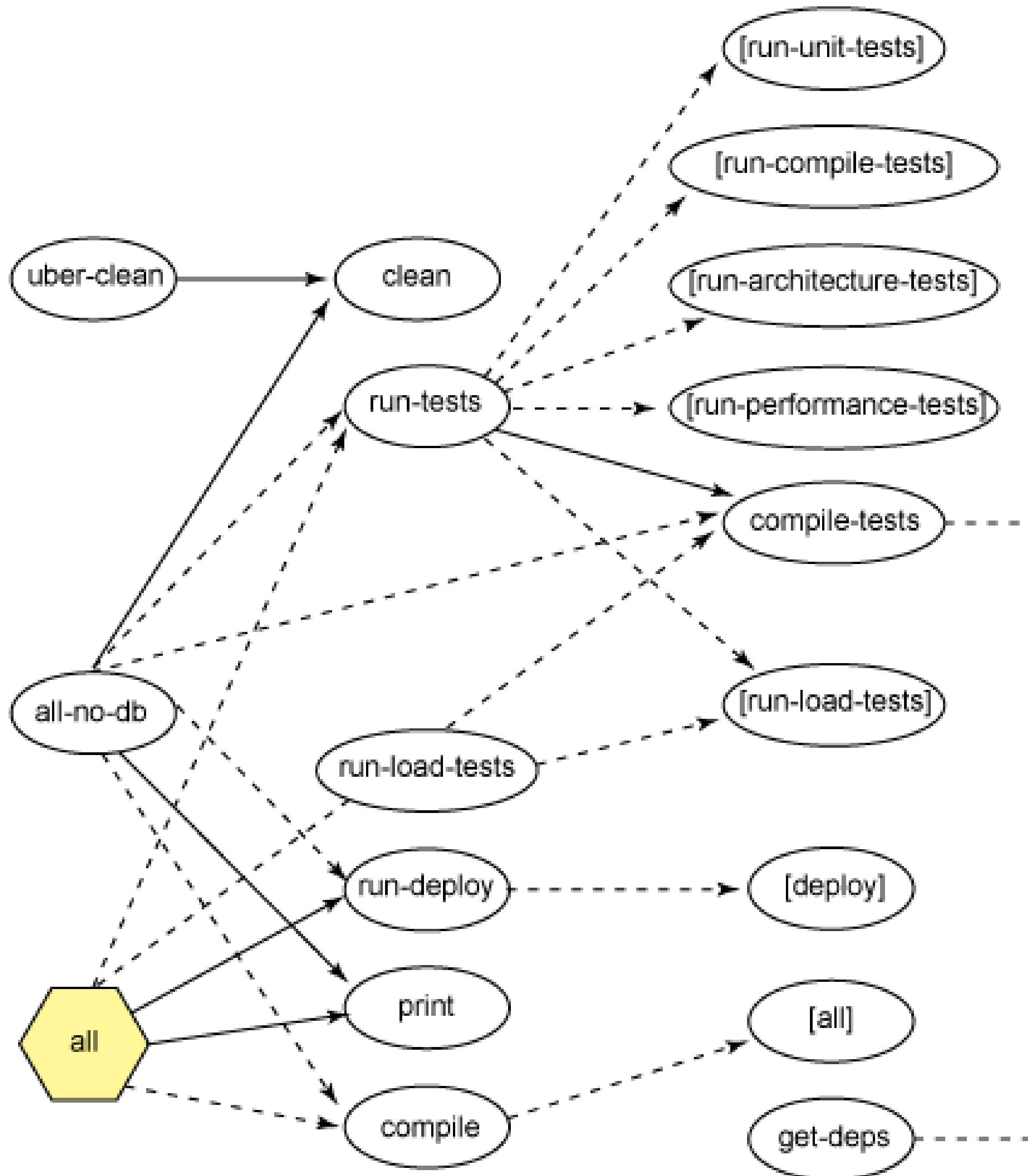
Listing 3. Creating diagram of Ant targets using Ant and Grand

```
<target name="create-ant-diagram">
  <property name="file.type" value="pdf" />
  <typedef resource="net/ggtools/grand/antlib.xml"
    classpath="grand-1.8.jar"/>
  <grand output="build.dot" buildfile="${basedir}/build.xml"/>
    <exec executable="dot" >
      <arg line="-T${file.type} -Gsize=11.69,8.27 -Grotate=90 -o build.${file.type}
    ${grand.output.file}"/>
    </exec>
</target>
```

In Listing 3, first I create the task by pointing to `grand-1.8.jar`. Next, I provide the name of the build file I'm analyzing, in the `buildfile` attribute. The `output` attribute is the Graphviz file (I named it `build.dot`) I'm creating. Finally, I call the `dot` executable to generate the PDF of the build targets of the `buildfile`. Figure 3 shows an example of this file:

Figure 3. Diagram of Ant targets generated by Ant, Grand, and GraphViz

© Copyright IBM Corporation 1994, 2008. All rights reserved.



I can use these diagrams to analyze a completed project or to refine the build process for a project that's still in development.

Documenting your code

If I want to remember what or possibly why I created a new Java class or method, I write a Java code comment. But if I want to get a more comprehensive view of the classes, methods, and attributes, I need a code documentation tool. For years, the

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Java platform has provided Javadoc as a way of generating code documentation for all classes in a code base. In fact, the `javadoc` task comes standard in Ant, and it generates code documentation. However, the onus is on the developer to embed a collection of unsightly HTML tags within the Java source code comments. For example, with Javadoc, your code comments might look like Listing 4:

Listing 4. Code comments for Javadoc

```
/**
  This is <i>the</i> LoginServiceImpl  class.
  <p>
  Refer to <a href="url.html">
  LoginService overview</a> for more details.
  <p>
  There is one type of supported {@link LoginServiceImpl }:
  <ul>
    <li>{@link LoginServiceImpl } (this class)</li>
  </ul>
*/
```

When you use Doxygen, code comments are much more terse, without the embedded HTML, as demonstrated in Listing 5:

Listing 5. Ant script using Doxygen documentation tool

```
/**
This is <i>the</i> LoginServiceImpl class.
Refer to \ref LoginService for more details.
There is one type of supported LoginService:
- LoginServiceImpl (this class)
*/
```

Both Javadoc and Doxygen provide support for automating the generation of code documentation, but Doxygen makes it easier by offering a more intuitive approach to annotating source code using comments. The Ant snippet in Listing 6 shows the most basic use of Doxygen:

Listing 6. Ant script using Doxygen documentation tool

```
<target name="generate-doxygen">
  <taskdef name="doxygen" classname="org.doxygen.tools.DoxygenTask"
    classpath="ant-doxygen.jar" />
  <doxygen configFilename="Doxyfile">
  <property name="INPUT" value="${src.dir}" />
  <property name="RECURSIVE" value="yes" />
  </doxygen>
</target>
```

In Listing 6, the `configFilename` value of `Doxyfile` is the name of a configuration file that is generated through Doxygen; you can customize it for your own use. Figure 4 shows an example HTML report generated for the

© Copyright IBM Corporation 1994, 2008. All rights reserved.

`LoginDaoImpl` class by the Doxygen example in Listing 6:

Figure 4. HTML Doxygen documentation

By creating Doxygen documentation as part of an automated build, you can get the most up-to-date usage information on an API.

User documentation

So far, you may have been thinking that generating technical documentation automatically is nice but that your pain lies in creating *user* documentation. In writing my own user documentation, I found I was spending quite a bit of time simply copying and pasting text throughout the document. When I started to analyze the process, I determined that I can automate items such as version numbers and file names by defining them in a single source. And by separating the content from the formatting using XML and Extensible Stylesheet Language (XSL), I can make wholesale changes to the document's look and feel with a simple change to a stylesheet.

DocBook, a tool that's been available for years, lets you define documents in XML and produce the document in many formats, including HTML and PDF. DocBook provides a schema that defines the templates for writers to use.

Listing 7 uses a combination of Ant, a DocBook XSL file, and XSL Formatting Objects (XSL-FO) to generate a PDF of my user documentation (see Resources):

Listing 7. Creating a PDF using Ant, DocBook, and FO

```
<target name="pdf" depends="init" description="Generates PDF files from DocBook XML">
  <property name="fo.stylesheet" value="${docbook.xsl.dir}/fo/docbook.xsl" />
  <xslt style="${fo.stylesheet}" extension=".fo" basedir="${src.dir}" destdir="${fo.dir}">
    <classpath refid="xalan.classpath" />
    <include name="${guide}.xml" />
```


© Copyright IBM Corporation 1994, 2008. All rights reserved.

```
  </xslt>
  <property name="fop.home" value="lib/fop-0.94" />
  <taskdef name="fop" classname="org.apache.fop.tools.anttasks.Fop">
    <classpath>
      <fileset dir="${fop.home}/lib">
        <include name="*.jar" />
      </fileset>
      <fileset dir="${fop.home}/build">
        <include name="fop.jar" />
        <include name="fop-hyph.jar" />
      </fileset>
    </classpath>
  </taskdef>
  <fop format="application/pdf" fofile="${fo.dir}/${guide}.fo"
    outfile="${doc.dir}/${guide}.pdf" />
</target>
```

Figure 5 shows a section of a PDF generated from the script in Listing 7:

Figure 5. PDF of user-guide documentation

72

Chapter 30. Installation

Requirements

Tested Environments

The Brewery 2.0.5 installation has been tested on Linux Red Hat Enterprise Linux AS 5 32-bit and the Windows 2003 environments. It's possible that the installation may work in other Linux and Windows environments, it has only been tested in these environments.

Required Software

Many of the servers and services that make up Brewery 2.0.5 are automatically installed as part of this installation. However, there are certain tools that must be manually installed and configured - as listed in Table 2. The software name, version, description, and URL hyperlinks (for download) are indicated in the table.

Because all of my content is defined in text-based XML, I can easily write scripts that make wide-sweeping changes to the documentation. Further, I can generate these documents as part of the build process, thus providing a single standard source to

© Copyright IBM Corporation 1994, 2008. All rights reserved.

define elements such as versions, definitions, abbreviations, file names, and directories.

Documentation doesn't need to be manual

I hope this article has convinced you that effective approaches to automating the generation of documentation are available. In my view, the most important aspect of documentation is communication, not the process of generating the document. The automation tools and techniques I've described can help you keep your documentation accurate and up to date, thereby eliminating the main culprit — outdated information — behind documentation's chronic underuse.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

Resources

Learn

- Manifesto for Agile Software Development: Read the Agile Manifesto.
- "Drawing UML Diagrams with UMLGraph" (Diomidis Spinellis, Athens University of Economics and Business): This article includes a list of class diagram options for UMLGraph.
- Graphviz Gallery (graphviz.org): A gallery of example diagrams to use with Graphviz.
- "10 Minutes to document your code" (Peter Chen, The Code Project, January 2003): An article describing how to set up Doxygen.
- Doxygen Ant blog: Get the latest information from the creator of the Doxygen Ant task.
- "*Java theory and practice*: I have to document THAT?" (Brian Goetz, developerWorks, August 2002): Get some guidelines on how to write more useful Javadoc.
- XSL and FO: Home base for these World Wide Web Consortium stylesheet-language recommendations.
- *DocBook XSL: The Complete Guide* (4th Edition) (Bob Stayton, Sagehill Enterprises, 2007): This book covers all aspects of DocBook publishing tools, including installing, using, and customizing the stylesheets and processing tools.
- *Java Power Tools* (John Ferguson Smart, O'Reilly, 2008): Chapter 30 of this phenomenal book covers SchemaSpy, UMLGraph, and Doxygen.
- DocBook resources on developerWorks: Learn more about expressing technical documentation in XML.
- *Automation for the people* (Paul Duvall, developerWorks): Read the complete series.
- Browse the technology bookstore for books on these and other technical topics.
- developerWorks Java technology zone: Hundreds of articles about every aspect of Java programming.

Get products and technologies

- Ant: Download Ant and start building software in a predictable and repeatable manner.
- Grand: Download Grand to diagram Ant build scripts.

© Copyright IBM Corporation 1994, 2008. All rights reserved.

- SchemaSpy: Download SchemaSpy to create ERDs for your database.
- Doxygen: Download Doxygen to generate detailed code documentation.
- UMLGraph: Download UMLGraph to reverse-engineer source code into UML diagrams automatically.
- DocBook: Download DocBook to generate content in different formats.

Discuss

- Improve Your Code Quality discussion forum: Regular developerWorks contributor Andrew Glover brings his considerable expertise as a consultant focused on improving code quality to this moderated discussion forum.
- Accelerate development space: Regular developerWorks contributor Andrew Glover hosts a one stop portal for all things related to developer testing, Continuous Integration, code metrics, and refactoring.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Paul Duvall
Paul Duvall is the CTO of Stelligent Incorporated, an Agile consultancy that helps development teams deliver production-ready software. He is the co-author of the Addison-Wesley Signature Series book, *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley Professional, 2007; Jolt Award 2008 winner). He also contributed to the *UML 2 Toolkit* (Wiley, 2003) and the *No Fluff Just Stuff Anthology* (Pragmatic Programmers, 2007).

© Copyright IBM Corporation 1994, 2008. All rights reserved.