API Reference

Adobe® XML/PDF Access API for Java™

Version 1.0

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

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

Contents

Preface

This API Reference is one of several resources available to help you learn about Adobe® XML/PDF Access API for Java™ (XPAAJ). XML/PDF Access API for Java is a class library that you can use to create applications that perform import and export operations on Adobe PDF documents and XDP files.

What's in this guide?

This reference provides information about the public XML/PDF Access API for Java interfaces and classes defined in the com.adobe.pdf package. The following table identifies each interface and class, and suggests how they can be used in a custom application:

Interface or class	Purpose
`PDFFactory` class	Instantiates a `PDFDocument` object from an input stream that references an existing PDF document or an XDP file. Given the input stream, the class can associate the file name and MIME type of a file attachment with the `PDFDocument` object.
`PDFDocument` interface	Provides methods for performing simple queries; importing, exporting, and deleting data; and saving and converting an existing data stream.
`FileAttachment` interface	Provides methods for accessing the file name and MIME values associated with a file attachment, and the data associated with the file attachment.
`PDFImage` interface	Provides methods for exporting and importing the XMP metadata associated with an embedded image.
`PDFWord` interface	Defines an interface for accessing one word from a list of the static-text words in a PDF document. The methods identify on which page the word occurs, provide the positions of the characters making up the word, and identify the Unicode characters representing the word.
`Quad` interface	Used in conjunction with `PDFWord`, provides methods for obtaining the locations and orientations of the characters in a word. Each location is defined as a set of four points. Each set represents a bounding quadrilateral that defines the area occupied by one or more characters.
`Point` interface	Used in conjunction with the `Quad` interface, implements methods for obtaining the X and Y coordinates associated with a single corner of a bounding quadrilateral.
`FormDataFormat` class	Indicates in which format to export form data, either XML Form Data Format (XFDF) or XML Forms Architecture (XFA) format.
`FormType` class	Determines whether a file contains a form and, if it does, the class identifies the form type.

Who should use this guide?

This reference is intended to help Java developers understand XML/PDF Access API for Java objects and methods. It is a companion reference to the *Developer Guide*.

Related documentation

The resources listed in this table can help you learn more about this product:

For information about	**See**
Installing the library in a development environment	*Getting Started*
Product architecture, programming techniques, and code examples	*Developer Guide*
Other Adobe services and products	www.adobe.com

1 PDFFactory Class

`PDFFactory` defines a static `PDFFactory` object. Using the `PDFFactory` object, you can associate a data stream representing an existing PDF document or an XDP file with a `PDFDocument` object that implements the `PDFDocument` interface. For information, see "PDFDocument Interface" on page 9.

You can also use a `PDFFactory` object to create a `FileAttachment` object that implements the `FileAttachment` interface. For information, see "FileAttachment Interface" on page 18.

For information about using a `PDFFactory` object to create either a `PDFDocument` object or a `FileAttachment` object, see the *Developer Guide*.

newFileAttachment

Creates a new `FileAttachment` object from a given data stream. This attachment can then be used to import a data stream into a PDF document using the `PDFDocument` interface's `importFileAttachment` method. For information, see "importFileAttachment" on page 15.

Syntax

```
public FileAttachment newFileAttachment(java.io.InputStream src,
                                        java.lang.String filename,
                                        java.lang.String mimetype)
```

Parameters

`src`	A `java.io.InputStream` object that represents a file that is used as a file attachment.
`filename`	The file name.
`mimetype`	The MIME type of the file attachment.

Returns

A `FileAttachment` object.

openDocument

Creates a new `PDFDocument` object based on a `java.io.InputStream` object that references an existing PDF or XDP file.

Syntax

```
public static PDFDocument openDocument(java.io.InputStream input)
                                throws java.io.IOException, PDFException
```

Parameters

`input`	A `java.io.InputStream` object that represents an existing PDF or XDP file.

Returns

A `PDFDocument` object.

Throws

`java.io.IOException` if the `java.io.InputStream` object is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

2 PDFDocument Interface

`PDFDocument` defines a public interface for manipulating PDF documents. Using a `PDFDocument` interface, you can programmatically perform the following tasks:

- Import data into a PDF document.
- Export data from a PDF document.
- Attach and retrieve file attachments from a PDF document.
- Save and convert PDF documents.
- Work with annotations.

Note: For information about performing these tasks, see the *Developer Guide*.

deleteAnnotations

Deletes all annotations in a PDF document.

Syntax

```
public void deleteAnnotations()
                        throws PDFException
```

Details

This method deletes all annotations in a PDF document that were added using XFDF. Individual annotations cannot be deleted. For a complete list of annotations that can be added using XFDF, see the *Developer Guide*.

Throws

`PDFException` if a PDF exception is raised. For more information, see "PDFDocument Interface" on page 9.

deleteFileAttachment

Deletes a file attachment in a PDF document that is associated with the given name.

Syntax

```
public void deleteFileAttachment(byte[] name)
                          throws PDFException
```

Parameters

`name`	A byte array that specifies the name of the file attachment to delete.

Details

An exception is not thrown if the specified name does not exist. In this situation, this method does not perform any action.

Throws

`PDFException` if a PDF exception is raised. For more information, see "PDFDocument Interface" on page 9.

exportAnnotations

Exports a PDF document's annotations as a stream of XFDF XML data.

Syntax

```
public java.io.InputStream exportAnnotations()
                                    throws java.io.IOException, PDFException
```

Returns

A java.io.InputStream object that contains annotations as an XFDF XML data stream.

Details

For a complete list of annotations that can be exported, see the *Developer Guide*.

Throws

`java.io.IOException` if the returned `java.io.InputStream` is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

exportFileAttachment

Exports data associated with the given name as a `FileAttachment` object.

Syntax

```
public FileAttachment exportFileAttachment(byte[] name)
                      throws java.io.IOException, PDFException
```

Parameters

`name`	A byte array that specifies the file attachment to export.

Returns

A `FileAttachment` object that contains the data associated with the specified name. For information, see "FileAttachment Interface" on page 18.

Details

This method returns null if the name does not exist.

Throws

`java.io.IOException` if a Java input error occurs.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

exportFormData

Exports a form's data as an XFA or an XFDF data stream as specified by the `format` parameter.

Syntax

```
public java.io.InputStream exportFormData(FormDataFormat format)
                                  throws java.io.IOException, PDFException
```

Parameters

`format`	A `FormDataFormat` object that specifies whether the form's data is exported as an XFA or an XFDF data stream.

Returns

A `java.io.InputStream` object that contains the form's data as an XFDF or an XFA data stream.

Details

A form's data can only be exported as an XFA data stream if the `getFormType` method's return value is `FormType.XML_FORM`. For information about determining the form type, see the *Developer Guide*.

Throws

`java.io.IOException` if the returned `java.io.InputStream` is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

exportXMP

Exports the XMP metadata that is associated with a PDF document.

Syntax

```
public java.io.InputStream exportXMP()
                              throws java.io.IOException, PDFException
```

Returns

A `java.io.InputStream` object that contains metadata belonging to a PDF document.

Throws

`java.io.IOException` if the returned `java.io.InputStream` is invalid.
`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

getFileAttachmentNames

Returns an array of names as byte arrays that references file attachments.

Syntax

```
public byte[][] getFileAttachmentNames()
                        throws PDFException
```

Returns

An array of byte arrays where each byte array represents a file attachment name.

Throws

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

getFormType

Returns the form type on which a PDF document is based.

Syntax

```
public FormType getFormType()
                        throws PDFException
```

Returns

A `FormType` object that specifies a PDF document's form type. For information, see "FormType" on page 28.

Throws

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29 .

getImages

Returns a list of all images that are referenced by a PDF document.

Syntax

```
public java.util.List getImages()
                        throws PDFException
```

Returns

A `java.util.List` collection object that contains `PDFImage` objects. For information, see "PDFImage Interface" on page 19.

Throws

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29 .

getNumberOfPages

Returns the number of pages within the PDF document.

Syntax

```
public int getNumberOfPages()
                        throws PDFException
```

Returns

An integer value that represents the number of pages.

Throws

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

getVersion

Returns the version of the PDF document.

Syntax

```
public java.lang.String getVersion()
                      throws PDFException
```

Returns

A `java.lang.String` value that represents the version of the PDF document.

Throws

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

getWords

Returns a list iterator for accessing the words in the content of the PDF document.

Syntax

```
public java.util.ListIterator getWords()
                        throws PDFException
```

Returns

A `java.util.ListIterator` object that can be used for traversing the list of `PDFWord` objects. For information, see "PDFWord Interface" on page 21.

Throws

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

importAnnotations

Imports into a PDF document annotations that are represented in an XFDF XML data stream.

Syntax

```
public void importAnnotations(java.io.InputStream xfdfAnnotations)
                              throws java.io.IOException, PDFException
```

Parameters

`xfdfAnnotations`	A `java.io.InputStream` object that represents an XFDF data stream containing annotations to import.

Throws

`java.io.IOException` if the `java.io.InputStream` argument is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

importFileAttachment

Imports the specified `FileAttachment` object into a PDF document under the given name. If the specified name already exists, the current data is replaced. For information about a `FileAttachment` object, see "FileAttachment Interface" on page 18.

Syntax

```
public void importFileAttachment(byte[] name,
                                 FileAttachment fileAttachment)
                                 throws java.io.IOException, PDFException
```

Parameters

`name`	A byte array that specifies the name of the file attachment.
`fileAttachment`	A `FileAttachment` object that is imported into a PDF document.

Throws

`java.io.IOException` if a Java input error occurs.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

importFormData

Imports the form data as an XFA or an XFDF data stream.

Syntax

```
public void importFormData(java.io.InputStream src)
                           throws java.io.IOException, PDFException
```

Parameters

`src`	A `java.io.InputStream` object that represents data to import.

Throws

`java.io.IOException` if the `java.io.InputStream` argument is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

importXMP

Ensures that the input stream that represents XML is valid and uses it to replace the metadata referenced by the PDF document.

Syntax

```
public void importXMP(InputStream src )
                      throws java.io.IOException, PDFException
```

Parameters

`src`	A `java.io.InputStream` containing new metadata.

Throws

`java.io.IOException` if the `java.io.InputStream` argument is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

save

Returns the modified PDF document data stream.

Syntax

```
public java.io.InputStream save()
                          throws java.io.IOException, PDFException
```

Returns

A `java.io.InputStream` that represents the modified PDF document data stream.

Throws

`java.io.IOException` if the returned `java.io.InputStream` is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

saveAsXDP

Returns the modified PDF document as an XDP data stream.

Syntax

```
public java.io.InputStream saveAsXDP()
                            throws java.io.IOException, PDFException
```

Returns

A `java.io.InputStream` that represents the XDP data stream.

Throws

`java.io.IOException` if the returned `java.io.InputStream` is invalid.

`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

3 FileAttachment Interface

`FileAttachment` defines a public interface for referencing embedded file streams that can be archived or transmitted along with a PDF file. An *embedded file stream* is a file attachment that a user can open and view while the PDF document is open in Adobe Reader® (or Adobe Acrobat® Professional and Acrobat Standard). Using this interface, you can perform file attachment operations. For example, you can create a new file attachment and embed it into a PDF document.

getData

Returns the data of the file attachment.

Syntax

```
public java.io.InputStream getData()
```

Returns

A `java.io.InputStream` that contains the file attachment data.

getFilename

Returns the file name associated with the data embedded in the document.

Syntax

```
public java.lang.String getFilename()
```

Returns

A `java.lang.String` object that represents the file name.

getMimetype

Returns the MIME type of the data embedded in the document.

Syntax

```
public java.lang.String getMimetype()
```

Returns

A `java.lang.String` object that represents the MIME type of the embedded file.

4 PDFImage Interface

`PDFImage` defines a public interface for exporting or importing image XMP metadata.

Each image in a PDF document is represented by a `PDFImage` object. You can obtain a list of `PDFImage` objects through a `java.util.List` object, which is created by calling the `getImages` method of a `PDFDocument` object. For information about the `PDFDocument` object's `getImages` method, see "getImages" on page 13.

For information about how to export or import image XMP metadata, including example code, see the "Working with XMP Metadata" chapter in the *Developer Guide*.

The `PDFImage` interface defines the methods described in this section.

exportXMP

Exports the XMP metadata associated with an image.

Syntax

```
public java.io.InputStream exportXMP()
                           throws java.io.IOException, PDFException
```

Returns

A `java.io.InputStream` object that references the image XMP metadata. The metadata is returned as a stream of bytes. This operation does not modify the exported metadata in any way.

Throws

`java.io.IOException` if the `java.io.InputStream` object could not be created.
`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

importXMP

Imports the XMP metadata to associate with an image. A Java input stream must be able to reference the XMP source (for example, an XML file or an XML data stream).

Note: XML/PDF Access API for Java does not validate the XMP input; it determines whether the input is well-formed XML. However, the application is responsible for ensuring that the XMP stream to import is meaningful and correct.

Syntax

```
public void importXMP(java.io.InputStream metadata)
              throws java.io.IOException, PDFException
```

Parameters

`metadata`	A `java.io.InputStream` object that represents the image XMP metadata to import.

Throws

`java.io.IOException` if the XMP input is not well-formed.
`PDFException` if a PDF exception is raised. For more information, see "XML/PDF Access API for Java Exceptions" on page 29.

5 PDFWord Interface

`PDFWord` defines a public interface for accessing one word from a list of the static-text words in a PDF document. Custom XML/PDF Access API for Java applications can use the `PDFWord` interface to perform these types of operations:

- Find text on a page in a known location.
- Search through and/or index PDF content.
- Search for and highlight individual words.

You can obtain a list of the words in a PDF document through a `java.util.ListIterator` object, which is created by calling the `getWords` method of a `PDFDocument` object. For information about obtaining a list of the words and using a list iterator to step through the words, see the "Extracting Text" chapter in the *Developer Guide*.

getPageNumber

Gets the number of the page on which a particular word occurs.

Syntax

```
public int getPageNumber()
```

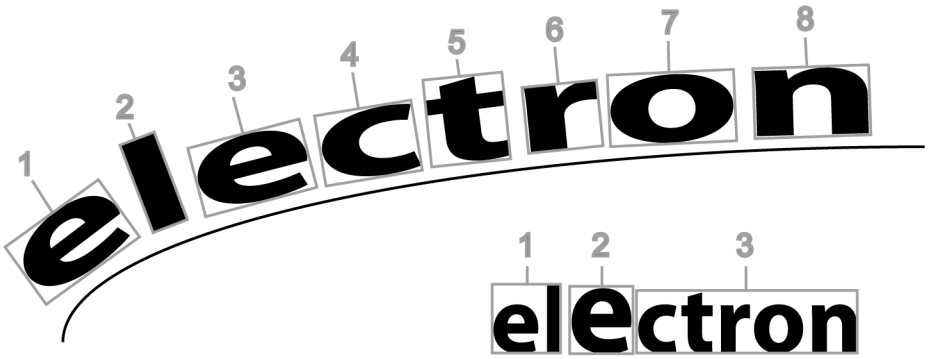
Returns

An integer that represents the page number.

getQuadList

Gets the list of `Quad` objects associated with a word. The items in the list represent the areas occupied by the characters in a word. A single `Quad` object may describe the area occupied by more than one character, depending on the attributes and orientations of the characters that make up the word. The characters could all be the same size and font, but that is not always the case. Also, because text can be aligned to a path, the orientation of each character could change from one character to the next.

The following conceptual diagram illustrates two examples of the same word and indicates how many `Quad` objects could be referenced in the list, depending on the attributes and orientation of each character in the word.



In the first example, the baseline of the text is aligned to a path. A path can be open (like an arc) or closed (like a circle). The characters are the same size and font; however, because the baseline associated with each character has a different orientation, more than one `Quad` object is required to accurately describe the orientation of each character.

In the second example, one of the characters is larger than the other characters in the word; therefore, three `Quad` objects are needed to accurately describe the overall area occupied by the word. If all the characters in a word have the same size, font, and orientation, a list referencing only one `Quad` object is returned.

Syntax

```
public java.util.List getQuadList()
```

Returns

A `java.util.List` list that references one or more `com.adobe.pdf.Quad` objects. You can access a `Quad` object through the `Quad` interface. For example code, see "Quad Interface" on page 23.

getString

Gets the Unicode characters that represent a particular word.

Syntax

```
public java.lang.String getString()
```

Returns

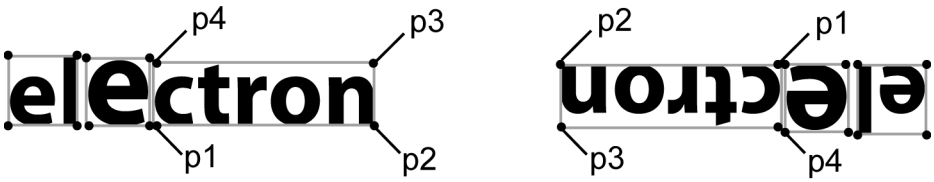
A `java.lang.String` object that identifies a Unicode-encoded string.

6 Quad Interface

`Quad` defines a public interface for obtaining the location and orientation of one or more characters in a word. The interface supports data extraction and text positioning. It also enables custom applications to highlight the static text in a PDF document.

The `getQuadList` method of a `PDFWord` object returns a list of `Quad` objects. (See `getQuadList` on page 21.) Each `Quad` object is associated with a set of four corner points: p1, p2, p3, and p4. The points represent the corners of a bounding quadrilateral that defines the area occupied by one or more characters. If the characters are aligned to a path, the points indicate the orientation of the characters along the path.

When the word is not rotated, corner point p1 defines the lower left corner of one or more characters, point p2 defines the lower right corner, p3 defines the upper right corner, and p4 defines the upper left corner. If the word were rotated 180°, p1 would be in the upper right corner and p3 would be in the lower left corner.



You can use the methods described in this chapter to get the locations of points p1, p2, p3, and p4. The following example shows how to get the locations:

```
//Iterate over every word in the PDF document.
ListIterator words = pdfDocument.getWords();
while (words.hasNext())
{
   PDFWord word = (PDFWord)words.next();

   //Iterate over every Quad object in the list.
   ListIterator quads = word.getQuadList();
   while (quads.hasNext())
   {
      Quad quad = (Quad)quads.next();

      //Get the point locations.
      Point p1, p2, p3, p4;
      p1 = quad.p1();
      p2 = quad.p2();
      p3 = quad.p3();
      p4 = quad.p4();
   }
}
```

p1

Gets the lower left corner of the bounding quadrilateral.

Syntax

```
public Point p1()
```

Returns

A `com.adobe.pdf.Point` object, which references the X-Y coordinate of point p1. You can access the object through the `Point` interface. (See "Point Interface" on page 26.)

p2

Gets the lower right corner of the bounding quadrilateral.

Syntax

```
public Point p2()
```

Returns

A `com.adobe.pdf.Point` object, which references the X-Y coordinate of point p2. You can access the object through the `Point` interface. (See "Point Interface" on page 26.)

p3

Gets the upper right corner of the bounding quadrilateral.

Syntax

```
public Point p3()
```

Returns

A `com.adobe.pdf.Point` object, which references the X-Y coordinate of point p3. You can access the object through the `Point` interface. (See "Point Interface" on page 26.)

p4

Gets the upper left corner of the bounding quadrilateral.

Syntax

```
public Point p4()
```

Returns

A `com.adobe.pdf.Point` object, which references the X-Y coordinate of point p4. You can access the object through the `Point` interface. (See "Point Interface" on page 26.)

7 Point Interface

`Point` defines a public interface for obtaining the X and Y values associated with one corner of the area that one or more characters occupy. The X value defines the horizontal position of the corner, and the Y value defines the vertical position of the corner. Both positions are relative to the upper left corner of the page, which has the X-Y coordinate 0,0.

After you create a `Quad` object, you can call the `p1`, `p2`, `p3`, or `p4` method to get the X-Y coordinate associated with one corner. (See "Quad Interface" on page 23). Afterward, you can use the methods described in this chapter to get the X and Y values of the corner.

Assuming that the locations p1, p2, p3, and p4 have been obtained, the following example shows how to get the X and Y values associated with each point:

```
double x1, y1, x2, y2, x3, y3, x4, y4;

//Get the values for point p1.
x1 = p1.x()
y1 = p1.y()

//Get the values for point p2.
x2 = p2.x();
y2 = p2.y();

//Get the values for point p3.
x3 = p3.x();
y3 = p3.y();

//Get the values for point p4.
x4 = p4.x();
y4 = p4.y();
```

x

Gets the horizontal position of a corner.

Syntax

```
public double x()
```

Returns

A 64-bit number with one or more digits after the decimal point.

y

Gets the vertical position of a corner.

Syntax

```
public double y()
```

Returns

A 64-bit number with one or more digits after the decimal point.

8 Related Classes

This chapter describes the XML/PDF Access API for Java classes that can be used in custom applications that export form data.

FormType

Before you can export form data, you need to determine whether a PDF document has a form and, if it does, what type of form it has. The `getFormType` method of a `PDFDocument` object returns a `FormType` object, which indicates the type of form. For information about how to determine the form type, see the "Invoking XML/PDF Access API for Java" chapter in the *Developer Guide*.

The `FormType` class defines these form types:

`FormType.NOT_A_FORM`	The PDF document does not contain a form.
`FormType.ACROFORM`	The PDF document contains an Acrobat form. Forms of type `ACROFORM` use the XFDF schema. (See "FormDataFormat" on page 28.)
`FormType.XML_FORM`	The PDF document contains an XML Forms Architecture stream. Forms of type `XML_FORM` use the XML Forms Architecture schema. (See "FormDataFormat" on page 28.)

FormDataFormat

The `FormDataFormat` class indicates in which format to export form data. You need to determine whether a file is a form before you can export any form data. (See `FormType` on page 28.) Once the form type is established, you can reference one of these `FormDataFormat` field values in a call to the `PDFDocument` object's `exportFormData` method:

`FormDataFormat.XFDF`	The XFDF schema is used to represent the data of an Acrobat form in XML.
`FormDataFormat.XFA`	The XML Forms Architecture schema is used to represent XFA stream data in XML.

The `exportFormData` method exports the form data as specified through the `FormDataFormat` input parameter. For information about exporting form data, see the *Developer Guide*.

A XML/PDF Access API for Java Exceptions

`PDFException` is the base exception for exceptions that are thrown by XML/PDF Access API for Java methods. You can handle exceptions using `PDFException`, as the following example shows:

```
try
   {
   //application logic
   }
catch (PDFException pdfException ){}
```

However, because `PDFException` is the base exception class, it represents a general non-descriptive exception. An exception that is derived from `PDFException`, such as `XMPException` or `XMPParseException`, is more descriptive then `PDFException`. For example, `XMPParseException` means there is a problem with the XMP data stream.

The `PDFException` hierarchy enables client applications to handle all PDF exceptions that may result from a specific operation or catch a particular exception such as `XMPParseException`. The following example shows how to handle `XMPParseException`:

```
   try {
      pdfImage.importXMP(xmpInputStream);
      }
   catch (PDFException pdfException) {
      if(pdfException instanceof XMPParseException)
         System.out.println("The XMP data stream is bad.");
       else
         System.out.println("The PDF exception was raised due to " +
        pdfException.getMessage());
      }
```

For information about the `PDFImage` interface's `importXMP` method, see `importXMP` on page 20.

The following table lists the possible PDF exceptions and the problem associated with each PDF exception:

Exception	Problem
`PDFException`	The base exception for all PDF exceptions raised by XML/PDF Access API for Java.
`PDFParseException`	Parsing a given PDF document. **Base exception:** `PDFException`
`PDFSecurityException`	The given PDF document is encrypted. **Base exception:** `PDFException`
`XFDFException`	Importing or exporting XFDF form data. **Base exception:** `PDFException`
`XFDFParseException`	Parsing a given XFDF stream. **Base exception:** `XFDFException`

Exception	Problem
`XFAException`	Importing or exporting XFA form data. **Base exception:** `PDFException`
`XFAParseException`	Parsing a given XFA stream. **Base exception:** `XFAException`
`XDPException`	Opening or saving a XDP stream. **Base exception:** `PDFException`
`XDPParseException`	Parsing a given XDP stream. **Base exception:** `XDPException`
`XMPException`	Importing or exporting a XMP stream. **Base exception:** `PDFException`
`XMPParseException`	Parsing a given XMP stream. **Base exception:** `XMPException`

Index

O

P

Q

S

X

Y