# *Software Book: Patent Printer*

## Patent Document Printer

*A Patent Document Printer fetches patent and patent application images from the US Patent and Trademark Office (USPTO) website and organizes them into a single Portable Document Format (PDF) file. Use this utility as a free alternative to pay-per-patent retrieval services.*


*Daniel Lanovaz*

*Los Gatos, California*


*September 10, 2004*

# Patent Printer

his software book describes how to programmatically fetch patent and patent application images from the U.S. Patent and Trademark Office (USPTO) website and organize them into a single Portable Document Format (PDF) file. Use this software book as an alternative to pay-per-patent sites.

## Introduction

The US Patent and Trademark Office website[1] ("site") contains a vast database of patent applications and issued patents ("patents"). The site implements a search mechanism that allows the public to find, read, and print patents.

The site implements the rendering of a patent within a web browser by converting the patent into Hypertext Markup Language (HTML). This display mechanism, unfortunately, does not faithfully reproduce the original patent as it omits drawings, original pagination, and other important information.

As an additional site service, each USPTO patent has a link to the "images" associated with the patent. These Tagged Image File Format (TIFF) images are scans of the original patent or application. There are, however, three problems with the implementation of image viewing on the USPTO site:

1. Users must install a TIFF viewing plug-in.
2. Only one page is viewable at a time.
3. Printing an entire patent (or application) is unsupported.

Others have noted the inability to fetch entire patent documents from the site. A useful utility called `pat2pdf` written by Oren Tirosh and Thomas Boege is the genesis of this application[2]. It is a Unix shell script that operates in a similar manner to this application. Oren and Thomas describe why they created pat2pdf:

> To help open source developers who increasingly find themselves facing software patent problems. Actually, the real reason is because I am a cheap bastard who doesn't want to pay for downloading a PDF

[1] http://www.uspto.gov

[2] http://www.tothink.com/pat2pdf/

> but I thought it would be a good place to make a point about the increasing abuse of software patents.

The main differences between `PatentPrinter` and pat2pdf is that `PatentPrinter` uses a slightly modified image fetching algorithm, is more easily used within [Java] application servers, and is written as a[n] [experimental] software book. In addition, this application attempts to reduce dependent technologies by bundling together PDF document generation code. While pat2pdf requires Bash, Ghostscript, Tiff2ps, and Lynx, `PatentPrinter` only requires the presence of a Java Virtual Machine (JVM)[1], and serves as a stepping-stone for building a web service such as the one implemented by the pat2pdf.com[2] site.

## Software Books

`PatentPrinter` is a software book: an Extensible Markup Language (XML) representation of software structured as a book rather than source text files. A software book compiler translates the text you are reading into the final application executed by a JVM. For further details on Software Books please read the essay, "Thinking with Style."[3]

In the appendix of this software book, we provide some self-analysis on using the model of a software book to write this patent printer utility.

## Disclaimer

The USPTO clearly states that patents are published in the public domain and are not subject to copyright restrictions[4]. They also publish a clear warning to clients who use software such as the PatentPrinter for automated download of patent images:

> Users employing third-party software which downloads multiple pages of a patent at once may find this practice subjects them to denial of access to the databases if they exceed PTO's maximum allowable activity levels.[5]

---

[1] The PatentPrinter application will be ported to .NET.

[2] The pat2pdf website is unfortunately charging a per-patent fee.

[3] http://www.lanovaz.org/daniel/Shared%20Documents/Thinking%20with%20Style

[4] http://www.uspto.gov/main/ccpubguide.htm

[5] http://www.uspto.gov/patft/help/images.htm

Please respect the wishes of the USPTO and do not abuse their site with this software.

## License



# Using the Patent Printer

This application is packaged as a Java archive (jar)[1] file and supports the following five arguments:

| Argument | Purpose |
|---|---|
| --help | Print usage information. |
| --patent <patentNumber> | Fetch a patent from the USPTO site and place it into a PDF file whose name is <patentNumber>.pdf. |
| --application <applicationNumber> | Fetch a patent application from the USPTO site and place it into a PDF file whose name is <applicationNumber>.pdf. |
| --file <filename> | Use the given filename rather than using the patent number. |
| --dir <directory> | Place the file in the given directory. |

## Examples

The following examples assume a JVM is installed on the target system.

The following example fetches patent 6,000,000 from the USPTO site and prints the patent to the file 6000000.pdf:

[1] Java archive (jar) files are a packaging mechanism for Java applications and libraries.

```
java –jar PatentPrinter.jar --patent 6000000
```

The following example fetches application number 2002006982 from the USPTO site and prints the application to the file 2002006982.pdf:

```
java –jar PatentPrinter.jar --application 2002006982
```

# Patent Printer Application

The patent printer application operates as shown in Figure 1. The command line options are processed **1** and the patent or application number as well as the destination location for the PDF file are calculated **2**. Based on the document number the application performs a network fetch to one of four USPTO image servers **3**. The algorithm used to determine which image server to connect with is described in the section “Patent Office Website” on page 14. The application then uses the iText PDF package to convert fetched TIFF images into the final PDF file **4**.

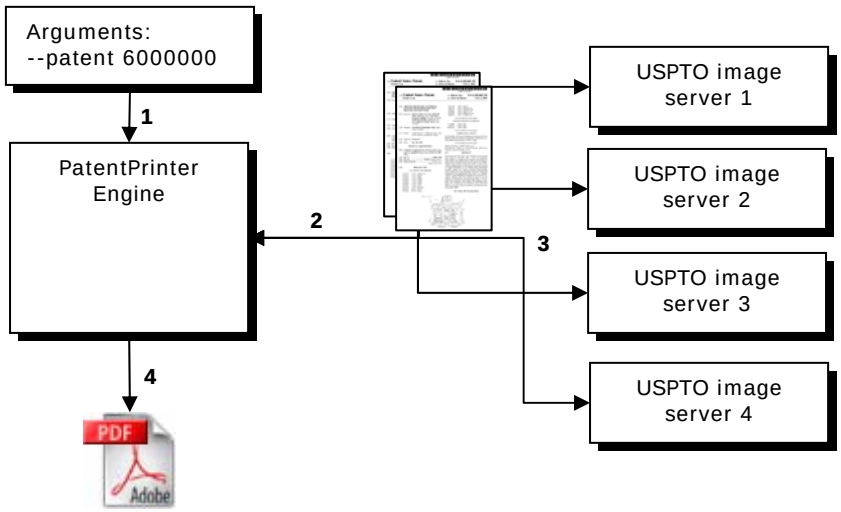


**Figure 1 Patent Printer Process**

```
package com.precedia.patents;
```

# Patent Printer Support Classes

The Patent Printer application is a collection of three packages bundled together in the final Java archive file:

| Package | Purpose |
|---|---|
| com.precedia.patents | The package that implements the patent fetching algorithm and implements the application’s main entry point. |

| | |
|---|---|
| com.lowagie.text | The iText[9] open-source package that implements the PDF writing logic. |
| gnu.getopt | A port of the GNU getopt() function for Java applications. |

The following five classes from the base Java Input/Output package are required to read images from the USPTO website and to write the final PDF file to the host file system.

```
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.File;
```

The following two classes are used to construct Uniform Resource Locators (URL) used to access TIFF images on the USPTO site.

```
import java.net.URL;
import java.net.MalformedURLException;
```

The following four classes are from the iText PDF processing package used to construct PDF files and insert images into pages within the file.

```
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Image;
import com.lowagie.text.pdf.PdfWriter;
```

The following two classes process the application's command-line arguments.

```
import gnu.getopt.Getopt;
import gnu.getopt.LongOpt;
```

## Class PatentPrinter

The main engine of the patent printer application is the `PatentPrinter` class. This class implements the main Java entrypoint (`main`), parses command-line arguments, fetches images from the USPTO site, and generates the final PDF file. This class is represented in Figure 1 as the "Patent Printer Engine."

[9] http://www.lowagie.com/iText/

```
public class PatentPrinter {
```

## Members

The PatentPrinter class maintains three private member variables that correspond to the command-line arguments.

The documentNumber variable is a string that comprises the --patent or --application command-line parameter value. Note that the current implementation does little validity checking on this argument. If the document number is invalid the PatentPrinter will be unable to fetch images from the USPTO site and a zero-sized document error will be reported.

```
private String documentNumber  = null;
```

The outputFilename variable is a string that comprises the --file command-line parameter. This filename will be used rather than the default document number.

```
private String outputFilename  = null;
```

The outputDirectory variable is a string that comprises the --dir command-line parameter. This directory is where the final PDF file will be located. The default is the application's current working directory.

```
private String outputDirectory = null;
```

# Public

## Main

This is the main entry point required by all Java applications. Its single argument is an array of String objects that represent the various command-line parameters presented to the application. The patent printer passes these arguments to the processArguments method for interpretation and then invokes the printPatent method to fetch the images and produce the PDF file. This method catches all exceptions and translates them into a message printed on standard error.

```
public static void main(String[] args)
{
   PatentPrinter printer = null;

   try
   {
      printer = new PatentPrinter();
      if (printer.processArguments(args))
```

```
            {
               printer.printPatent();
            }
         }
         catch (Exception e)
         {
            System.err.println(
               "Failed to print document.");
            System.err.println(e.getMessage());
            printUsage();
         }
      }
```

## printUsage

Print on standard output a brief synopsis of the command-line parameters accepted by the patent printer.

```
public static void printUsage()
{
   System.out.println(
      "Usage: com.precedia.patent.PatentPrinter");
   System.out.println("\t--help <print this help>");
   System.out.println("\t--patent <patent number>");
   System.out.println("\t--application <application number>");
   System.out.println("\t--file <output filename>");
   System.out.println("\t--dir <output directory>");
}
```

## PatentPrinter

Create a new instance of the patent printer. This constructor must be followed by a sequence of calls to initialize the document number, the output filename, or the output directory.

```
public PatentPrinter()
{
}
```

Create a new instance of the patent printer initialized to print the patent document numbered `documentID`. The `outputFilename` specifies the name of the resulting PDF file. If this argument is null or the empty string, the `documentID` is used. The `outputDirectory` argument specifies the directory where the PDF file must be placed. If this argument is null or the empty string the current working directory is used.

```
public PatentPrinter(
   String documentID,
   String outputFilename,
   String outputDirectory)
{
```

```
        setDocumentNumber(documentID);
        setOutputFilename(outputFilename);
        setOutputDirectory(outputDirectory);
    }
```

## processArguments

This method is a straightforward application of the GNU “Getopt” class to process the application’s command-line argument as identified by the argument, args.

The arguments supported by this method are documented in the section “Using the Patent Printer” on page 3 and example usage is described in the section “Examples” on page 4.

This method returns `true` if the command-line arguments were processed correctly, otherwise it returns `false`. This method calls `printUsage()` if the `--help` command-line switch is present.

```
public boolean processArguments(String[] args)
{
    boolean continueProcessing = true;
    final char OPTION_HELP        = 'h';
    final char OPTION_PATENT      = 'p';
    final char OPTION_APPLICATION = 'a';
    final char OPTION_FILE        = 'o';
    final char OPTION_DIR         = 'd';

    int aChar;
    LongOpt[] longOptions = new LongOpt[5];

    longOptions[0] =
        new LongOpt("help", LongOpt.NO_ARGUMENT, null,
        OPTION_HELP);
    longOptions[1] =
        new LongOpt("patent", LongOpt.REQUIRED_ARGUMENT, null,
        OPTION_PATENT);
    longOptions[2] =
        new LongOpt("application", LongOpt.REQUIRED_ARGUMENT,
        null, OPTION_APPLICATION);
    longOptions[3] =
        new LongOpt("file", LongOpt.OPTIONAL_ARGUMENT, null,
        OPTION_FILE);
    longOptions[4] =
        new LongOpt("dir", LongOpt.OPTIONAL_ARGUMENT, null,
        OPTION_DIR);

    Getopt getOpt =
        new Getopt("PatentPrinter", args, "", longOptions);
    getOpt.setOpterr(false);

    while ((aChar = getOpt.getopt()) != -1)
    {
```

```
            switch (aChar)
            {
               case OPTION_HELP:
                  printUsage();
                  continueProcessing = false;
                  break;
               case OPTION_PATENT:
                  setDocumentNumber(getOpt.getOptarg()); break;
               case OPTION_APPLICATION:
                  setDocumentNumber(getOpt.getOptarg()); break;
               case OPTION_FILE:
                  setOutputFilename(getOpt.getOptarg()); break;
               case OPTION_DIR:
                  setOutputDirectory(getOpt.getOptarg()); break;
               default:
                  throw new IllegalArgumentException("Invalid
                  parameters."
               );
            }
         }

         return continueProcessing;
      }
```

## printPatent

The `printPatent` method instructs the `PatentPrinter` to perform the following operations:

1. Connect to the USPTO website.
2. Fetch the images associated with the printer's document number.
3. Print each image on a page in the PDF file.

```
public void printPatent()
{
   PatentOfficeWebsite website =
      new PatentOfficeWebsite();
   PatentDocument document =
      website.getDocument(documentNumber);
   printPatentDocument(document);
}
```

## Properties

The following three methods initialize the patent printer with the patent or application document number, the output filename for the PDF file, and the output directory. If any of the arguments are null the method throws an `IllegalArgumentException`.

See the `PatentPrinter` constructor for further details on how these methods are used.

### setDocumentNumber

```
public void setDocumentNumber(String documentID)
{
   if (documentID == null)
      throw new IllegalArgumentException(
         "Invalid document number");
   documentNumber = documentID;
}
```

### setOutputFilename

```
public void setOutputFilename(String filename)
{
   if (filename == null)
      throw new IllegalArgumentException(
         "Invalid out put filename");
   outputFilename = filename;
}
```

### setOutputDirectory

```
public void setOutputDirectory(String directory)
{
   if (directory == null)
      throw new IllegalArgumentException(
         "Invalid out put directory");
   outputDirectory = directory;
}
```

# Protected

## printPatentDocument

This method is the workhorse of the PatentPrinter class. It accepts an instance of a PatentDocument (see the section “} Patent Documents” on page 21 for details of a patent document) that represents the document (patent or application) to be fetched from the USPTO and placed into a PDF file. This method creates an instance of the iText class PdfWriter and passes to the PDF writer each patent image as they arrive from the USPTO website.

Details on how an image is fetched from the USPTO site is found in the section “Patent Office Website” on page 14.

```
protected void printPatentDocument(
   PatentDocument patentDocument)
{
```

```
        Document pdfDocument = new Document();

        try
        {
            File outputFile = getOutputFile();
            PdfWriter writer =
               PdfWriter.getInstance(
                  pdfDocument,
                  new FileOutputStream(outputFile));
            pdfDocument.open();
            int pageCount = patentDocument.getPageCount();

            System.out.print("Printing document ");
            System.out.print(outputFile.getAbsolutePath());
            System.out.print(" [");
            System.out.print(pageCount);
            System.out.println(" pages].");

            for (int i = 0; i < pageCount; i++)
            {
               System.out.print("Fetching page ");
               System.out.print(i + 1);
               System.out.print(" of ");
               System.out.print(pageCount);
               System.out.println(".");

               Image pageImage = patentDocument.getImage(i + 1);
               pageImage.setAlignment(Image.MIDDLE);
               pageImage.setAbsolutePosition(0, 0);
               pageImage.scaleAbsolute(
                  pageImage.scaledWidth() / pageImage.getDpiX() *
                  72f,
                  pageImage.scaledHeight() / pageImage.getDpiY() *
                  72f);
               pdfDocument.add(pageImage);
               pdfDocument.newPage();
            }

            System.out.println("Done.");

            pdfDocument.close();
        }
        catch(DocumentException e) {
            System.err.println(e.getMessage());
        }
        catch(IOException e){
            System.err.println(e.getMessage());
        }
    }
```

# Private

## getOutputFile

Answer the full pathname for the output PDF file, taking into consideration the command-line options that specify the document number (--patent, --application), the output filename (--file), and the output directory (--dir). If no output filename or output directory is specified, this method constructs the default output filename which is the document number followed by a '.pdf' file extension. If either the output filename or directory is specified a file path is constructed incorporating that filename or directory.

```
private File getOutputFile()
{
   final String PDF_FILE_EXTENSION = ".pdf";
   final String UNKNOWN_FILENAME   = "unknown";

   File file = null;
   String filename = outputFilename;

   if ((filename == null) || (filename.length() == 0))
   {
      if ((documentNumber != null)
         && (documentNumber.length() > 0))
      {
         filename = documentNumber;
      }
      else
      {
         filename = UNKNOWN_FILENAME;
      }
      filename += PDF_FILE_EXTENSION;
   }

   if (outputDirectory != null
      && (outputDirectory.length() > 0))
   {
      file = new File(outputDirectory, filename);
   }
   else
   {
      file = new File(filename);
   }

   return file;
}
}
```

# Patent Office Website

The class `PatentOfficeWebsite` is the interface to the USPTO site. This class contains knowledge of the location of the image servers and how to fetch specific images for a given patent document number. Note that any changes in the implementation of the USPTO site will affect the implementation of this class.

The image-fetching algorithm implemented by this class is slightly different from the algorithm used in the pat2pdf script we mentioned in the section "Introduction" on page 2. The pat2pdf script performs a successive sequence of HTML page fetches, parsing each page to extract enough information to fetch the next HTML page. It acts as if an end-user were browsing through the USPTO website.

In contrast, the implementation of this class uses information posted on J. Matthew Buchanan's website, Promote the Progress[10], a blog focused on intellectual property and technology law issues.

In Matthew's blog posting, he notes that the USPTO maintains two image servers: patimg1.uspto.gov and patimg2.uspto.gov. In fact, the USPTO maintains four image servers, two for patent images, and two for application images. The application image servers are aiw1.uspto.gov and aiw2.uspto.gov.

If the patent document number has the final two digits of 00 to 49, use the '1' image server (patimg1.uspto.gov or aiw1.uspto.gov). If the final two digits are 50 to 99 use the '2' image server (patimg2.uspto.gov or aiw2.uspto.gov).

For example, given the patent number 6,185,183, one would fetch images from patimg2.uspto.gov, since the last two digits of the patent document number are 83. Figure 2 shows the patent document image servers.

[10] http://www.promotetheprogress.com/2004/04/deep_linking_to.html

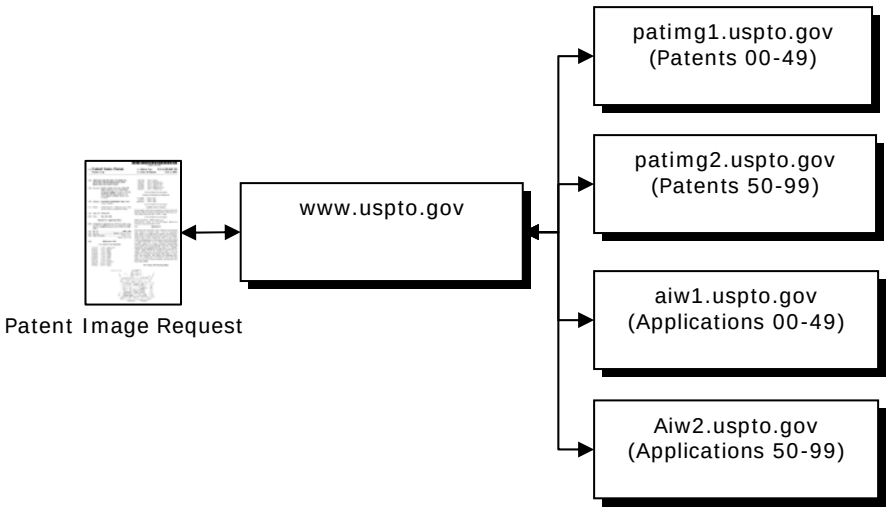


**Figure 2 USPTO Image Load Balancing**

```
class PatentOfficeWebsite {
```

# Public

## PatentOfficeWebsite

Create a new instance of the `PatentOfficeWebsite`.

```
public PatentOfficeWebsite()
{
}
```

## getDocument

Given a patent or application document number represented as a String, return an instance of a PatentDocument object. The PatentDocument object encapsulates all knowledge on how to fetch images from the given USPTO website. See the section "} Patent Documents" on page 21 for details on how to use the `PatentDocument` object.

This method is the workhorse of the USPTO site interface. For each document number we must:

1. Perform an HTTP GET of a "patent image page." The USPTO website will respond with HTML that has embedded within it the number of pages for the given patent document and an URL for where to fetch the document's images. The method that implements this step is `fetchPatentImagePage`.

2. Parse the HTML document returned in the previous step looking for the embedded page count. It turns out that the page count is embedded in a comment of the form <!—NumPages=XXX -->. The method that implements this step is `patentPageCount`.

3. Parse the HTML document returned in the first step looking for the URL where the actual document images are stored. This URL is identified by an <embed> tag. The method that implements this step is `patentImageURL`.

```
public PatentDocument getDocument(String documentNumber)
{
   String patentPage  = fetchPatentImagePage(documentNumber);
   int pageCount  = patentPageCount(patentPage);
   URL patentImageURL =
      patentImageURL(patentPage, documentNumber);

   return new Patent(
      documentNumber, pageCount, patentImageURL);
}
```

# Protected

## patentPageCount

Return the number of document pages embedded inside the USPTO web page represented by the argument. The argument must be an HTML page fetched from the USPTO website using the `fetchPatentImagePage` method.

This method has an intimate understanding of the format of the HTML page returned by the USPTO site. Specifically, it assumes the HTML contains a comment of the form <!-- NumPages = XXX -->, where XXX is an integer representing the number of pages in the patent document.

```
protected int patentPageCount(String patentPage)
{
   final String NUMBER_OF_PAGES_PREFIX = "-- NumPages=";
   final String NUMBER_OF_PAGES_SUFFIX = " --";

   int pageCount = 0;
   int pageIndex = patentPage.indexOf(NUMBER_OF_PAGES_PREFIX);

   if (pageIndex != -1)
   {
      try
      {
         String pageCountString = patentPage.substring(
            pageIndex + NUMBER_OF_PAGES_PREFIX.length());
         if (pageCountString != null)
         {
```

```
            int pageCountEnd =
               pageCountString.indexOf(NUMBER_OF_PAGES_SUFFIX)
               ;
            if (pageCountEnd != -1)
            {
               String pageCountNumber =
                  pageCountString.substring(
                                 0, pageCountEnd);
               if (pageCountNumber != null)
               {
                  pageCount =
                     Integer.parseInt(pageCountNumber);
               }
            }
         }
      }
      catch (NumberFormatException e) { }
   }

   return pageCount;
}
```

## patentImageURL

Return the URL of the image reference embedded inside the USPTO web page represented by the argument. The argument must be an HTML page fetched from the USPTO website using the `fetchPatentImagePage` method.

This method has an intimate understanding of the format of the HTML page returned by the USPTO site. Specifically, it assumes the HTML contains an `<embed>` tag whose 'src' attribute is the URL used to fetch individual page images for a given patent document.

The format of this URL is documented in the section "} Patent Documents" on page 21.

```
protected URL patentImageURL(
   String patentPage,
   String documentNumber)
{
   final String DOCUMENT_PREFIX = "<embed src=\"";
   final String DOCUMENT_SUFFIX = " ";

   URL imageURL  = null;
   int documentIndex = patentPage.indexOf(DOCUMENT_PREFIX);

   if (documentIndex != -1)
   {
      String documentString =
      patentPage.substring(documentIndex +
      DOCUMENT_PREFIX.length());
      if (documentString != null)
```

```
            {
               int documentEnd =
               documentString.indexOf(DOCUMENT_SUFFIX);
               if (documentEnd != -1)
               {
                  PatentDocumentNumber documentID = new
                  PatentDocumentNumber(documentNumber);
                  imageURL = imageURLForPatentDocumentNumber(
                  documentID,
                  documentString.substring(0, documentEnd));
               }
            }
         }

         return imageURL;
      }
```

# Private

## fetchPatentImagePage

As we described earlier in this software book, two important pieces of information are extracted from a USPTO website page for a given document: the page count and an URL on where to obtain its images.

This method performs all the work to take a document number, `patentNumber`, and return the proper USPTO website HTML page to extract the page count and image URL. This method returns the entire contents of the HTML page.

Any failure to fetch the document page will result in the return of an empty string. It is possible that this method returns a partial page.

The URL used to fetch the HTML page is documented in the method imageURLForPatentDocumentNumber.

```
private String fetchPatentImagePage(String patentNumber)
{
   PatentDocumentNumber patentDocumentNumber = new
   PatentDocumentNumber(patentNumber);
   URL serverURL =
      imageURLForPatentDocumentNumber(patentDocumentNumber);
   String page = null;

   try
   {
      InputStream imagePageStream = serverURL.openStream();

      StringBuffer result = new StringBuffer();
```

```
        BufferedReader reader = null;
        try
        {
           reader = new BufferedReader( new
           InputStreamReader(serverURL.openStream()) );
           String line = null;
           while ( (line = reader.readLine()) != null) {
           result.append(line);
           }
        }
        catch ( IOException ex )
        {
           System.err.println("Cannot retrieve contents of: " +
           serverURL);
        }
        finally
        {
           if (reader != null)
           {
              reader.close();
           }
        }
        page = result.toString();
     }
     catch (IOException e)
     {
        e.printStackTrace();
     }

     return page;
  }
```

## imageURLForPatentDocumentNumber

As we discussed in the method fetchPatentImagePage, an URL must be constructed for a given patent document number that represents the proper location on the UPSTO website to return patent page count and image URL information. This method contains the knowledge on how to build that URL given a patent document number.

The URL for a patent document has the form:

```
http://patimg1.uspto.gov/.piw?Docid=0documentNumber&idkey=NONE
```

The URL for a patent application document has the form:

```
http://aiw1.uspto.gov/.aiw?Docid=documentNumber&idkey=NONE
```

This method substitutes “documentNumber” in the above URL with the number associated with the argument, `documentID`. The method imageURLForPatentDocumentNumber/2 is responsible for calculating the appropriate server domain name and using the appropriate suffix (1 or 2)

of the patimg or aiw domain name attribute depending on the last two digits of documentID.

```
private URL imageURLForPatentDocumentNumber(
   PatentDocumentNumber documentID)
{
   String suffix = "/";

   if (documentID.isPatent())
   {
      suffix += ".piw?Docid=0";
   }
   else if (documentID.isPatentApplication())
   {
      suffix += ".aiw?Docid=";
   }
   suffix += documentID.toString();
   suffix += "&idkey=NONE";

   return imageURLForPatentDocumentNumber(
      documentID,
      suffix);
}
```

## imageURLForPatentDocumentNumber

This method works in conjunction with imageURLForPatentDocumentNumber/1. The responsibility of this method is to construct the proper server domain name, including which of the two load-balancing servers (1 or 2) is used for the given documentID. The algorithm to select which server to use is described in the section “Patent Office Website” on page 14.

```
private URL imageURLForPatentDocumentNumber(
   PatentDocumentNumber documentID,
   String suffix)
{
   String urlString = "http://";
   URL url   = null;

   if (documentID.isPatent())
   {
      urlString += PATENT_IMAGE_WEBSITE_PREFIX;
   }
   else if (documentID.isPatentApplication())
   {
      urlString += APPLICATION_IMAGE_WEBSITE_PREFIX;
   }
   else
   {
      throw new IllegalArgumentException(
               "Unknown patent document type");
   }
```

The following code extracts the last two characters from the document number and determines which of the two image servers (1 or 2) contain the document images.

```
   String patentNumberString = documentID.toString();
   if (patentNumberString.length() > 2)
   {
      patentNumberString = patentNumberString.substring(
         patentNumberString.length() - 2);
   }
   int patentNumber =
   Integer.valueOf(patentNumberString).intValue();
   urlString += ((patentNumber % 100) < 50) ? "1" : "2";
   urlString += ".";
   urlString += PATENT_IMAGE_WEBSITE_SUFFIX;
   urlString += suffix;

   try
   {
      url = new URL(urlString);
   }
   catch (MalformedURLException e)
   {
      e.printStackTrace();
   }

   return url;
}
```

### Static Variables

```
private static final String
   PATENT_IMAGE_WEBSITE_PREFIX  = "patimg";
private static final String
   APPLICATION_IMAGE_WEBSITE_PREFIX = "aiw";
private static final String
   PATENT_IMAGE_WEBSITE_SUFFIX  = "uspto.gov";
}
```

## Patent Documents

You have now arrived at some support classes that encapsulate the behavior of patent documents and patent document numbers.

Every patent document has an associated patent document number. The document number is different depending on the type of document: patents have one numbering system and patent applications have a different number systems.

The class `PatentDocumentNumber` can answer simple questions such as “Are you a patent?” or “Are you an application?” It can also answer an integer representation of itself. Note that integer conversion is currently

problematic because application numbers can overflow Java's native `int` type. The interface for a `PatentDocumentNumber` needs to change to accommodate this.

There is a small hierarchy of patent document classes. The base PatentDocument class is an abstract superclass for PatentApplication and Patent documents as shown in Figure 3.

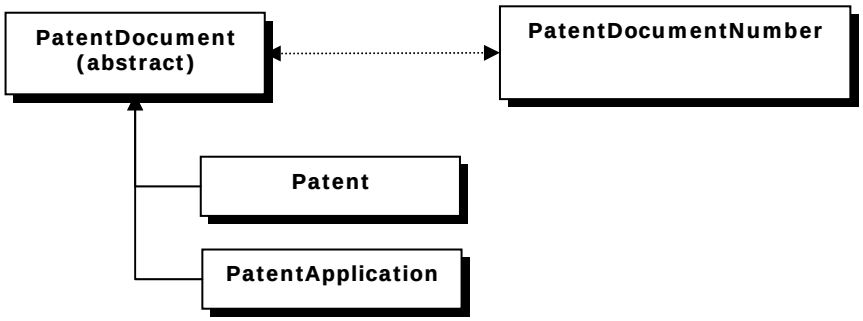


**Figure 3 Patent Document Hierarchy**

Each patent document class is responsible for implementing how to fetch images associated with the document (`getImage()`).

Readers should note that the patent number and patent document class hierarchy and interfaces are not well thought out. Feel free to extend, enhance, or re-write.

# PatentDocumentNumber

```
class PatentDocumentNumber {
```

## Private

### Members

```
private String id = null;
```

## Public

### PatentDocumentNumber

```
public PatentDocumentNumber(String aNumber)
{
   if (aNumber == null || aNumber.length() < 2)
   {
```

```
            throw new IllegalArgumentException(
               "Invalid patent document number");
         }

         id = aNumber;
      }
```

### isPatent

```
      public boolean isPatent()
      {
         return id.length() <= 7;
      }
```

### isPatentApplication

```
      public boolean isPatentApplication()
      {
         return id.length() > 7;
      }
```

### intValue

```
      public int intValue()
      {
         int value = 0;

         try
         {
            value = Integer.parseInt(id);
         }
         catch (NumberFormatException e)
         {
         }

         return value;
      }
```

### toString

```
      public String toString()
      {
         return id;
      }
      }
```

# PatentDocument

The PatentDocument class is an abstract base class for the various types of documents available from the USPTO site. Subclasses are responsible for determining the number of pages in the document and how to fetch a specific page image.

```
abstract class PatentDocument
{
   abstract public PatentDocumentNumber getDocumentNumber();
   abstract public int getPageCount();
   abstract public Image getImage(int page)
      throws DocumentException, IOException;
}
```

# PatentApplication

The class PatentApplication represents a patent application document on the USPTO site.

```
class PatentApplication extends PatentDocument {
```

## Private

### Members

A PatentApplication stores the document number (id), number of pages in the document (numberOfPages) as extracted from a USPTO site page, and an URL used to fetch images from the USPTO site ( (imageURL)). The imageURL must be modified depending on which page is fetched as documented in the getImage method.

```
private String id = null;
private int numberOfPages = 0;
private URL imageURL = null;
```

## Public

### PatentApplication

Create an instance of a PatentApplication for the given document number. The page count for this application was previously determined by

parsing a USPTO site page. The argument, images, is a prototype URL used to fetch document images. The format of the images URL is:

```
http://aiw1.uspto.gov/.DImg?Docid=US006000000&PageNum=1&IDKey=645DB7471AD7
&ImgFormat=tif
```

The getImage method will modify the PageNum URL parameter depending on which page is fetched.

```
public PatentApplication(
   String documentNumber, int pageCount, URL images)
{
   id = documentNumber;
   numberOfPages = pageCount;
   imageURL = images;
}
```

## getDocumentNumber

```
public PatentDocumentNumber getDocumentNumber()
{
   return new PatentDocumentNumber(id);
}
```

## getPageCount

```
public int getPageCount()
{
   return numberOfPages;
}
```

## getImage

```
public Image getImage(int page)
   throws DocumentException, IOException
{
   final String PAGE_NUMBER = "PageNum=";

   Image  image = null;
   String url   = imageURL.toString();
   int index = url.indexOf(PAGE_NUMBER);
   if (index != -1)
   {
      String suffix = url.substring(index +
      PAGE_NUMBER.length());
      int ampersandIndex = suffix.indexOf('&');
      if (ampersandIndex != -1)
      {
```

```
            suffix = suffix.substring(ampersandIndex + 1);
            String prefix = url.substring(0, index);
            URL newURL = new URL(
                  prefix + "PageNum=" + page + "&" + suffix);
            image = new PatentOfficeImage(newURL).getImage();
         }
      }

      return image;
   }
   }
```

# Patent

The class `PatentDocument` represents a patent document on the USPTO site.

```
class Patent extends PatentDocument {
```

## Private

### Members

A `PatentDocument` stores the document number (`id`), number of pages in the document (`numberOfPages`) as extracted from a USPTO site page, and an URL used to fetch images from the USPTO site ( (`imageURL`)). The `imageURL` must be modified depending on which page is fetched as documented in the `getImage` method.

```
private String id = null;
private int numberOfPages = 0;
private URL imageURL  = null;
```

## Public

### Patent

Create an instance of a `Patent` for the given document number. The page count for this application was previously determined by parsing a USPTO site page. The argument, `images`, is a prototype URL used to fetch document images. The format of the images URL is:

```
http://patimg1.uspto.gov/.DImg?Docid=US006000000&PageNum=1&IDKey=645DB7471
AD7&ImgFormat=tif
```

The getImage method will modify the PageNum URL parameter depending on which page is fetched.

```
public Patent(
   String documentNumber, int pageCount, URL images)
{
   id = documentNumber;
   numberOfPages = pageCount;
   imageURL = images;
}
```

## getDocumentNumber

```
public PatentDocumentNumber getDocumentNumber()
{
   return new PatentDocumentNumber(id);
}
```

## getPageCount

```
public int getPageCount()
{
   return numberOfPages;
}
```

## getImage

```
public Image getImage(int page) throws DocumentException,
IOException
{
   final String PAGE_NUMBER = "PageNum=";

   Image  image = null;
   String url   = imageURL.toString();
   int index = url.indexOf(PAGE_NUMBER);
   if (index != -1)
   {
      String suffix = url.substring(index +
      PAGE_NUMBER.length());
      int ampersandIndex = suffix.indexOf('&');
      if (ampersandIndex != -1)
      {
         suffix = suffix.substring(ampersandIndex + 1);
         String prefix = url.substring(0, index);
         URL newURL = new URL(
               prefix + "PageNum=" + page + "&" + suffix);
```

```
            image = new PatentOfficeImage(newURL).getImage();
         }
      }

      return image;
   }
   }
```

# Patent Images

The PatentOfficeImage class is a small helper class that stores a page image URL and performs the actual network fetch of the image from the USPTO site.

```
class PatentOfficeImage {
```

## Private

### Members

```
private URL url = null;
```

### Public

#### PatentOfficeImage

```
public PatentOfficeImage(URL imageURL)
{
   url = imageURL;
}
```

#### getImage

```
public Image getImage() throws DocumentException, IOException
{
   return Image.getInstance(url);
}
}
```

# Conclusions

The Patent Document Printer application was both an experiment in writing a Software Book and in building a utility that we have found useful in our patent infringement, litigation, and intellectual property work. Although there was an existing USPTO website fetch utility (pat2pdf), it is our hope that this simple port to an application server language may prove useful to others.

It is also our hope that some of the concepts we introduced in our essay, "Thinking with Style" will influence how others write software. The Appendix contains some thoughts and insights gleaned from writing this software book.

# References


1. Lanovaz, Daniel, "Thinking with Style," May 3, 2004. http://www.lanovaz.org/daniel/Shared%20Documents/Thinking%20with%20Style/Thinking_with_Style.pdf
2. Tirosh, Oren and Boege, Thomas, pat2pdf. http://www.tothink.com/pat2pdf/
3. Sullivan, Sean C., Dynamically Creating PDFs in a Web Application. http://www.onjava.com/pub/a/onjava/2003/06/18/dynamic_files.html


# Appendix

This appendix contains notes on what we learned from writing the patent printer software book using existing tools, in our case Microsoft Word 2003 Professional (with XML support).

1. A software book can quickly grow very large and there needs to be an effective way to split a "Master Book" into a series of slave books and have them linked properly. It is unclear at this time how to best structure a multi-book solution although following programming language package or module guidelines are a good starting point.
2. A software book editor needs a better understanding of the Compiled Code style similar to modern programming language text editors, such as automatically creating headings and highlighting keywords. Word does an excellent job manipulating the English components of the book, but needs further tools for the foreign language (programming language) components of the book.
3. From a programming language design perspective, we found that software book headings remove the need for language-specific modifier keywords. For example, when source code appears in a particular section such as "Public" there is no reason to prefix language statements with a "public" keyword. That is, book section modifiers replace some programming language modifiers.

4. Indentation of compiled code in a software book argues for the removal of block enclosure syntax such as the curly braces ('{', '}') used in languages such as C, C++, Java, C#, etc. Similar to Python, indentation on the page is more than adequate to signal to the language compiler the block structure.

5. Creating headings for programming language constructs such as classes, visibility designators (public, private, protected, etc.), and functions/methods allowed us to use Word's "Document Map" to quickly navigate the software book. The document map is shown in the following Figure 4. This proved useful in navigating the book.

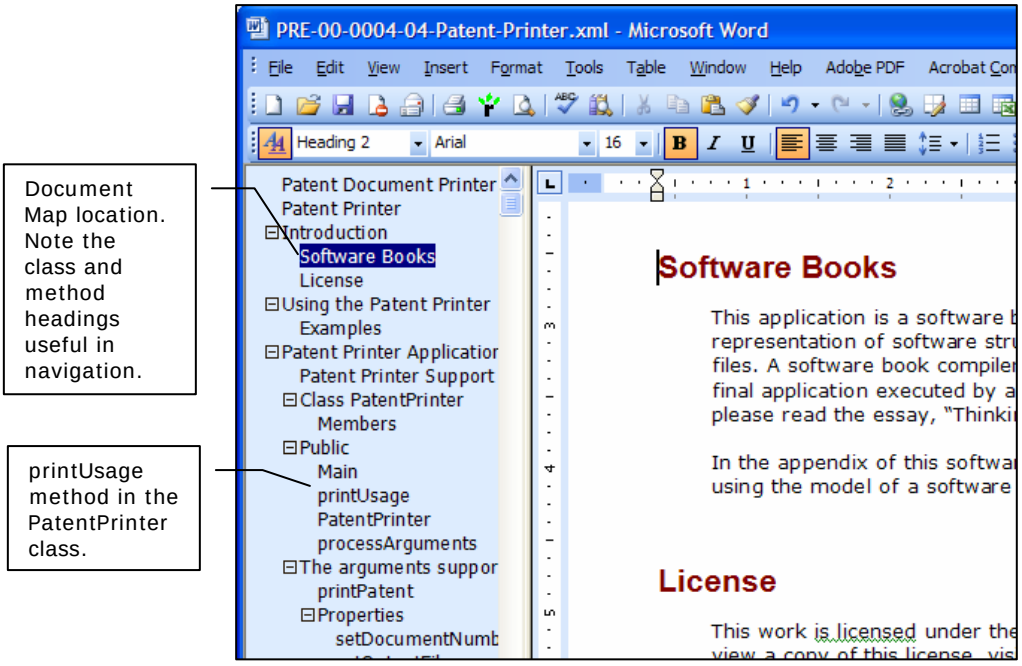


**Figure 4 Word Document Map**

6. There is no support in Word 2003 for embedding the RDF mark-up of Creative Commons Licenses, although the Creative Commons does support embedding licenses into PDF files.

7. The current implementation of the software book compiler is not integrated into Word 2003. The book is translated into Java source and then compiled by a Java compiler. Programming language errors are manually mapped to a location in the Word document. We need to create Word macros that automate this compilation step, or word with other individuals who are creating software book editors using OpenOffice, Squeak, or other programming environments.

8. Word automatically changes single quotes around a character constant into an open signle quote followed by a close single quote. E.g. 'a' becomes 'a'. This causes problems with the Java language compiler. Care must be taken to correct Word's quote autocorrect feature.

9. Word can automatically capitalize words it believes start a sentence. If you have this feature enabled, Word may capitalize programming

language constructs. Either watch out for this automatic change or turn off automatic capitalization.

10. We speculate that there must be some programming language design concepts that will allow a language to be better integrated into a software book, but we have no firm proposals at this time.