This story appeared on JavaWorld at
http://www.javaworld.com/javaworld/jw-02-2006/jw-0227-maven.html

Get the most out of Maven 2 site generation

Steps for creating a Maven-based Website

By John Ferguson Smart, JavaWorld.com, 02/27/06

Team communication is an essential part of any project. And wasting time looking for technical project information can be costly and frustrating. Clearly, any IT project will benefit from having its own dedicated technical Website.

That's where the Maven site generator steps in. With little effort, you can have a professional-quality, low maintenance project Website up and running in no time. Maven lets you generate a one-stop center of information about your project, including:

- General project information such as source repositories, defect tracking, and team members
- Unit test and test coverage reports
- Automatic code reviews with Checkstyle and PMD
- Configuration and versioning information
- Dependencies
- Javadocs
- Source code in indexed and cross-referenced HTML format
- And much more

Maven sites are frequently used on open source projects (see Resources for some examples). A typical project site contains information about the project, largely derived from the pom.xml file, some generated reports (unit tests, Javadocs, Checkstyle code audits, etc.), as well as some project-specific content matter.

In this article, we will walk though what you need to know to get a Maven site up and running in minimal time. Note: The source code used in this tutorial can be downloaded from Resources.

Iteration 1: The project information section

Your Maven site will be one of the first places a newcomer will look to get to know your project, and the project information page is the place they will expect to find a summary of your project's organization. This part of the Website is built entirely using information found in the pom.xml file. The default pom.xml file created will look something like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v400.xsd>
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.javaworld.hotels</groupId>
 <artifactId>HotelWeb</artifactId>
 <packaging>jar</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>Maven Quick Start Archetype</name>
 <url>http://maven.apache.org</url>


 <dependencies>
   <dependency>
     <groupId>junit</groupId>
```

```
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Now you will want to add some details about your project. All zones are optional, but you should put as much information as is relevant to your project. The following sections discuss some useful items to add.

Project name and description

Add an appropriate project name, a description, and the project site URL. This appears on your project's homepage, so don't be stingy:

```
  ...
   <name>Hotel Database tutorial application</name>
   <url>http://your.project.url</url>
   <description>Write a few paragraphs to say what your project is about.</description>
```

Issue tracking

Now put the name and URL of your project's issue management system (Bugzilla, Jira, Scarab, or your own favorite issue tracking system), using the `issueManagement` tag:

```
  ...
   <issueManagement>
      <system>Bugzilla</system>
      <url>https://bugzilla.wakaleo.com/</url>
   </issueManagement>
```

Continuous integration

If your project uses a continuous integration tool of some sort, such as CruiseControl or Continuum, you can provide details in the `ciManagement`
tag, as shown in the code below. (If you are not, consider using one!) Maven 2 integrates well with Continuum: you can install a Maven 2 project onto a Continuum server just by providing the pom.xml file.

```
  ...
   <ciManagement>
      <system>Continuum</system>
      <url>http://integrationserver.wakaleo.com/continuum</url>
      <notifiers>
         <notifier>
            <type>mail</type>
            <address>duke@wakaleo.com</address>
         </notifier>
      </notifiers>
   </ciManagement>
```

The project team

People like to know who they are working with, especially these days, when a project team can be spread across organizations and continents. In the `developers` section, list team member details. The `timezone` field is useful for international teams; this field is offset from Greenwich Mean Time (GMT), or London time, and lets people see what time it is wherever the team member is located. For example, -5 is for New York time, +1 for Paris, and +10 for Sydney.

```
  ...
   <developers>
      <developer>
         <id>duke</id>
         <name>Duke Java</name>
         <email>duke@wakaleo.com</email>
         <roles>
            <role>Project Manager</role>
            <role>Architect</role>
         </roles>
         <organization>Acme.com</organization>
         <timezone>-5</timezone>
       </developer>
   </developers>
```

Mailing lists

If your project uses mailing lists, describe them in the `mailingLists` page. A typical mailing list configuration might look like this:

```
  ...
   <mailingLists>
     <mailingList>
       <name>HotelDatabase project mailing list</name>
       <subscribe>dev-subscribe@wakaleo.com</subscribe>
       <unsubscribe>dev-unsubscribe@wakaleo.com</unsubscribe>
      <post>dev@wakaleo.com</post>
      <archive>http://mail-archives.wakaleo.com/modmbox/dev/</archive>
     </mailingList>
   </mailingLists>
```

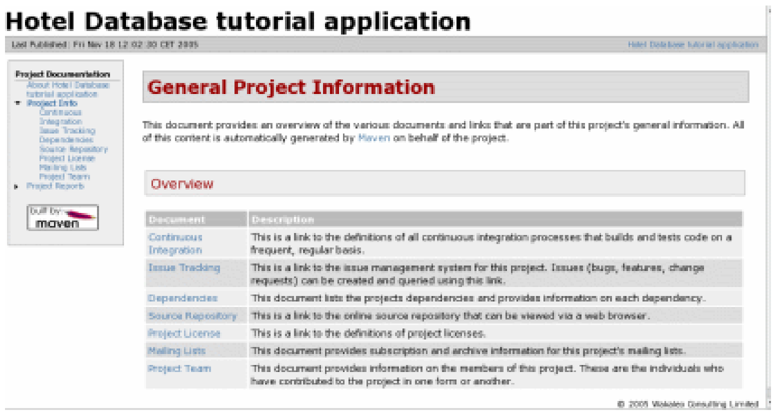
The source repository

Another vital part of any project is the source repository. The `scm` tag lets you document the configuration of your source repository, both for the Maven Website and for use by other plug-ins. If you are using CVS or Subversion, the source repository page will also give detailed, tool-dependent instructions on how to use the repository. Here is an example of a typical SCM (software configuration management) configuration:

```
  ...
   <scm>
      <connection>scm:svn:http://svn.wakaleo.com/hoteldatabase/</connection>
      <developerConnection>scm:svn:http://svn.wakaleo.com/hoteldatabase/</developerConnection>
      <url>http://svn.wakaleo.com/viewcvs.cgi/hoteldatabase/</url>
   </scm>
```

Now you can try out your new Maven site! The command to generate the Maven site is: `mvn site`.

Your site will be generated in `target/site`. Take a look at the project information link; you should find everything you just entered and more (see Figure 1)!



Hotel Database tutorial application

Last Published: Fri Nov 18 12:02:30 CET 2005 Hotel Database tutorial application

Project Documentation
- About Hotel Database tutorial application
- Project Info
  - Continuous Integration
  - Issue Tracking
  - Dependencies
  - Source Repository
  - Project License
  - Mailing Lists
  - Project Team
- Project Reports

built by maven

General Project Information

This document provides an overview of the various documents and links that are part of this project's general information. All of this content is automatically generated by Maven on behalf of the project.

Overview

Document	Description
Continuous Integration	This is a link to the definitions of all continuous integration processes that builds and tests code on a frequent, regular basis.
Issue Tracking	This is a link to the issue management system for this project. Issues (bugs, features, change requests) can be created and queried using this link.
Dependencies	This document lists the projects dependencies and provides information on each dependency.
Source Repository	This is a link to the online source repository that can be viewed via a web browser.
Project License	This is a link to the definitions of project licenses.
Mailing Lists	This document provides subscription and archive information for this project's mailing lists.
Project Team	This document provides information on the members of this project. These are the individuals who have contributed to the project in one form or another.

© 2005 Wakaleo Consulting Limited

Figure 1. The project information zone in your new Maven 2 project site

Iteration 2: Add reports

Maven also provides an extensive range of plug-ins for automatic report generation. Report generation in Maven 2 is easy: you just add the report plug-ins you need into the `reporting` section at the end of the pom.xml file.

Javadocs

Most likely, you will want to start by publishing your classes' Javadocs. This is as simple as adding `javadoc` to the `reporting` section of your pom.xml file. While you're at it, add the `jxr` plug-in as well; this will generate an indexed and cross-referenced HTML version of your source code:

```
<reporting>
   <plugins>
     <plugin>
       <artifactId>maven-javadoc-plugin</artifactId>
     </plugin>
     <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>jxr-maven-plugin</artifactId>
     </plugin>
   </plugins>
 </reporting>
```

Unit-test reports

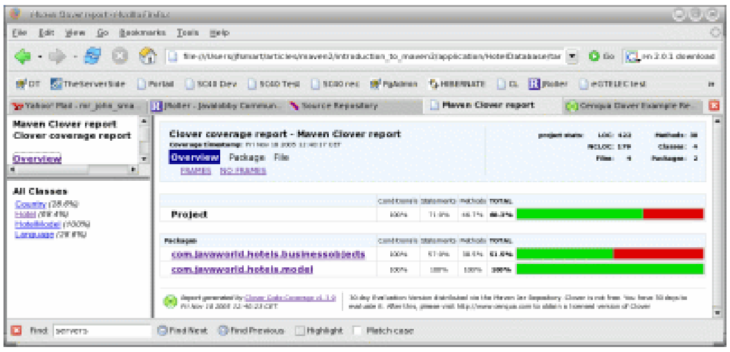
Writing unit tests for as much code as possible is highly recommended. Maven fully integrates unit testing into the build process—by default, all unit tests run at each build. Publishing test results for all to see is beneficial, as it tends to encourage developers to fix any broken unit tests.

The `surefire` plug-in runs all the unit tests and generates a detailed report:

```
<reporting>
   <plugins>
     ...
     <plugin>
       <artifactId>maven-surefire-plugin</artifactId>
     </plugin>
   </plugins>
 </reporting>
```

Test coverage

Test coverage can be a useful indication of the quality of your unit tests. It basically tells you how much of your code is actually run by your unit tests, which, in turn, can give you a good idea of the tests' quality. In some projects, requiring a certain percentage of test coverage in all classes is recommended. Tools such as Clover (a robust commercial test coverage tool) or Cobertura (a promising open source tool that replaces the Maven 1 JCoverage plug-in) generate test coverage reports. Clover has a proven ability to perform test coverage for large projects, where other tools would often fail. Figure 2 illustrates a Clover test coverage report.



Clover coverage report - Maven Clover report

	Conditionals	Statements	Methods	TOTAL
Project	100%	[illegible]	[illegible]	[illegible]

Packages	Conditionals	Statements	Methods	TOTAL
com.javaworld.hotels.businessobjects	100%	[illegible]	[illegible]	[illegible]
com.javaworld.hotels.model	100%	100%	100%	100%

Figure 2. A Clover-generated test coverage report

To date, only the Clover plug-in is available for Maven 2, though Cobertura is in development at the time of this writing.

To add a Clover report to your Maven site, simply add the `maven-clover-plug-in` to the `reporting` section:

```
<reporting>
   <plugins>
     ...
     <plugin>
       <artifactId>maven-clover-plugin</artifactId>
     </plugin>
   </plugins>
 </reporting>
```

Code analysis

Automatic code analysis is a useful way of improving code quality and encouraging good coding habits. Checkstyle runs a wide range of tests aimed at enforcing coding standards and best practices. PMD concentrates more on semantic errors and potential bugs. Both can provide useful information, though you may have to fine-tune them (especially Checkstyle) to obtain only the errors meaningful for your project.

Sometimes you will need to pass parameters to a plug-in. In Maven 2, you can do this using the `configuration` tag, which will inject the parameter value into the plug-in. In the `pmd` plug-in, set the `targetjdk` parameter to 1.5 so that PMD will be able to handle Java 1.5 source code. You can also specify other parameters, such as the rules to be used, the output format, and whether hyperlinks to the code source should be generated:

```
<reporting>
   <plugins>
     ...
     <plugin>
        <groupId>org.apache.maven.plugins</groupId>


        <artifactId>maven-pmd-plugin</artifactId>
        <configuration>
           <targetjdk>1.5</targetjdk>
           <rulesets>
              <ruleset>/rulesets/basic.xml</ruleset>
              <ruleset>/rulesets/controversial.xml</ruleset>
           </rulesets>
           <format>xml</format>
           <linkXref>true</linkXref>
           <sourceEncoding>utf-8</sourceEncoding>


           <minimumTokens>100</minimumTokens>
        </configuration>
     </plugin>
   </plugins>
 </reporting>
```

Change and configuration management

Documenting changes is important in any project. Maven 2 provides a couple of useful features to make this task a little easier.

The `changes-maven-plugin`
plug-in uses a special XML file (src/changes/changes.xml) to track releases and changes in each release. This file looks something like this:

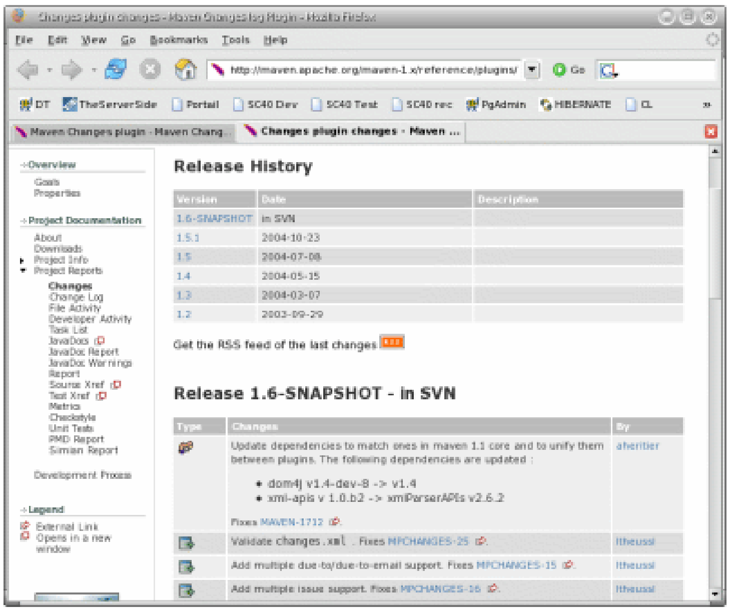
```
 <?xml version="1.0" encoding="ISO-8859-1"?>
<document>
   <properties>
      <title>Hotel Database tutorial application</title>
      <author email="duke@wakaleo.com>Duke Java</author>
   </properties>
   <body>
      <release version="current" description="Current work version>
           <action dev="Duke Java" type="add>
               A new cool feature.
           </action>
         </release>
      <release version="1.0.1" date="2005-11-18" description="Release fix>
           <action dev="Duke Java" type="add>
               Added a cool feature.
           </action>
           <action dev="Duke Java" type="fix" issue="1254>
               Fixed a nasty bug.
           </action>
           <action dev="Duke Java" type="delete>
               Removed a feature that nobody liked.
           </action>
         </release>
   </body>
</document>
```

Here, you list your releases and describe the actions associated with each release: a new feature or evolution (`add`), a bug fix (`fix`), or something removed (`delete`). You should detail the modification, who made the change, and what issue was addressed. Using this file gives a clearer and more readable record of changes and release history.

Now add the `changes` plug-in to the Maven 2 reports:

```
 <reporting>
    <plugins>
      ...
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>changes-maven-plugin</artifactId>
      </plugin>
    <plugins>
  </reporting>
```

Figure 3 shows an example of a real-world change report.



Release History

Version	Date	Description
1.6-SNAPSHOT	in SVN	
1.5.1	2004-10-23	
1.5	2004-07-08	
1.4	2004-05-15	
1.3	2004-03-07	
1.2	2003-09-29	

Get the RSS feed of the last changes

Release 1.6-SNAPSHOT - in SVN

Type	Changes	By
	Update dependencies to match ones in maven 1.1 core and to unify them between plugins. The following dependencies are updated : • dom4j v1.4-dev-8 -> v1.4 • xml-apis v 1.0.b2 -> xmlParserAPIs v2.6.2 Fixes MAVEN-1712.	aheritier
	Validate changes.xml . Fixes MPCHANGES-25.	ltheussl
	Add multiple due-to/due-to-email support. Fixes MPCHANGES-15.	ltheussl
	Add multiple issue support. Fixes MPCHANGES-16.	ltheussl

Figure 3. A real-world change report

Another more development-oriented option is to use your SCM repository to track changes. The `changelog` plug-in generates a nice report describing which files have been changed and by whom:

```
<reporting>
   <plugins>
     ...
     <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>changelog-maven-plugin</artifactId>
     </plugin>
   </plugins>
 </reporting>
```

Finally, if you use `@todo` tags to remind you of things to be done (which is a good coding practice), the `taglist` report will generate a list of all the items marked `@todo` or `TODO`:

```
<reporting>
   <plugins>
     ...
     <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>taglist-maven-plugin</artifactId>
     </plugin>
   <plugins>
 </reporting>
```

Add specific site content

You can also add your own original content to your site. You can put FAQ pages, technical documentation, or whatever takes your fancy.

Site content is stored in the `src/site` directory and organized in three main directories, as shown in this example:

```
- src/
  + site/
     + apt/
     |  + vision-statement.apt
     |     ...
     |
     + fml/
```

```
      |  + faq.fml
      |     ...
      |
      + xdoc/
      |  + best-practices.xml
      |     ...
      |
      + site.xml
```

To define site layout and navigation, you must write a site descriptor (placed in the site.xml file). This file basically describes the banners and menus to appear on your site. In our simple example, this file takes the following form:

```
 <?xml version="1.0" encoding="ISO-8859-1"?>
<site>
  <bannerLeft>
    <name>Hotel Database</name>
    <src>http://maven.apache.org/images/apache-maven-project.png</src>
    <href>http://maven.apache.org/</href>
  </bannerLeft>
  <bannerRight>
    <src>http://maven.apache.org/images/maven-small.gif</src>
  </bannerRight>
  <body>
    <links>
      <item name="Maven" href="http://maven.apache.org/"/>
      <item name="Apache" href="http://www.apache.org/"/>
    </links>


    <menu name="The Hotel Database project>
      <item name="Vision Statement" href="/vision-statement.html"/>
      <item name="Best Practices" href="/best-practices.html"/>
      <item name="FAQs" href="/faqs.html"/>
    </menu>


    ${reports}


  </body>
</site>
```

Supported file formats

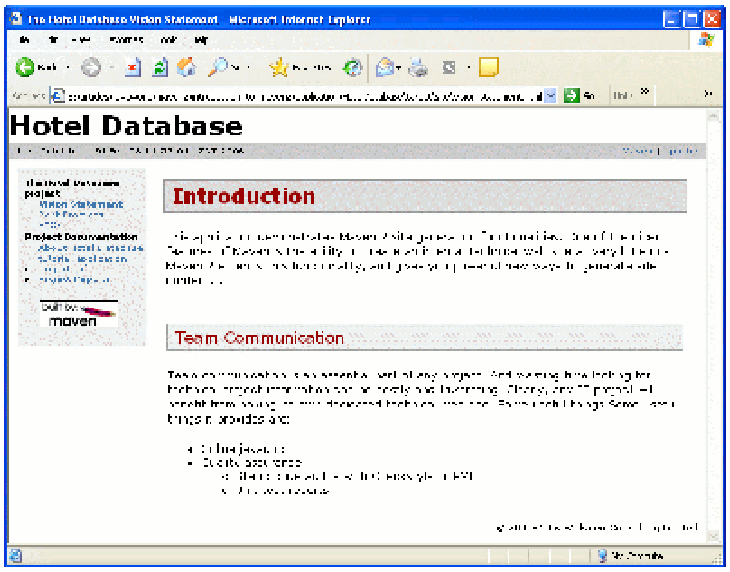
Site content can be added in a variety of formats. The traditional Maven documentation format is XDoc, a loosely structured general-purpose XML format for site content. XDoc files are stored in the `xdoc` directory. XDoc resembles XHTML and will prove familiar to anyone knowing HTML. A simple example is shown here:

```
 <document>
     <properties>
       <author email="user@company.com> The Wakaleo Team< /author>
       <title> The Hotel Database Vision Statement< /title>
     </properties>
     <body>


       <section name="Introduction">
         <p>
         This application demonstrates Maven 2 site generation functionalities. One of the nicer f
         at very little cost. Maven 2 extends this functionality, and gives you powerful new ways
         to generate site content...
         </p>


          <subsection name="Team Communication">


            <! --Team communication is an essential part of any project. And wasting
     time looking for technical project information can be costly and frustrating. Clearly,
    any IT project will benefit from having its own dedicated technical web site. Some
    useful things it provides are:
         <ul>
           <li>Online javadoc</li>
          <li>Quality assurance</li>
           <ul>
             <li>Static code audits with Checkstyle or PMD
```

Maven 2 introduces a new format, the APT (almost plain text) format, designed to be more convenient for writing technical site content. The APT format is a wiki-like text format handy for writing simple structured documents and for using simple text formatting and indentation rules. The APT files (`*.apt`) go in the `apt` directory. A full description of the APT format can be found in Resources. A simple example is shown here:

```
 -----
 The Hotel Database Vision Statement
 -----
 The Wakaleo Team
 -----
 January 2006

Introduction


 One of the nicer features of Maven is the ability to create an internal technical web site
 at very little cost. Maven 2 extends this functionality, and gives you powerful new ways
 to generate site content...


* Sub-section title


 Team communication is an essential part of any project. And wasting time looking for technical
 project information can be costly and frustrating. Clearly, any IT project will benefit from
 having its own dedicated technical web site...

 * Item 1

 * Item 2

   * Item 2.1

   * Item 2.2
```

Both documents generate a page similar to the one shown in Figure 4.



The Hotel Database Vision Statement - Microsoft Internet Explorer

Hotel Database

The Hotel Database project
Vision Statement
Project Documentation
Built by maven

Introduction

[illegible]

Team Communication

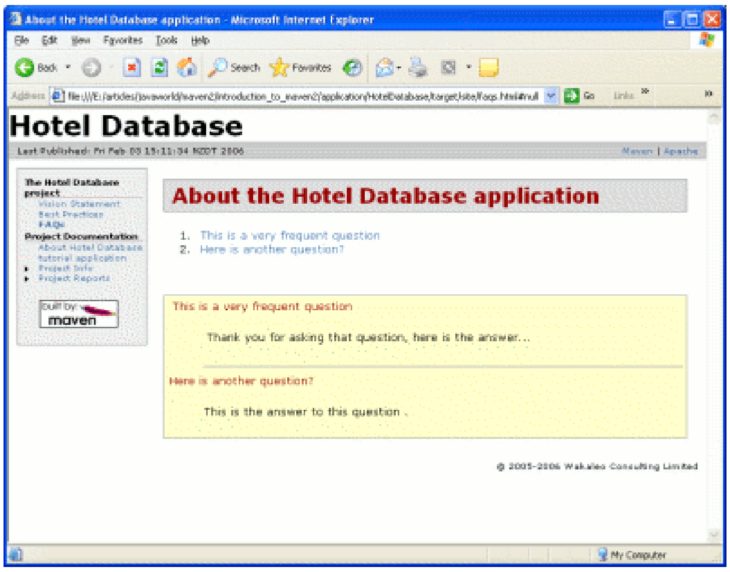
[illegible]

Figure 4. Some site content

The `fml`
directory is for FAQ pages, which are written using the FML format. The FML format is an XML format designed specifically for FAQ pages. A simple example is shown here:

```
 <?xml version="1.0"?>
<faqs title="About the Hotel Database application">


  <part id="about">
    <faq>
      <question> This is a very frequent question</question>
      <answer>
        <p>
          Thank you for asking that question, here is the answer...
        </p>
      </answer>
    </faq>


    <faq>
      <question> Here is another question?< /question>
      <answer>
        <p>
          This is the answer to this question...
        </p>
      </answer>
    </faq>
  </part>
</faqs>
```

This would generate the page shown in Figure 5.


About the Hotel Database application - Microsoft Internet Explorer

Hotel Database

Last Published: Fri Feb 03 15:11:34 NZDT 2006

Maven | Apache

The Hotel Database project
- Vision Statement
- Best Practices
- FAQs

Project Documentation
- About Hotel Database tutorial application
- Project Info
- Project Reports

About the Hotel Database application

1. This is a very frequent question
2. Here is another question?

This is a very frequent question

Thank you for asking that question, here is the answer...

Here is another question?

This is the answer to this question .

© 2005-2006 Wakaleo Consulting Limited

File Edit View Favorites Tools Help
Back Search Favorites
Address file:///E:/articles/javaworld/maven2/introduction_to_maven2/application/HotelDatabase/target/site/faqs.html#null Go Links
built by maven
My Computer

Figure 5. An example of a simple FAQ page

Other formats, such as DocBook, are also in development at the time of this writing.

Deploying your site

To deploy your site, you will first have to tell Maven where to deploy it by defining the location in the pom.xml file, as follows:

```
 <distributionManagement>
  <site>
    <id> website</id>
    <url> scp://www.mycompany.com/www/docs/project/</url>
  </site>
```

```
</distributionManagement>
```

Now you can deploy your site. Run `mvn site-deploy`.

The site will be copied, using `scp` (the only method currently accepted), to the destination server for all to see.

Site generation and continuous integration

How often should you update your site? This is often a matter of personal and team preference. Don't forget that site generation is a processor-intensive process. A really big project on a busy server may take 10 or 15 minutes to generate. So if you run site generation every 5 minutes, your system administrator will probably get annoyed (I've seen servers go down like this). Often, once or twice a day is enough.

Conclusion

In this article, we have walked through the basics of setting up a Maven 2 Website. Maven 2 site generation is powerful and flexible, providing many useful standard technical reports out of the box; much can be done with relatively little initial effort. In addition to the traditional XDoc and FML formats, the APT format provides a new, convenient way of writing specific technical content. Site generation is also much faster in Maven 2. Overall, although some reports are still in development, the site generation functionalities of Maven 2 are well worth investigating.

Author Bio

John Ferguson Smart has been involved in the IT industry since 1991 and in J2EE development since 1999. His specialties are J2EE architecture and development, and IT project management, including offshore project management. He has wide experience in open source Java technologies. He has worked on many large-scale J2EE projects involving international and offshore teams for government and businesses in both hemispheres, and also writes technical articles in the J2EE field. His technical blog can be found at http://www.jroller.com/page/wakaleo.