Ashkelon Users Guide

Eitan Suez

ashkelon

Eitan Suez

Table of Contents

1. Overview 1
    1.1. Usage Scenarios 1
2. Download 3
3. Installation 4
    3.1. Requirements 4
    3.2. Configuration 4
        3.2.1. Environment 4
        3.2.2. Configuration Files 5
        3.2.3. Prepare Database 5
        3.2.4. Build ashkelon manager codebase 6
        3.2.5. Verify "ashkelon" cmd works 7
        3.2.6. Finish setting up database schema 8
    3.3. Populate Database 8
    3.4. Setup the Ashkelon Web Application 9
4. Administrator Guide: Using the Repository Manager 10
    4.1. Documenting Source Code 10
    4.2. The apixml Tool 11
    4.3. Example: Populating DOM4J 12
    4.4. Working with J2SE 14
5. System Architecture 15
    5.1. Database Schema Description 15
6. Viewer Application Screen Shots 16
7. FAQ 20
8. Credits 21
9. Miscellaneous 22
    9.1. Project History 22
    9.2. Origin of Project Name 23

Chapter 1. Overview

ashkelon is an open source project. It is a Java API documentation tool designed for Java developers. Its goals are the same as the goals of the well-known javadoc tool that comes with J2SE, whose user interface most java developers are quite familiar with (see: http://java.sun.com/j2se/javadoc/ if you'd like to learn more about javadoc)

ashkelon addresses the following issues that javadoc does not:

- ashkelon is a multi-API documentation system, unlike javadoc which produces single-API documentation
- ashkelon's documentation system leverages SQL to provide a queriable/searchable repository for documentation
- The DHTML GUI produced by ashkelon (works with all major browsers) is markedly different from the defacto html 3.2 user interface produced by javadoc, which I termed "classic."
- ashkelon's user interface is a Java web application, not a set of static html pages

The javadoc tool is composed of two parts:

1. The documentation generator, used to produce html (or other) documentation
2. a graphical user interface used to browse and peruse through the documentation (static html pages)

Sometimes the same person performs both the generation and referencing the documentation. however, the more common case is that of an API publisher producing javadocs for the API user who references them.

Similarly, ashkelon is also composed of two parts:

1. the *repository manager*, used to populate and otherwise manage a database with documentation information
2. The *viewer application* , a web application that provides access to the documentation in a variety of ways (search, browse, cross-reference, etc..). this application is markedly different from javadoc's user interface in a number of ways.

1.1. Usage Scenarios

In a workgroup environment, a development team needs to reference a number of APIs from a number of sources, including:

1. Internal company code
2. Sun APIs (J2SE, J2EE, etc..)
3. Popular third party APIs (from Apache, Sourceforge, or other)

A developer may be appointed to the task of administrator. This individual would be in charge of using the repository manager to populate the ashkelon database with the API documentation. Alternatively, this person may choose to automate the task entirely and enhance the nightly build process to automatically rebuild the ashkelon repository with the latest docs.

The entire development team would use the viewer application as its window into the documentation. One of the strengths of ashkelon is the various ways in which someone can access information about a Programming Element:

- Hierarchical navigation to a target class or method
- Search by name
- Search by wildcard
- Navigating programmatic relationships
- Cross referencing classes or interfaces
- Power search feature (search by meta information such as author, modifier (et al) or a combination thereof

An important difference with ashkelon is that as an administrator, you will use the repository manager repeatedly to add new APIs into the repository. You don't have to (and should not try to) add all APIs in one step.

In a large community environment, it is conceivable that a single high-availability site be made available to the general Java developer community. Such a site might provide API publishers with online tools to register and populate their APIs in this "global" repository. The end-result would be publicly-accessible Java API documentation for many APIs which would be constantly kept up-to-date by the various API publishers.

Chapter 2. Download

Here are the relevant project and download URLs:

askhelon Project Home Page	http://ashkelon.sourceforge.net/
askhelon's SourceForge project page	http://sourceforge.net/projects/ashkelon/
Download URL	http://sourceforge.net/project/showfiles.php?group_id=54364

Visit the project download page. Select to download ashkelon in whichever compression format you prefer (.zip or .tgz). After the download is complete, unpack the compressed file. You should now have an ashkelon working directory.

We are more than happy to assist you with getting ashkelon properly configured and installed in your environment. Feel free to email Eitan directly at `<eitan-keyword-deux.352d02@u2d.com>` or the appropriate mailing list.

ashkelon Users mailing list	`<ashkelon-users@lists.sourceforge.net>`
ashkelon Developers mailing list	`<ashkelon-devs@lists.sourceforge.net>`

Chapter 3. Installation

3.1. Requirements

1. J2SE v1.4.x or higher
2. A servlet & JSP container (I use Jakarta Tomcat 5.5 at the moment, but have used or received reports from many users and/or vendors that ashkelon works with their servlet container)
3. postgresql (http://www.postgresql.org/) or mysql (http://www.mysql.org)
4. Jakarta Ant (v1.5 or higher) (http://ant.apache.org)

3.2. Configuration

3.2.1. Environment

1. Download & unpack the ashkelon tarball
2. Make sure your `$JAVA_HOME` environment variable is set
3. Set up `ASHK_HOME` environment variable

   Example 3.1. Windows

   ```
   set ASHK_HOME=c:\ashkelon
   ```

   Example 3.2. bash

   ```
   export ASHK_HOME=~/ashkelon
   ```

4. Add `$ASHK_HOME/bin` to your path:

   ```
   $ export PATH=$PATH:$ASHK_HOME/bin
   ```

3.2.2. Configuration Files

3.2.2.1. SourceCache Base Path

In version 0.9 ashkelon now has the capability to auto-fetch API source code from either subversion or cvs version control systems. ashkelon therefore will use a temporary directory as the base path for keeping a cache of source code.

In the file `src/org/ashkelon/ashkelon-config.xml`, specify the base path that ashkelon will use for its source cache. If the path does not exist, ashkelon will attempt to create the specified directory.

3.2.2.2. Database Connection Information

> **Note**
>
> As you read this section, substitute the database type you'll be using (mysql or postgres) wherever you see the "placeholder" text *${dbtype}*

1. Edit `db/` *${dbtype}* `/org/ashkelon/db/conn-info.properties` to specify the proper database connection information:

```
#
dbtype=mysql
jdbcDriverName=com.mysql.jdbc.Driver
connectionURL=jdbc:mysql://localhost/ashkelon
user=ashkelon
password=ashkelon
```

or

```
#
dbtype=postgres
jdbcDriverName=org.postgresql.Driver
connectionURL=jdbc:postgresql://localhost/ashkelon
user=ashkelon
password=ashkelon
```

3.2.3. Prepare Database

1. Create the ashkelon database:

Example 3.3. For mysql

```
c:\> mysql -u root -p
mysql> create database ashkelon;
mysql> exit
```

Example 3.4. For postgresql

```
~$ createdb -U postgres ashkelon
```

2. Generate the schema. Run script: `db/${dbtype} /org/ashkelon/db/init.sql`

Example 3.5. Using mysql

```
c:\ashkelon> mysql -u root -p ashkelon
                      < db\mysql\org\ashkelon\db\init.sql
```

Example 3.6. Using postgresql

```
~/ashkelon> psql -U ashkelon ashkelon
ashkelon#= \i db/postgres/org/ashkelon/db/init.sql
ashkelon#= \q
```

3.2.4. Build ashkelon manager codebase

Build and create the ashkelon jar file:

Example 3.7. For mysql

```
$ ant -Ddbtype=mysql jar-manager
```

Example 3.8. For postgresql

```
$ ant -Ddbtype=postgres jar-manager
```

Rather than specify dbtype from the command line, feel free to edit the build.xml file directly. This way you won't have to remember how to construct your ant command line each time.

3.2.5. Verify "ashkelon" cmd works

Verify that the "ashkelon" command works. Invoke it with no parameters from the command line.

```
$ ashkelon
```

should produce this help/usage screen

```
Ashkelon Repository Manager (v 0.9)
----------
Description:
  Manages a Java Documentation Repository.
  For more information, visit http://ashkelon.sourceforge.net/

Examples:
  ashkelon reset
  ashkelon add hibernate.xml
  ashkelon list
  ashkelon remove Hibernate
  ashkelon html -d junit-javadocs apis/junit.xml

Usage:
  ashkelon {command} [options]

  Valid commands:

    add         Add an API to repository
    remove      Remove specified API from the repository
    update      Update an API
    list        List APIs currently residing in the repository
    reset       Reset repository (i.e. delete everything, use with care)
    export      Dump list of APIs as XML to System.out
    html        Produce static HTML pages (with ashkelon look and feel)

For command-specific help, type:
  ashkelon {command} -h

Caveats:
  No work done yet to support J2SE 5.
----------
Copyleft
```

3.2.6. Finish setting up database schema

```
$ ashkelon reset
```

3.3. Populate Database

Example 3.9. Populating DOM4J API

```
~ashkelon$ cd apis
~ashkelon/apis$ ashkelon add dom4j.xml
[press enter]
```

If everything goes well, you should end up with your database populated with the Java API documentation for DOM4J.

Ashkelon basically downloads the source code for the project (per the cvs or subversion repository information specific in the xml file) and places a copy in its "source cache" area on disk. Ashkelon the proceeds to invoke javadoc with itself as the doclet, in order to parse and populate the database with the API information.

You can list the APIs in the ashkelon database like this:

```
eitan@ubuntu:~/projects/ashkelon/apis$ ashkelon list
ashkelon@ubuntu:~/projects/ashkelon/apis$ ashkelon list
Ashkelon: 1 APIs in ashkelon:
Ashkelon:    dom4j
```

Continue to add as many APIs as you desire. They all get added to the same database and cross-references between APIs are resolved.

> **Note**
>
> There's a bug at the moment with v0.9 regarding inability to autodetect and supply a cvs password. If for some reason ashkelon appears to have "hanged", usually after a line that resemples this:
>
> CVS password: add: Logging in to :pserver:guest@cvs.dev.java.net:2401/cvs
>
> Then this means that the invoked cvs command is waiting for a password before

> proceeding. Since almost all open source projects use a blank password for anonymous cvs login, simply press enter. I apologize for the inconvenience.

3.4. Setup the Ashkelon Web Application

```
eitan@ubuntu:~/projects/ashkelon$ ant war
[output]
Buildfile: build.xml
init:
compile:
bind:
jar-model:
      [jar] Building jar: /home/eitan/projects/ashkelon/build/ashkelon-model.jar
jar-taglib:
      [jar] Building jar: /home/eitan/projects/ashkelon/build/ashkelon-taglib.jar
jar-webapp:
      [jar] Building jar: /home/eitan/projects/ashkelon/build/ashkelon-webapp.jar
     [copy] Copying 1 file to /home/eitan/projects/ashkelon/build
war:
      [war] Building war: /home/eitan/projects/ashkelon/dist/ashkelon.war

BUILD SUCCESSFUL
Total time: 3 seconds
[end output]
eitan@ubuntu:~/projects/ashkelon$ ls dist/
ashkelon.war
eitan@ubuntu:~/projects/ashkelon$ cp dist/ashkelon.war ~/devel/tomcat/webapps
```

You need to do two things to deploy the ashkelon war file: [1] copy the war file to the tomcat webapps directory, as shown above, and [2] copy $JAVA_HOME/lib/tools.jar to $CATALINA_HOME/common/lib

If necessary, restart tomcat or load the new web application and explore http://localhost:8080/ashkelon/

Chapter 4. Administrator Guide: Using the Repository Manager

The ashkelon command-line tool is called "ashkelon". It is a wrapper shell script for a Java program. On win32, the file is "ashkelon.bat" while on unix it's "ashkelon." These files reside in $ASHK_HOME/bin

You will primarily use the ashkelon tool to add content to your database. Here is a listing of some of the subcommands:

ashkelon add	use to add APIs to your database
ashkelon reset	use to "nuke" the contents of the database (start from scratch)
ashkelon list	lists the APIs currently documented in the database
ashkelon remove	opposite of ashkelon add, use it to remove APIs from the repository

You can obtain a listing of all the subcommands by invoking ashkelon with no arguments. For help per subcommand, invoke ashkelon like this:

```
ashkelon subcommandname -h
```

ashkelon reset and **ashkelon list** require no parameters and likewise require no explanation.

ashkelon add is richer. Let's talk about "ashkelon add".

4.1. Documenting Source Code

Traditionally (i.e. with javadoc) one documents a single API. So the inputs to the javadoc tool are a list of package names, such as {java.lang, java.io, java.net, ..}

When documenting multiple APIs in a single repository, it's important to also keep track of what API each java package belongs to. That is, `org.apache.tools.ant` is part of the ant API whereas `org.apache.log4j` is part of the log4j API.

The ashkelon project has devised a simple XML structure to describe an API. Here's an example API XML file for DOM4J (see apis/dom4j.xml):

Example 4.1. DOM4J API XML file descriptor for ashkelon

```
<?xml version="1.0" encoding="UTF-8"?>
<api>
   <name>dom4j</name>
   <summarydescription>The flexible XML framework for Java</summarydescription>
   <description>dom4j is an easy to use, open source library...</description>
   <publisher>SourceForge</publisher>
   <download_url>http://www.dom4j.org/</download_url>
   <release_date>2005-04-15T08:00:00</release_date>
   <version>1.6</version>
   <packages>
      <package>org.dom4j</package>
      <package>org.dom4j.bean</package>
      <package>org.dom4j.datatype</package>
      <package>org.dom4j.dom</package>
      <package>org.dom4j.dtd</package>
      <package>org.dom4j.io</package>
      <package>org.dom4j.jaxb</package>
      <package>org.dom4j.rule</package>
      <package>org.dom4j.rule.pattern</package>
      <package>org.dom4j.swing</package>
      <package>org.dom4j.tree</package>
      <package>org.dom4j.util</package>
      <package>org.dom4j.xpath</package>
      <package>org.dom4j.xpp</package>
   </packages>
   <repository>
     <type>cvs</type>
     <url>:pserver:anonymous@cvs.sourceforge.net:/cvsroot/dom4j</url>
     <modulename>dom4j</modulename>
     <sourcepath>src/java</sourcepath>
   </repository>
</api>
```

This file is similar to the `package.html` file that you usually have to write for javadoc package documentation. Except that this file is xml-based and thus more structured, and easier to interpret by a program.

These xml api descriptors are the primary input to the ashkelon command (aside from the actual source code that you're trying to document). Many descriptors already exist in in ashkelon's `apis` directory.

4.2. The apixml Tool

ashkelon provides a tool called "apixml" that assists you in producing ashkelon API descriptors.

For a given API, search its javadocs directory for a file named `package-list`. This file is a sort of signature that javadoc leaves behind after processing an API.

Feed apixml this file and it will produce an almost completed xml descriptor file for you. In the example session below, I process junit's package-list through **apixml**. The xml output is produced by apixml. Now simply specify the missing pieces: API name, description, publisher, url, and version control system information.

```
eitan@ubuntu:~/devel/junit/javadoc$ apixml package-list


<?xml version="1.0" encoding="UTF-8"?>
<api>
   <name/>
   <summarydescription/>
   <description/>
   <publisher/>
   <download_url/>
   <release_date>2006-02-03T20:21:45.151</release_date>
   <version/>
   <packages>
      <package>junit.extensions</package>
      <package>junit.framework</package>
   </packages>
   <repository>
      <type>cvs</type>
      <url>:pserver:anonymous@cvs.sourceforge.net:/cvsroot/[projectname]</url>
      <modulename>[projectname]</modulename>
      <tagname/>
      <sourcepath>src/java</sourcepath>
   </repository>
</api>
eitan@ubuntu:~/devel/junit/javadoc$
</screen>
```

4.3. Example: Populating DOM4J

Let's run through a quick example. We first produce dom4j.xml:

```
eitan@ubuntu:~/devel/dom4j/docs/apidocs$ apixml package-list > dom4j.xml
```

Check out `apis/dom4j.xml` to see how the base XML descriptor produced by apixml was completed. We're now ready to ask ashkelon to put DOM4J into our database. Let's first see what we have already..

```
eitan@ubuntu:~/projects/ashkelon/apis$ ashkelon list
Ashkelon: 9 APIs in ashkelon:
Ashkelon:    Hibernate
Ashkelon:    jEdit
Ashkelon:    FlexDock
Ashkelon:    Servlet
Ashkelon:    L2FProd.com Common Components
Ashkelon:    log4j
Ashkelon:    JiBX
```

```
Ashkelon:    JUnit
Ashkelon:    J2SE
eitan@ubuntu:~/projects/ashkelon/apis$
```

Ok, let's add dom4j:

```
eitan@ubuntu:~/projects/ashkelon/apis$ ashkelon add dom4j.xml
```

And here's the output (less the javadoc warnings)

```
add: cmd is: cvs -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/dom4j -q update -d  -r HEAD dom4j/s
Loading source files for package org.dom4j...
Loading source files for package org.dom4j.bean...
Loading source files for package org.dom4j.datatype...
Loading source files for package org.dom4j.dom...
Loading source files for package org.dom4j.dtd...
Loading source files for package org.dom4j.io...
Loading source files for package org.dom4j.jaxb...
Loading source files for package org.dom4j.rule...
Loading source files for package org.dom4j.rule.pattern...
Loading source files for package org.dom4j.swing...
Loading source files for package org.dom4j.tree...
Loading source files for package org.dom4j.util...
Loading source files for package org.dom4j.xpath...
Loading source files for package org.dom4j.xpp...
Constructing Javadoc information...
[a bunch of javadoc side-effect warnings suppressed]
Ashkelon: 14 packages to process..
Ashkelon: Processing package org.dom4j..
Ashkelon: Processing package org.dom4j.bean..
Ashkelon: Processing package org.dom4j.datatype..
Ashkelon: Processing package org.dom4j.dom..
Ashkelon: Processing package org.dom4j.dtd..
Ashkelon: Processing package org.dom4j.io..
/home/eitan/projects/sourcecache/dom4j/src/java/org/dom4j/io/HTMLWriter.java:181:
warning - Tag @link: missing '#': "OutputFormat.isXHTML() OutputFormat.isXHTML()"
/home/eitan/projects/sourcecache/dom4j/src/java/org/dom4j/io/HTMLWriter.java:181:
warning - Tag @link: can't find OutputFormat.isXHTML() in org.dom4j.io.HTMLWriter
Ashkelon: Processing package org.dom4j.jaxb..
Ashkelon: Processing package org.dom4j.rule..
Ashkelon: Processing package org.dom4j.rule.pattern..
Ashkelon: Processing package org.dom4j.swing..
Ashkelon: Processing package org.dom4j.tree..
Ashkelon: Processing package org.dom4j.util..
Ashkelon: Processing package org.dom4j.xpath..
Ashkelon: Processing package org.dom4j.xpp..
Ashkelon: Add Time: 7 seconds
Ashkelon: Updating Internal References..
Ashkelon:       Processing FIELD references..
Ashkelon:       Processing METHOD references..
Ashkelon:       Processing IMPL_INTERFACE references..
Ashkelon:       Processing THROWNEXCEPTION references..
Ashkelon:       Processing PARAMETER references..
Ashkelon:       Processing SUPERCLASS references..
Ashkelon:       Processing seetag PACKAGE references..
Ashkelon:       Processing seetag CLASSTYPE references..
Ashkelon:       Processing seetag MEMBER references..
Ashkelon:       Processing seetag EXECMEMBER references..
Ashkelon: Ref. Time: 8 seconds
```

```
Ashkelon: done
102 warnings
eitan@ubuntu:~/projects/ashkelon/apis$
```

The warnings are harmless. They're a side effect of the javadoc parser attempting to resolve referenced libraries from the DOM4J classes. We see there were a couple of javadoc syntax warnings that we might want to fix. Otherwise, ashkelon ensured that its source cache was up to date by doing a CVS update. It then proceeded to process the 14 packages that comprise the dom4j javadocs. This took seven seconds. Eight more seconds were spent resolving inter-api references between existing APIs in the database and the newly added one.

Oftentimes you'll want to update an API you've already populated into your database with an updated version. You do this with the **ashkelon update** command. This command first deletes the api from the repository, and then proceeds to fetch, parse, and populate the updated version.

Say, for example, that dom4j released a new version. You'd update the xml descriptor, maybe the tag name in cvs to fetch the code from. Then you'd invoke:

```
        ashkelon update dom4j
```

4.4. Working with J2SE

With J2SE v1.4, javadoc introduced the -source flag for processing APIs that used J2SE v1.4 specific features, such as the assert keyword. So, if you're going to populate such an API, or J2SE itself, you must specify ashkelon's corresponding flag: --source, like this:

```
$ ashkelon add --source 1.4 j2sdk14.xml
```

Another concern is that although it's very convenient to have ashkelon fetch the source code on our behalf directly from version control, this is not always possible or practical. For example, you'll more likely manually download J2SE source code for processing. ashkelon has a provision for this. You can define a SOURCEPATH environment variable containing however many paths you need where source code resides that you want to process through ashkelon.

So, assume you download J2SE to ~/devel/j2se/sources. Simply add that path to your SOURCEPATH environment variable (in a completely analogous fashion to the way we work with CLASSPATH). Then, simply omit the <repository> section from the api descriptor file (it's optional). ashkelon will simply fall back to looking for the j2se source code in your source path.

Chapter 5. System Architecture

Here are two complemtentary sources on ashkelon's architecture are:

July 2002 java.sun.com main story/article	http://java.sun.com/features/2002/07/dbdoc.html
JavaOne (2002) alternate presentation (ppt)	http://ashkelon.sourceforge.net/files/1145suez.pdf

5.1. Database Schema Description

I have created an entity relationship diagram or sorts (pdf) representing the database schema used by ashkelon. Click on the thumbnail for a full-size png:

ERD

The script that constructs the schema for a given database is located in `$ASHK_HOME/db/` *`$dbtype`* `/org/ashkelon/db/ashkelon.sql`

Chapter 6. Viewer Application Screen Shots

Here is a collection of screenshots of the ashkelon web application. This particular instance shows only a half dozen APIs. You should be able to notice how the search results include items from a variety of APIs. Again, inter-API cross referencing is complete. For example: the documentation page for a method in API A that returns a type defined in API B will show the return type hyperlinked.

Table 6.1. APIs and Package Listings

API Listing	J2SE Package Listing	Packages View

Table 6.2. Sample Package: java.awt

java.awt at a glance	java.awt Class Listing	java.awt Interfaces Listing

java.awt Main View	java.awt Inheritance Hierarchy	

Table 6.3. Sample Class: java.lang.String

String class at a glance	String constructor variants	String Class Main View
String Method Listing	String Static Method Listing	

Table 6.4. Search

 Simple Search	 Simple Search Results	 Power Search
 Power Search Results	 Wildcard Search	

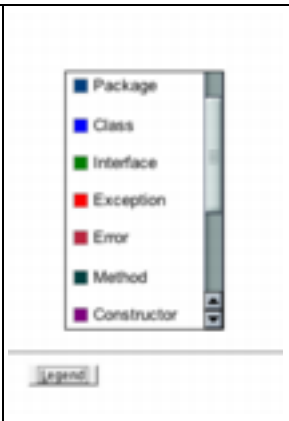
Table 6.5. Method Detail & Cross Reference Lookup

 Method Detail	 Cross Reference Lookup	

Table 6.6. Stats

Author Stats	Package Stats	Stats

Table 6.7. Miscellaneous

Author View	Settings	 Legend: Color coding and Styles

Chapter 7. FAQ

Q:
Is ashkelon Java 5 Compatible?

A:
You can build and run ashkelon with Java 5.

Q:
Does ashkelon support Java 5?

A:
At this time, no. ashkelon has not been retrofitted to be aware of and document the new Java 5 constructs (enums, generics, et al)

Q:
Does ashkelon support multiple versions of the same API in a single database?

A:
Not at this time.

Q:
Is there an Eclipse Plugin for ashkelon?

A:
No. We would welcome your contribution of one. Eitan has written a primitive IDEA plugin for ashkelon though at this time it has not been published.

Q:
Does javadoc require source code to produce documentation?

A:
Yes.

Chapter 8. Credits

Conceived, designed, & built by Eitan Suez, Austin, Texas (http://u2d.com/).

Sincere thanks goes to the following individuals:

Andreas Schaefer	For writing the initial ant build file for this project, and for being the first individual to step up to the plate and actively contribute to this project.
Bob Hays (alephnaught.com)	For actively helping on the project, contributing a number of patches and features, including an ashkelon ant task
Ben Walding	Who wrote a maven plugin for ashkelon and who planted the idea in my mind to write a maven POM file for ashkelon (forthcoming)

Chapter 9. Miscellaneous

9.1. Project History

I had the idea that javadocs should be a web application relying on a backend database around 1998. It took me a couple of years to actually act on this. On a short vacation trip to hawaii in 2000, I wrote my first stab at a doclet to populate a db. It worked beautifully. The results were promising but with only a few days' work invested in this project, there wasn't much to show for: only a populator doclet.

The idea simmered a little longer and a year later I decided that I could no longer sit idle with this idea in my head without doing anything about it. So I took a 2-week vacation around March 2001 and shut myself in a small hotel room in houston and coded ~ 19 hour days for 14 days straight. The result was 90% of what ashkelon is today. This is what I accomplished:

- developed my own MVC framework to support the web application (a la struts)
- redesigned and implemented the database schema from scratch
- refactored the javadoc html doclet into jsp pages
- learned css
- wrote an entirely new gui for javadocs, employing dhtml and css vigorously
- wrote the viewer application in jsp
- wrote my own crappy database connection pool
- Added entirely new sections to the viewer app, going beyond the scope of just javadocs (stats section, authors section)

Over the next three years, I added these:

- A second GUI, in the style of javadoc (html 3.2)
- postgres and mysql support besides the original oracle version
- mozilla support for the DHTML GUI
- Added a "power search" feature to complement the simple search feature that had originally been built into the viewer app

One deduction from this story is that much more can be accomplished in two weeks

of immersion compared to three years of trying to squeeze in an hour here or there in the middle of the night.

9.2. Origin of Project Name

The project name "ashkelon" is definitely influenced by names of other open source projects, especially jakarta projects. There are two main reasons for selecting "ashkelon" as the name for this project. One is selfish, the other is subtle.

I became aware of a Jakarta project called "Alexandria" (now defunct). As I read about Alexandria, I realized that my project had many goals similar to Alexandria's: making many api docs available in a single place. One cannot find a more aptly named project, as the city of Alexandria was the host of the largest known library in antiquity.

It turns out that Ashkelon is also a city, an ancient city, not 250 miles from Alexandria on the mediterranean coast of Israel. The thought that two projects so close to each other conceptually were also named after two cities close to each other geographically amused me. Furthermore, the Alexandria project was hosted on what I considered to be a high profile and high visibilitly open source site: The Apache Jakarta project. My project came from relative obscurity (ashkelon likewise is not known for being the source of great libraries or documentation projects). I thought to myself.. "maybe a great project can emerge from relative obscurity.."

What tipped the scales in favor of naming the project 'ashkelon' is the additional fact that the city of Ashkelon was my home for a number of years in my youth.