Getting Started

Adobe® XML/PDF Access API for Java™

Version 1.0

Adobe® XML/PDF Access API for Java™ Getting Started
July 2004

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

1 Getting Started

Thank you for downloading the Adobe® XML/PDF Access API for Java™ (XPAAJ) Software Development Kit (SDK). The XML/PDF Access API for Java SDK enables custom Java applications to manipulate Adobe PDF documents and their data. You can use the SDK to perform data input and output operations on PDF documents, including these operations:

- Extract and insert field data.
- Convert PDF documents to XDP format.
- Add metadata and file attachments.
- Add, replace, and delete embedded data objects, file attachments, and annotations.

XML/PDF Access API for Java includes all the features expected in a Java API, such as standard packaging. In addition to the class library, the SDK contains code samples that demonstrate these operations on PDF documents:

- Convert the PDF document to an XDP file.
- Extract and insert PDF document data, XMP metadata, embedded data objects, file attachments, and annotations.
- Extract static text.

The product package integrates easily with J2SE and J2EE systems. You can integrate the API into your custom application by using standard Java programming techniques.

XML/PDF Access API for Java applications

Custom applications that include XML/PDF Access API for Java may be found in academic institutions, advertising agencies, and corporate and government workplaces, to name only a few. A typical XML/PDF Access API for Java application may extract data from an interactive PDF form or insert data into a PDF form for presentation to users. Consider the following example application scenarios:

Processing application forms A college uses electronic forms to process applications from those seeking admission:

- An electronic form, created using Adobe Acrobat® Professional or Acrobat Standard, is emailed to applicants.
- The applicant fills the PDF form online, attaches an electronic check, and saves the form locally before emailing a copy of the form back to the college.
- At the registrar's office, a custom application processes the PDF form programmatically. First, the file attachment representing the check is extracted and saved to a folder on the registrar's computer. Next, the file attachment is deleted from the form. Finally, the form is routed to the academic department for review.

Archiving photographs A geological survey receives photographic images packaged in a PDF "envelope" that contains document and image metadata:

- A photographer at a remote location creates a PDF document and inserts digital photographs into the document.

- The photographer updates the document properties to include a document title, a subject line, and some keywords, and then emails the PDF document to the office.
- At the office, a custom application processes the PDF document programmatically. First, the document and image XMP metadata is extracted and saved to an archival folder. Then, the images in the PDF document are extracted and saved as a series of separate files that have sequentially numbered file names.

Exchanging corporate data A company processes information at a satellite office through PDF forms but stores the form data in an off-site central repository:

- A company uses Adobe Designer to create a PDF form that is capable of submitting an XDP file to a custom enterprise server application.
- The form is deployed to the satellite office.
- An employee at the satellite office fills the PDF form and submits it to a custom enterprise application as an XDP file. The XDP file encodes the PDF form in XML along with the form data.
- At the central repository, the custom enterprise application processes the XDP file programmatically. The XDP file is opened, and the encoded XML is converted to the schema used at the central repository. The form data is extracted and stored in the database.

Collecting data from various sources A government organization produces reports and circulates them for review as PDF documents:

- An information officer renders the report as a PDF document and posts the document on an intranet.
- Subject matter experts are asked to submit input as annotations and file attachments.
- Reviewers use a web browser to access the PDF document and use the commenting features of Adobe Reader® to mark up the report. In addition to adding annotations, some reviewers attach text files before uploading the revised PDF document to a web-based custom server application.
- At the publishing office, the custom server application processes the input. The annotations are exported and saved to a folder. The file attachments are extracted and saved to the same folder as a series of separate files that include the reviewer's name in the file name. The information officer is notified through email whenever a reviewer submits input.

Client applications can be delivered as Java applets or a stand-alone Java applications or, if web-based server-side processing is required, developers could implement JavaServer Page technology or a Java servlet (a typical enterprise application would deploy such applications as J2EE components).

What you received

The product download contains the following items:

- Product JAR file, XPAAJ.jar, which contains the product's class library.
- Product documentation, which comprises these documents:
  - Documentation Map (PDF)
  - Getting Started (PDF)
  - *Developer Guide* (PDF)
  - *API Reference* (PDF and Javadoc format)
- Sample applications, which demonstrate the capabilities of the product, including data querying, extraction, deletion, insertion, and conversion.

Installing the software

It takes only a few minutes to install the software. Explanations of the required installation steps are provided in the following sections.

➤ **To install the software:**

1. Download and decompress the ZIP file. (See "Downloading and decompressing the ZIP file" on page 6.)
2. Check the system requirements. (See "System requirements" on page 6.)
3. Move the JAR file into an appropriate location in your development environment. (See "Adding the class library to your development environment" on page 6.)

Downloading and decompressing the ZIP file

If you are reading this document, you have probably already downloaded and decompressed the ZIP file. It is assumed that you saved a copy of the ZIP file locally after you downloaded the software.

The ZIP file, XPAAJ.zip, contains the product software, documentation, and sample applications. These files and locations are created when you decompress the ZIP file:

- The XPAAJ.jar class library.
- The doc location, which contains the PDF product documentation and a copy of the *API Reference* in Javadoc format.
- The samples location, which contains the sample applications

It is recommended that you add the class library to your Java programming environment.

System requirements

XML/PDF Access API for Java is dependent on Java 2 Standard Edition 1.4.1 or later. The SDK has been tested on the following platforms:

- Microsoft® Windows XP® Professional
- Sun™ Microsystems™ SunOS™ 8
- Red Hat® Linux® 9.0

Adding the class library to your development environment

When you configured your Java programming environment, you set the `CLASSPATH` environment variable. The `CLASSPATH` environment variable defines where the Java compiler searches for Java class definitions at compile time and where the Java Virtual Machine (JVM) searches for Java class definitions at run time.

Whenever you compile a custom XML/PDF Access API for Java application using the java command, be sure to include the `-classpath` command-line parameter to explicitly specify the location of the XPAAJ.jar file. If you use an IDE, update the environment settings to include the XPAAJ.jar class library.

Accessing the documentation

When you decompressed the ZIP file, the Documentation Map, all supporting PDF documentation, and a Javadoc version of the *API Reference* were written to the doc location. You can access both types of documentation by using a web browser.

➤ **To access the PDF documentation through the Documentation Map:**

- Using a web browser enabled with Acrobat or Adobe Reader, navigate to the doc location and open the Documentation Map, the Doc_Map.pdf file.

➤ **To access the API Reference in Javadoc format:**

- Using a web browser, navigate to the doc/javadoc location and open the index.html file.

You can bookmark these pages in your web browser.

Running the sample applications

XML/PDF Access API for Java contains a number of sample applications. When you decompressed the ZIP file, three of the sample applications and two test files (test.pdf and HouseView.jpg) were written to the samples\command-line\ConsoleSample location.

About the sample applications

To demonstrate some of the basic capabilities of XML/PDF Access API for Java, you can compile the supplied .java files and run these three sample applications according to the procedures given in this guide:

- PDFExtractData.class extracts information from test.pdf and writes the data to a number of external files.
- PDFDeleteData.class deletes the comments and attachments from test.pdf and saves the changed version as a new PDF document.
- PDFImportData.class imports new information (HouseView.jpg) and some of the previously extracted data into a PDF document.

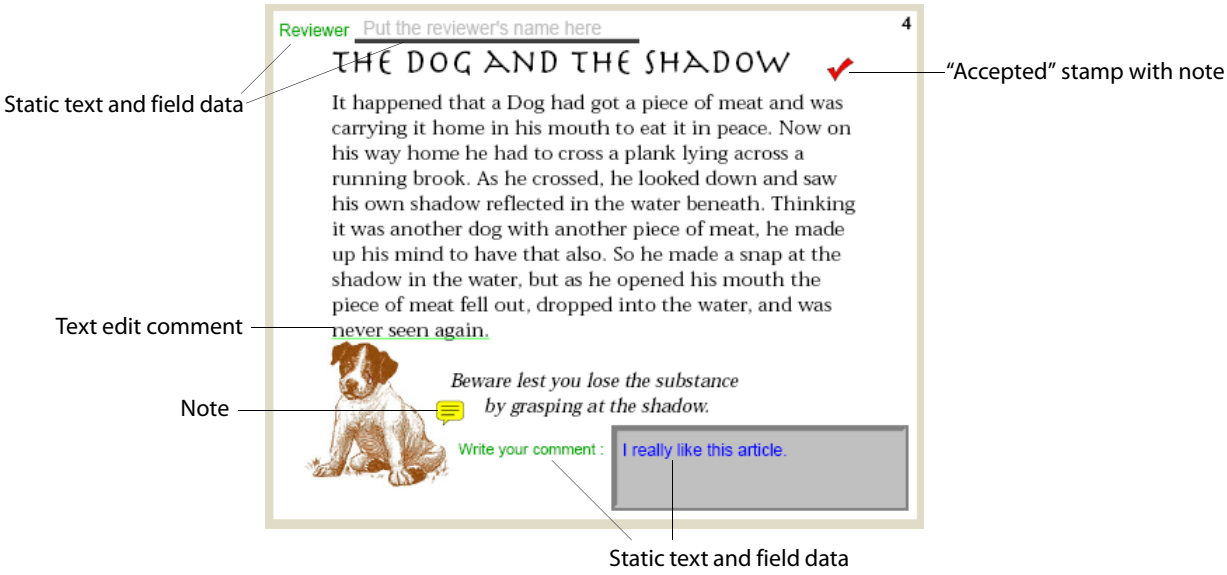
These sample applications demonstrate the basic usage and syntax of the product classes and interfaces. You can preview the sample .java code to determine how the applications work.

Under SunOS and Linux, these sample applications may be run from a terminal or Xterm. If you are a Windows user, run the applications from the Windows Command Prompt.

Note: The images and example commands in this section feature a Windows environment. If you work in a SunOS or Linux environment, the path separator is a forward slash (/), not a backslash (\).

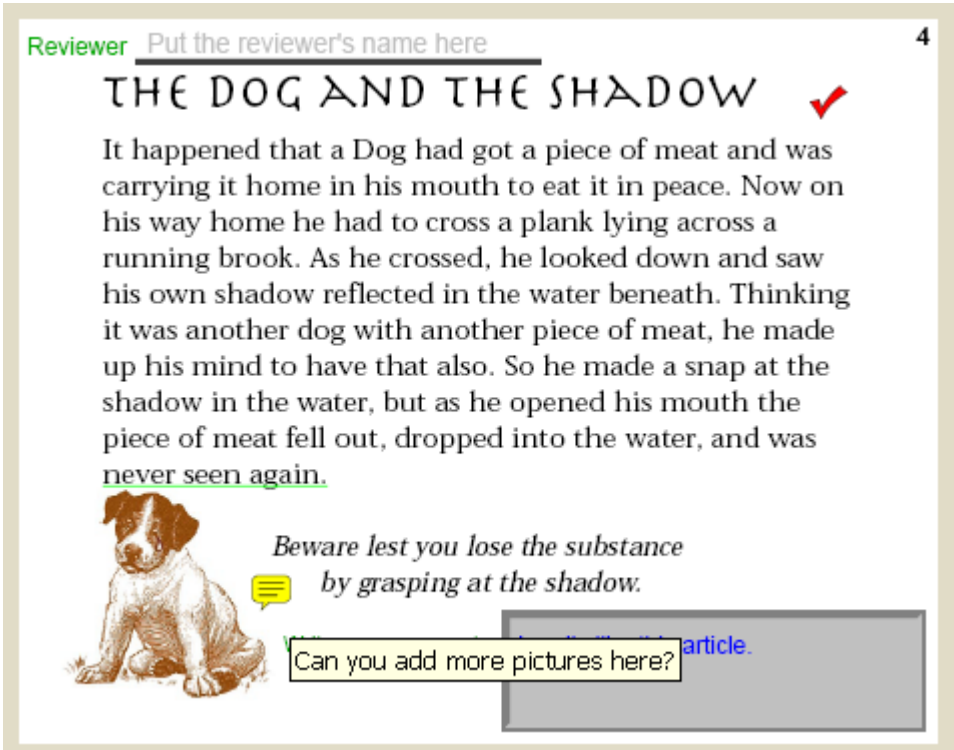
About test.pdf

The sample test file, test.pdf, contains static text objects, field data, and three embedded annotations. This file is also associated with four file attachments.



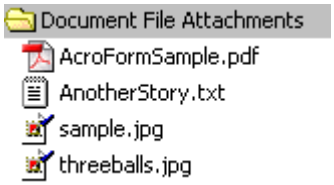
➤ **To view the note comments in test.pdf:**

1. In Adobe Reader, open test.pdf.
2. Select the Hand tool , and hover the pointer over the note icon.



➤ **To view the list of files attached to test.pdf:**

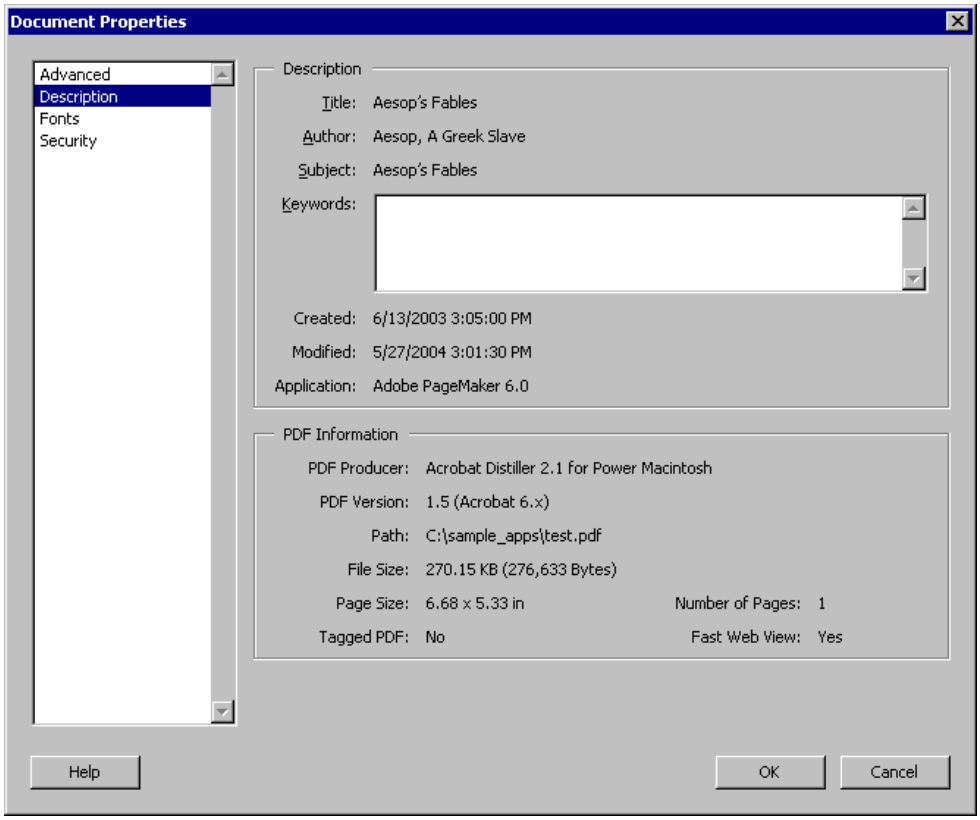
- In Adobe Reader, select Document > File Attachments. The list is displayed on the left side of the File Attachments dialog box:



The test.pdf file also stores information about the file itself (for example, PDF version and number of pages) and includes metadata, such as the document title, author, and subject.

➤ **To view the file information and metadata associated with test.pdf:**

1. In Adobe Reader, select File > Document Properties, and then select Description.



Configuring the run-time environment

Before you run the sample command-line applications, consider setting up the run-time environment according to these guidelines:

- Create a working directory (for example, C:\sample_apps).
- Compile the supplied .java files and copy PDFExtractData.class, PDFDeleteData.class, PDFImportData.class, test.pdf, and HouseView.jpg to the working directory.
- The sample applications use relative paths to refer to input and output files. Unless otherwise stated, all input files must be in the directory from which you execute the java command, and all output files are created in the directory from which you execute the java command.
- When you execute the java command, specify the location of the XPAAJ.jar file explicitly by including the `-cp` command-line parameter.

Extracting data from the PDF document

Run the PDFExtractData.class application to extract data from the sample test file. It is recommended that you run the sample application in your working directory.

➤ **To extract data from the PDF document:**

1. In the command window, switch to your working directory; for example:

   ```
   cd C:\sample_apps
   ```

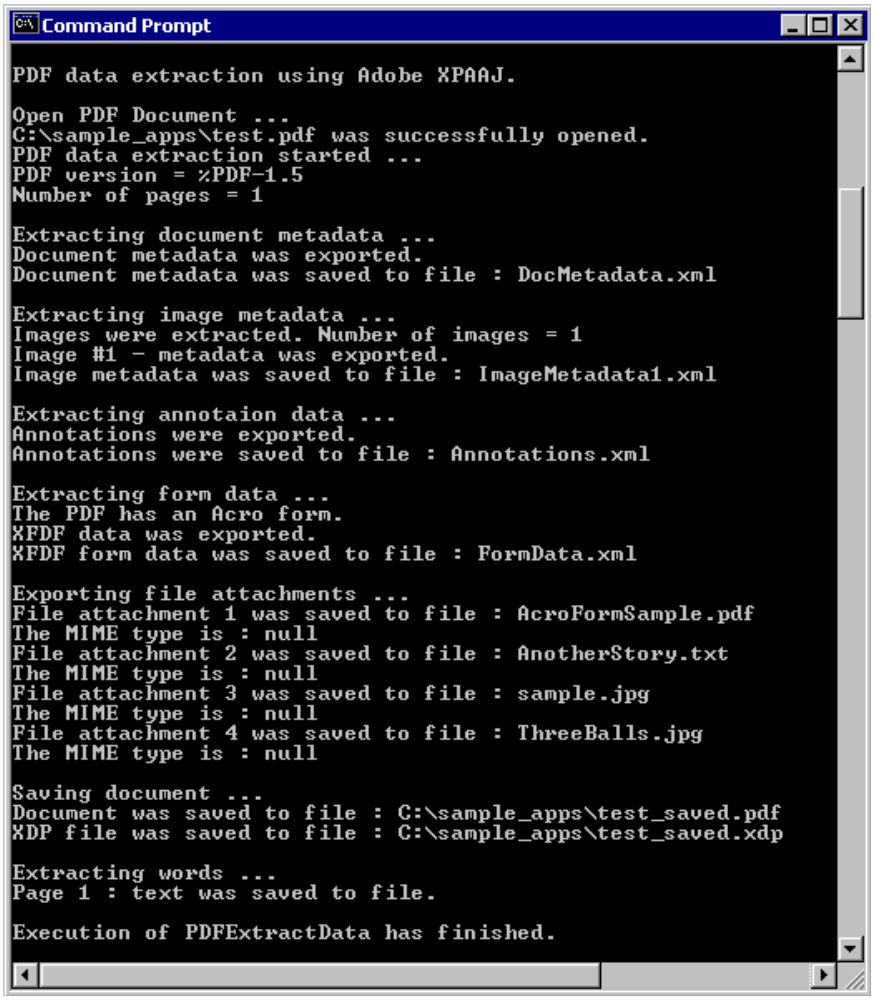
2. Type this command:

   ```
   java -cp <lib_path> PDFExtractData <PDF_path>
   ```

   where `<lib_path>` is the location of the product's class library and `<PDF_path>` is the location of the sample test file, test.pdf; for example:

   ```
   java -cp C:\Adobe\XPAAJ.jar PDFExtractData C:\sample_apps\test.pdf
   ```

Several files are created. The expected results are captured in the following image:



The application performs the following extraction operations on test.pdf:

- The PDF version and number of pages are displayed in the command window.
- The associated metadata is saved in a file called DocMetadata.xml.
- The image metadata is saved in a file called ImageMetadata1.xml.
- The embedded annotations are saved in a file called Annotations.xml.
- The static text and field data are saved in a file called FormData.xml.
- The file attachments are stored separately in files called AcroFormSample.pdf, AnotherStory.txt, sample.jpg, and ThreeBalls.jpg.
- A copy of the PDF document (test_saved.pdf) is written to disk, and an XDP variant of the file (test_saved.xdp) is created.
- The static text making up the page number and story are saved in a file called test_page1.txt.

You can open these files individually and examine their contents.

Deleting data from the PDF document

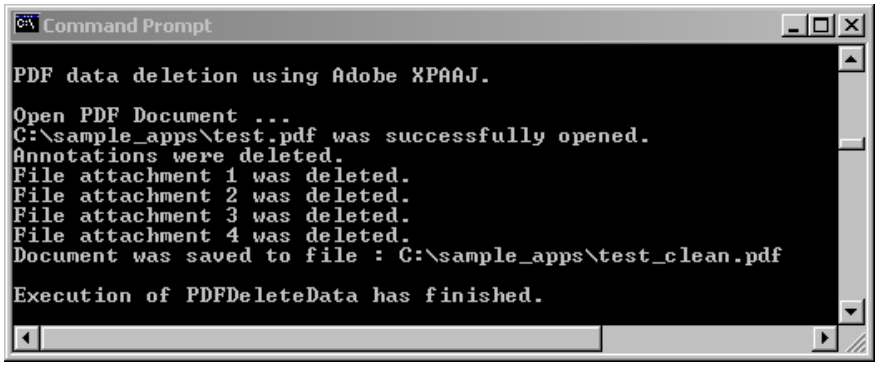
The next sample application, PDFDeleteData.class, deletes the embedded annotations and file attachments from test.pdf and saves the results in a new PDF document.

➤ **To delete data from the PDF document:**

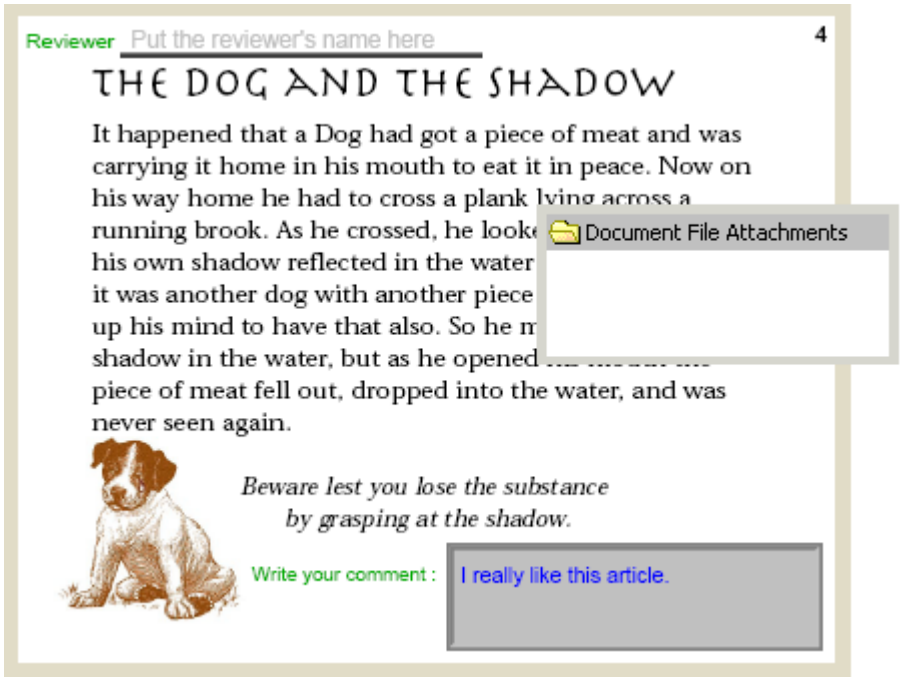
- In the command window, type this command:

```
java -cp <lib_path> PDFDeleteData <PDF_path>
```

where `<lib_path>` is the location of the product's class library, and `<PDF_path>` is the location of the sample test file, test.pdf. The expected results are captured in the following image:



A new output file called test_clean.pdf is created in your working directory. Opening the file in Adobe Reader shows that all annotations and file attachments were removed:



Importing data into the PDF document

The third sample application, PDFImportData.class, imports data from the files that were created when you extracted data from the sample test file. In addition, PDFImportData.class adds the sample input file, HouseView.jpg, to the PDF document as a file attachment. To clearly see the additions in the recipient PDF document after the import operation, we'll start with test_clean.pdf, which has no annotations or file attachments.

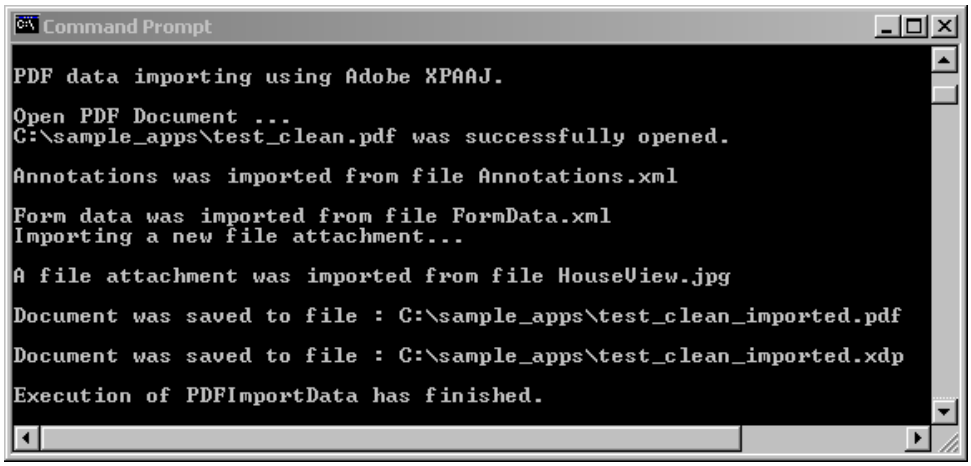
Note: For the purposes of this demonstration, the previously extracted data (test_clean.pdf) and HouseView.jpg must exist in the directory where you execute the java command. It is recommended that you run the sample application in your working directory.

➤ **To add data to the PDF document:**

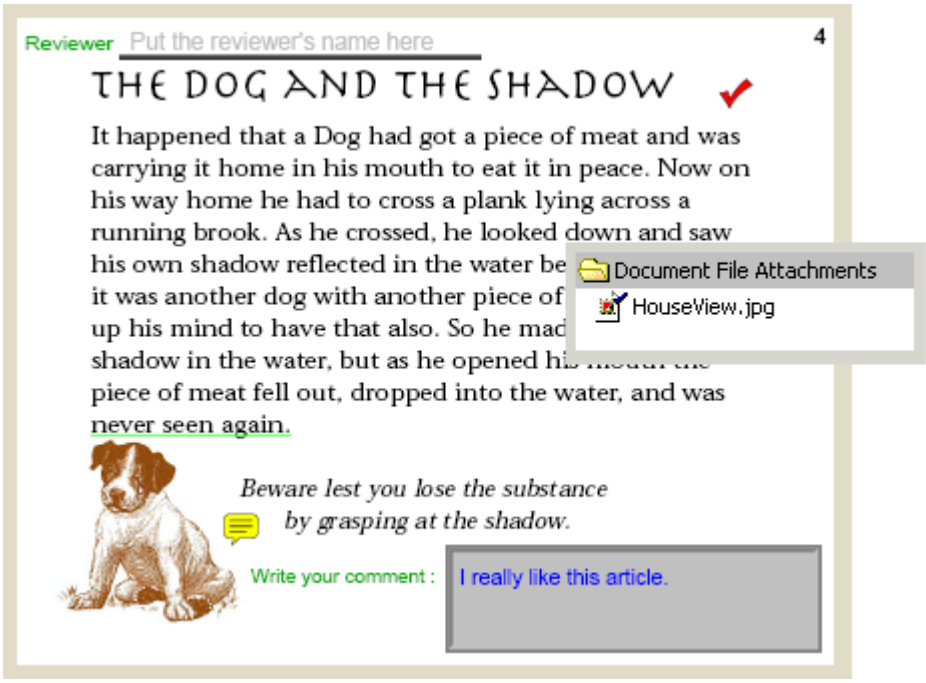
- In the command window, type this command:

  ```
  java -cp <lib_path> PDFImportData <PDF_path>
  ```

  where `<lib_path>` is the location of the product's class library, and `<PDF_path>` is the location of the new test file, test_clean.pdf. The expected results are captured in the following image:



The new output file, test_clean_imported.pdf, contains three embedded annotations, and the HouseView.jpg file is available as a file attachment.



What's next?

Additional sample applications are available. They demonstrate how to use XML/PDF Access API for Java in an enterprise workflow. Information about how to install them is provided in the samples location.