Communications – Document Manipulation APIs and DDX Reference

Forms as a Cloud Service

March 2022
Legal Notices
For more information, see help.adobe.com/en_US/legalnotices/index.html.
Dynamically insert forms or form fragments into an XFA form ................................................................. 40
Resolve references ........................................................................................................................................... 41
Package an XDP document as PDF .................................................................................................................. 42
  PDF documents from single XFA-based forms ............................................................................................... 43
Assemble XFA-based forms with other documents ......................................................................................... 43
  PDF documents from Acrobat forms ............................................................................................................. 45
Package a PDF document as XDP .................................................................................................................... 45

6 Assembling PDF Packages and Portfolios .................................................................................................. 46
Understanding PDF packages .......................................................................................................................... 46
About PDF package and portfolio properties ................................................................................................. 47
  PDF Package property: package files and package specifications ................................................................. 47
PDF Portfolio properties .................................................................................................................................. 48
Folders .......................................................................................................................................................... 50
Navigation welcome page and navigation heading ......................................................................................... 50
Creating a PDF Portfolio ................................................................................................................................... 51
Creating a PDF package ........................................................................................................................................ 52
Change the cover sheet for an existing PDF package or portfolio ................................................................. 53
  Choose a new cover sheet ................................................................................................................................ 53
Add or remove pages to an existing cover sheet ............................................................................................. 54
Creating a package or portfolio specification from other ones ......................................................................... 55
  Creating a package or portfolio specification by aggregating existing ones .................................................. 55
Selecting the package specification from an existing package ........................................................................... 55
Overriding properties in merged package or portfolio specifications ............................................................ 56
Modifying the package files in a PDF package or portfolio ............................................................................ 57
  Adding single files to an existing PDF package or portfolio ......................................................................... 57
Adding documents from a PDF package or portfolio to another ..................................................................... 57
  Modifying selected files in a PDF package or portfolio ................................................................................. 58
Exporting and importing package files ............................................................................................................. 58
Converting a PDF package or portfolio into a single PDF .................................................................................. 59

7 Disassembling PDF Documents .................................................................................................................. 61

8 Working with Bookmarks and Thumbnails ............................................................................................... 63
Including and excluding bookmarks ................................................................................................................. 63
Exporting and importing bookmarks .................................................................................................................. 64
  Exporting book marks from a PDF document ................................................................................................. 64
Importing bookmarks into a PDF document .......................................................................................................... 65
Creating bookmarks from source documents .................................................................................................... 65
Sorting bookmarks ................................................................................................................................................ 67

9 Working with Annotations ........................................................................................................................... 71
Including and excluding comments .................................................................................................................... 71
Importing and exporting comments ...................................................................................................................... 72
Selecting specific comments ............................................................................................................................... 74
Working with links ................................................................................................................................................ 77
  Removing links ................................................................................................................................................ 78
Rationalizing links ............................................................................................................................................... 78

10 Working with File Attachments .................................................................................................................. 80
Preserving and deleting file attachments ............................................................................................................ 80
Attaching files to a PDF document ...................................................................................................................... 81
Document-level file attachments ....................................................................................................................... 81
11 Adding Table of Contents or Blank Pages to an Assembly .................................................... 85
   Adding a table of contents ........................................................................................................... 85
   Formatting a table of contents.................................................................................................... 87
   Applying page properties and content to particular pages....................................................... 87
   Applying entry styles to specific line levels ............................................................................ 88
   Adding blank pages .................................................................................................................. 89

12 Setting Other Document Properties ...................................................................................... 91
   Working with metadata .............................................................................................................. 91
   Modifying metadata properties ............................................................................................... 92
   Working with layers ................................................................................................................. 93
   Setting the initial view .............................................................................................................. 93
   Using document-level JavaScript .......................................................................................... 93

13 Setting Page Properties .......................................................................................................... 95
   Applying page properties ........................................................................................................... 95
   Page size and rotation .............................................................................................................. 95
   Changing page size ................................................................................................................ 96
   Rotation and orientation ....................................................................................................... 97
   Interaction of page properties and content .......................................................................... 98
   Prepress settings .................................................................................................................... 99

14 Adding and Manipulating Page Content ............................................................................... 102
   Adding and removing headers and footers ........................................................................... 102
   Adding headers and footers ................................................................................................. 103
   Removing headers and footers ............................................................................................. 105
   Adding and removing watermarks and backgrounds .......................................................... 106
   Adding page content ............................................................................................................ 108
   Overlaying and underlaying pages ....................................................................................... 108
   Understanding rendering order ............................................................................................ 109
   Understanding blending color spaces ................................................................................ 110
   Specifying styled text ............................................................................................................ 111
   Style attributes .................................................................................................................... 112
   Applying identifying labels ................................................................................................. 113
   Built-in keys ....................................................................................................................... 114
   Using style profiles .............................................................................................................. 115
   Formatting dates .................................................................................................................. 117

15 Specifying Page Labels .......................................................................................................... 121
   About page labels .................................................................................................................. 121
   Specifying page labels ........................................................................................................... 123
   Removing page labels .......................................................................................................... 127

16 Working with Secured Documents ....................................................................................... 129
   Specifying passwords .......................................................................................................... 129
   Accessing a password-protected document ........................................................................ 131
   Digital signatures ................................................................................................................. 131

17 Querying Documents ................................................................................................................ 133
   Getting document information ................................................................................................. 133
Getting the text of a document ...................................................................................................................... 133
Getting information about the DDX processor .............................................................................................. 134

Part II: DDX Reference

18 DDX Concepts ............................................................................................................................................... 137
  Element relationships and roles ...................................................................................................................... 137
  Attributes, child elements, and text content ................................................................................................. 137
    Attribute names, formatting, and possible values ............................................................................................ 137
    Child elements .................................................................................................................................................. 138
    Text content ................................................................................................................................................... 139
  Element categories .......................................................................................................................................... 139
    Document assembly ...................................................................................................................................... 140
    Document components ................................................................................................................................. 141
    Document disassembly ................................................................................................................................. 143
    Document properties .................................................................................................................................... 143
    Page labels .................................................................................................................................................... 144
    Page properties ............................................................................................................................................. 144
    Page content ................................................................................................................................................ 145
    Profile ............................................................................................................................................................ 145
    Query ............................................................................................................................................................. 146
  Built-in keys .................................................................................................................................................. 146
    _AdobeCoverSheet ........................................................................................................................................ 148
    Color-specifier .............................................................................................................................................. 150
    External Data URL ....................................................................................................................................... 150
    Page and document ranges .......................................................................................................................... 154
    length-specifier ............................................................................................................................................ 154

19 DDX Language Elements ............................................................................................................................ 156
  About ............................................................................................................................................................. 156
  ArtBox ........................................................................................................................................................... 156
  AttachmentAppearance ................................................................................................................................. 157
  Author ............................................................................................................................................................ 158
  Background .................................................................................................................................................. 159
  BlankPage .................................................................................................................................................... 161
  BleedBox ....................................................................................................................................................... 162
  Bookmarks .................................................................................................................................................... 163
    Bookmarks result .......................................................................................................................................... 164
    Bookmarks source ....................................................................................................................................... 164
    Bookmarks filter ......................................................................................................................................... 165
    Center .......................................................................................................................................................... 166
    ColorScheme ............................................................................................................................................... 166
  Comments ...................................................................................................................................................... 166
    Comments result .......................................................................................................................................... 167
    Comments source ....................................................................................................................................... 169
    Comments filter .......................................................................................................................................... 170
  DatePattern .................................................................................................................................................. 167
  DDX ............................................................................................................................................................... 171
  DDXProcessorSetting .................................................................................................................................... 174
  DisplayOrder ................................................................................................................................................ 175
  DocumentInformation ................................................................................................................................. 177
<table>
<thead>
<tr>
<th>DocumentText</th>
<th>177</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>178</td>
</tr>
<tr>
<td>Field contained in Schema element</td>
<td>178</td>
</tr>
<tr>
<td>Field contained in DisplayOrder element</td>
<td>179</td>
</tr>
<tr>
<td>Field contained in SortOrder element</td>
<td>179</td>
</tr>
<tr>
<td>FieldData</td>
<td>180</td>
</tr>
<tr>
<td>File</td>
<td>181</td>
</tr>
<tr>
<td>FileAttachments</td>
<td>181</td>
</tr>
<tr>
<td>FileAttachments result</td>
<td>181</td>
</tr>
<tr>
<td>FileAttachments document-level source</td>
<td>183</td>
</tr>
<tr>
<td>FileAttachments page-level source</td>
<td>183</td>
</tr>
<tr>
<td>FilenameEncoding</td>
<td>184</td>
</tr>
<tr>
<td>FileSize</td>
<td>186</td>
</tr>
<tr>
<td>Folder</td>
<td>187</td>
</tr>
<tr>
<td>Footer</td>
<td>188</td>
</tr>
<tr>
<td>Header</td>
<td>190</td>
</tr>
<tr>
<td>Header (portfolio navigation pane)</td>
<td>193</td>
</tr>
<tr>
<td>InitialViewProfile</td>
<td>193</td>
</tr>
<tr>
<td>JavaScript</td>
<td>196</td>
</tr>
<tr>
<td>Keyword</td>
<td>197</td>
</tr>
<tr>
<td>Keywords</td>
<td>198</td>
</tr>
<tr>
<td>Left</td>
<td>198</td>
</tr>
<tr>
<td>LinkAlias</td>
<td>199</td>
</tr>
<tr>
<td>Links</td>
<td>199</td>
</tr>
<tr>
<td>Links result</td>
<td>200</td>
</tr>
<tr>
<td>Links source</td>
<td>200</td>
</tr>
<tr>
<td>Links filter</td>
<td>201</td>
</tr>
<tr>
<td>MasterPassword</td>
<td>201</td>
</tr>
<tr>
<td>Metadata</td>
<td>202</td>
</tr>
<tr>
<td>Metadata result</td>
<td>202</td>
</tr>
<tr>
<td>Metadata source</td>
<td>203</td>
</tr>
<tr>
<td>MetadataSchemaExtension</td>
<td>203</td>
</tr>
<tr>
<td>Navigator</td>
<td>204</td>
</tr>
<tr>
<td>NoBackgrounds</td>
<td>204</td>
</tr>
<tr>
<td>NoBookmarks</td>
<td>205</td>
</tr>
<tr>
<td>NoComments</td>
<td>205</td>
</tr>
<tr>
<td>NoFileAttachments</td>
<td>205</td>
</tr>
<tr>
<td>NoFooters</td>
<td>206</td>
</tr>
<tr>
<td>NoForms</td>
<td>206</td>
</tr>
<tr>
<td>NoHeaders</td>
<td>207</td>
</tr>
<tr>
<td>NoJavaScripts</td>
<td>207</td>
</tr>
<tr>
<td>NoLinks</td>
<td>208</td>
</tr>
<tr>
<td>NoPackage</td>
<td>208</td>
</tr>
<tr>
<td>NoPackageFiles</td>
<td>208</td>
</tr>
<tr>
<td>NoPageLabels</td>
<td>209</td>
</tr>
<tr>
<td>NoPortfolio</td>
<td>210</td>
</tr>
<tr>
<td>NoThumbnails</td>
<td>210</td>
</tr>
<tr>
<td>NoWatermarks</td>
<td>210</td>
</tr>
<tr>
<td>OpenPassword</td>
<td>212</td>
</tr>
<tr>
<td>Outputintent</td>
<td>213</td>
</tr>
<tr>
<td>Package</td>
<td>213</td>
</tr>
</tbody>
</table>
Part III: Supporting XML Grammars Reference

20 Extended Services .......................................................................................................................... 300

21 About Language ............................................................................................................................ 305

22 Document Information Language ............................................................................................... 307

23 Bookmarks Language ................................................................................................................... 318
About the Bookmarks language ................................................................................................................................. 318
Intent of bookmarks in a PDF document .......................................................................................................................... 318
XML representation of bookmarks .................................................................................................................................. 318
Action .................................................................................................................................................................................. 322
Bookmark ........................................................................................................................................................................... 322
Bookmarks .......................................................................................................................................................................... 323
Desc ................................................................................................................................................................................... 323
Dest ..................................................................................................................................................................................... 323
File ...................................................................................................................................................................................... 323
Fit ......................................................................................................................................................................................... 324
FitB ....................................................................................................................................................................................... 324
FitBH ...................................................................................................................................................................................... 325
FitBV ...................................................................................................................................................................................... 325
FitH ....................................................................................................................................................................................... 326
FitR ....................................................................................................................................................................................... 326
FitV ....................................................................................................................................................................................... 327
GoTo ..................................................................................................................................................................................... 328
GoToE .................................................................................................................................................................................... 328
GoToR .................................................................................................................................................................................... 330
Launch .................................................................................................................................................................................. 330
Named .................................................................................................................................................................................. 330
Target .................................................................................................................................................................................. 331
Title ................................................................................................................................................................................... 332
URI ...................................................................................................................................................................................... 333
Win ...................................................................................................................................................................................... 333
XYZ .................................................................................................................................................................................... 334
Supported character encodings ........................................................................................................................................... 335

24 Document Text Language ............................................................................................................................................. 337
About the DocText XML language ....................................................................................................................................... 337
Text encoding .................................................................................................................................................................................. 338
DocText ..................................................................................................................................................................................... 339
Page ......................................................................................................................................................................................... 339
Paragraph ................................................................................................................................................................................... 339
ParagraphsPerPage ................................................................................................................................................................. 339
P1 .............................................................................................................................................................................................. 340
P2 .............................................................................................................................................................................................. 340
P3 .............................................................................................................................................................................................. 340
P4 .............................................................................................................................................................................................. 340
Quad .......................................................................................................................................................................................... 340
TextPerPage ............................................................................................................................................................................. 341
WithQuads ............................................................................................................................................................................ 341
Word ......................................................................................................................................................................................... 341

25 File Attachments Language ............................................................................................................................................ 342
About the Attachments XML language ................................................................................................................................ 342
Attachment ............................................................................................................................................................................. 342
Attachments .......................................................................................................................................................................... 343
Description ........................................................................................................................................................................... 343
File ........................................................................................................................................................................................ 344
FileName .............................................................................................................................................................................. 344
Location .................................................................................................................................................................................. 345
26 PackageFiles Language

About the PackageFiles language

PackageFiles reference

Description

DisplayOrder

FieldData

File

Folders

Folder

Package

PackageFile

PackageFiles (root element)

Schema

SortOrder

Part IV: Special Topics

27 Handling Out of Memory Problems

Operation checkpoints (DDXProcessorSetting)

About operation checkpoints

Determine a checkpoint value
The Communications - Document Manipulation APIs can assemble, disassemble, and manipulate PDF and XDP documents. For assembly, it combines multiple source documents into a resultant document. For disassembly, it breaks a source document into multiple resultant documents.

The Document Manipulation APIs can also perform these tasks:

- Insert additional content such as headers, footers, and a table of contents
- Preserve, import or export existing content such as annotations, file attachments, annotations, and bookmarks
- Encrypt and apply usage rights to documents
- Convert documents into a PDF/A compliant file for use in archiving

Each job submitted to the Document Manipulation APIs includes a Document Description XML (DDX) document and a set of source PDF and XML documents. The DDX document provides instructions on how to use the source documents to produce a set of result documents. The set of result documents usually includes one or more PDF and XDP documents, but it can also include XML.

To know the limitations of Document Manipulation APIs, see Considerations, known issues, limitations, and best practices.
Part I: DDX User Document

This section is an in-depth description of using DDX to represent specific results.
Introducing Document Description XML

Document Description XML (DDX) is a declarative markup language whose elements represent building blocks of documents. These building blocks include PDF and XDP documents, and other elements such as comments, bookmarks, and styled text.

DDX documents describe resultant documents in terms of source documents. They describe the desired characteristics of source documents as they appear in assembled resultant documents. They do not provide instructions on how to convert or assemble documents. DDX processors such as the Communications – Document Manipulation APIs determine the best way to produce the desired result.

DDX documents are templates for the documents that the Document Manipulation APIs produces. A single DDX can be used with a range of source documents.

**DDX document structure**

At the root of any DDX document is the `DDX` element. Every DDX document has the structure shown here.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <!-- Other DDX elements -->
</DDX>
```

DDX elements must be in the DDX namespace: `http://ns.adobe.com/DDX/1.0/`. The trailing slash character (`/`) is required. All elements and attributes described in this specification, unless otherwise indicated, belong to the DDX namespace.

**Note:** The `DDX` element is required in all DDX documents, but for brevity, most of the examples in this document abbreviate it or omit it entirely.

The DDX schema, `ddx.xsd`, is provided with the Document Manipulation APIs.

The Document Manipulation APIs runs a complete validation on DDX documents that you submit and reports violations that it finds.

**DDX building blocks**

The primary function of the Document Manipulation APIs is to assemble multiple PDF or XDP documents into a single document. The Document Manipulation APIs can also generate output types other than PDF. These other types represent either components of PDF documents, such as bookmarks, or information related to PDF documents.

You use DDX elements to specify everything about the document or documents you want the Document Manipulation APIs to generate. DDX elements can be grouped according to the following categories. Each
category represents a building block of a PDF or other file type. The names of DDX elements reflect their content type:

**Document assembly** elements represent PDF documents, pages from existing PDF documents (the **PDF** element), XDP documents, or fragments from XDP documents. In addition, the **TableOfContents** and **BlankPage** elements add new pages to an assembly.

**Document disassembly** elements (**PDFsFromBookmarks**) create multiple PDF documents from a single document.

**Document component** elements represent parts of PDF documents that can be imported and exported but are not pages or page content; for example, **Bookmarks** and **Links**.

**Document property** elements represent information associated with a document, such as **Metadata** and **InitialViewProfile**.

**Page content** elements specify new content that is added to pages in PDF documents, such as **PageContent**, **Header** and **Watermark** elements.

**Page property** elements specify how pages are viewed or printed, for example, the **PageSize** element.

**Page label** elements (**PageLabel**) specify page identifiers that are used for navigation in viewer applications. Page label elements can also add content to the page.

**PDF Package** elements contain metadata about file attachments used for PDF packages and portfolios. These elements also provide viewing information for the package or portfolio and schemas that define characteristics of custom metadata. PDF packages are compatible with PDF Portfolios available in Acrobat 9.

**Query** elements specify XML documents containing specific types of information about PDF documents. They include elements such as **DocumentInformation** and **DocumentText**.

These elements and their usage are described in detail throughout this document. For information, see “**DDX Concepts**” on page 137.

**DDX principles**

The DDX grammar uses principles that make it easier to understand the role of elements and how they relate to other elements. The following example describes a single PDF document called doc1.pdf that assembles the contents of two existing PDF documents, doc2.pdf and doc3.pdf.

**Example: Basic assembly**

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <PDF result="doc1.pdf">
    <PDF source="doc2.pdf"/>
    <PDF source="doc3.pdf"/>
  </PDF>
</DDX>
```

Here are some important points illustrated by this example.

First, DDX element names reflect the content that they represent:

- The **PDF result** element in the example represent PDF documents.
- The **PDF source** elements can specify PDF or other types of documents. If you provide a Microsoft Word document with a PDF source element, the Document Manipulation APIs attempts to convert it to PDF.
before assembling it in the resultant document. If that service cannot convert the document to PDF, it raises an exception.

Other DDX elements such as Bookmarks, Links, and Headers represent specific content that can be added to or extracted from a PDF document. These types of content also implicitly contained within the PDF element.

Second, a number of DDX elements can be used in different ways and are categorized depending on the context in which they are used. These elements are identified using the following terminology:

A **result** element typically has a **result** attribute and represents data being returned. Result elements have no initial content but accumulate the content of their child elements. See “Result elements” on page 16 for details.

A **source** element typically has a **source** attribute and represents initial content. See “Source elements” on page 18 for details.

A **filter** element is used much the same way as a source element, but its content comes from source elements nested within it. Filter elements contain child elements of a certain type. They do not contain a **result** or a **source** attribute. See “Filter elements” on page 19 for details.

A **PDF** element can be either a result or source element, depending on the presence of the **result** attribute or the **source** attribute. These elements are called PDF source or PDF result elements, rather than simply PDF elements.

Third, the strings appearing as values of the **result** and **source** attributes are names that identify data streams. These attributes can specify External Data URLs or names in the output and input HashMap objects.

**Note:** The source and result elements can use External Data URLs to specify documents or lists of documents. (See “Using External Data URLs for source and result values” on page 23.)

The element name determines the type of data (for example, **PDF** or **Bookmarks**). The names can have extensions for clarity (for example, **.pdf** or **.xml**), although using extensions is not required. The names do not reference files in the file system, although data is typically associated with files. The data specified by source elements typically originate from files. The data specified by result elements is typically saved to files. The exception is package files in a portfolio. With the advent of folders in portfolios, Acrobat 9 treats component files (what DDX calls package files) the same as files in a filesystem. For the best experience when viewing a portfolio in Acrobat, it is important that all component or package files be given filename extensions. (See “Input and output” on page 22.)

Fourth, **sibling elements** aggregate content of the same kind. In the example, the two **PDF** source elements combine their PDF pages to contribute content, in the order specified, to the parent element. This principle does not apply to page property and content elements. (See “Scope of elements that affect PDF or XDP properties” on page 24.)

### Result elements

Each DDX document typically contains, as children of the **DDX** element, one or more **result** elements. A result element has a **result** attribute and represents data being returned.

**Note:** The **PDFsFromBookmarks** element is an exception; it is a result element that does not have a **result** attribute. (See “Disassembling PDF Documents” on page 61.)

The example from above is repeated here. This example has a single **PDF** result element, which means that the Communications - Documentation Manipulation APIs and DDX Reference service creates a stream
named doc1.pdf. This stream contains a PDF document. The Document Manipulation APIs returns the stream to the client.

```xml
<PDF result="doc1.pdf">
  <PDF source="doc2.pdf"/>
  <PDF source="doc3.pdf"/>
</PDF>
```

In addition to the PDF element, the following element types can contain a result attribute: Bookmarks, Links, Comments, FileAttachments, XDP, PackageFiles, DocumentInformation, About, DocumentText, and Metadata. The following example has three result elements (also known as result blocks): two that return PDF and one that returns bookmarks.

**Example: Result elements**

```xml
<DDX>
  <PDF result="doc1.pdf">
    <!-- Additional elements here -->
  </PDF>
  <PDF result="doc2.pdf">
    <!-- Additional elements here -->
  </PDF>
  <Bookmarks result="doc3.xml">
    <!-- Additional elements here -->
  </Bookmarks>
</DDX>
```

Result elements must be direct children of the DDX element. They have no content initially but take their content from their children, which can include the following elements:

- Source elements specifying various types of content (see **Source elements** on page 18)
- Filter elements (see **Filter elements** on page 19)
- Other elements providing additional pages, page content, or modifications to existing content.

Result elements have a return attribute that can be either true (the default) or false. If true, the result is returned to you as a stream (see **Input and output** on page 22). You can set this attribute to false if you do not need the data returned. Use this setting when the result is used later in the DDX document as a source element for a subsequent result element.

**Note:** The FileAttachments and PackageFiles result elements do not have a return attribute. Instead, they have an extract attribute that serves a similar purpose (see **Working with File Attachments** on page 80).
Source elements

The content of result elements comes from source elements, which typically contain a `source` attribute.

**Note:** Some source elements can represent a list of streams. A list of streams is an ordered list when the order is important, such as when assembling PDF sources. To select the sources from the input map by their key names, create this list by specifying a `sourceMatch` attribute. Not all source elements support the `sourceMatch` attribute. (See “Specifying multiple input streams” on page 31.)

Source elements represent existing content, which can be one of the following:

- Content that has been provided to the Document Manipulation APIs (see “Input and output” on page 22).
- Content that is provided as an External Data URL. Such a URL resolves to a document or list of documents.
- The content of a previous result element in the same DDX document.

In the example, repeated here, doc2.pdf and doc3.pdf reference streams containing PDF content provided by the client or obtained from a previous result element.

```xml
<PDF result="doc1.pdf">
  <PDF source="doc2.pdf"/>
  <PDF source="doc3.pdf"/>
</PDF>
```

Sibling and child elements

The PDF source elements doc2.pdf and doc3.pdf are *siblings*. That is, they appear at the same level of the hierarchy and have the same parent. When sibling elements provide the same type of content, their content is aggregated to contribute content to their parent. In the example, the pages from doc2.pdf and doc3.pdf are combined to produce the result.

In some cases, the order of sibling elements is significant in determining the result. For PDF elements, as in the example above, the order is significant because a PDF document contains sequential pages. Therefore, the pages from doc3.pdf are appended to the pages from doc2.pdf to produce the result, since doc3.pdf appears after doc2.pdf. Other elements whose order is significant when they appear as siblings are Bookmarks, TableOfContents, and BlankPage.

For other elements, such as Comments, order is not significant because the comments have identifying characteristics indicating their position in the document.

The next two examples show the distinction between siblings and children.

**Example: Aggregating bookmark content using sibling elements**

```xml
<PDF result="doc1.pdf">
  <PDF source="doc2.pdf"/>
  <Bookmarks source="doc3.xml"/>
</PDF>
```

In the previous example, the PDF result element obtains its content from its child elements. The first child is a PDF source element that provides PDF content to the result. The second child is a Bookmarks element that specifies an XML stream containing bookmarks. (See “Bookmarks” on page 163.)
The source document doc2.pdf provides PDF page content but also provides other content types that are inherent to PDF. These types include bookmarks, links, comments, and file attachments. The Bookmarks source element in the example above is a sibling to the PDF source element. As a result, the bookmarks from doc3.xml are appended to any existing bookmarks in doc2.pdf. That aggregation becomes the bookmarks in the result.

In the following example, by contrast, bookmarks from one source replace bookmarks in another. (Unlike result elements, source elements can be children of other source elements.)

**Example: Using a child element to replace content in the parent**

```xml
<PDF result="doc1.pdf">
  <PDF source="doc2.pdf">
    <Bookmarks source="doc3.xml"/>
  </PDF>
</PDF>
```

In this example, as in the previous one, doc2.pdf provides the PDF page content of the resultant document. In contrast, the Bookmarks element is a child of the PDF source element rather than a sibling. The child element provides the bookmarks used in doc2.pdf. As a result, the bookmarks in doc3.xml replace preexisting bookmarks in doc2.pdf.

**Filter elements**

In the previous examples, Bookmarks elements appeared as children of PDF elements. Bookmarks are one of the content types that can be contained in PDF documents. By contrast, the following example shows a PDF source element as the child of a Bookmarks result element.

**Example: Using a bookmarks result element**

```xml
<Bookmarks result="doc1.xml">
  <PDF source="doc2.pdf"/>
</Bookmarks>
```

Because the PDF source element is the child of a Bookmarks element, only the bookmarks in doc2.pdf contribute to the result element. Unlike a PDF element, which implicitly contains other types of content, a Bookmarks element can contain only bookmarks. Therefore, doc1.xml is an XML representation of the bookmarks that came from doc2.pdf. (See “Exporting and importing bookmarks” on page 64 for more information on bookmarks.)

This example can be extended to have two PDF source elements. Since they are siblings, their content is concatenated and the bookmarks from the combined document constitute the result.

**Example: Getting bookmarks from two source documents**

```xml
<Bookmarks result="doc1.xml">
  <PDF source="doc2.pdf"/>
  <PDF source="doc3.pdf"/>
</Bookmarks>
```

**Note:** The bookmarks in doc3.pdf are updated to adjust to their new page locations. For example, if doc2.pdf has five pages, a bookmark in doc3.pdf that references page 2 is updated in the result document to reference page 7.
In the following example, the result (doc1.xml) is used as a source element to provide content to a PDF result element. The data from a result element can be specified as the input data in a source element in a subsequent result block. The order of result elements matters. That is, if the PDF result block appeared before the Bookmarks result block, an error occurs.

**Example: Using a result as a source element in a subsequent step**

```xml
<Bookmarks result="doc1.xml">
    <PDF source="doc2.pdf"/>
    <PDF source="doc3.pdf"/>
</Bookmarks>

<PDF result="doc4.pdf">
    <PDF source="doc5.pdf"/>
    <Bookmarks source="doc1.xml"/>
</PDF>
```

The following example produces an equivalent result, except that only doc4.pdf is returned and no Bookmarks result is generated in the process.

**Example: Using a bookmarks filter element**

```xml
<PDF result="doc4.pdf">
    <PDF source="doc5.pdf"/>
    <Bookmarks>
        <PDF source="doc2.pdf"/>
        <PDF source="doc3.pdf"/>
    </Bookmarks>
</PDF>
```

In this case, the Bookmarks element is called a filter element. It can function as a source or result element. The role it plays depends on its relationship to other elements. In the above example, the Bookmarks filter element contains the result of filtering the bookmarks from doc2.pdf and doc3.pdf. The same element also provides source bookmark content to doc4.pdf.

**Note:** A filter element, just like a source element of the same type, provides only content of its type. That is, no content from doc2.pdf and doc3.pdf other than bookmarks is included in the assembly.

### Profile elements

Some DDX elements are profiles that contain information used by other elements. They are children of the root DDX element. Therefore, they are never children of other elements but have a name attribute that lets other element reference them. Using profiles lets you avoid rewriting the same DDX.

**Note:** The value of the name attribute must be unique among profiles of a given type.

The DDX profile elements are discussed in detail elsewhere in this document:

- **InitialViewProfile** (See “Setting the initial view” on page 93.)
- **PasswordEncryptionProfile** (See “Specifying passwords” on page 129.)
- **PasswordAccessProfile** (See “Accessing a password-protected document” on page 131.)
- **StyleProfile** (See “Using style profiles” on page 115.)
- **PDFAProfile**
Grouping PDF sources

As described in “Scope of elements that affect PDF or XDP properties” on page 24, page content, page property, and page label elements modify pages within their scope.

The following example has three source documents. doc2 and doc3 are specified to have a page size of 8.5 x 14 inches. doc4 is specified to have a page size of 8.5 x 11 inches.

**Example: Specifying different page sizes**

```xml
<PDF result="doc1">
  <PDF source="doc2">
    <PageSize width="8.5in" height="14in"/>
  </PDF>
  <PDF source="doc3">
    <PageSize width="8.5in" height="14in"/>
  </PDF>
  <PDF source="doc4">
    <PageSize width="8.5in" height="11in"/>
  </PDF>
</PDF>
```

To avoid having to specify the same PageSize information multiple times, you can group the source elements that it applies go by using the PDFGroup element. The PDFGroup element provides its own scope that can include more than one source document.

**Example: Using the PDFGroup element**

```xml
<PDF result="doc1">
  <PDFGroup>
    <PageSize width="8.5in" height="14in"/>
    <PDF source="doc2"/>
    <PDF source="doc3"/>
  </PDFGroup>
  <PDF source="doc4">
    <PageSize width="8.5in" height="11in"/>
  </PDF>
</PDF>
```

The source documents doc2 and doc3 are in the PDFGroup element scope and therefore use all properties specified for that scope. They also inherit properties from higher scope that are not specified in the PDFGroup scope. In addition, individual source elements can have their own scope, providing properties that override the corresponding PDFGroup properties. (PDFGroup elements can also be nested.)

**Example: Using the PDFGroup element**

```xml
<PDF result="doc1">
  <PageSize width="8.5in" height="11in"/>
  <PDFGroup>
    <PageSize width="8.5in" height="14in"/>
    <PDF source="doc2"/>
    <PDF source="doc3">
      <PageSize width="11in" height="17in"/>
    </PDF>
  </PDFGroup>
  <PDF source="doc4"/>
</PDF>
```
In this example, the page size property from the PDFGroup scope (8.5 x 14 inches) applies to doc2 and doc4. doc3 overrides this property by specifying its own page size (11 x 17 inches). doc5 inherits the page size from the top-level scope (8.5 x 11 inches).

PDFGroup elements group page elements that have similar properties. A PDFGroup element does not represent a PDF document by itself. In particular, note the following points:

**Page numbering:** A PDFGroup element does not have independent page numbering (although it can specify page labels). Consider a PDFGroup element that contains `<BlankPage forceEven="true"/>`. In this case, a page is odd or even, based on its position in the resultant document. The odd or even determination is independent of the page position within the PDFGroup.

**Order of applying properties:** Placing documents in a PDFGroup element does not change the order in which DDX properties are applied. That is, DDX elements are applied to documents in a PDFGroup in the same order they would be if the DDX were constructed without PDFGroup elements. (See “Interaction of page properties and content” on page 98.)

### Grouping XDP sources and content

The XDP element lets you create an XDP document. In addition, the XDP (generic) element lets you package an XDP document in a PDF result.

### Input and output

A DDX document can specify inputs and outputs by using input and output maps or URLs. A DDX document can use a mix of these conventions.

#### Using input and output maps

When you invoke the Document Manipulation APIs using a client interface, you can provide the following information:

- A data stream containing a DDX document that specifies the output you want.
- One or more input data streams. These data streams, which represent PDF and other input types, are mapped to names that appear in source elements in the DDX document.

**Note:** A source name (for PDF, XDP, and PackageFiles source elements) can be mapped to a single data stream or to an ordered list of streams. The relationship between source names and streams is described in detail in “Specifying multiple input streams” on page 31.

After the Document Manipulation APIs processes the DDX document, it returns one or more data streams (unless it throws an exception). These data streams are mapped to the names specified by result attributes of result elements in the DDX document. If the result is specified with an External Data URL, then the resultant data streams are output directly to those locations. The resultant streams are not returned in the output map.

There is no relationship between source names and the names of files. Likewise, result names mapped to output data streams are not related to filenames. In fact, the result names may not even be valid filenames,
particularly in the case of the `PDFsFromBookmarks` element, which generates names based on bookmarks. (See “Disassembling PDF Documents” on page 61.)

Therefore, names used in DDX can be generic names. DDX support for source and result names that are independent of filenames lets you reuse the DDX with varying sets of data. You use the client interface to specify where the source data originates and where the results are stored.

**Note:** Using filename extensions such as `.pdf` for source or result attributes is not necessary but can be useful for clarity.

In particular, the original files that provided the input data streams are not modified unless you explicitly overwrite them using the client interface. In the following example, the input stream `doc1` has no relationship with the result `doc1`. Also, if a subsequent PDF source element references `doc1`, the original source for `doc1` is used.

**Example: Using the same name for source and result**

```xml
<PDF result="doc1">
  <PDF source="doc1"/>
  <PDF source="doc2"/>
</PDF>
```

**Note:** Do not modify the input data streams though the result element. Doing so can cause the Document Manipulation APIs to hang.

**Note:** Having a result and source with the same name can affect which source document is treated as the base document. (See “About base documents” on page 29.)

Input data streams are not returned to the client. If a result element contains multiple references to the same source, each instance references the original input data. A typical reason to list the same source document more than once is when specifying different page ranges for each one. (See “Page ranges” on page 30.) The following example would concatenate two copies of `doc2` and return them as `doc1`.

**Example: Specifying a source document twice**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc2"/>
</PDF>
```

**Using External Data URLs for source and result values**

The Document Manipulation APIs enables the use of URL references instead of input map keys within a DDX document. With the URL reference capability, you do not need to provide an input data stream because the Document Manipulation APIs retrieves it directly. Here are examples of files that you can reference by using External Data URLs:

- Assigned to process variables. The Process URL variant is not available through the client interface. It is available only when the Document Manipulation APIs is invoked as part of process created in Workbench.
- Located in the repository.
- Accessible from the Internet or intranet.
- Accessible on the server’s locally accessible file system.
Using External Data URLs for string values

The Document Manipulation APIs enables the use of URL references instead of hardwired strings within a DDX document. You can use URLs to add dynamically generated text to a document. For example, if a URL references a process variable containing a string, that string can be used in a page header in the resulting PDF document. (See “External Data URL” on page 150.)

Scope of elements that affect PDF or XDP properties

Some DDX elements affect PDF page properties or XDPContent properties. The scope of these elements applies to the parent element and to any of the parent’s children.

DDX provides flexibility in applying different page content and page properties to different pages in the result document. The following examples use the PageSize element to illustrate the concept of scope in DDX. The same principles apply to other page property and page content elements.

When you use DDX to specify XDPContent, the properties apply to the descendents.

Scope of PDF page properties

In the first example, the PageSize element is a child of the PDF result element. The scope of the PageSize element is its parent element. Therefore, it applies to the entire result document, and it affects all source documents that make up the result.

**Example: Specifying a page property for the entire result document**

```xml
<PDF result="doc1">
  <PageSize width="8.5in" height="11in"/>
  <PDF source="doc2"/>
  <PDF source="doc3"/>
  <PDF source="doc4"/>
</PDF>
```

The next example is equivalent to the previous one, even though the PageSize element appears after the PDF source elements. There can be only one PageSize element for any scope, and the order of this element relative to its siblings does not matter. (The order of the PDF source elements does matter because they specify the order of pages in the result document.)

**Example: Specifying a page property for the entire result document**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc3"/>
  <PDF source="doc4"/>
  <PageSize width="8.5in" height="11in"/>
</PDF>
```
In the next example, an additional `<PageSize>` element appears as a child of the first `<PDF>` source element. This addition creates a new scope. The properties specified by this `<PageSize>` element apply only to the pages in `doc2`, which have a page size of 8.5 x 14 inches. `doc3` and `doc4` retain the page size specified by the `<PageSize>` element at the top level (8.5 x 11 inches).

**Example: Overriding a page property at a lower scope**

```xml
<PDF result="doc1">
  <PageSize width="8.5in" height="11in"/>
  <PDF source="doc2">
    <PageSize width="8.5in" height="14in"/>
  </PDF>
  <PDF source="doc3"/>
  <PDF source="doc4"/>
</PDF>
```

**Note:** The top-level scope is considered to be that of result elements that are direct children of the `<DDX>` element. For example, a `<PageSize>` element cannot be a child of the `<DDX>` element with the intention of having it apply to all result documents.

The `<PDFGroup>` element can be used to create another level of scoping. (See “PDFGroup” on page 245.)

For more information on how `<DDX>` elements affect the contents or properties of PDF pages, see “Setting Page Properties” on page 95, “Adding and Manipulating Page Content” on page 102, and “Specifying Page Labels” on page 121.

Odd and even pages

Some page property and page content elements have an **alternation** attribute that lets you specify different values for even and odd pages within a given scope. You use this feature by specifying the element twice and setting the `alternation` attribute to `OddPages` for one and `EvenPages` for the other.

The default value for `alternation` is `None`. This value means that there is no difference between odd and even pages and the element can appear only once in a given scope.

The `alternation` attribute applies only to the following elements:

- **Page properties:** `PageMargins`, `ArtBox`, `TrimBox`, and `BleedBox`
- **Page content:** `Footer`, `NoFooters`, `Header`, `NoHeaders`, `Watermark`, `NoWatermarks`, `Background`, `NoBackgrounds`, `PageContent`

**Scope of XDPContent**

Similar to `<PDF>` source element, the `<XDPContent>` element can contain other `<XDPContent>` elements. The properties specified by an `<XDPContent>` element apply to the element and to the children of the element. For example, the nested `<XDPContent>` element inherits the value of its parent’s `includeSubFolders` attribute.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <XDP result="myFormResult">
    <XDP source="myFormSource">
      <XDPContent fragment="myFragment" includeSubFolders="false" insertionPoint="myInsertionPoint" source="myFragmentSource">
        <XDPContent fragment = "myOtherFragment"/>
      </XDPContent>
    </XDP>
  </XDP>
</DDX>
```
Specifying length

Attributes of some DDX elements specify information about distance or length. These values are expressed as a positive or negative value, along with a unit of measurement without any spaces. These are examples of the allowable units:

- "1.0in": inches
- "25.4mm": millimeters
- "2.54cm": centimeters
- "72.0pt": points (1/72 inch)

Dynamic document assembly

DDX provide several mechanisms that let you dynamically create documents.

Optional source documents

All DDX source elements are required by default, meaning that an error is thrown if any source content is missing. However, PDF and XDP source data can be specified as optional by specifying the `required` attribute with a value of `false`. This setting instructs the Document Manipulation APIs to skip over missing source data without reporting errors. Use the `required` attribute for sources with these conditions:

- DDX specifies the source
- At execution time, some of the sources are missing

For example, a stock brokerage house sends statements to its customers every month describing their total portfolio. Some customers own stocks or bonds, while others do not. In the latter case, source documents describing the stocks or bonds are absent. Consider the following example.

```xml
<PDF result="CompletePortfolio">
  <PDF source="Stocks" required="false"/>
  <PDF source="Bonds" required="false"/>
  <PDF source="Options" required="false"/>
  <PDF source="TotalFunds"/>
</PDF>
```

Lists of documents

The PDF, XDP, and PackageFiles source elements can have a single data stream or an ordered list of data streams. The ordered list can be supplied via an entry in the input map or an External Data URL.
Lists obtained from input maps or External Data URLs

Lists are often used when the Document Manipulation APIs is being driven from a watched folder. They are also useful when you use a single DDX document to assemble an indeterminate number of documents. Because all of the files in the list are included in a single PDF, XDP, or PackageFiles element, they are all treated identically. That is, they simply occur in the resulting file in the order they occur in the list.

Lists constructed with the sourceMatch attribute

Lists can be constructed by the DDX document at run-time by using the sourceMatch attribute in the PDF, XDP, or PackageFiles elements. The value of sourceMatch is a regular expression pattern (in Java™ syntax) that matches keys in the input map. The result of apply the sourceMatch attribute is a list of data streams with names that the match the regular expression. If the value of a matching key is itself a list, then the contents of that list are included in the constructed list. Typically, however, sourceMatch is used to build a list from files specified individually in the input map or URL.

The constructed list can be modified by using the optional sortOrder, sortLocale, and matchMode attributes. By default, the constructed list is sorted in alphanumeric ascending order. You can change this behavior by setting the sortOrder attribute value to Descending. By default, the locale used for sorting is the default locale. The TargetLocale element specifies the default locale. The locale used for sorting can be changed with the sortLocale attribute. Finally, a list can be built from keys not matched by specifying a matchMode value of Exclude. Setting matchMode to Exclude is useful for including documents that omitted by a previous sourceMatch list construction.

Lists obtained by apply the sourceMatch attribute to from input maps or External Data URLs

To achieve even more flexibility, both source and sourceMatch can be specified within a single PDF, XDP, or PackageFiles element. Using both attributes enables the input map to have different configurations with either the list being built before the Document Manipulation APIs is invoked or using the Document Manipulation APIs to build the list at run-time capability.

Automatic Conversion of source documents to PDF documents

If the type of a source file for a <PDF> element is not PDF, the Document Manipulation APIs invokes another AEM Forms service to convert the file to PDF. In particular, the Document Manipulation APIs invokes the following services depending on the type of file and other DDX elements:

- Generate PDF service. Converts native file formats to PDF. Examples of native file formats are Microsoft Word, Excel, and HTML. The supported native file formats depends on the configuration of the Generate PDF service.

- Generate PDF service. Converts into PDF any XDP element that appears in a PDF result element. The result is a PDF document that can be assembled with sibling elements in the parent PDF result element.

- DocConverter service. Converts a PDF document to a PDF/A document.

- Generate 3D PDF service. Converts CAD file formats to PDF (Windows only).
Distiller service. Converts an Adobe PostScript® file to PDF.

Forms service. Merges XML data into an XML form (XFA template). The XML data is specified with the XMLData element. The result is a PDF file containing the filled form.

Output service. Flattens a dynamic XML form template (XFA template) to remove its interactive qualities. (See “Flattening forms” on page 36.)

Reader Extensions service. Enables Adobe Reader® users to digitally sign the resulting PDF document.

The PDFGenerationSettings and XFAConversionSettings elements specify conversion settings for PDF and XFA conversion, respectively. (See “Extended Services” on page 300.)
Assembling PDF Documents

The Communications - Document Manipulation APIs can assemble PDF and XDP documents from other documents.

**Note:** If a PDF file contains invalid XMP metadata, the Document Manipulation APIs throws an exception when processing that file. The error message states that a "SyncMetadataTask task failed for document."

Such errors can occur with documents produced by outdated software, such as old versions of Adobe Distiller® or Adobe FrameMaker®. You can avoid such problems by regenerating the input PDF document using updated software.

Specifying source documents

The Document Manipulation APIs can assemble documents from one or more source documents. It also supports several options for specifying source documents.

**Note:** The non-PDF documents can be used as PDF source documents. Document Manipulation APIs automatically converts such documents to PDF documents. The Document Manipulation APIs performs the conversion by using the Generate PDF service. The PDFGenerationSettings element specifies conversion parameters. (See "PDFGenerationSettings" on page 300.)

About base documents

When assembling multiple PDF source documents, one of them is considered to be the base document. The base document is used as the source of certain properties of the result document, including document properties, form data, document-level JavaScript code, and viewer preferences. Other source documents do not contribute these properties to the result. However, they do contribute the other PDF building blocks, such as page content and properties (see "DDX building blocks" on page 14) to the result document.

For document-level components, like file attachments, they are only included from a document once, even if the document is specified multiple times.

**Note:** Document-level file attachments are assembled from a non-base document when the entire PDF document is part of the assembly. If only some pages from a non-base document are assembled, then the document-level file attachments for that PDF are not included.

The Document Manipulation APIs determines the base document by applying these considerations (in order):

1. The PDF source element that contains a baseDocument attribute whose value is true is the base document. The default value of baseDocument is false. It is an error for more than one source element in a PDF resultant block to have baseDocument set to true.

2. Otherwise, the first PDF source element, if any, whose source attribute has the same value as the PDF resultant element’s result attribute is the base document.

3. Otherwise, the first PDF source element in the PDF result block is the base document.
The following example illustrates these cases.

**Example: Setting the base document**

```xml
<PDF result="doc3">
  <PDF source="doc1"/>
  <PDF source="doc2" baseDocument="true"/> <!--base document (explicit)-->
</PDF>

<PDF result="doc2">
  <PDF source="doc1"/>
  <PDF source="doc2"/> <!--base document (source matches result)-->
</PDF>

<PDF result="doc3">
  <PDF source="doc1"/> <!--base document (first source document)-->
  <PDF source="doc2"/>
</PDF>
```

**Page ranges**

Specify the range of pages to include for each source document by setting the `pages` attribute of each `PDF` source element. If this attribute is not set, all pages are included by default. Page ranges can include one or more of the following items:

- A positive integer that specifies a single page number in the source (where the pages are numbered starting with 1)
- The keyword `last` means the last page number
- The keyword `penultimate` means the next-to-last page number
- A continuous, increasing range of pages separated by a hyphen (for example, `1-last`)
- A discontinuous range of pages separated by commas (for example, `1,3,5,10`)

**Note:** Spaces are allowed in page ranges.

**Example: Specifying page ranges**

```xml
<PDF source="doc1" pages="5"/> <!--page 5 only-->
<PDF source="doc1" pages="2-4"/> <!--pages 2 through 4-->
<PDF source="doc1" pages="1-3,6-9,20-21"/> <!--pages 1,2,3,6,7,8,9,20,21-->
<PDF source="doc1" pages="2-penultimate"/> <!--all but first & last pages-->
<PDF source="doc1"/> <!--all pages by default; equivalent to "1-last" -->
```

The following example assembles three PDF documents, specifying page ranges for the first two pages. It is equivalent to deleting pages 4-4 from doc1.pdf, deleting page 7 from doc2.pdf, and then concatenating the documents.

**Example: Assembling pages from documents**

```xml
<PDF result="doc4.pdf">
  <PDF source="doc1.pdf" pages="1-2,5-last"/>
  <PDF source="doc2.pdf" pages="1-6,8-last"/>
  <PDF source="doc3.pdf"/>
</PDF>
```
**Note:** For PDF source's that specify multiple input documents, use the `select` attribute to specify document ranges. (See “Specifying multiple input streams” on page 31.)

### Other source attributes

In addition to the `source` and `pages` attributes, PDF source elements can have these attributes, which are discussed further in other sections:

- **bookmarkTitle** specifies the title of a bookmark that is added to the resultant document (see “Creating bookmarks from source documents” on page 65).
- **includeInTOC** controls whether the source document appears in the table of contents, if present (see “Adding a table of contents” on page 85).
- **access** specifies a password for opening the source document if it is encrypted (see “Accessing a password-protected document” on page 131).

A number of attributes (`matchMode`, `required`, `select`, `sortLocale`, `sortOrder`) are used to specify documents from a selection of multiple source documents. (See “Specifying multiple input streams” on page 31). The `sortLocale`, `sortOrder`, and `matchMode` attributes apply only when the `sourceMatch` attribute is used to create an ordered list.

**Note:** Data specified by a PDF source element can be in XML Data Package (XDP) format or PDF. XDP is an XML format that can contain PDF data and form data.

### Specifying multiple input streams

In most of the examples in this document, a PDF source element represents a single PDF document. You can also map multiple documents to a single PDF source element, XDP source element, XDPContents, or PackageFiles source element.

Names are mapped to streams with an input map. An input map contains entries that specify names and corresponding values. The values can represent either a single input stream or an ordered list of input streams.

A source element can have a `sourceMatch` attribute instead of a `source` attribute. (If neither is present, the DDX is invalid.) The Document Manipulation APIs uses these attributes and other source element attributes to generate an ordered input list of one or more streams. The ordered input list is used to generate content for the parent result element.

The Document Manipulation APIs uses the following process to create the input list for a PDF, XDP, XDPContent, or PackageFiles element:

If the `source` attribute resolves to a document or list of document, the input is initially set to the stream or ordered list of streams mapped to the name. If the source attribute is not a URL, the Document Manipulation APIs looks for the named document or document list in the input map or among the previous result elements.

If `source` is present and the name is not found in the input map or from a previous result, `sourceMatch` is used, if present. If `sourceMatch` is not present, an error occurs if the `required` attribute is set to `true`. Otherwise, the source element is ignored and contributes no content to the resultant document.

`sourceMatch` specifies a regular expression whose syntax is implemented in the `java.util.regex` package for Java. If `source` is not present, `sourceMatch` must be present. The `sourceMatch` regular expression is used to select one or more source names from the input map. If
source is present and contains an URL, then the sourceMatch regular expression is used to select one or more source names from the URL. The URL can resolve to a document or a list of documents. If source is present and its value is a key in the input map, then sourceMatch is ignored.

When the names are matched, matchMode indicates whether to include or exclude the matched names. This list of names is then sorted according to sortOrder to result in an ordered list of names. The matching names are sorted based on the value of the sortOrder attribute, either Ascending (the default) or Descending. The Document Manipulation APIs creates an ordered list of streams by taking each name in the sorted list and adding each of its streams, in order, to the input list. (The sort order can be refined for different languages by using the sortLocale attribute.)

The final input list is created by applying the select attribute, which specifies a range of documents in the list. Its syntax is the same as the syntax used for page ranges (see “Page ranges” on page 30). For example, "2-last" selects the second-through-last document in the list. If the range results in no documents being selected, the input list is empty.

If the value of matchMode is Exclude, the select attribute value is inverted. That is, the input list includes all streams other than those streams specified by select.

If the final input list is empty (consists of no streams), the source element does not add any content to the assembly. For example, if the select attribute is set to "3" and only two source documents are in the input map, then the final input list is empty. You can specify that an empty list is acceptable by setting the value of the required attribute to false. If the value of required is true (the default), an error occurs.

List defined by a source that specifies a name in the input map

In the following example, there are two keys in the input map, Cover and Files. Assume that Cover is supplied as a single document and Files is supplied as a list of documents. The first file in the assembly, Cover, does not have a Header applied. Each file in the Files list has a Header containing the value of the Title keyword in metadata of the first file in the list using the built-in key SourceTitle. The same DDX can be used when Files is a single file or a list of files. Consider the following example.

Example: Input map that specifies multiple input streams

```
<PDF result="AnAssembly">
  <PDF source="Cover"/>
  <PDF source="Files">
    <Header><Center>
      <StyledText><p>_SourceTitle</p></StyledText>
    </Center></Header>
  </PDF>
</PDF>
```

Note: When Files in the above example refers to a list of documents, the metadata properties of the first file in the list are used for built-in keywords like SourceTitle.

List defined by a source that specifies URL

You can use External Data URLs as the value of source attributes. The URL can resolve to a single document or to a folder. If the URL resolves to a folder, the Document Manipulation APIs (by default) imports the contents of the folder.

```
<PDF result="AnAssemblyAlso">
  <PDF source="File:///c:/myDirectory/myFile" select="1" /></PDF>
  <PDF source="File:///c:/myDirectory/myFolder" select="2-last">
```

List defined by the matchSource and select attributes acting on source

The following example shows how the `select` and `sourceMatch` attributes act on the `source` attribute. It shows how to use regular expressions to select documents from a list of documents. The value "chap\[\d\]+" is a regular expression that uses the following conventions:

- `[ ]` encloses the characters to match
- `\d` specifies any digit
- `+` matches one or more of the preceding search criterion

The regular expression translates into this English expression: Find files that have names that include the string `chap` followed by one or more digits.

**Example: sourceMatch selects files from a source that is an External Data URL**

```xml
<PDF result="mybook" save="Full"/>
<PDF source="intro"/>
<PDF source="Chapters" baseDocument="true"
sourceMatch="chap\[\d\]+" select="1"/>
<PDF source="Chapters"
sourceMatch="chap\[\d\]+" select="2-last"/>
<PDF source="Appendices"
sourceMatch="appendix\[\d\]+"/>
<PDF source="index"/>
</PDF>
```

The following two examples are of input maps that can be used with this DDX document. The first one maps each logical name to a single data stream as follows:

<table>
<thead>
<tr>
<th>Logical name</th>
<th>Data stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro</td>
<td>stream_1</td>
</tr>
<tr>
<td>chap1</td>
<td>stream_2</td>
</tr>
<tr>
<td>chap2</td>
<td>stream_3</td>
</tr>
<tr>
<td>chap3</td>
<td>stream_4</td>
</tr>
<tr>
<td>chap4</td>
<td>stream_5</td>
</tr>
<tr>
<td>appendix1</td>
<td>stream_6</td>
</tr>
<tr>
<td>appendix2</td>
<td>stream_7</td>
</tr>
<tr>
<td>index</td>
<td>stream_8</td>
</tr>
</tbody>
</table>

In the example, the `source` attributes of the first and last `PDF` source element (intro and index) match names in the input map. Therefore, those streams are used in the resultant document.
The source attributes of the other PDF source elements (Chapters and Appendices) do not match any of the names in the input map. Therefore, the sourceMatch attribute is checked. The regular expression "chap\[d]+" matches chap1, chap2, chap3, and chap4 in the input map. The select attribute "1" selects the first of those streams (chap1) and "2-last" selects the rest of them. (If only one name matched "chap[d]+", then "2-last" would generate an error unless the value of required was false.)

Similarly, the sourceMatch expression "appendix\[d]+" matches appendix1 and appendix2.

In the following example, the input map maps each logical name to an ordered list of data streams.

<table>
<thead>
<tr>
<th>Logical name</th>
<th>Data streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro</td>
<td>stream_1</td>
</tr>
<tr>
<td>Chapters</td>
<td>stream_2, stream_3, stream_4, stream_5</td>
</tr>
<tr>
<td>Appendices</td>
<td>stream_6, stream_7</td>
</tr>
<tr>
<td>index</td>
<td>stream_8</td>
</tr>
</tbody>
</table>

In this case, all of the names match source attributes in the DDX. In the case of Chapters and Appendices, multiple streams are used for the source elements.

PDF, XDP, or PackageFiles source elements that represent multiple streams are equivalent to multiple versions of those elements representing individual streams. The other attributes in the “sourceMatch selects files from a source that is an External Data URL” on page 33 example have these meanings:

- **baseDocument**: If true, the first document in the ordered list is marked as the base document.
- **bookmarkTitle**: If present, the value is a built-in key (see “Built-in keys” on page 114); otherwise, the bookmarks are identical for all selected input documents. For example, bookmarkTitle="_SourceTitle" uses the metadata title of each PDF document as the bookmark title.
- **includeInTOC**: If true, it applies for all selected input documents.
- **pages**: If set, the page range is applied separately to each selected input document.
- **access**: If set, the same password specified by PasswordAccessProfile is used to open each selected input document.

### Saving PDF documents

The Document Manipulation APIs provides several options for saving PDF documents.

**Note**: The term save in this discussion does not mean actually saving a file to disk. It means that the result stream is structured in such a way that it can be saved as a PDF file.

The PDF result element’s save attribute specifies the save options:

- **Incremental**: Performs an incremental save. This means that changes to the document are placed at the end of the file, and the bytes corresponding to the original file are unchanged. Use this option to
maximize the speed of the save operation and to preserve certifying signatures (certification) and reader enabled rights.

If unspecified, the default is to save incrementally.

Incremental saves are relative to the base document (see “About base documents” on page 29). That is, the bytes in the result stream begin with the original bytes of the base document, followed by updates. As a result, an incrementally saved document is always larger than the original, even if pages were removed. The removed pages are retained in the PDF document.

**Full**: Performs a full save, which means that the file is restructured so that duplicate and obsolete information is removed. This results in a smaller file size than when doing an incremental save. Use this option to minimize the size of the resultant PDF file.

**Note**: Typically, an incremental save takes less time to perform than a full save. However, when deleting pages from a document (by specifying page ranges), there is not a significant performance difference in the current version of the Document Manipulation APIs.

Because the bytes corresponding to the original base document are not preserved, all digital signatures in the document are invalidated. Such invalidated signatures include signatures for certification and rights enabled for Adobe Reader®.

**FastWebView**: Restructures the file so that it is optimized for viewing on the web. This optimization (called linearization in the *PDF Reference* guide) allows individual pages to be loaded quickly, without the need to read the entire file. Use this option to minimize the size of the resultant PDF file; FastWebView results in the smallest file sizes.

This type of optimization is considered a full save because incremental updates are incorporated into the file when it is restructured. As with Full, all digital signatures in the document, including those applying to certification and rights enabled for Adobe Reader, are invalidated.

When the value of `save` is unspecified, the Document Manipulation APIs follows this behavior:

If the PDF version is 1.4 or later, an incremental save is performed.

If the PDF version is less than 1.4, a full save is performed.
PDF documents can have interactive form fields that are used to collect data from a user. These form fields include elements such as text boxes, radio buttons, and lists. Collectively, the form fields are called the document’s form. PDF documents can have two types of forms:

- Traditional Acrobat forms.
- Forms based on the XML Forms Architecture (XFA) that are created in Designer ES4. XFA forms have two varieties:
  - Static XFA forms have a fixed layout of the form fields.
  - Dynamic XFA forms can have a variable arrangement of form fields and a variable number of pages depending on the data that is provided.

There are restrictions on the operations the Communications - Document Manipulation APIs can perform depending on the presence of forms. Documents containing Acrobat forms have the fewest restrictions, and dynamic XFA forms have the most restrictions. See “Restrictions on documents containing forms” on page 37 for details.

To avoid these restrictions, you can flatten form fields in your source documents, as described in the next section.

### Flattening forms

Flattening means that form fields retain their graphical appearance but are no longer interactive. For example, a check box still appears as a rectangle, but the user cannot click it to indicate a selection. In addition, scripts associated with the fields no longer function.

To flatten all form fields for Acrobat forms or XFA-based forms, use the NoForms element as a child of a PDF or PDFGroup element. The flattening is performed for the specified scope. Flattening is useful, for example, if you want to assemble documents that contain XFA forms. If a non-base document contains XFA-based forms, an error is thrown when that document is aggregated with other documents. To avoid such an error, use the NoForms element to flatten all forms in the document. Alternatively, use the NoXFA element to flatten all XFA-based forms in the document. The properties in the XFAConversionSettings element specify how XFA-based forms are flattened. (See “Assemble XFA-based forms with other documents” on page 43 and “XFAConversionSettings” on page 302.)

**Note:** If the PDF document is a dynamic XML form template (XFA form) and the flatten attribute is true, the Document Manipulation APIs uses the Output service to flatten the form. If that service is unavailable, an exception is thrown.

The Document Manipulation APIs itself flattens static XML forms and Acrobat forms.

In this example, any form fields in doc3.pdf are flattened. If doc2.pdf contains an Acrobat or XFA form, it is retained.

**Example: Flattening forms**

```xml
<PDF result="doc1.pdf">
  <PDF source="doc2.pdf"/>
</PDF>
```
To flatten only XFA forms, you use the NoXFA element as a child of a PDF or PDFGroup element. If the documents in the scope of the NoXFA element do not have XFA forms, they are not modified.

The following example flattens XFA form fields in doc2.pdf and assembles doc2.pdf with doc1.pdf. The resultant document is returned through the doc3.pdf output map.

**Example: Flattening XFA forms**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf">
    <NoXFA/>
  </PDF>
</PDF>
```

When the base document is an XFA form, you have the option of saving the result document in XDP format instead of PDF. XDP is an XML format that can contain PDF data as well as form data. To save a document as XDP, you set the format attribute to XDP as in this example.

**Example: Saving an XFA form as XDP**

```xml
<PDF result="doc3.pdf" format="XDP">
  <PDF source="doc1.pdf"/> <!--Base document; must be XFA form-->
  <PDF source="doc2.pdf"/>
</PDF>
```

**Note:** Flattening form fields can have side effects. Consider a document that contains JavaScript code that runs when the document is opened. If the JavaScript references a specific form field and forms have been flattened, an error occurs.

**See also**

*Create a Flat PDF*

*“Creating and Modifying Acrobat and XML (XFA) Forms” on page 39*

**Restrictions on documents containing forms**

There are restrictions that apply to documents that contain forms. These restrictions do apply when the forms are flattened by using the NoXFA or NoForms elements.

**Acrobat forms**

When you assemble multiple documents that contain only Acrobat forms, the Document Manipulation APIs retains all form fields by default. In general, there are no restrictions on the operations that the Document Manipulation APIs can perform on documents containing Acrobat forms. However, if a document contains signature fields, certain operations invalidate the signatures. (See *“Digital signatures” on page 131*.)
XFA-based forms

You can assemble a PDF document containing a static XFA-based form with other documents if these conditions are satisfied:

- PDF document containing the form is the base document
- Form is static XFA. A dynamic XFA document cannot be assembled with any other documents.
- None of the other PDF documents can contain XFA forms. (They can contain Acrobat forms.)

If any of these conditions are violated, an exception is thrown.

You can assemble multiple XFA-based forms into a single form, export the result as a PDF document, and then assemble that PDF document with other PDF documents. (See “Package an XDP document as PDF” on page 42.)

In addition to assembly restrictions, static XFA documents also have these restrictions:

- You can disassemble a static XFA document, but the result documents are not XFA documents.
- Using the PageOverlay or PageUnderlay elements throws an exception.
- Attempting to change the page size or rotation of a document or to transform the page contents throws an exception.

For unflattened dynamic XFA documents, no operations are permitted that modify the document, add content, change page properties, or the initial view. The following operations are permitted with dynamic XFA documents:

- Query operations
- Export operations (for example, exporting bookmarks, links, and annotations).
- Specifying encryption
- Removal of certification or usage rights
- Saving as XDP
- Disassembly (but the result documents are not XFA documents)
Creating and Modifying Acrobat and XML (XFA) Forms

You can use the Communications Document Manipulation APIs to dynamically assemble multiple XDP documents into a single XDP document or into a PDF document. For source XDP files that include insertion points, you can specify the fragments or entire forms to insert at the insertion point.

Here are some of the ways you can assemble XDP documents:

- “Assemble a simple XDP document” on page 39
- “Dynamically insert forms or form fragments into an XFA form” on page 40
- “Package an XDP document as PDF” on page 42
- “Package a PDF document as XDP” on page 45

See also

Guidelines for Dynamically Assembling Customized Forms and Documents

Assemble a simple XDP document

The following illustration shows three source XDP documents being assembled into a single resultant XDP document. The resultant XDP document contains the three source XDP documents and the data for the base XDP document. The resultant document obtains basic attributes from the base document. The base document is the XDP source that includes a `baseDocument="true"` attribute or the first source XDP document.

Here is a DDX expression that produces the result illustrated above.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <XDP result="MyXDPResult"/>
  <XDP source="sourceXDP1"/>
  <XDP source="sourceXDP2"/>
</DDX>
```
<XDP source="sourceXDP3"/>
</XDP>
</DDX>

The first XDP source element is the base document, provided it does not contain a fragment. The configuration and data in the XDP result is obtained from the base document.

Dynamically insert forms or form fragments into an XFA form

The Document Manipulation APIs lets you create an XFA form by inserting forms or form fragments into another XFA form.

Support for dynamic insertion of form fragments supports single-source control. You maintain a single source of commonly used components. For example, you can create a fragment for your company banner. If the banner changes, you only have to modify the fragment. The other forms that include the fragment are unchanged.

Form designers use Designer 11 to create form fragments. These fragments are uniquely named subforms within an XFA form. The form designers also use Designer 11 to create XFA forms that have uniquely named insertion points. You (the programmer) write DDX expressions that specify how fragments are inserted into the XFA form.

You can control the fragments or forms that are inserted into a form by the removal or retention of insertion points. For example, if an insertion is removed after a fragment is inserted, then subsequent fragments are not inserted into that same insertion point.

The following illustration shows two XML forms (XFA templates). The form on the left contains an insertion point named myInsertionPoint. The form on the right contains a fragment named myFragment.

When the Document Manipulation APIs interprets the following DDX expression, it creates an XML form that contains another XML form. The myFragment subform from the myFragmentSource document is inserted at the myInsertionPoint in the myFormSource document.

Example: Dynamic assembly of form fragments

<DDX xmlns="http://ns.adobe.com/DDX/1.0/"></DDX>
XDP documents can contain images referenced either through absolute or relative references. Communications - Document Manipulation APIs, by default, retains the references to the images in the resultant XDP document.

You can specify how the Communications - Document Manipulation APIs handles the images references in the source XDP documents, when assembling the source documents. References in the source documents can be absolute or relative. You can choose to have all the images embedded in the resultant so that it contains no relative or absolute references. You define this by setting the value of the resolveAssets tag, which can take any of the following options:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Does not resolve any references. All references are retained.</td>
</tr>
<tr>
<td>all</td>
<td>Embeds all referenced images in the source XDP documents.</td>
</tr>
<tr>
<td>relative</td>
<td>Embeds all the images referenced through relative references in the source XDP document.</td>
</tr>
<tr>
<td>absolute</td>
<td>Embeds all the images referenced through absolute references in the source XDP document.</td>
</tr>
</tbody>
</table>

You can specify the value of the resolveAssets attribute either in the XDP source tag or in the parent XDP result tag. If the attribute is specified to the XDP result tag, it will be inherited by all the XDP source elements which are children of XDP result. However, explicitly specifying the attribute for a source element overrides the setting of the result element for that source document alone.

**Example: Resolve all source links in an XDP document**

To convert all references in the source XDP documents, specify the resolveAssets tag for the resultant document to *all*, as in the example below:

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     result="result.xdp" resolveAssets="all">

    <XDP source="input1.xdp" />
    <XDP source="input2.xdp" />
    <XDP source="input3.xdp" />

</XDP>
</DDX>
```

You can also specify the attribute for all the source XDP documents independently to get the same result.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     result="result.xdp">

    <XDP source="input1.xdp" resolveAssets="all" />
    <XDP source="input2.xdp" resolveAssets="all" />

</DDX>
```
Example: Resolve selected source links in an XDP document

You can selectively specify the source references that you want to resolve by specifying the resolveAssets attribute for them. The attributes for individual source documents override the resultant XDP document’s setting. In this example, the fragments included are also resolved.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <XDP result="result.xdp" >
    <XDP source="input1.xdp" resolveAssets="all">
      <XDPContent source="fragment.xdp" insertionPoint="MyInsertionPoint" fragment="myFragment"/>
    </XDP>
    <XDP source="input2.xdp" />
  </XDP>
</DDX>
```

Example: Selectively resolve absolute or relative references

You can selectively resolve absolute or relative references in all or some of the source documents, as shown in the example below:

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <XDP result="result.xdp" resolveAssets="relative">
    <XDP source="input1.xdp" />
    <XDP source="input2.xdp" />
  </XDP>
</DDX>
```

Package an XDP document as PDF

You can use the Document Manipulation APIs to package an XDP document as a PDF document. The XDP assembly must be contained within an XDP element that omits a source or result attribute. Such an element is called an XDP (generic) element. The XDP (generic) element provides a PDF representation for inclusion in the PDF assembly. However, the PDF with XDP cannot be successfully assembled unless the XDP is flattened with NoForms or NoXFA.

In this example, the XDP source files are combined into a single XDP stream before being assembled into the PDF result. The data from the base document is retained in the result. The data from other XDP files is lost.

Example: Assembling multiple XFA-based forms into a single form

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <PDF result="interactive_form.pdf" encryption="passEncProfile1">
    <XDP >
      <XDP source="sourceXDP3"/>
      <XDP source="sourceXDP4"/>
    </XDP>
  </PDF>
</DDX>
```
Within the XDP (generic) element, you can use the XFAData element to populate the XFA form data fields. Data supplied in the base document is replaced with data supplied in the XFAData element.

**Example: Assembling multiple XFA-based forms with external data**

```xml
<PDF result="final.pdf">
  <XDP>
    <XDP source="doc1.xdp"/>
    <XDP source="doc2.xdp"/>
    <XDPContent insertionPoint="Disclaimer"
      source="disclaimer.xdp" fragment="US"/>
    <XFAData source="data.xml"/>
  </XDP>
</PDF>
```

**PDF documents from single XFA-based forms**

When the PDF result contains only a single XDP (generic) element (XFA-based form), the XML form in the resultant PDF retains its fillable characteristics. That is, the XFA-based form is not flattened.

**Example: Assembling a PDF document that contains an interactive XFA-based form (case 1)**

```xml
<PDF result="result.pdf">
  <XDP>
    <!-- These XFA-based forms are assembled into a single form.-->
    <XDP source="doc1.xdp"/>
    <XDP source="doc2.xdp"/>
  </XDP>
</PDF>
```

**Example: Assembling a PDF document that contains an interactive XFA-based form (case 2)**

```xml
<PDF result="result.pdf">
  <PDF source="doc1.xdp"/>
</PDF>
```

**Assemble XFA-based forms with other documents**

You can assemble XFA-based forms with other documents. In some cases, the forms must be flattened.

**Form is flattened:** If the PDF result element contains an XDP (generic) element and other source documents, then flatten the XFA-based forms before assembly.

**Form remains interactive:** If the PDF result element’s base document is a PDF source element assembled from XFA-based forms, then the resultant document can be interactive. The PDF result can also include other source documents that do not contain XFA-based forms.

**Flatten assembly of multiple XFA-based forms**

The Document Manipulation APIs cannot assemble an XDP (generic) result with other documents. If the following conditions occur, then Document Manipulation APIs throws an error:

- PDF result element contains multiple document sources.
- One or more of those sources are XDP (generic) elements
Before assembling an XFA-based form with another document, use the NoForms or NoXFA elements to flatten the form. The Document Manipulation APIs uses the Output service to flatten dynamic XFA forms. The Document Manipulation APIs flattens static XFA-based forms and Acrobat forms by itself. (See “Assemble XFA-based forms with other documents” on page 43.)

Example: **Flatten the XFA-based forms before assembling with PDF sources**

```
<PDF result="result.pdf">
  <XDP>
    <XDP source="Summary.xdp"/>
  </XDP>
  <PDF source="doc1.pdf"/>
  <NoForms/>
</PDF>
```

Example: **Error when assembling a PDF containing an XFA-based form with other PDF documents**

```
<PDF result="result.pdf">
  <XDP>
    <XDP source="doc1.xdp"/>
  </XDP>
  <PDF source="doc2.pdf"/>
</PDF>
```

Example: **Error when assembling a PDF containing an XFA-based form with same**

```
<PDF result="result.pdf">
  <XDP>
    <XDP source="doc1.xdp"/>
  </XDP>

  <XDP>
    <XDP source="doc2.xdp"/>
  </XDP>
</PDF>
```

**Single XFA-based form remains interactive**

You can assemble non-interactive PDF documents with an interactive PDF document, provided these conditions are satisfied:

- PDF result contains at most one PDF source element that contains a single XFA-based form. (Other PDF source elements provide non-interactive documents or Acrobat forms.)
- PDF source element containing the XFA-based form is the base document.
- XFA-based form is static (not dynamic).

Example: **Assembling an interactive XFA-based form with other documents**

```
<PDF result="intermediate_result.pdf" return="false">
  <XDP>
    <XDP source="doc1.xdp"/>
    <XDP source="doc1.xdp"/>
  </XDP>
</XDP>
```

```<PDF result="final_result.pdf">```
PDF documents from Acrobat forms

The Document Manipulation APIs can successfully assemble PDF documents from multiple PDF documents containing Acrobat forms. By default, Document Manipulation APIs retains all form fields.

The following example successfully returns a fillable form, if the two sources are Acrobat forms.

**Example: Assembling a PDF document from multiple Acrobat forms**

```xml
<PDF result="result.pdf">
    <PDF source="acroform1.pdf"/>
    <PDF source="acroform2.pdf"/>
</PDF>
```

Package a PDF document as XDP

The Document Manipulation APIs can package a PDF document containing an XFA-based form into as XDP, as shown in this example.

**Example: Packaging a PDF document containing an XFA-based form as XDP**

```xml
<DDX>
    <PDF result="doc.xdp" format="XDP">
        <PDF source="xfa-form.pdf"/>
    </PDF>
</DDX>
```

**Note:** The DDX fails if the PDF source element does not contain an XFA-based form.
Assembling PDF Packages and Portfolios

You can use DDX expressions to create PDF packages and portfolios.

**PDF packages** are containers for a collection of documents. A PDF package includes metadata to support efficient viewing, sorting, and searching of documents in the package. Acrobat 8 added support for PDF packages.

**PDF Portfolios** extend the capability of PDF packages, by adding a customizable user interface (navigator), folders, navigation header, and navigation welcome pages. The navigator is a compiled ActionScript program that can use resources that are independent of the program. The resources can include localized text string, custom color schemes, and graphic resources. Navigators conform to Navigator format and navigation welcome pages and navigation headers conform to the Navigator Template Format.

When designing a PDF Portfolio, consider how the portfolio appears in Acrobat 9 and Acrobat 8:

- **Acrobat 8**: Supports only PDF packages. A PDF Portfolio viewed in Acrobat 8 appears to be a PDF package. The cover sheet is displayed as if it was added as a package file and set as the default initial document. The navigator, navigation welcome page, and navigation header are not visible and files do not appear to be in folders.

- **Acrobat 9**: Supports PDF Portfolios and packages. A PDF package when viewed in Acrobat 9 appears to be a PDF Portfolio.

See also

- Adobe Developer Center

- The PDF Developer Junkie Web: Customizing PDF Portfolio Layouts

- AEM Forms Doc Team: Using Document Manipulation to Create PDF Portfolios (PDF Packages)

Understanding PDF packages

Before PDF version 1.7, all PDF documents were *single PDF documents* consisting of pages and possibly document-level file attachments. A PDF document as a container for a collection of documents, known as a **PDF package**. A PDF package consists of a cover sheet, package files, and a package specification. In Acrobat 9, a PDF package is called a **PDF Portfolio** or simply a **portfolio**.

A *package specification* provides information about how the files in the collection are displayed. It can also contain a schema that defines the syntax of custom metadata that can be used to organize the files in the package or portfolio. If a package specification is added to a single PDF, it becomes a PDF package, and any preexisting document-level file attachments automatically become package files.

With packages, you can perform these tasks:

- Create a collection of documents that cannot be assembled in a single document. For example, some forms cannot be assembled in a single PDF but can be collected together in a package.

- Flatten a package into a single PDF if the documents it contains are modifiable. For example, an encrypted PDF document would remain as a document-level attachment.

- Add or change metadata in packages.
Add documents to packages.

Modify documents in a package. For example, you can change the headers in the documents if they are modifiable.

Export documents from a package and then reimport them into the package in a workflow. For example, you could export the documents, digitally sign them, and reimport them into the package.

About PDF package and portfolio properties

DDX provides properties you can use to specify PDF packages and portfolios.

PDF Package property: package files and package specifications

Here is a basic DDX expression that creates a PDF package:

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <PDF result="outDoc"/>
  <PDF source="cover" baseDocument="true"/>
  <Package/>
  <PackageFiles/>
  <PackageFiles/>
  <PackageFiles/>
</PDF>
</DDX>
```

Base file (the cover sheet)

The `<PDF source="cover" baseDocument="true"/>` expression defines the base document for the PDF package. The base document provides a cover page for the PDF package or Portfolio. It must be a PDF document that has at least one page. It also provides other basic characteristics such as page size and orientation.

If the base document is not specified, then a locale-specific default cover sheet is used ([AdobeCoverSheet](#)).

PackageFiles

The `PackageFiles` element specifies the PDF and non-PDF documents to add to the PDF package or portfolio. It also provides the metadata (the schema field values) for those documents. Here is a DDX expression that shows the basic structure of a `PackageFiles` element. This expression does not specify folders.

```xml
<PackageFiles>
  <PDF source="Elwood.pdf"/>
  <FieldData name="Genre">Movies</FieldData>
  <FieldData name="Location">Chicago</FieldData>
</PackageFiles>
```

The `FieldData` element specifies metadata associated with the folder.

If the source attribute for the `PackageFiles` element is a URL that references a folder, then the contents of the folder are added. If the `PackageFiles` is a child of a `Folder` element, then the folder structure is retained. (See "Folders" on page 50.)
Note: Document Services navigates down into the subfolders to include files in subfolders.

Package or Portfolio element

The Package or Portfolio element specifies information about custom metadata fields to use for a PDF package. It also specifies the display order, sort order for the default, and custom metadata fields. Here is a DDX expression that shows the structure of a Package element. The Portfolio element is the same with the addition of other properties specific for PDF Portfolios.

```xml
<Package>
    <Schema>
        <Field name="Character" type="Filename"/>
        <Field name="Genre" type="Text"/>
        <Field name="Location" type="Text"/>
    </Schema>
    <DisplayOrder/>
    <SortOrder/>
</Package>
```

Here are elements that can be children of the Package element. These child elements define package characteristics:

- **Schema** element defines the custom metadata for the PDF package or portfolio.
- **DisplayOrder** element defines the display order for the result, in which the columns describing the order in which viewing applications display the package files.
- **SortOrder** element defines the priority viewing applications apply to the fields when sorting. It also defines the order of the package files when assembling into a single PDF document.

Files in a PDF package have default metadata such as filename and file size. They can also have custom metadata that is defined in FieldData elements for the files. Metadata that exists in other forms such as Acrobat metadata (author, title, and subject) is not used in PDF packages.

**PDF Portfolio properties**

Document Services (9.0) and Acrobat 9 add support for navigators, folders, and navigation welcome pages.

The Communications Document Manipulation APIs cannot help you create a custom navigator, a navigation welcome page, or a navigator header. But it lets you include such resources in a PDF navigator result.

Use Acrobat Pro to create a navigation welcome page or navigator header. Use Adobe Flex® Builder™ and the Acrobat SDK to build a custom navigator. See the Adobe Developer Center for details.
Navigators

Your PDF Portfolio can include navigators from these sources:

Document Services. These navigators are available from the Adobe Navigators LCA. This LCA initially contains three basic multi-lingual navigator resources. These resources are similar to three basic layouts that are available in Acrobat Pro: On an Image, Revolve, and Sliding Row.

These multi-lingual navigator resources support the same 28 languages that are supported for the multi-lingual cover sheet. The locale-specific version is selected by using the TargetLocale. See _AdobeCoverSheet for information about localization.

You can add custom navigators to this LCA. Here are navigators included in the Navigator LCA:

AdobeOnImage.nav. Requests a background image. The schema includes X and Y viewer coordinates that specify placement of files and folders on the background image when viewed. (See the example at “DDX that creates PDF Portfolio that uses the AdobeOnImage navigator” on page 51.)

AdobeRevolve.nav

AdobeSlidingRow.nav

Adobe Acrobat Pro. These navigators reside in the installation directory.

Other sources. Custom navigators that you or others develop.

PDF Portfolios. Your DDX can specify the navigator from an existing PDF Portfolio that contains a custom navigator.

Here is a DDX example that accesses a navigator in the default Adobe Acrobat Pro installation directory. This example assumes that the application is installed on the Document Services server. This example does not include any package files.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
<PDF result="myPortfolio.pdf">
    <Portfolio>
        <Navigator source="file:///C:/Program Files/Adobe/Acrobat 9.0/Acrobat/Navigators/AdobeRevolve.nav"/>
    </Portfolio>
</PDF>
</DDX>
```

Here is a DDX example that obtains the navigator from a PDF file.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
<PDF result="myPortfolio.pdf">
    <Portfolio>
        <Navigator source="existingPortfolio.pdf"/>
    </Portfolio>
</PDF>
</DDX>
```
Folders

Folders are analogous to directories in a hierarchical file system and allow files to be grouped in a recursive manner. Folders provide a scalable and efficient mechanism for arranging files in a portable collection into folders, while maintaining a high degree of compatibility with older viewers.

You can create a folder with attributes and place files into specific folders regardless of their source. Folders are visible only when viewing a PDF Portfolio in Acrobat 9. Therefore, adding a Folder element for a single PDF turns the PDF into a portfolio. That is, adding a Folder element has the same as adding a portfolio element.

You can create a folder hierarchy by nesting Folder elements. Each Folder element can include a graphic to use as a thumbnail image associated with the folder.

Here is an example of a nested folder structure. The path to the sole package file is /QE/Beta Test Files/cool.pdf.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"
<PDF result="pkg.pdf">
  <PDF source="doc1.pdf"/>
  <Folder name="QE" thumbnail="qeImage.jpg">
    <Description>QE Root folder</Description>
    <Folder name="Beta Test Files" thumbnail="betaImage.jpg">
      <Description>Beta Test Files</Description>
      <PackageFiles source="cool.pdf">
        <Description>test for coolness</Description>
        <FieldData name="Owner">Beta Boop</FieldData>
      </PackageFiles>
    </Folder>
  </Folder>
</PDF>
</DDX>
```

Here is a DDX that shows the basic structure of a DDX expression that creates a nested Folder in a PDF package:

```xml
<Folder name="Characters">
  <Description>Fictional characters.</Description>
  <Folder name="Movies">
    <PackageFiles>
      <PDF source="Elwood.pdf"/>
      <FieldData name="Location">Chicago</FieldData>
    </PackageFiles>
  </Folder>
</Folder>
```

Navigation welcome page and navigation heading

The navigation welcome page appears only once when the portfolio is opened in the viewer and the navigation header appears across all navigation panes. These features are specified as XML that conforms to the Navigator Template Format. Acrobat 9 or Adobe Reader 9 introduced support for these features.

**Note:** The Document Manipulation APIs cannot help you create the navigation header (not to be confused with a page header). Nor can the service help you create a navigation welcome page. The best tool for creating navigation headers and navigation welcome pages is Acrobat Pro Extended 9.
You can specify a navigation header or navigation welcome page file, and its resources. Alternatively, you can specify an existing PDF Portfolio from which to obtain a header or welcome page.

The navigation welcome page is a resource with a name of "welcome/model.xml". If the welcome page source is specified, then all resources excluding the navigation header ("header/model.xml") in the PDF source are specified. Only resources referenced by a "header/model.xml" or "welcome/model.xml" are visible in the PDF Portfolio navigation pane (also called the **PDF Portfolio Layout pane**). If the source specified is an XML source, then any resources it references must also be specified. Such resources include images and localized strings.

The navigation header is a resource with a name of "header/model.xml". If the header source is specified, then all resources excluding the WelcomePage ("welcome/model.xml") in the source are specified. Only resources referenced by a "header/model.xml" or "welcome/model.xml" are visible in the portfolio navigation pane.

**Example: Choosing a new welcome page for a PDF Portfolio**

```xml
<PDF result="newPackage">
  <Portfolio>
    <WelcomePage source="myXMLWelcomePage"/>
    <Header source="myXMLHeader"
  </Portfolio>
  <PackageFiles>
    <PDF source="pkg1.pdf"/>
    <PDF source="pkg2.pdf"/>
  </PackageFiles>
</PDF>
```

**Creating a PDF Portfolio**

The following example creates a PDF Portfolio that uses the AdobeOnImage navigator. For each file or folder that is added, the DDX supplies a `FieldData` element that specifies the location for the icon. The DDX also supplies an image in the `Resource` element.

The PDF Portfolio produced by this example works well in Acrobat 8 and 9. When the PDF Portfolio is viewed in Acrobat 8, the cover sheet is not displayed.

**Example: DDX that creates PDF Portfolio that uses the AdobeOnImage navigator**

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <PDF result="portfolio1.pdf">
    <Portfolio>
      <Navigator source="AdobeOnImage.nav">
        <!-- Or source="myCustomNav.nav" or source="myCustomNav.pdf" -->
        <!-- AdobeOnImage uses the following resource. -->
        <!-- Each navigator is unique in its use of resources. -->
        <Resource name="navigator/image.xxx" source="myImage.png"/>
      </Navigator>
    </Portfolio>
    <Folder name="LC ES4 Testing">
      <Folder name="Single Files">
        <PackageFiles source="process:///process_data/@doc1" required="false">
          <!-- These FieldData entries are specific to the AdobeOnImage navigator.-->
        </PackageFiles>
      </Folder>
    </Folder>
  </PDF>
</DDX>
```
Creating a PDF package

In the example below, a new package is created. The pages from single1 become the cover sheet for resultDoc, and documents doc2 and doc3 are added as package files. The InitialViewProfile element "letterCollection", which contains the package specification, suggests how the viewing applications initially display the document. Viewing applications that support only PDF packages behave differently from applications that support PDF Portfolios.

Example: Creating a PDF package

<DDX>
  <PDF result="resultDoc" initialView="letterCollection">
    <Package styleReference="letterCollection" />
    <PDF source="single1" />
    <PackageFiles source="doc2">
      <File filename="/documents/data" mimetype="application/pdf"/>
      <Description>email from Employee B</Description>
      <FieldData name="to">Employee A</FieldData>
      <FieldData name="from">Employee B</FieldData>
      <FieldData name="date">2005-06-21T09:47:00Z</FieldData>
      <FieldData name="subject">Re: Lunch on Friday!</FieldData>
    </PackageFiles>
    <PackageFiles source="doc3">
      <File filename="/documents/data" mimetype="application/pdf"/>
      <Description>email from Employee A</Description>
      <FieldData name="to">Employee B</FieldData>
      <FieldData name="from">Employee A</FieldData>
      <FieldData name="date">2005-06-21T10:12:00Z</FieldData>
      <FieldData name="subject">Re: Lunch on Friday!</FieldData>
    </PackageFiles>
  </PDF>
</DDX>
Change the cover sheet for an existing PDF package or portfolio

You can change the cover sheet for an existing PDF package or portfolio.

**Note:** Acrobat 8 is more likely to display the cover sheets in PDF packages or portfolios. Acrobat 9 displays those cover sheets only when specific viewer preferences are specified.

Choose a new cover sheet

Specify the new the cover sheet with a PDF source element and specify the PDF package or portfolio to modify in a PackageFiles element. The PDF source must specify a single document, not an array of documents. If the cover sheet is not already PDF, the Document Manipulation APIs attempts to convert it to PDF.

The following PDF package contains these parts:

- Cover sheet that uses the pages from `single1`
- Package files from `pkg1.pdf`, `pkg2.pdf`
- Any document-level file attachments from `single1`

Page-level file attachments remain for every page included in the cover sheet (`single1`). No page-level file attachments are included from `pkg1.pdf` and `pkg2.pdf`. The package specification is defined by merging the package specification from `pkg2.pdf` with the package specification from `pkg1.pdf`.

**Example: Choosing a new cover sheet for a PDF package**

```
<PDF result="newPackage">
```
Add or remove pages to an existing cover sheet

As shown in the example below, to add pages to an existing cover sheet, specify the additional source documents. In the following example, pages from `prepend-pages` and `append-pages` are added to the cover sheet contained in `pkg1.pdf`.

**Example: Adding pages to a cover sheet**

```xml
<PDF result="resultDoc">
  <PDF source="prepend-pages"/>
  <PDF source="pkg1.pdf" baseDocument="true"/>
  <PDF source="append-pages"/>
</PDF>
```

As shown in the example below, to remove pages from an existing cover sheet, specify the pages to include in the source document. In the following example, pages 2-4 are removed from the cover sheet.

**Example: Removing pages from a cover sheet**

```xml
<PDF result="resultDoc">
  <PDF source="pkg1.pdf" pages="1,5-last"/>
</PDF>
```

As shown in the example below, if no cover sheet is specified, the `_AdobeCoverSheet` is used as the default.

**Example: Use a default cover sheet**

```xml
<PDF result="resultDoc">
  <PackageFiles>
    <PDF source="pkg1.pdf"/>
    <PDF source="pkg2.pdf"/>
  </PackageFiles>
</PDF>
```

As shown in the example below, the `AdobeCoverSheet` is provided in 15 languages and selected by specifying a locale. To support other languages, create locale-specific PDF documents to use as the cover sheet. You can also use your custom cover sheet to replace the cover sheet in an existing PDF package or portfolio. The Document Manipulation APIs cannot select custom cover sheets based on locale.

If you are using a default package specification, the same locales that are supported for the Adobe cover sheet are also available to select localized display strings for the default package specification. In the following example, the German version of the Adobe cover sheet is specified.

**Example: Localizing a cover sheet**

```xml
<PDF result="resultPkg">
  <TargetLocale locale="de_DE"/>
  <PDF source="_AdobeCoverSheet"/>
```
Creating a package or portfolio specification from other ones

A package or portfolio specification provides organizational information about the package or portfolio. Portfolio specifications also identify the navigator and data that the navigator consumes.

Creating a package or portfolio specification by aggregating existing ones

You can also create a portfolio specification by aggregating portfolio specifications in existing PDF Portfolios.

In the following example, two package or portfolio specifications are assembled. The new package specification is created by aggregating the two package specifications.

If the PDF source elements contain PDF packages, the Schema, DisplayOrder, and SortOrder elements are aggregated.

If the PDF source elements contain PDF Portfolios, only the Schema, DisplayOrder, and SortOrder elements are aggregated. The other elements in the PDF result's Portfolio element are taken from the first portfolio, in this case pkg2.pdf.

If any PDF source is a package, the resultant document will be a package.

If none of the PDF sources are packages, the result is a single PDF. To force the result to be a package, without overriding any package specifications, add <Package/> within the <PDF> result block.

Example: Assembling packages

```xml
<PDF result="newPackage">
  <PDF source="pkg2.pdf"/>
  <PDF source="pkg3.pdf"/>
</PDF>
```

Selecting the package specification from an existing package

In the following example, the explicit Package element specifies only the package specification that exists in pkg1.pdf. As the package specification is explicitly specified, it does not include the specifications from pkg2.pdf or pkg3.pdf. The cover sheet is aggregated from the cover sheets in pkg2.pdf and pkg3.pdf.

This behavior is also true for a portfolio specification and the Portfolio element.

Example: Determining the content of a package specification

```xml
<PDF result="newPackage">
  <Package>
    <PDF source="pkg1.pdf"/>
  </Package>
</PDF>
```
In the example above, if `pkg1.pdf` is only a single PDF, it contributes nothing to the assembly. The package specification starts with `pkg2.pdf`, followed by the specification that is contained in `pkg3.pdf`. Considering that `pkg1.pdf` is only a single PDF, the following DDX achieves the same result.

**Example: Using a default package specification**

```xml
<PDF result="newPackage">
  <Package />
  <PDF source="pkg2.pdf"/>
  <PDF source="pkg3.pdf"/>
</PDF>
```

**Overriding properties in merged package or portfolio specifications**

You can assemble a PDF package or portfolio specification from the package or portfolio specifications in other packages or portfolios. By default, the existing specifications are aggregated the package or portfolio specification of resultant document. You can override this default behavior by specifying a `Package` element as a child of the result. The `Package` element specifies the Schema, DisplayOrder, SortOrder, and TargetLocale that you want to replace. For example, if the `Package` element supplies only a Schema element, then the schema in the package specification of the resultant PDF package contains that schema. All other properties are taken from the merged package specifications.

This behavior is also true for a portfolio specification and the Portfolio element.

**Changing the metadata display order in an existing PDF package or portfolio**

You can change the display order in an existing PDF package, without changing the schema or sort order. Make this change by adding a new package specification with the changes you want. The package specification for the resultant PDF package or portfolio merges the new package specification with the other package specifications included in the PDF result element.

The following example creates a package specification with a new display order for the metadata. It also retains the same schema and sort order already defined in `pkg1.pdf`.

**Example: Specifying a different display order in a package specification**

```xml
<PDF result="newPackage">
  <Package>
    <DisplayOrder>
      <Field name="Phone" />
      <Field name="Last Name" />
      <Field name="First Name" />
      <Field name="Address" />
    </DisplayOrder>
  </Package>
  <PDF source="pkg1.pdf"/>
</PDF>
```
Modifying the package files in a PDF package or portfolio

The DDX grammar lets you add, modify, and extract the package files in an existing PDF package or portfolio.

Adding single files to an existing PDF package or portfolio

You can add individual files to a PDF package or portfolio. The Document Manipulation APIs retains the documents’ custom metadata if the `Schema` element defines that metadata. The service removes any custom metadata that the Schema element does not define. The removed metadata can be pre-existing or newly specified metadata (by a `FieldData` element).

In the following example, `single` is a single PDF and `pkg` is a PDF package. The result, `newPackage`, is a package file that contains the package specification that existed within `pkg`.

**Example: Adding a single PDF as a package file (a document in the collection)**

```xml
<PDF result="newPackage">
  <PDF source="pkg" />
  <PackageFiles source="single">
    <File filename="test.pdf" mimetype="application/pdf"/>
    <FieldData name="Description">example</FieldData>
  </PackageFiles>
</PDF>
```

Adding documents from a PDF package or portfolio to another

Using a `PackageFiles` source, filter, or import element, you can specify the package files and specifications to assemble in the resultant PDF package or portfolio.

Using nameKeys to select files from a PDF package or portfolio file

The following example uses the `nameKeys` attribute to specify which files to include in a PDF package. This example produces a resultant PDF Portfolio that includes the following files:

- Cover sheet and all package files from `portfolio.pdf`
- All package files from `pkg2.pdf`
- `single4`
- `doc1.pdf` and `doc2.pdf` from `pkg3.pdf`

The document `single5` is omitted because the `nameKeys` filter does not select it. The document is a single PDF and has no `namekey`. The package specifications from the source PDF packages and portfolios are merged into one, beginning with `portfolio.pdf`, followed by `pkg2.pdf`, and then `pkg3.pdf`.

**Example: Using a PackageFiles filter element to select package files**

```xml
<PDF result="newPortfolio">
  <PDF source="portfolio.pdf"/>
  <PackageFiles>
    <PDF source="pkg2.pdf"/>
    <PDF source="single4"/>
  </PackageFiles>
</PDF>
```
Using the nameKeys to select folders from a PDF Portfolio

Here is an example that uses the nameKeys attribute to specify which folders to include from a PDF Portfolio. The PackageFiles in the resultant PDF Portfolio includes all files from portfolio_1.pdf and only those files in portfolio_2.pdf that reside in the Drafts folder.

Example: Using a PackageFiles filter element to select folders and their files

```xml
<PDF result="newPortfolio">
  <PDF source="portfolio_1.pdf"/>
  <PackageFiles nameKeys="/Drafts">
    <PDF source="portfolio_2.pdf"/>
  </PackageFiles>
</PDF>
```

Modifying selected files in a PDF package or portfolio

Using a PackageFiles select element, you can select package files that are contained in the parent PDF result or parent PDF source document. The PackageFiles select element is typically used to modify the package files or to add metadata. (See “PackageFiles select elements” on page 220.)

In the following example, package files that the nameKey selects from portfolio_1.pdf are marked with a "Draft" watermark. All package files from portfolio_1.pdf are then marked with an additional "For Review" watermark. Selected files are marked only if they are modifiable PDF documents.

Example: Using a select element to modify package files

```xml
<PDF result="newPortfolio.pdf">
  <PDF source="portfolio_1.pdf"/>
  <PackageFiles nameKeys="/Draft/doc1.pdf">
    <Watermark><StyledText><p>Draft</p></StyledText></Watermark>
  </PackageFiles>
  <PackageFiles>
    <Watermark verticalOffset="2in" replaceExisting="false">
      <StyledText><p>For Review</p></StyledText>
    </Watermark>
  </PackageFiles>
</PDF>
```

Exporting and importing package files

You can use the PackageFiles result to extract, modify, and reimport the files in a PDF package or portfolio.

The PackageFiles result element exports the package files and a descriptive XML document:
**Exported package files:** These files are turned in a document map. Each exported file has one entry in the map. The entry names correspond to nameKey attributes from the PackageFiles XML file.

**Descriptive XML file (PackageFiles XML file):** This file provides information about each of the package files, such as the metadata and the unique, internal nameKey. The filename of each exported package file is represented in the nameKey attribute within the resultant XML document. For package files that are in a PDF Portfolio with folders, the nameKey is the full path. Consider a portfolio that contains a main folder named Reports. The Reports folder contains a subfolder named June. The value of the nameKey attribute for the file is /Reports/June//summary.pdf. (See “PackageFiles Language” on page 347.)

The following examples show how you can export, modify, and import package files. In the first example, the package files and the descriptive XML file are exported. The package files are then processed elsewhere. When the Document Manipulation APIs is invoked for the second example, the input map contains entries for each of the modified files. The entry names correspond to the nameKey entries in the descriptive XML file. The mypkg.xml is unmodified.

**Example: Exporting package files**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Export package files -->
<DDX>
  <PackageFiles result="mypkg.xml" extract="true">
    <PDF source="myPkg"/>
  </PackageFiles>
</DDX>
```

**Example: Importing modified package files**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Import package files -->
<DDX>
  <PDF result="myPkg">
    <PDF source="myPkg"/>
    <PackageFiles import="mypkg.xml"/>
  </PDF>
</DDX>
```

**Converting a PDF package or portfolio into a single PDF**

The NoPackage and NoPortfolio elements assemble a single PDF document from the contents of its parent PDF element. It assembles the documents by appending each PDF package file to the cover sheet in sort order. Any package files that cannot be converted into PDF (such as XFA-based forms), remain as file attachments of the result.

Consider a PDF result that assembles pkg1.pdf and pkg2.pdf. The pkg1.pdf package contains single1a and single1b as package files. The pkg2.pdf package contains single2a and single2b as package files. The aggregated package specification produces the package files sorted in this order:

```
[single1a, single2a, single1b, single2b]
```

The pages in the resultant document consist of the cover sheets from pkg1.pdf and pkg2.pdf, followed by pages from single1a, single2a, single1b, and single2b. The file attachments in the resultant document are obtained from these sources:

```
Any document-level file attachments in single1a, single1b, single2a, and single2b
```
Other package files that are not assembled. They become document-level file attachments.

All page-level file attachments from all six files.

Encrypted PDF documents, dynamic XFA forms, and non-PDF documents cannot be assembled into a single PDF.

The following example shows the usage of the NoPackage element in this case.

**Example: Converting a package to a single PDF**

```xml
<PDF result="newSingle">
  <NoPackage/>
  <PDF source="pkg1.pdf"/>
  <PDF source="pkg2.pdf"/>
</PDF>
```

You can also use the NoPackage element to assemble a single PDF from PDF packages. In this case, the Package/> child of a PDF> source element promotes a single PDF into a PDF package. The pages become the cover sheet and the document-level file attachments become package files.

**Example: Assembling a single PDF from two PDF documents**

```xml
<PDF result="newSingle">
  <NoPackage/>
  <PDF source="doc1">
    <Package/>
  </PDF>
  <PDF source="doc2">
    <Package/>
  </PDF>
</PDF>
```
Disassembling PDF Documents

You can use the Communications Document Manipulation APIs to disassemble a single PDF document into multiple documents. Disassembly is useful when the document was originally created from many individual documents, such as a collection of bank statements.

**Note:** If a PDF file contains invalid XMP metadata, the Document Manipulation APIs throws an exception when processing that file. The error message states that a "SyncMetadataTask task failed for document."

Such errors can occur with documents produced by outdated software, such as old versions of Adobe Distiller® or Adobe FrameMaker®. You can avoid such problems by regenerating the input PDF document using updated software.

To disassemble a document, use the PDFsFromBookmarks element, as in the following example.

**Example: Disassembling documents**

```xml
<PDFsFromBookmarks prefix="stmt">
  <PDF source="doc1.pdf"/>
</PDFsFromBookmarks>
```

The PDFsFromBookmarks element is a result element and can be a child only of a DDX element. (It does not have a result attribute because it can result in the generation of multiple documents.)

PDFsFromBookmarks generates a single document for each level 1 bookmark in the source document (doc1.pdf in this example). The Document Manipulation APIs generates a name for each document that is the concatenation of the following items:

- A string specified by the prefix attribute
- A 6-digit sequence number (This number could be used to re-create the original order of the pages after the document is disassembled.)
- The bookmark title
- The filename extension .pdf

**Note:** Bookmarks can contain characters that are not legal in filenames. When saving the result streams as files, the client is responsible for specifying appropriate filenames.

In the example, assume that the level 1 bookmarks in doc1.pdf are as follows:

"Chapter 1" (beginning on page 3)
"Chapter 2" (beginning on page 13)
"Chapter 3" (beginning page 21)

There are three result documents:

stmt.000001.Chapter 1.pdf, containing pages 4-12
stmt.000002.Chapter 2.pdf, containing pages 13-20
stmt.000003.Chapter 3.pdf, containing pages 21 through the last page.

Here are more details about how the documents are separated:
Any level 1 bookmark that does not point to a page is ignored.

The bookmarks must be in sequence. That is, they must point to pages that are in ascending order. Otherwise, an error is returned.

A single page is never extracted into two separate PDF documents. The first page of the first extracted document is the destination page of the first level 1 bookmark. Pages are included until either the end of the document is reached (the last page is included) or another level 1 bookmark is reached (the page before that is included).

In the example, if Chapter 1 ends on page 13 and Chapter 2 begins in the middle of the page, then the entire page 13 appears in stmt.000002.Chapter 2.pdf instead of in stmt.000001.Chapter 1.pdf.

If more than one level 1 bookmark occurs on a page, all bookmarks except the first one are ignored.

The separated PDF documents are created as if they came from non-base documents (see “About base documents” on page 29). Document-level elements such as properties, attachments, and initial views are not included in the resultant PDF documents.

If the source document is encrypted, provide the master password for the document to disassemble it. (See “Accessing a password-protected document” on page 131.)

As with PDF result elements, you can specify these attributes for the PDFsFromBookmarks element:

- encryption can be set to either the name of a PasswordEncryptionProfile element that specifies password encryption or None (the default). A value of None indicates that the documents are not encrypted. (See “Working with Secured Documents” on page 129.)
- save can be set to Full (the default) or FastWebView (see “Saving PDF documents” on page 34). Incremental saving is not applicable to disassembled documents.
Working with Bookmarks and Thumbnails

In PDF documents, bookmarks are a tree-structured hierarchy of outline items that provide a means of navigating the document. When a user in a viewer application clicks a bookmark, an action is triggered. Typically, a bookmark action specifies a particular location in the document to which the viewer navigates. However, bookmarks can also trigger actions such as opening web pages or running JavaScript code.

For information on creating a table of contents from the bookmarks in a document, see “Adding a table of contents” on page 85.

Including and excluding bookmarks

By default, all bookmarks from all source documents are included in the result document. Therefore, you do not need to do anything special to preserve bookmarks. In the following example, the result document contains the bookmarks from doc2, doc3, and doc4, in that order. The first (level 1) bookmark from doc3 follows the last bookmark from doc2, and so on.

**Example: Preserving bookmarks**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc3"/>
  <PDF source="doc4"/>
</PDF>
```

To exclude bookmarks from a source document, use the `NoBookmarks` element.

**Example: Removing bookmarks from a source document**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc3">
    <NoBookmarks/>
  </PDF>
  <PDF source="doc4"/>
</PDF>
```

In the example above, bookmarks from doc2 and doc4 are included in the result document; bookmarks from doc3 are not included.

In the following example, because the `NoBookmarks` element is a child of the `PDF` result element, none of the bookmarks in its scope are included.

**Example: Removing all bookmarks**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc3"/>
  <PDF source="doc4">
    <NoBookmarks/>
  </PDF>
</PDF>
```
In the following example, the `NoBookmarks` element is a child of a `PDFGroup` element. Therefore, no bookmarks are included from the source documents within the `PDFGroup` element. Only the bookmarks from `doc4` are included in the result.

**Example: Selectively removing bookmarks**

```xml
<PDF result="doc1">
  <PDFGroup>
    <PDF source="doc2"/>
    <PDF source="doc3"/>
    <NoBookmarks/>
  </PDFGroup>
  <PDF source="doc4"/>
</PDF>
```

`NoBookmarks` and `Bookmarks` cannot be siblings.

You can also include bookmarks from PDF documents other than your PDF source documents. You can restrict the bookmarks that are included by using the `Bookmarks` element as a filter element.

**Example: Using a bookmarks filter element**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <Bookmarks>
    <PDF source="doc3"/>
    <PDF source="doc4"/>
  </Bookmarks>
</PDF>
```

In this example, the `Bookmarks` element filters the bookmarks from `doc3` and `doc4` and adds them to the result. No content from `doc3` and `doc4` other than bookmarks are included in the result. `doc2` provides PDF page content as well as bookmarks (if any) to the result.

The same task could be accomplished by exporting the bookmarks from `doc3` and `doc4` to an XML representation and then importing the bookmarks into `doc2`. The next section describes this process in more detail.

**Exporting and importing bookmarks**

The Communications Document Manipulation APIs supports an XML representation of bookmarks. This representation conforms to a schema whose namespace is `http://ns.adobe.com/xpdf/1.6`. You can export bookmarks to this format or import them from this format.

**Exporting bookmarks from a PDF document**

To export bookmarks, use a `Bookmarks` result element. (See "Bookmarks" on page 163.)

The following example exports bookmarks from `doc2` and `doc3` to `doc1.xml`.

**Example: Exporting bookmarks**

```xml
<Bookmarks result="doc1.xml">
  <PDF source="doc2"/>
```
A Bookmarks result element can have any number of PDF source or PDFGroup elements as children. It can also have Bookmarks source or filter elements as children. The bookmarks appear in the order they would appear if all the source documents were assembled into a PDF document. In the above example, if doc2 has 5 pages, a bookmark in doc3 that references page 2 is updated in the result document to reference page 7.

**Importing bookmarks into a PDF document**

You can save a bookmarks XML document to a file and use it in a separate part of your workflow. Within a DDX file, you can use a bookmarks XML document exported from one document to import bookmarks into a result PDF document. You can also create a bookmarks XML document and import that into a result PDF document.

**Example: Adding bookmarks from one document to another**

```xml
<Bookmarks result="doc1.xml" return="false">
  <PDF source="doc2"/>
</Bookmarks>
<PDF result="doc3">
  <PDF source="doc4"/>
  <Bookmarks source="doc1.xml"/>
</PDF>
```

In this example, the bookmarks from doc2 are included in doc1.xml, which in turn is used to add bookmarks to doc3. Note the following features of this example:

- The return attribute of the Bookmarks result element is set to false because doc1.xml is being used only in the current DDX file and therefore does not need to be returned as a stream to the client. (By default, return is true.)
- The bookmarks from doc1.xml are appended to the bookmarks from doc4 (if any) in the result document doc3. To instead replace the bookmarks from doc4 with those from doc1.xml, make the Bookmarks element a child rather than a sibling of the PDF source element, as in the following example.

**Example: Replacing bookmarks**

```xml
<PDF result="doc3">
  <PDF source="doc4">
    <Bookmarks source="doc1.xml"/>
  </PDF>
</PDF>
```

**Creating bookmarks from source documents**

For each PDF source document in an assembly, you can specify the name of a bookmark in the result document by using the bookmarkTitle attribute. The bookmark appears in the result document as a level 1 bookmark whose destination is the first page of the source document.

In the following example, each chapter in the book has a level 1 bookmark. All bookmarks that are present in the source documents are moved down a level.
Example: Creating bookmarks from source documents

```xml
<PDF result="TheBook">
  <PDF source="Chap1" bookmarkTitle="Chapter 1"/>
  <PDF source="Chap2" bookmarkTitle="Chapter 2"/>
  <PDF source="Chap3" bookmarkTitle="Chapter 3"/>
</PDF>
```

For example, suppose the bookmarks in each of the chapters are:

- Section A
- Section B
- Section C

The bookmarks in the result document have this hierarchy:

Chapter 1
  - Section A
  - Section B
  - Section C

Chapter 2
  - Section A
  - Section B
  - Section C

Chapter 3
  - Section A
  - Section B
  - Section C

To create more than one level of bookmarks from the source documents, use intermediate results. (You cannot, for example, specify the `bookmarkTitle` attribute on the `PDFGroup` element.) The following example illustrates this point.

Example: Creating bookmarks from source documents

```xml
<PDF result="PartI">
  <PDF source="Chap1" bookmarkTitle="Chapter 1"/>
  <PDF source="Chap2" bookmarkTitle="Chapter 2"/>
  <PDF source="Chap3" bookmarkTitle="Chapter 3"/>
</PDF>
<PDF result="PartII">
  <PDF source="Chap4" bookmarkTitle="Chapter 4"/>
  <PDF source="Chap5" bookmarkTitle="Chapter 5"/>
</PDF>
<PDF result="PartIII">
  <PDF source="Chap6" bookmarkTitle="Chapter 6"/>
  <PDF source="Chap7" bookmarkTitle="Chapter 7"/>
  <PDF source="Chap8" bookmarkTitle="Chapter 8"/>
</PDF>

<PDF result="BigDoc">
```
Here are the resulting bookmarks:

Part I
  Chapter 1
    ...bookmarks from Chap1
  Chapter 2
    ...bookmarks from Chap2
  Chapter 3
    ...bookmarks from Chap3

Part II
  Chapter 4
    ...bookmarks from Chap4
    ...etc...

**Sorting bookmarks**

You can sort bookmarks in a result PDF. Sorting is relative to the target page numbers in the resultant document. This feature is helpful when you insert one document into another document, where both documents contain bookmarks. For example, if you use the following DDX to assemble documents that have bookmarks, the bookmarks in resultant document are ordered according to the destination page number.

**Example: Inserting one document into another with sorting**

```xml
<DDX>
  <PDF result="final.pdf" sortBookmarks="true">
    <PDF source="doc1" pages="1-2" />
    <PDF source="doc2" />
    <PDF source="doc1" pages="3-last"/>
  </PDF>
</DDX>
```

In contrast, if you use the following DDX to assemble the same documents, the bookmarks in the resultant document are logically out of order.

**Example: Inserting one document into another without sorting**

```xml
<DDX>
  <PDF result="final.pdf">
    <PDF source="doc1" pages="1-2" />
    <PDF source="doc2" />
    <PDF source="doc1" pages="3-last"/>
  </PDF>
</DDX>
```

Consider using the above DDX examples with the documents described in the following tables.
Bookmark structure of Doc1

<table>
<thead>
<tr>
<th>Bookmark</th>
<th>Target page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ba1</td>
<td>1</td>
</tr>
<tr>
<td>-- Ba1-1</td>
<td>2</td>
</tr>
<tr>
<td>-- Ba1-2</td>
<td>4</td>
</tr>
<tr>
<td>Ba2</td>
<td>5</td>
</tr>
<tr>
<td>-- Ba2-1</td>
<td>7</td>
</tr>
<tr>
<td>-- Ba2-2</td>
<td>10</td>
</tr>
</tbody>
</table>

Bookmark structure of Doc2

<table>
<thead>
<tr>
<th>Bookmark</th>
<th>Target page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bb1</td>
<td>1</td>
</tr>
<tr>
<td>-- Bb1-1</td>
<td>2</td>
</tr>
<tr>
<td>-- Bb1-2</td>
<td>3</td>
</tr>
<tr>
<td>Bb2</td>
<td>4</td>
</tr>
</tbody>
</table>

Bookmark structure of the resultant document with sorting

When the DDX at "Inserting one document into another with sorting" on page 67 is applied to the example source documents, it produces a 14-page document with the following bookmark structure. When the PDF result element includes the sortBookmarks="true" attribute, the order of the bookmarks is consistent with the order of the target pages.

In some cases, the sort process inserts new bookmarks between sibling child bookmarks. To retain the relative structure of the bookmarks in such cases, the ancestor bookmarks are repeated. However, the repeated ancestor bookmarks have no destinations. In the following example, the Ba1 bookmark has the child bookmarks Ba1-1 and Ba1-2. The sorting process inserts several bookmarks between those sibling bookmarks. The Ba1 bookmark is repeated to show the parent relationship with the Ba1-2 bookmark.

<table>
<thead>
<tr>
<th>Bookmark</th>
<th>Target page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ba1</td>
<td>1</td>
</tr>
<tr>
<td>-- Ba1-1</td>
<td>2</td>
</tr>
<tr>
<td>Bb1</td>
<td>3</td>
</tr>
<tr>
<td>-- Bb1-1</td>
<td>4</td>
</tr>
<tr>
<td>-- Bb1-2</td>
<td>5</td>
</tr>
<tr>
<td>Bb2</td>
<td>6</td>
</tr>
<tr>
<td>Ba1</td>
<td>No target page</td>
</tr>
<tr>
<td>-- Ba1-2</td>
<td>8</td>
</tr>
</tbody>
</table>
Bookmark structure of the resultant document without sorting

Without sorting, the resultant document is a 14-page document with the following bookmark structure. The order of the bookmarks is inconsistent with the order of the target pages.

<table>
<thead>
<tr>
<th>Bookmark</th>
<th>Target page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ba1</td>
<td>1</td>
</tr>
<tr>
<td>-- Ba1-1</td>
<td>2</td>
</tr>
<tr>
<td>-- Ba1-2</td>
<td>8</td>
</tr>
<tr>
<td>Ba2</td>
<td>9</td>
</tr>
<tr>
<td>-- Ba2-1</td>
<td>11</td>
</tr>
<tr>
<td>-- Ba2-2</td>
<td>14</td>
</tr>
<tr>
<td>Bb1</td>
<td>3</td>
</tr>
<tr>
<td>-- Bb1-1</td>
<td>4</td>
</tr>
<tr>
<td>-- Bb1-2</td>
<td>5</td>
</tr>
<tr>
<td>Bb2</td>
<td>6</td>
</tr>
</tbody>
</table>
Removing thumbnails

A thumbnail is a small image of a PDF page. Acrobat can display thumbnails, allowing the user to navigate to a page by clicking the image. Thumbnails can be embedded in a document so that they can be displayed quickly when the document is opened. Acrobat generates thumbnails if they are not embedded.

Beginning with version 9.0, PDF packages and portfolios can include thumbnails for package files. However, Acrobat 9 does not recognize or display package file thumbnails.

A disadvantage of embedding thumbnails is that they increase the size of the document. For some workflows, such as when archiving documents for long periods of time, reducing file size is more important than display speed. Therefore, the Document Manipulation APIs provides the capability of removing embedded thumbnails from a document.

To remove thumbnails from a document, you use the `NoThumbnails` element, as in the following example. This element removes all thumbnails, including package file thumbnails.

**Example: Removing thumbnails**

```xml
<PDF result="doc3.pdf">
    <NoThumbnails/>
    <PDF source="doc1.pdf" />
    <PDF source="doc2.pdf" />
</PDF>
```

In the following example, `NoThumbnails` applies only to one of the source elements. Therefore, only thumbnails in doc1.pdf are removed.

**Example: Removing thumbnails from part of a document**

```xml
<PDF result="doc3.pdf">
    <PDF source="doc1.pdf">
        <NoThumbnails/>
    </PDF>
    <PDF source="doc2.pdf"/>
</PDF>
```
Working with Annotations

PDF documents can contain annotations that appear on a page but are not considered to be part of the page content. The Communications Document Manipulation APIs enables you to perform operations on annotations, such as importing them into and exporting them from PDF documents. You can work with the following annotation types:

- **Comments**, also known as *markup annotations*. These are items such as text notes, highlights, lines, and circles. They can be used in review and comment workflows.

- **Link** annotations are areas on a page that users can click to perform an action, typically to navigate to another part of the same document.

- **File attachment** annotations represent files attached to a page. They are described in "Working with File Attachments" on page 80.

**Note:** Other annotation types, such as 3D graphics and multimedia, cannot be imported and exported directly from PDF documents. However, they can be removed when working with the other annotation types.

PDF annotations are described in detail in the *PDF Reference*.

Including and excluding comments

By default, all comments from source documents are included in the result document. Therefore, you do not need to do anything special to preserve comments. In the following example, the result document contains the comments from doc2, doc3, and doc4.

**Example: Preserving comments**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc3"/>
  <PDF source="doc4"/>
</PDF>
```

Comments remain associated with the pages on which they appear in the source document. The following example assembles a subset of pages from doc2. Any comments (or other annotations) associated with pages that are not included do not appear in the result.

**Example: Assembling a subset of pages**

```xml
<PDF result="doc1">
  <PDF source="doc2" pages="1-5"/>
  <PDF source="doc2" pages="10-15"/>
</PDF>
```

To exclude comments from the pages in a source element, use the `NoComments` element.

**Example: Removing comments from a source document**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
</PDF>
```
In the example above, comments from doc2 and doc4 are included in the result document; comments from doc3 are not included.

**Note:** The NoComments element removes annotations (also known as comments). It removes all annotation types (including 3D and multimedia, for example). The exception is that Link annotations and Widget annotations cannot be removed. Use the NoLinks element to remove Link annotations.

In the following example, because the NoComments element is a child of the PDF result element, none of the comments in its scope are included.

**Example: Removing all comments**

```xml
<PDF result="doc1">
  <PDF source="doc2"/>
  <PDF source="doc3"/>
  <PDF source="doc4"/>
  <NoComments/>
</PDF>
```

In the following example, the NoComments element is a child of a PDFGroup element. Therefore, no comments are included from the source documents within the PDFGroup element. Only the comments from doc4 are included in the result.

**Example: Removing comments from a group of sources**

```xml
<PDF result="doc1">
  <PDFGroup>
    <PDF source="doc2"/>
    <PDF source="doc3"/>
    <NoComments/>
  <PDFGroup/>
  <PDF source="doc4"/>
</PDF>
```

NoComments and Comments cannot be siblings.

**Importing and exporting comments**

You can export a representation of the comments in a PDF document to a file and you can import comments from a file. The following file formats can be used to store comments:

- **Forms Data Format (FDF)** is based on PDF and is documented in the *PDF Reference*.
- **XFDF** is an XML representation based on FDF. It is documented at [http://www.adobe.com/devnet/topics/xml.html](http://www.adobe.com/devnet/topics/xml.html).

**Note:** In addition to comments, FDF can also be used to store information about links and form fields. In the Document Manipulation APIs, however, you can use FDF only for comments. Links must be
imported and exported using XFDF, and the Document Manipulation APIs cannot export form fields. For more information, see “Working with links” on page 77).

Some comment types are of special interest:

The Document Manipulation APIs provides special functionality with regard to file attachments. Page-level (not document-level) file attachments are considered comments and can be imported and exported in the same way as all other comments. For more information on file attachments, see “Working with File Attachments” on page 80.

Stamp annotations are also considered comments. However, they are not exported to XFDF.

Link annotations are not comments but can be imported and exported separately using XFDF.

The Comments element can be used in several ways:

As a source element, it specifies an FDF or XFDF stream containing comments.

As a result element, it contains comments from the aggregation of its child elements and is returned as an FDF or XFDF string. A Comments result element can appear only as a child of the DDX element.

As a filter element, it is like a result element except that it is not returned to the user and can be used as a source.

You can also select specific comments (see “Selecting specific comments” on page 74).

In this example, all comments from doc1.pdf are exported in XFDF format.

**Example: Exporting comments as XFDF**

```xml
<Comments result="doc1comments.xfdf" format="XFDF">
  <PDF source="doc1.pdf"/>
</Comments>
```

In this example, doc2comments.xfdf contains comments that were exported previously. Because the Comments source element is a child of the PDF source, the comments from doc2comments.xfdf replace the existing comments in doc1.pdf.

**Example: Replacing comments**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf">
    <Comments source="doc2comments.xfdf"/>
  </PDF>
</PDF>
```

In this example, the Comments source element is a sibling of the PDF source. Therefore, its comments are combined with the existing comments in the PDF source, doc1.pdf. The combined comments are included in the result.

**Example: Importing additional comments**

```xml
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <Comments source="comments.xfdf"/>
  </PDF>
</PDF>
```

In this example, comments are exported from one document and then imported into another. The Comments result element from the first step is then used as a source element to create a PDF result.
Example: Exporting and importing comments

```
<Comments result="doc1comments.xfdf" format="XFDF">
  <PDF source="doc1_rev1.pdf"/>
</Comments>

<Comments result="doc2.pdf">
  <PDF source="doc1_rev2.pdf"/>
  <Comments source="doc1comments.xfdf"/>
</Comments>
</PDF>
```

The same result can be accomplished with a Comments filter element. A Comments filter element is like a result element except that it can be used in the same way as a source element. The comments are exported from doc1_rev1.pdf and imported into doc2.pdf without the use of XFDF.

Example: Using a comments filter element

```
<PDF result="doc2.pdf">
  <PDF source="doc1_rev2.pdf"/>
  <Comments>
    <PDF source="doc1_rev1.pdf"/>
  </Comments>
</PDF>
```

The next section describes how you can select specific comments to export or import.

Selecting specific comments

A Comments result or filter element can specify a subset of the comments in their children rather than all of them. You can set several attributes specifying criteria by which comments are selected:

- **filter** specifies whether comments are included or excluded. If omitted, its default value is Include. If you specify a filter attribute with a value of Exclude, the selected comments are excluded from the result and the other comments are included.
- **beforeDate** and **afterDate** select comments dated before or after a particular date, respectively. The date is specified as an 8-character string of the format YYYYMMDD, where YYYY is the year, MM is the month, and DD is the day.
- **byAuthor** selects comments that match an author’s name.
- **byType** selects comments by the annotation type, for example, Text or Highlight. Annotation types are listed in the PDF Reference.
byCategory selects the comments from a category of annotation types. Here are examples of annotation types:

- **Notes**: Text annotations.
- **DrawingMarkups**: Line, PolyLine, Square, Circle, Polygon, and Ink annotations.
- **TextEditingMarkups**: Highlight, Underline, Squiggly, StrikeOut, Caret, and FreeText annotations.
- **Stamps**: Stamp annotations.
- **Attachments**: FileAttachment and Sound annotations.
- **All**: all of the above. If filter is set to Exclude, note that annotations in addition to the types listed here are removed from the result, as if NoComments were specified. The exception is that Link annotations and Widget annotations cannot be removed. Use the NoLinks element to remove Link annotations.

The selection criteria are additive. That is, all comments satisfying any of the specified criteria are included. In the following example, the result includes all comments from doc1_byGeorge.pdf that are dated before July 4, 2005 or are drawing markups.

**Example: Including selected comments**

```xml
<PDF result="doc4.pdf">
  <PDF source="doc1.pdf"/>
  <Comments beforeDate="20050704" byCategory="DrawingMarkups" filter="Include"/>
  <PDF source="doc1_byGeorge.pdf"/>
</Comments>
</PDF>
```

If the previous example specified filter="Exclude", the result would include all comments except comments that are dated before July 4, 2005 or are drawing markups.

This example exports all comments meeting any of these criteria: entered after June 1, 2005, Notes annotations, or authored by Joe User.

**Example: Exporting selected comments**

```xml
<Comments result="doc6.fdf" format="FDF" filter="Include"
  afterDate="20050601" byCategory="Notes" byAuthor="Joe User">
  <PDF source="doc1.pdf"/>
</Comments>
```

To include only those comments that meet all of a set of criteria (rather than any), use nested Comments elements. In the following example, the innermost Comments filter element includes only comments from doc1.pdf authored by Joe User. Its parent element uses those comments as a source and then selects comments that are Text annotations. Finally, the outermost Comments element narrows down the selection to comments created after June 1, 2005, and exports them in FDF format.
Example: Using nested selection criteria

```
<Comments result="doc6.fdf" format="PDF" filter="Include"
  afterDate="20050601">
  <Comments byCategory="Notes" filter="Include">
    <Comments byAuthor="Joe User" filter="Include">
      <PDF source="doc1.pdf"/>
    </Comments>
  </Comments>
</Comments>
```

In this example, doc1.pdf is a source PDF document containing no comments. The other three documents (doc1_fromTom.pdf, for example) are the same PDF document but contain comments from reviewers. The Comments element extracts all the comments from these three documents, excluding any that are dated after June 1, 2005, and imports them into the result document.

Example: Excluding specific comments

```
<PDF result="doc1_comments.pdf">
  <Comments afterDate="20050601" filter="Exclude">
    <PDF source="doc1_fromTom.pdf"/>
    <PDF source="doc1_fromDick.pdf"/>
    <PDF source="doc1_fromHarry.pdf"/>
  </Comments>
  <PDF source="doc1.pdf"/>
</PDF>
```

When exporting comments, you can also use selection attributes on a Comments result element. For example, you could export the comments in the previous example to XFDF, as in this example.

Example: Exporting selected comments

```
<Comments result="doc8.xfdf" format="XFDF"
  afterDate="20050601" filter="Exclude">
  <PDF source="doc1_fromTom.pdf"/>
  <PDF source="doc1_fromDick.pdf"/>
  <PDF source="doc1_fromHarry.pdf"/>
</Comments>
```

If you want to specify different selection attributes for each source document, use separate Comments filter elements as children of the Comments result element, as shown in the following example.

Example: Using several comments filter elements

```
<Comments result="doc9.xfdf" format="XFDF">
  <PDF source="doc1_nocomment.pdf"/>
  <Comments byCategory="Notes">
    <PDF source="doc1_fromTom.pdf"/>
  </Comments>
  <Comments byCategory="DrawingMarkups">
    <PDF source="doc1_fromDick.pdf"/>
  </Comments>
  <Comments byCategory="TextEditingMarkups">
    <PDF source="doc1_fromHarry.pdf"/>
  </Comments>
</Comments>
```

There are some important things to note about this example:

The `Comments` result element must have at least one `PDF` source as a child element. It cannot have only `Comments` filter elements as children; otherwise, an error occurs.

The `Comments` filter elements select different types of comments from their respective source elements. The comments are aggregated and effectively imported into the source document `doc1_nocomment.pdf`, then exported as XFDF.

The source document `doc1_nocomments.pdf` is not returned to the user. Therefore, its page contents are ignored. But it must contain enough pages to include all the comments from the original documents. It must also contain no comments of its own originally.

### Working with links

Links in PDF documents are interactive elements that represent either a hypertext link to a destination in the same (or other) PDF document or an action to be performed. In PDF, links are a type of annotation.

You can use the `Links` element to specify the following results:

- Export of a representation of the links in a PDF document into XFDF.
- Import of links in XFDF format into a PDF document. These links can replace or add to the existing links in the document.
- Removal of the links in a document by using the `NoLinks` element.

This example extracts the links from `doc1.pdf` and returns the data to the client as an XFDF stream specified by the `Links` result element.

**Example: Exporting links as XFDF**

```xml
<Links result="links1.xfdf">
  <PDF source="doc1.pdf" />
</Links>
```

If a `Links` element has multiple `PDF` source elements as children. The children are effectively assembled into a single document from which the link information is exported, as in this example. This means that any links in `doc1.pdf` or `doc2.pdf` that reference the other document (that is, cross-document links) are rationalized in the result document. See "Rationalizing links" on page 78 for details.

**Example: Exporting links from multiple sources**

```xml
<Links result="links1.xfdf">
  <PDF source="doc1.pdf" />
  <PDF source="doc1a.pdf" />
</Links>
```

You can save `links1.xfdf` to use in a separate workflow or within the same DDX. In the following example, the result document `doc3.pdf` imports the links from `links1.xfdf`. Because the `PDF` and `Links` source elements are siblings, the links from both sources are aggregated and included in the result.

**Example: Adding links**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc2.pdf" />
  <Links source="links1.xfdf" />
</PDF>
```
In this example, the Links element is a child of the PDF source and therefore the links from links1.xfdf replace the links in doc2.pdf. The result is returned as doc3.pdf.

**Example: Replacing links**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc2.pdf">
    <Links source="links1.xfdf" />
  </PDF>
</PDF>
```

If you do not need the XFDF outside the current DDX document, you can skip the step of exporting links to XFDF and then importing them. Instead, you can use a Links filter element to effectively export and import within a single result element. A filter element is like a result in that its content is provided by its child elements. A filter element is also like a source in that it provides content to its parent. For more information, see "Filter elements" on page 19.

In the following example, the Links filter element can be thought of as containing the links from its child, doc1.pdf. Those links are then imported into the result document doc2.pdf.

**Example: Using a links filter element**

```xml
<PDF result="doc2.pdf">
  <Links>
    <PDF source="doc1.pdf"/>
  </Links>
</PDF>
```

**Removing links**

You use the NoLinks element to specify that links within a scope should not be included in the result. In the following example, the NoLinks element is in the scope of doc1.pdf. Therefore, any links in doc1.pdf are effectively removed during the assembly. Any links in doc2.pdf and doc3.pdf are included in the result.

**Example: Removing links**

```xml
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <NoLinks />
  </PDF>
  <PDF source="doc2.pdf">
    <PDF source="doc3.pdf"/>
  </PDF>
</PDF>
```

**Note:** You cannot specify both a Links and NoLinks element as siblings.

**Rationalizing links**

Links in PDF documents can reference other documents. Such links are called cross-document links. When the user clicks such a link in a viewer application such as Acrobat, the location in the other document is displayed.

When assembling documents, some source documents can have cross-document links to documents that are represented by other sources in the assembly. In this example, suppose that you have two documents called chapter1.pdf and chapter2.pdf and want to assemble them using the following DDX.
**Example: Assembling documents with cross-document links**

```
<PDF result="book.pdf">
  <PDF source="chapter1.pdf"/>
  <PDF source="chapter2.pdf"/>
</PDF>
```

Suppose further that chapter1.pdf contains one or more cross-document links to chapter2.pdf and chapter2.pdf contains one or more cross-document links to chapter1.pdf. In this case, have the links reference the destination in the result document, rather than pointing to an external document.

A difficulty with referencing the destinations in the resultant document is that the names of source elements in DDX are unrelated to the original filenames. If the Document Manipulation APIs encounters a link whose destination is a file called chapter2.pdf, it does not assume that this file is the same as the PDF source element chapter2.pdf.

To ensure the resultant document’s cross-documents links work, specify the source names that correspond to filenames that appear in cross-document links. To specify the source names, use the `LinkAlias` element as a child of a `PDF` source, as shown in this example.

**Example: Specifying link aliases**

```
<PDF result="book.pdf">
  <PDF source="chapter1.pdf" baseDocument="true">
    <LinkAlias>Chapter1</LinkAlias>
    <LinkAlias>chapter1.pdf</LinkAlias>
  </PDF>
  <PDF source="chapter2.pdf">
    <LinkAlias>Chapter2</LinkAlias>
    <LinkAlias>chapter2.pdf</LinkAlias>
  </PDF>
</PDF>
```

For example, any cross-document links in the base document to files named "Chapter2" or "chapter2.pdf" would resolve to the correct location in the result document.
Working with File Attachments

PDF documents can contain file attachments consisting of any type of data. The data in these attachments (also called embedded files) is separate from the page content and other document information. Any external file can be attached to a PDF document. Once attached, it can be extracted to an external file.

PDF supports two types of file attachments:

- **Document-level attachments** are associated with the document as a whole and are identified by name.
- **Page-level attachments** are associated with a particular page in a document and do not have names. These are also called file attachment annotations.

See the PDF Reference for more information.

You use the FileAttachments and NoFileAttachments elements to specify information about the file attachments in a document.

Preserving and deleting file attachments

By default, attachments from all source documents are preserved in the resultant document. File attachments are included from a source document only once, even if the source document is specified several times. That is, if the base document contains three file attachments, then the result PDF document contains three file attachments.

**Note:** Document-level file attachments are assembled from a non-base document when the entire PDF document is part of the assembly. If only some pages from a non-base document are assembled, then the document-level file attachments for that PDF are not included.

To exclude attachments, use the NoFileAttachments element. (See “NoFileAttachments” on page 205.)

**Example: Excluding attachments from a source document**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf">
    <NoFileAttachments/>
  </PDF>
  <PDF source="doc2.pdf"/>
</PDF>
```

The example above excludes attachments from doc1.pdf and includes attachments from doc2.pdf. The following example excludes all file attachments by making the NoFileAttachments element a child of the result element.
Example: Excluding all file attachments

```xml
<PDF result="doc3.pdf">
  <NoFileAttachments/>
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf"/>
</PDF>
```

Note: The `<NoFileAttachments>` element and the `<FileAttachments>` element cannot be siblings.

Attaching files to a PDF document

You can attach files to a resultant document using the `<FileAttachments>` source element. As mentioned earlier, there are two types of `<FileAttachments>` source elements: page-level and document-level.

Document-level file attachments

To attach a file at the document level, you use a `<FileAttachments>` source element and specify the following information:

- The `source` attribute specifies the document to attach.
- The `nameKey` attribute suggests a unique identifier for the document. Beginning with 9.0, the `nameKey` is deprecated and can be omitted.
- If you provide the `nameKey`, set its value to the filename of the attachment. The `nameKey` must be unique. If duplicate filenames are specified, then the filename and `nameKey` are made unique by appending a number to the root of the filename. For example, the second `data.pdf` added is identified as `data_0001.pdf`.
- The `<File>` subelement specifies the filename for the attachment and optionally the MIME type, creation date, and modification date. The filename must be unique.
- The `<FilenameEncoding>` subelement specifies the encoding for the filename.
- The `<Description>` subelement provides descriptive text.

This example attaches the file `data.pdf` to the resultant document.

```xml
<PDF result="doc2.pdf">
  <FileAttachments source="faData.pdf">
    <File filename="data.pdf" mimetype="application/pdf"/>
    <FilenameEncoding encoding="UTF-8"/>
    <Description>What this file does</Description>
  </FileAttachments>
  <PDF source="doc1.pdf"/>
</PDF>
```

In this example above, the `<FileAttachments>` source element is a sibling of the PDF source element. Therefore, all file attachments in the original document `doc1.pdf` are preserved. In the following example, `data.pdf` would be attached to the source (and hence the result), but all other attachments in `doc1.pdf` would be deleted.
Example: Replacing file attachments

```xml
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <FileAttachments source="data.pdf" ... additional attributes />
  </PDF>
</PDF>
</PDF>
```

Page-level file attachments

You can attach a file to any page of a PDF document as a file attachment annotation. To attach a page-level file attachment, specify a `FileAttachments` element as a child of a `PDF` element. (See “FileAttachments” on page 181.)

The syntax is similar to the syntax for document-level file attachments, with these exceptions:

- Page-level file attachments do not use the `nameKey` attribute.
- Page-level file attachments must have an additional subelement, `AttachmentAppearance`, that specifies the appearance of the annotation icon on the page that represents the file attachment. Beginning with version 9.0, the presence of this element is what distinguishes a page-level attachment from a document-level attachment.

The file is attached to the first page represented by the parent element. Therefore, to attach a file to any page other than the first page, use multiple `PDF` source elements, as in the following example.

Example: Attaching a file to a page

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf" pages="1-2"/>
  <PDF source="doc2.pdf" pages="3-last">
    <FileAttachments source="meetingnotes.txt">
      <File filename="meetingnotes.txt" mimetype="text/plain"/>
      <FilenameEncoding encoding="ISO-8859-1"/>
      <Description>comments from meeting</Description>
      <AttachmentAppearance icon="Paperclip" color="red" opacity="75"
        x="36.0" y="360.0" />
    </FileAttachments>
  </PDF>
</PDF>
```

This example creates a resultant document by assembling doc1.pdf and doc2.pdf. The goal is to attach the file meetingnotes.txt to page 3 of the second source document, doc2.pdf. (Assuming doc1.pdf has three pages, this file attachment appears on page 6 of the resulting doc3.pdf.) The doc2.pdf document is broken into two parts by using two appearances of the PDF source element. Each appearance uses the `pages` attribute to specify the pages in the part. The first part specifies pages 1-2, and the second part specifies pages 4-last. The `FileAttachments` element causes the file to be attached to the first page of the range 3-last; that is, page 3.

Extracting file attachments

You can use the `FileAttachments` result element to return one or more file attachments from source documents. Each file attachment is returned as a separate data stream, and the Document Manipulation APIs maps each stream to a unique name in the outputs map.
Information about the mapping between names and streams is returned in an XML document that also contains information about each returned file attachment.

This example returns all file attachments associated with the document doc3.pdf.

**Example: Extracting all file attachments**

```
<FileAttachments result="attachmentInfo.xml" nameKeys="*" extract="true">
  <PDF source="doc3.pdf"/>
  <FilenameEncoding encoding="ISO-8859-1"/>
</FileAttachments>
```

Note the following points about this example:

The `result` attribute specifies the destination for the resultant XML document. This XML file contains information about the file attachments. This document conforms to the FileAttachments schema. (See “FileAttachments” on page 181.)

All page-level file attachments in the PDF pages specified for the source document are returned; you cannot specify individual page-level file attachments to return. If you specify the `pages` attribute on the `PDF` source element, only file attachments on that range of pages would be returned. In the example, all page-level file attachments are returned.

The `nameKeys` attribute is a string specifying a list of document-level file attachments that are returned. It can be a single name or a comma-separated list such as "doc1, doc2, doc3". The value "*", as in the example above, requests that all document-level file attachments be returned. If you do not specify this attribute, no document-level file attachments are returned.

The `extract` attribute has a default value of true, so it is shown in the example only for convenience. If you specify a value of false, the file attachments are not returned to you as streams. Only the XML document is returned.

The `FilenameEncoding` element specifies an encoding to use to decode the filenames of the file attachments. You can specify more than one of these elements in case one of them is unknown to the client. (See “FilenameEncoding” on page 184.)

The following example extracts a single document-level file attachment. It provides several encodings that can be used to decode the stored filename. The result XML document is encoded with UTF-8.

**Example: Extracting a single file attachment**

```
<FileAttachments result="attachmentInfo.xml" nameKeys="data1" >
  <PDF source="doc3.pdf"/>
  <FilenameEncoding encoding="ISO-10646-UCS-2"/>
  <FilenameEncoding encoding="UTF-8"/>
  <FilenameEncoding encoding="ISO-8859-1"/>
</FileAttachments>
```

This example extracts multiple files attached to the PDF document.

**Example: Extracting multiple file attachments**

```
<FileAttachments result="attachmentInfo.xml" nameKeys="doc1,doc2,doc3" >
  <PDF source="doc3.pdf"/>
  <FilenameEncoding encoding="ISO-8859-1"/>
</FileAttachments>
```

In this example, page-level file attachments are extracted from pages 2-3 only. Default filename encoding is used to decode the stored filename.
Example: Extracting file attachments from specific pages

```xml
<FileAttachments result="attachmentInfo.xml">
  <PDF source="doc3.pdf" pages="2-3"/>
  <FilenameEncoding encoding="ISO-8859-1"/>
</FileAttachments>
```

Understanding filename encoding

The `FilenameEncoding` element specifies character encodings to use for encoding and decoding the names of files being attached or extracted. The `FilenameEncoding` element is optional. However, that element can be useful for processing documents that conform to PDF 1.6 or earlier. With such documents, the filenames are not stored as Unicode strings. In such cases the package files are document-level file attachments before the PDF to which they were attached became a PDF package. If the original host encoding is unknown, and if more than one encoding is provided, the first encoding that successfully decodes the bytes in the filename is used. However, there is no guarantee that the result is the expected result. (See “FilenameEncoding” on page 184.)
Adding Table of Contents or Blank Pages to an Assembly

Most of the pages in assembled documents originate from source documents. You can also add pages to your documents in the form of a table of contents (TOC) or blank pages.

Adding a table of contents

To create a table of contents, use a `TableOfContents` element. Only one `TableOfContents` element can exist in a resultant document.

The placement of the `TableOfContents` element relative to the PDF source documents determines where it is located in the resultant document. The `TableOfContents` element can be a child of a PDF result or a `PDFGroup` element. It can also appear as a child of a `StyleProfile` element.

The table of contents consists of entries that are derived from bookmarks in the resultant document. Each entry contains the bookmark title and the page that the bookmark links to. You can specify the number of levels of bookmarks to include in the table of contents.

**Note:** Only bookmarks that link to pages in the resultant document are included in the table of contents. Some bookmarks in a document can instead trigger actions such as running a script. (See “Working with Bookmarks and Thumbnails” on page 63.)

Also, a mixture of Simplified-Chinese, Traditional-Chinese, Japanese, or Korean text in PDF bookmarks within a given table of contents can result in illegible characters. To avoid problems with such a mixture, specify the font to use for a particular Asian text. Because you cannot indicate the preferred font or language within a given bookmark, mixed Asian languages within the same table of contents are not supported.

In the following example, the table of contents is inserted after Intro and before Chap1. By default, the table of contents includes the entries for all source elements after the `TableOfContents` element. It omits entries from source elements before the `TableOfContents` element. Therefore, the table of contents in this example does not include entries that represent the bookmarks in Intro.
Example: Creating a table of contents

```xml
<PDF result="doc5">
  <PDF source="Intro"/>
  <TableOfContents/>
  <PDF source="Chap1"/>
  <PDF source="Chap2"/>
  <PDF source="Summary"/>
</PDF>
```

The following example overrides the default behavior by setting the `includeInTOC` attribute explicitly. In this case, entries from Intro, Chap1, and Chap2 are included in the table of contents, and entries from Summary are not included.

Example: Including extra sources in a table of contents

```xml
<PDF result="doc5">
  <PDF source="Intro" includeInTOC="true"/>
  <TableOfContents/>
  <PDF source="Chap1"/>
  <PDF source="Chap2"/>
  <PDF source="Summary" includeInTOC="false"/>
</PDF>
```

By default, the table of contents includes only bookmarks at the top level of the outline hierarchy. You can include additional bookmarks by setting the `maxBookmarkLevel` attribute to one of the following values:

- A positive integer, which specifies the level of bookmarks to include. The default value is 1, which means that only the level 1 bookmarks are included.
- `infinite`, which means that all bookmarks are included in the table of contents.

The following example specifies that three levels of bookmarks appear in the table of contents.

Example: Specifying table of contents levels

```xml
<PDF result="doc5">
  <TableOfContents maxBookmarkLevel="3" createLiveLinks="false"
                  bookmarkTitle="Table of Contents" includeInTOC="true"/>
  <PDF source="Chap1"/>
  <PDF source="Chap2"/>
  <PDF source="Chap3"/>
</PDF>
```

This example also uses the following attributes:

- `createLiveLinks` specifies whether entries in the table of contents have PDF links associated with them. The default is `true` but can be set to `false`, such as if the resulting document is intended solely for print.
- `bookmarkTitle` specifies that a bookmark are created for the table of contents with the given title.
- `includeInTOC` specifies that the string specified for `bookmarkTitle` is used to generate a table of contents entry for the table of contents itself. Its default value is `true`; however, the entry in the table of contents is not generated unless `bookmarkTitle` is specified.
A `TableOfContents` element can also appear as a child of a `StyleProfile` element. (See “Using style profiles” on page 115.) The `StyleProfile` element lets you define a table of contents that multiple result elements reference via the `styleReference` attribute.

**Example: Using a style profile to create a table of contents**

```xml
<PDF result="finalDoc">
  <TableOfContents styleReference="myTOCStyle"/>
  <PDF source="Chap1"/>
  <PDF source="Chap2"/>
  <PDF source="Chap3"/>
</PDF>

<StyleProfile name="myTOCStyle"/>
  <TableOfContents maxBookmarkLevel="3" createLiveLinks="false"/>
  <!--Elements describing the table of contents-->
  </TableOfContents>
</StyleProfile>
```

**Formatting a table of contents**

You can format a table of contents in the following ways:

- **Add content and properties.** Specify elements such as `PageMargins`, `Header`, `Watermark`, and `PageLabels` as children of the `TableOfContents` element. The table of contents also inherits these elements if they are specified in a parent element.

- **Specify different properties and content for different pages.** For example, you can specify one set of properties for the first page of the table of contents and all other pages. To specify such properties, add one or two `TableOfContentsPagePattern` elements as children of the `TableOfContents` element. The `TableOfContentsPagePattern` element has a `pages` attribute that can be set to `1` or `2-last`. Any child elements that you specify for the `TableOfContentsPagePattern` element apply only to the pages specified.

- **Specify style information for line levels.** For example, you can specify the style to use depending on the bookmark level they apply to. Use the `TableOfContentsEntryPattern` element.

**Applying page properties and content to particular pages**

By default, page properties and content applied to the `TableOfContents` element (or its parents) apply to all pages of the table of contents. You can override this behavior by using the `TableOfContentsPagePattern` element. This element lets you specify the pages in the table of content that have the page properties or content.

In the following example, a header and footer are defined for the table of contents as a whole. However, the first page overrides the header and the remaining pages override the footer.

**Example: Formatting a table of contents**

```xml
<TableOfContents maxBookmarkLevel="3">
  <Header styleReference="alpha"/>
  <Footer styleReference="beta"/>
  <TableOfContentsPagePattern pages="1">
    <Header styleReference="gamma"/>
  </TableOfContentsPagePattern/>
  <TableOfContentsPagePattern pages="2-last"/>
```
Applying entry styles to specific line levels

By default, entry styles defined in the TableOfContentsEntryPattern element apply to the entry style for each level in the table of contents. You can override this behavior by using the applicableLevel attribute. Set this attribute to “1” for a style applied to the first level of entries. Set it to “2” for a style applied to the second level of entries.

Specify an entry style by adding a StyledText element as a child of the TableOfContentsEntryPattern element. This element can specify any of the styled text attributes of the Communications - Document Manipulation APIs (see “Specifying styled text” on page 111).

The following example shows a table of contents that uses two levels of bookmarks.

**Example: Formatting table of contents entries**

```xml
<TableOfContents maxBookmarkLevel="2">
  <TableOfContentsEntryPattern applicableLevel="1">
    <StyledText>
      <p font-family="MyriadPro" font-size="12pt">
        <_BookmarkTitle/><leader leader-pattern="dotted"/>
        <_BookmarkPageCitation/>
      </p>
    </StyledText>
  </TableOfContentsEntryPattern>

  <TableOfContentsEntryPattern applicableLevel="2">
    <StyledText>
      <p font-family="MyriadPro" font-size="10pt">
        Section <_BookmarkTitle/><leader leader-pattern="space"/>
        <_BookmarkPageCitation/>
      </p>
    </StyledText>
  </TableOfContentsEntryPattern>
</TableOfContents>
```

If you do not specify an entry pattern for a specific bookmark level, the pattern specified for the next higher level is used. In the example above, level 3 bookmarks use the same style as bookmarks that are specified for level 2. If you do not specify any TableOfContentsEntryPattern elements, a default style is used, which corresponds to the following example.

**Example: Default style for table of contents entries**

```xml
<StyledText><p>
  <_BookmarkTitle/>
  <leader leader-pattern="dotted"/>
  <_BookmarkPageCitation/></p>
</StyledText>
```

This style causes each entry to look like the following example:

Chapter 1............................................................3
Chapter 2...........................................................25
Adding blank pages

You can use the `BlankPage` element to add pages to your document. Such pages are blank in the sense that they begin with no text or graphics. However, as with all pages in an assembled document, you can add page content such as headers and watermarks. You can also set the page properties and page labels.

A typical use for adding a blank page is to make the number of pages in a chapter, section, or document even. As a result, the first page of the next section starts on an odd-numbered page. To force the next chapter, section, or document to start on an odd-numbered page, specify true for the value of the `forceEven` attribute. A blank page is added only if the number of pages up to that point in the resultant document is odd.

In the following example, each chapter is forced to contain an even number of pages before the next chapter is assembled into the resultant document.

**Example: Adding blank pages**

```xml
<PDF result="doc.pdf">
  <PDF source="Chap1.pdf"/>
  <BlankPage forceEven="true"/>
  <PDF source="Chap2.pdf"/>
  <BlankPage forceEven="true"/>
  <PDF source="Chap3.pdf"/>
  <BlankPage forceEven="true"/>
</PDF>
```

**Note:** The ordinal page number of a page in the resultant document determines whether that page is odd or even. Whether a page is odd or even is independent of the page's ordinal page number in a PDF source or PDFGroup element. It is also independent of the PageLabel element that applies to the page.

In the next example, a blank page is added only at the end of the resultant document, if necessary, to make the pages even. The page, if it exists, has a watermark specified by the `Watermark` element that is a child of the `BlankPage` element.

**Example: Adding a blank page with a watermark**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf"/>
  <BlankPage forceEven="true">
    <Watermark>
      <StyledText font-family="Helvetica" font-size="14">
        <p>This page intentionally left blank</p>
      </StyledText>
    </Watermark>
  </BlankPage>
</PDF>
```

As with other pages, pages that the `BlankPage` element adds inherit page properties and content that the parent elements specify. Page properties can also be specified through the `BlankPage` element. If a value for either `PageSize` or `PageRotation` is not specified within the scope of the `BlankPage` element, then the value is taken from other sources (in order of consideration):

1. Resultant PDF document page just before the blank page
2. Resultant PDF document page just after the blank page

If a page label is not specified within the scope of the blank page, the following result occurs:
   If the document contains no other page labels, the blank page has no page labels.
   If the document contains page labels, the blank page takes its labeling style from the previous page in
   the assembly. This behavior is equivalent this setting: `<PageLabel mode="Continue"/>`. If no
   pages are present before the blank page, the blank page takes a filler label that is equal to the ordinal
   page number.

See also

“Scope of elements that affect PDF or XDP properties” on page 24.
Some properties of PDF documents apply to the document as a whole rather than to individual pages. The Communications - Document Manipulation APIs provides options for determining how these properties are set.

By default, document-level properties are taken from the base source document (see “About base documents” on page 29).

Document properties also include security settings, which are discussed in “Working with Secured Documents” on page 129.

### Working with metadata

PDF documents contain metadata (information about the document) in an XML format called Extensible Metadata Platform (XMP). PDF metadata includes properties such as the title, author, and date created.

When assembling documents, the resultant document contains the metadata from the base source document.

**Example: Using metadata from the base document**

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf"/>
</PDF>
```

The metadata in doc1.pdf becomes the metadata for the resultant document doc3.pdf. However, you can modify the document metadata in several ways.

You can use DDX to export the metadata from a PDF document, as in the following example.

**Example: Exporting metadata**

```xml
<Metadata result="doc1.xmp">
  <PDF source="doc1.pdf"/>
</Metadata>
```

Saved metadata can be imported into a PDF document to replace the existing metadata, as in this example:

**Example: Importing saved metadata**

```xml
<PDF result="doc4.pdf">
  <Metadata source="doc1.xmp"/>
  <PDF source="doc2.pdf"/>
  <PDF source="doc3.pdf"/>
</PDF>
```

The metadata that was previously exported as doc1.xmp replaces any metadata that existed in doc2.pdf (the base document) and becomes the metadata for doc4.pdf.
Alternatively, you can combine the previous examples without having the XMP returned as a stream by specifying the return attribute:

**Example: Using a temporary metadata result**

```xml
<Metadata result="temp" return="false">
  <PDF source="doc1.pdf"/>
</Metadata>
<PDF result="doc4.pdf">
  <Metadata source="temp"/>
  <PDF source="doc2.pdf"/>
  <PDF source="doc3.pdf"/>
</PDF>
```

### Modifying metadata properties

You can modify individual metadata items in the result PDF with the Author, Title, Subject, and Keywords elements. In this example, doc1.pdf provides all the metadata for doc2.pdf, except for the Title property, which is set explicitly to "My Memoirs".

**Example: Setting a metadata property**

```xml
<PDF result="doc2.pdf">
  <Title value="My Memoirs"/>
  <PDF source="doc1.pdf"/>
</PDF>
```

In the next example, the Subject element is a sibling of the Metadata source element. The metadata from doc1.xmp replaces the metadata in doc3.pdf. Then the value provided in the Subject element overrides the Subject property.

**Example: Overriding a metadata property**

```xml
<PDF result="doc2.pdf">
  <Metadata source="doc1.xmp"/>
  <Subject value="politics"/>
  <PDF source="doc3.pdf"/>
</PDF>
```

The Author, Title, and Subject elements contain a single string attribute, value. The Keywords element contains Keyword subelements that each have a value attribute.

The Keywords element’s mode attribute lets you replace or amend keywords in the resultant document. A value of Set (the default) replaces the keywords with the specified set. A value of Append adds the keywords to the existing ones. In this example, the Keywords document property for Reference.pdf is set to "PDF, language". Any existing keywords are overwritten.

**Example: Specifying metadata keywords**

```xml
<PDF result="Reference.pdf">
  <Keywords>
    <Keyword value="PDF"/>
    <Keyword value="language"/>
  </Keywords>
  <PDF source="Reference.pdf"/>
</PDF>
```
Working with layers

Graphical content in PDF documents can be stored on different layers, which can be displayed or hidden under user control. Such content is also called optional content.

Each layer in a PDF document has a name. When assembling two or more PDF documents, you can distinguish between the layers that originated from different source documents. To standardize the layers in the resultant document, use the layerLabel attribute of PDF source elements and the mergeLayers attribute of the PDF result element.

**Label layers.** Layers in a PDF document can be grouped under a label name. Use the layerLabel attribute of a PDF source element to specify the name of a top-level label. This label is a heading for the layers of the source document. Labeling layers enables readers to distinguish between layers from different source documents.

**Merge layers.** The mergeLayers attribute of the PDF result element determines whether layers with the same name from different source documents are kept as separate layers in the resultant document. If mergeLayers is false (the default), layers from source documents are kept distinct in the resultant document. If mergeLayers is true, layers with the same name are merged into a single layer in the resultant document.

Setting the initial view

You can specify how a resultant document is viewed when it is opened in a viewer application such as Adobe Reader. This information is equivalent to the document’s Initial View properties, which can be set in Acrobat. It includes magnification level and other page properties. It also includes properties for viewing PDF packages or portfolios. (See “InitialViewProfile” on page 193.)

You specify the initial view for a document with an InitialViewProfile element, which must be a child of the root DDX element. The initialView attribute of a PDF result element references the InitialViewProfile element, as shown in the following example:

**Example: Setting the initial view**

```xml
<PDF result="doc3.pdf" initialView="demo">
  <PDF source="doc1.pdf" />
  <PDF source="doc2.pdf" />
</PDF>
<InitialViewProfile name="demo" show="BookmarksPanel"
  magnification="FitPage" openToPage="2"/>
```

Using document-level JavaScript

The JavaScript element specifies a document-level script to add to the resultant PDF document. When the PDF document is opened, all document-level scripts are executed. Use the NoJavaScripts element to omit JavaScript in the resultant document. (See “NoJavaScripts” on page 207.)

In the following example, a document level JavaScript named "onOpen" is added to the resultant document. Even if pdf1 contains a document level JavaScript named "onOpen", it is not included in the result. The input data stream associated with "js1" is included in resultDoc as the JavaScript named "onOpen" instead.
Example: Adding a document level JavaScript to the resultant document

```xml
<PDF result="resultDoc">
  <PDF source="pdf1"/>
  <JavaScript source="js1", name="onOpen"/>
</PDF>
```

In the next example, the resultant document contains only the JavaScript from pdf1, which is the base document. Any scripts contained in pdf2, pdf3, or pdf4 are excluded from the result. The NoJavaScripts element lets you exclude all JavaScript from the resultant document.

Example: Including document level JavaScript only from the base document

```xml
<PDF result="resultDoc">
  <PDF source="pdf1"/>
  <PDFGroup>
    <NoJavaScripts/>
    <PDF source="pdf2"/>
    <PDF source="pdf3"/>
    <PDF source="pdf4"/>
  </PDFGroup>
</PDF>
```
The Communications - Document Manipulation APIs allows you to set several properties of PDF pages, such as their size and rotation.

Applying page properties

The page property elements are PageSize, PageRotation, PageMargins, ArtBox, BleedBox, and TrimBox. You can specify these elements for different pages in an assembly by including them at the appropriate level. (See "Scope of elements that affect PDF or XDP properties" on page 24.)

Page property elements can be children of the following elements: PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, and BlankPage.

You can specify each page property element only once for a given scope, except when you use the alternation attribute to specify odd and even pages. This restriction exists to ensure that individual pages do not have conflicting properties. See "Odd and even pages" on page 25 for more details.

In addition to setting page properties, you can specify additional page content with page content elements. (See "Adding and Manipulating Page Content" on page 102.)

The next section explains the relationship between these elements.

Page size and rotation

You use the PageSize element to define the page size of a PDF page. In DDX, the term page size is interpreted in the following way:

From the user’s point of view, it is the dimensions of the visible page. If you use the Crop Pages command in Acrobat, you see the page size displayed as "Cropped page size". A standard letter-size page is 8.5 x 11 inches. If the page is rotated 90°, the page size is 11 x 8.5 inches.

In the PDF file, these dimensions correspond to the intersection of the /MediaBox and /CropBox entries in the page dictionary after applying the /Rotate entry. (See the PDF Reference for details.)

The page size corresponds to what is returned when using the DocumentInformation query element to obtain information about the document. The XML that is returned includes a PageSize element. (See "Getting document information" on page 133.)

There is an interaction between the PageSize and PageRotation elements. See "Rotation and orientation" on page 97 for more information.

The width and height attributes of the PageSize element specify the dimensions, which default to the standard letter size of 8.5 x 11 inches. These attributes can be specified in inches, millimeters, centimeters, or points. (See "Specifying length" on page 26 for more information.) The default values are for letter size: "612pt" (8.5 inches) for width and "792pt" (11 inches) for height.

In the following example, suppose that doc2 contains letter size pages (8.5 x 11 inches) and doc1 contains legal size pages (8.5 x 14 inches). To make all the pages the same size, you can specify legal size for the pages in doc2.
Example: Setting the page size

```xml
<PDF result="newdoc">
  <PDF source="doc1"/>
  <PDF source="doc2">
    <PageSize width="8.5in" height="14in"/>
  </PDF>
</PDF>
```

Changing page size

If the page size changes, you can specify whether the page contents are scaled and how the original page is anchored. The page size changes if the result page size is different from the source page size.

By default, page contents are not scaled. If the page size increases, white space is added around the original page. If the page size decreases, the page contents are cropped to the new page size (possibly resulting in some visible content being hidden).

To modify the default behavior, specify true for the attributes `scaleUp` and `scaleDown`. The `scaleUp` attribute applies when the page size increases in both dimensions. The `scaleDown` attribute applies when the page size decreases in at least one dimension.

If the new specified page is larger than the previous size in one dimension but smaller in the other and `scaleDown` is true, scaling is performed. `scaleUp` never results in content being cropped; cropping can occur only if the page size has been reduced and `scaleDown` is false.

Scaling never alters the aspect ratio of the page's contents. If performed, scaling is always the same in the horizontal and vertical dimensions. If the new page size has a different aspect ratio from the previous page size, the scale factor is the largest one that accommodates the page. In the other, non-critical dimension, either more of page's contents are made visible or white space is added.

The `horizontalAnchor` and `verticalAnchor` attributes determine where extra white space (or previously hidden content) appears if the page size increases. They also determine how cropping occurs if the page size decreases and scaling is not performed.

For example, if you specify a page size of 8.5 x 14 inches, any source pages that were 8.5 x 11 inches grows vertically. The default value for the `verticalAnchor` attribute is Middle. With this setting, the old page size is centered vertically compared to the new size and white space appears equally at the top and bottom. If you set the `verticalAnchor` attribute to Top or Bottom, the white space is added at the bottom or top, respectively. The same applies for `horizontalAnchor`, which can be set to Center (the default), Left, or Right.

By contrast, if the vertical dimension is decreased (say from 14 to 11) and `verticalAnchor` is Middle, the page is cropped equally at top and bottom. If `verticalAnchor` is Bottom, the top of the page is anchored to the new top and all cropping is done at the top. The following example and the figures below show this cropping behavior.

Example: Decreasing the page size

```xml
<PDF result="newdoc">
  <PageSize width="8.5in" height="11in"
     verticalAnchor="Bottom"/>
  <PDF source="doc1"/>
  <PDF source="doc2"/>
```
Rotation and orientation

You can use the `PageRotation` element to change the rotation angle of a page. The `rotate90` attribute specifies the rotation angle in increments of 90°, where a positive value is clockwise rotation and a negative value is counterclockwise. This rotation is absolute, not relative. That is, if the `rotate90` attribute is set to 90, then the page rotation is 90°. For example, if the original rotation angle was 90 and you specify 90 in the `PageRotation` element, there is no change.

The Document Manipulation APIs always assumes that the pages are set the way you want to view them; whether a rotation value is specified or not. The Document Manipulation APIs assumes that the pages are set the way you want to view them. For example, when you add a footer to a rotated page, the footer is at the bottom of the newly oriented page.

Example: Specifying a rotation angle

```xml
<PDF result="newdoc">
  <PDF source="doc1">
    <PageRotation rotate90="90"/>
    <Footer styleReference="myStyle"/>
  </PDF>
</PDF>
</PDF>
```
Changing page rotation affects page size. If you change PageRotation, the effective page size changes. However, if you change PageSize, the rotation does not change. The DocumentInformation query returns page rotation and page size.

For purposes of this discussion, the following terms are used:

A portrait page is one whose width is less than or equal to its height.

A landscape page is one whose width is greater than its height.

If you specify a page size for your document, it is possible that some of the pages have a different orientation from the specified one. To control what happens, set the select attribute of the PageSize element to one of the following values:

Auto: The orientation of each page is preserved if the page size is changed. If the specified orientation does not match the existing orientation of a page, the width and height values for PageSize are effectively swapped. For example, if the page size is 8.5 x 11 and the new page size is specified as 14 x 8.5, the effective page size becomes 8.5 x 14. As a result, the orientation is preserved.

Portrait: The new values apply only to pages that are already in portrait mode. For example, if the old page size is 11 x 8.5 (landscape) and the new page size is 8.5 x 14, the page size is not changed.

Landscape: The new values apply only to pages that are already in landscape mode. For example, if the old page size is 8.5 x 11 (portrait) and the new page size is 14 x 8.5, the page size is not changed.

All: The new values apply to all pages regardless of their previous orientation.

Interaction of page properties and content

It is important to understand the effect of specifying multiple page property or page content elements to a set of pages specified by a DDX scope. The elements are applied in the following order:

PageRotation: Specifies a rotation angle applied to the original page size of the source element. Including this element can change the page orientation from portrait to landscape or the reverse. (See "Rotation and orientation" on page 97.)
**PageSize**: Specifies the new page size. “Changing page size” on page 96 describes how this element is used.

**Transform**: A page content element that transforms the existing page content, including scaling, translation, and rotation (see “Transforming page content” on page 119). If PageRotation changed the orientation of the page, Transform rotates, scales, or translates the content relative to that new orientation.

**PageMargins**: Defines a border that affects the initial placement of page content elements such as headers and footers. (See “Page margins” on page 100.)

**Page content elements (for example Header and Footer)**: Applies new content to the pages. Elements that appear earlier in a DDX can modify the page content’s dimensions, orientation, and content margins. Applying page content elements ensure that the resultant document bears the same additional content. See “Adding and Manipulating Page Content” on page 102.)

---

**Prepress settings**

You can specify information for prepress production workflows by setting the ArtBox, BleedBox, and TrimBox elements. These elements correspond to the /ArtBox, /BleedBox, and /TrimBox entries in the page dictionary of a PDF file. (See section 10.10.1 in the PDF Reference for details.)

Unlike PDF, the prepress elements do not specify the exact coordinates of the boxes. Instead, their positions are specified as offsets (margins) from the edge of the visible page. The PageSize element specifies the dimensions of the visible page. (See “Page size and rotation” on page 95.)

These elements all have the same attributes:

- `left`, `top`, `right`, and `bottom` specify the distance from the edge of the page to the corresponding edge of the box. Their default values are `0pt`, meaning that the size of each box is the same as the page size. (See “Specifying length” on page 26 for details.)

- `alternation` specifies whether the settings apply to all pages in the current scope, odd pages, or even pages. (See “Odd and even pages” on page 25.)

This example shows the art box being set differently for odd and even pages. All offsets are set to 36 points (.5 inch) from the edge of the page. The exception is a 72 point (1 inch) offset for the left side of the art box on odd pages and the right side for even pages.

**Example: Setting the art box for alternating pages**

```xml
<PDF result="doc2">
  <PDF source="doc1">
    <ArtBox left="72pt" top="36pt" right="36pt" bottom="36pt" alternation="OddPages"/>
    <ArtBox left="36pt" top="36pt" right="72pt" bottom="36pt" alternation="EvenPages"/>
  </PDF>
</PDF>
```

Specifying a new value for the PageSize element does not adjust the settings of these elements. If these values are important to your workflow, set them explicitly.
Page margins

The PageMargins element lets you specify the initial placement of page content elements such as headers, watermarks, and tables of contents. (See “Adding and Manipulating Page Content” on page 102.)

The PageMargins element has four attributes, left, top, right, and bottom, which specify each margin as a distance from the edge of the page. The PageSize element specifies the page dimensions. The default margins are 36 points (.5 inches).

In this example, all the margins are set to values other than the default. The Header and Footer elements specify the header and footer. (See “Adding and removing headers and footers” on page 102.)

Example: Setting page margins

```xml
<PDF result="doc1">
  <PDF source="doc2">
    <PageMargins left="1in" top="0.75in" right="0.75in" bottom="1in"/>
    <Header padding="0.5in">
      <Center>
        <StyledText><p>Header content</p></StyledText>
      </Center>
    </Header>
    <Footer>
      <Center>
        <StyledText><p>Footer content</p></StyledText>
      </Center>
    </Footer>
  </PDF>
</PDF>
```

The page margins determine the placement of the header:

The top of the header is aligned with the top margin
The left side of the Left header is aligned with the left margin
The right side of the Right header is aligned with the right margin

Footers behave in a similar manner. Watermarks, backgrounds, and tables of contents also use the margins to determine where to place new content, as described in the individual sections on those elements.
This figure shows the placement of the margins, header, and footer defined in the example.

**Note:** The page margins determine where to place the text or graphics but do not provide a clipping boundary. For example, the figure shows a centered header. If the header has too many characters, it can overflow both the left and right margins. Similarly, a left-justified header can overflow the right margin.

You can specify `PageMargins` only once within a given scope. The exception is when you use the `alternation` attribute and specify both `OddPages` and `EvenPages` as values. (See "Odd and even pages" on page 25.)
14 Adding and Manipulating Page Content

You can add and remove several types of content to the pages of assembled PDF documents:

*Headers* and *footers* consist of text or graphics that appear at the top or bottom of a page. Typically, they contain information such as the date, page number, and document title.

*Watermarks* and *backgrounds* consist of text or graphics that can appear anywhere on a page. Watermarks can appear on top of the existing content, and backgrounds can appear in back of the existing content.

*Overlays* and *underlays* consist of entire PDF pages that are placed on top of the existing content (overlays) or in back of the existing content (underlays).

Page content, added in a manner similar to a watermark or background, which can be made screen readable for structured (tagged) PDF documents.

For the Communications Document Manipulation APIs, reapplying a page content element to a PDF file causes the file to grow in size. This growth occurs even if you set the PDF element’s save attribute to Full, as shown in the example below. The page content elements include the Header, Footer, Watermark, Background, PageContent, PageOverlay, and PageUnderlay elements.

The reason the file grows is that the Header, for example, references an optional content group (OCG) in the PDF document. Because different PDF objects can reference the same OCG, it is assumed that other PDF objects must be able to access the OCG associated with the Header. For this reason, reapplying the Header adds a new OCG associated with the new Header. It does not remove the associated OCG.

For example, the following DDX causes the file to grow in size each time it is applied to the file:

```xml
<PDF result="result.pdf" save="Full">
  <PDF source="input.pdf" />
  <Header> <Center>
    <StyledText>
      <p>For review</p>
    </StyledText> </Center>
  </Header>
</PDF>
```

**Adding and removing headers and footers**

You can specify headers and footers that appear in your resultant documents.
Adding headers and footers

The Header and Footer elements specify page content that appears at the top and bottom of the page, respectively. Headers and footers are located on the page as follows:

- **Outer margins.** The PageMargins element specifies the outer margins for the given page. (See “Page margins” on page 100.) For headers, the left, top, and right margins apply. For footers, the left, bottom, and right margins apply. The default margins are 36 points (.5 inches).

- **Padding between the header or footer and the content.** The Header or Footer element’s padding attribute specifies the distance between the header or footer and the body content.

- **Background color.** The Header or Footer element’s backgroundColor attribute specifies the color to use for filling the background area for the header or footer area.

- **Placement of text.** The Header or Footer element’s margin attribute provides additional control over placement of text.

Header and Footer elements have the child elements Left, Center and Right, which specify an anchor for placement of content within the header or footer:

- **Left** supplies content that is justified to the left margin.
- **Center** supplies content that is centered on the page.
- **Right** supplies content that is justified to the right margin.

**Note:** The margins and Header or Footer margins determine where to place the text or graphics. These margins do not provide a clipping boundary. It cannot be assumed that a DDX processor can wrap the text. For example, a left-justified header can overflow the right margin if it is too wide.

Header and Footer elements can be children of the following elements, which correspond to the pages on which they can appear:

- **PDF or PDFGroup elements** (source pages)
- **TableOfContents or TableOfContentsPagePattern elements** (table of contents pages)
- **BlankPage elements** (blank pages)

You can also specify Header and Footer elements within named StyleProfile elements. Parent elements can then reference those StyleProfiles by name. (See “Using style profiles” on page 115.)

You can retain pre-existing headers and footers by setting the replaceExisting attribute to false.

**Note:** The Document Manipulation APIs can remove only page content added with Acrobat 8 or earlier. It cannot remove watermarks, backgrounds, headers, and footers added with later versions. Acrobat 9 and later does not distinguish between watermarks, backgrounds, headers, and footers. Also, the Document Manipulation APIs does not remove page content that contains Bates numbers.

You can specify a different Header or Footer for even and odd pages within a given scope (see “Odd and even pages” on page 25).

If Header or Footer are not explicitly specified for the current scope but are specified for a parent scope, the parent Header or Footer apply to the current scope.

You can specify the contents of the header and footer fields in one of two ways:

- By using styled text specified by a StyledText element (see “Specifying styled text” on page 111)
- By using a page from a PDF document (specified by a PDF source element) as a graphic. The first page from the pages specified by the PDF source is used.
The next figure shows the elements of a header and footer including the padding below the header, and padding above the foot and the margins. The following example shows the corresponding DDX.

**Example: Adding headers and a footer**

```xml
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <Header padding=".3in">
      <Right>
        <StyledText><p>Right-justified header</p></StyledText>
      </Right>
      <Left>
        <StyledText><p>Left-justified header</p></StyledText>
      </Left>
    </Header>
    <Footer padding=".25in">
      <Center>
        <StyledText><p>A centered footer</p></StyledText>
      </Center>
    </Footer>
  </PDF>
</PDF>
```

Here is an alternative way to specify the same information:

**Example: Specifying headers and footers with the styleReference attribute**

```xml
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <Header styleReference="myProfile"/>
    <Footer styleReference="myProfile"/>
  </PDF>
</PDF>
```
In the example, the Header and Footer elements have a styleReference attribute that references a style profile. Style profiles (specified by the StyleProfile element) can contain information about headers, footers, watermarks, backgrounds, tables of contents, or date patterns. See “Using style profiles” on page 115 for more information.

Note: In terms of scope, Header and Footer elements that reference definitions within StyleProfile elements are treated as if the definition appeared directly inline.

The next example shows different headers specified for odd and even pages and also specifies different page margins for odd and even pages. In addition, the Header elements specify true for the shrinkContentToFit attribute. This setting means that the content of the page is reduced in size to fit between the header and the bottom of the page. The default value is false, which means the header could possibly overlap some of the content at the top of the page.

Example: Specifying headers and footers for alternating pages

```xml
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <PageMargins left="1in" top="0.5in" right="0.5in" bottom="0.5in" alternation="OddPages"/>
    <PageMargins left="0.5in" top="0.5in" right="1in" bottom="0.5in" alternation="EvenPages"/>
    <Header alternation="EvenPages" shrinkContentToFit="true">
      <Center> <StyledText><p>Confidential</p></StyledText></Center>
      <Left><StyledText><p>Draft</p></StyledText></Left>
    </Header>
    <Header alternation="OddPages" shrinkContentToFit="true">
      <Center> <StyledText><p>Confidential</p></StyledText></Center>
      <Right><StyledText><p>Draft</p></StyledText></Right>
    </Header>
  </PDF>
</PDF>
```

Removing headers and footers

You can use the NoHeaders and NoFooters elements to remove headers and footers from the pages in a given scope. You can specify the alternation attribute for either of these elements, as described in “Odd and even pages” on page 25.
**Note:** You cannot specify both Header and NoHeaders or both Footer and NoFooters in the same scope.

This example removes headers from the document. Any headers that were added to either of the source documents previously, for example by Acrobat or the Document Manipulation APIs, are removed.

**Example: Removing headers**

```xml
<PDF result="headlessDoc.pdf">
  <NoHeaders/>
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf"/>
</PDF>
```

In this example, a header is specified for the resultant document. The presence of NoHeaders in a lower scope means that the header does not apply to the pages of doc2.pdf.

**Example: Removing headers from specific sources**

```xml
<PDF result="assembledDoc.pdf">
  <Header styleReference="general"/>
  <PDF source="doc1.pdf">
    <PDF source="doc2.pdf">
      <NoHeaders/>
    </PDF>
    <PDF source="doc3.pdf"/>
  </PDF>
</PDF>
```

**Note:** The NoWatermarks and NoBackgrounds elements are used in the same way for watermarks and backgrounds.

### Adding and removing watermarks and backgrounds

Like headers and footers, watermarks and backgrounds specify text or graphics to add to the existing page content. Watermarks are placed on top of the existing page content and are specified with the Watermark element. Backgrounds are placed behind the existing page content and are specified with the Background element.

Watermarks and backgrounds are similar to headers and footers in these ways:

- You can retain pre-existing watermarks and backgrounds by setting the replaceExisting attribute to false.

**Note:** You can specify a different watermark or background for even and odd pages within a given scope (see “Odd and even pages” on page 25).

A Watermark or Background element can specify its content either as direct child elements or with the styleReference attribute. The styleReference attribute references a StyleProfile element that can contain a watermark or background description.

The content of a watermark or background can be either a StyledText element or a single PDF source element. When using a PDF source, the first page in the pages specified for the source is used as the watermark or background.

If Watermark or Background are not explicitly specified for the current scope but are specified for a parent scope, the parent Watermark or Background apply to the current scope.
You can use the NoWatermarks and NoBackgrounds elements to ensure that a group of pages contains no watermarks or backgrounds.

Watermarks and backgrounds differ from headers and footers in that they are not restricted to specific areas of the page. You can specify the placement of a watermark or background using several attributes. You can also specify rotation and transparency.

Watermark and Background elements can be children of the following elements, which correspond to the pages on which they can appear:

- **PDF** or **PDFGroup** elements (source pages)
- **TableOfContents** or **TableOfContentsPagePattern** elements (table of contents pages)
- **BlankPage** elements (blank pages)

You can specify a different Watermark or Background for even and odd pages within a given scope. (See “Odd and even pages” on page 25.)

**Note:** The Document Manipulation APIs supports only unfiltered data, ASCIIHex filters, and ASCII85 filters for inline images in watermarks and backgrounds. All other filters in inline images are unsupported in these operations.

Watermarks and backgrounds (as with all other page content) follow scoping rules. (See “Scope of elements that affect PDF or XDP properties” on page 24.) In the following example, the Watermark element is a child of the PDF result element. It specifies a watermark that applies to all other children of the PDF result element that do not specify a watermark. Therefore the watermark applies to all the pages in doc1 and doc2.

**Example: Adding a watermark**

```xml
<PDF result="myPDF">
  <Watermark rotation="45">
    <StyledText><p><b>Draft</b></p></StyledText>
  </Watermark>
  <PDF source="doc1"/>
  <PDF source="doc2"/>
</PDF>
```

In the following example, the watermark specified for the PDF result still applies to the pages from doc1. However, the pages from doc2 have a different watermark, supplied by a PDF page (the second page of myWatermark). The pages of doc 3 have no watermark because of the presence of the NoWatermarks element.

**Example: Adding a watermark from a PDF page**

```xml
<PDF result="myPDF">
  <Watermark rotation="45">
    <StyledText><p><b>Draft</b></p></StyledText>
  </Watermark>
  <PDF source="doc1"/>
  <PDF source="doc2">
    <Watermark>
      <PDF source="myWatermark" pages="2"/>
    </Watermark>
  </PDF/>
  <PDF source="doc3">
```
Backgrounds follow the same rules as watermarks, including the ability to use the `NoBackgrounds` element.

### Adding page content

Page content can be added in a manner similar to adding a watermark or background. The difference between adding page content and adding a watermark or background is:

1. If the PDF document is already tagged (structured), text can be supplied to specify the text read by a screen reader. Adding untagged pages to a base-document which is a structured PDF document, does not result in their being tagged. Even adding a `PageContent` element to such a document does not result in the added page being tagged. This behavior also applies when adding page content by using the `TableOfContents` or `BlankPage` elements. (See “PageContent” on page 223.)

2. The page content added in this manner is not removable by other DDX elements.

3. Multiple `PageContent` elements can be specified per page.

It is recommended that the alternate text match the text within the page content element or accurately describe the graphic within the page content. In the following example, `doc1` is a tagged PDF document:

**Example: Adding page content for screen reading**

```xml
<PDF result="doc2">
  <PageContent appears="Behind" alternateText="This is highly Adobe Confidential.">
    <StyledText><p><This is highly <graphic source="AdobeConfLogo.pdf"/></p></StyledText>
  </PageContent>
</PDF>
```

### Overlaying and underlaying pages

Overlays and underlays specify PDF page content that is added to pages in a document. They differ from watermarks and backgrounds in the following ways:

- The content must come from PDF pages and cannot be specified using styled text.
- Overlay and underlay are not removable by other DDX elements.
- You can add multiple overlays or underlays to a page or pages.

You can use the `PageOverlay` and `PageUnderlay` elements to add page content over or under the existing content, respectively. You specify them as child elements of the `PDF` elements identifying the destination pages.

**Note:** The Document Manipulation APIs supports only unfiltered data, ASCIIHex filters, and ASCII85 filters for inline images on overlay pages. All other filters in inline images are unsupported in these operations.
Note: When overlaying PDF documents whose versions are 1.3 or earlier by using the PageOverlay element, the PDF document is changed to version 1.4.

This example overlays page 5 from doc2.pdf onto all but the first page of doc1.pdf.

Example: Overlaying pages

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf" pages="2-last">
    <PageOverlay>
      <PDF source="doc2.pdf" pages="5"/>
    </PageOverlay>
  </PDF>
</PDF>
```

In this example, the parent of the PageOverlay element is a PDF source element that specifies the pages with the overlay. The child of the PageOverlay element specifies the document from which the overlay comes. In the example, it specifies a single page (5). If it does not specify a single page, the first page of the range is used.

Items such as bookmarks, page labels, document structure, and document-level JavaScript code are not copied from overlays or underlays to their destination document. Any page properties that conflict with the destination page are ignored. Form fields and annotations can be copied by specifying a value of true for the embedFormsAndAnnots attribute (the default is false). In this case, form-level JavaScript code associated with forms on the page is included with the copied form.

The source (overlay or underlay) page is placed relative to the destination page so that the user space origin (lower left corner) of both pages coincide. If the source page size is smaller than the destination page size, no content is removed. In effect, the page size for the source overlay page is made larger.

Understanding rendering order

The PageContent, PageOverlay, and PageUnderlay elements add page content to the resultant document. The content they add is indistinct from the other page content. That is, the added content cannot be removed from the resultant document.

The order in which watermarks and backgrounds are added affects rendering order. The Document Manipulation APIs determines the rendering order of the content in a resultant document. The result element does not determine rendering order.

Within a PDF result element, the following page content elements add page content over the existing page content:

- PageContent with the appears attribute set to OnTop
- PageOverlay
- Watermark

The following page content elements add page content under the existing page content:

- PageContent with the appears attribute set to Behind (the default value)
- PageUnderlay
- Background
For PageContent, PageOverlay, and Watermark elements in the same result block, the Document Manipulation APIs overlays page content in the following order:

1. Watermark
2. PageContent
3. PageOverlay

The PageOverlay appears over the PageContent, which appears over the Watermark. The order in which these elements appear in the result block have no bearing on their overlay order.

Similarly, for PageContent, PageUnderlay and Background in the same result block, the Document Manipulation APIs underlays page content in the following order:

1. Background
2. PageContent
3. PageUnderlay

The PageUnderlay appears behind or underneath the PageContent, which appears behind the Background.

Adding a Watermark over a PageOverlay or a Background under a PageUnderlay requires two result blocks. To add a Watermark over a PageOverlay, perform these steps:

1. Create a transient result block that applies the PageOverlay. (A result block is transient if its return attribute is false.)
2. In another result block, add a source element that specifies the transient result block. Within that result block, apply the Watermark or Background.

Understanding blending color spaces

Adding a page content element to a PDF document can change the appearance of other page content under certain conditions. The page content elements include the elements PageContent, Watermark, Background, PageOverlay, and PageUnderlay.

If the assembled page content elements or the target page in the document has the following characteristics, it changes the appearance of other page content:

- Has opacity of less than 100%
- Contains RGB content
- Added to a page in a document that does not have a blending color space specified on each PDF page

Source documents or page content that use RGB colors in the added content are distorted in the resultant document. The distortion occurs because RGB colors are converted to CMYK, which is the default blending color space. Colors within added content are always in the RGB color space, even if all requested colors are black or gray. A color shift also occurs if an explicit CMYK or ICC blending color space is specified. DDX processors assume that such a conversion and subsequent color shift is intended and no warning is issued.

To prevent color distortion from occurring, add one of the following elements to the DDX:
Add an RGB or RGB-compatible ICC blending color space to the original PDF document. (Use Acrobat to add such color spaces to the PDF document. The Document Manipulation APIs cannot add blending color spaces to a resultant document.)

Set the opacity in the DDX to 100% (1.0).

**Note:** To change blending profiles in Acrobat, select Advanced > Print Production > Convert Colors > Page-Level Transparency Blending Space.

For example, the following DDX file adds a watermark with opacity of 25% to the file *test.pdf*. The text in the resulting file, *result.pdf*, is darker in appearance than in the original. This darkening occurs because the `/ExtGState` is applied to the entire page instead of the watermark itself. Removing the watermark removes the `/ExtGState` and the text returns to its original lighter color:

**Example:** Adding a watermark with opacity less than 100%

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <PDF result="result.pdf">
    <PDF source="test.pdf"/>
    <Watermark rotation="45" opacity="25%">
      <StyledText><p font-size="72pt">Draft</p></StyledText>
    </Watermark>
  </PDF>
</DDX>
```

**Specifying styled text**

When you add page content elements or TableOfContentsEntryPattern elements to a PDF document, you can specify style information for the styled text. The page content elements include the `Header`, `Footer`, `Watermark`, `Background`, `PageContent`, `PageOverlay`, and `PageUnderlay` elements. To specify style information add the `StyledText` element,

which can be the child of a `Header`, `Footer`, `Watermark`, `Background`, `PageContent`, or `TableOfContentsEntryPattern` element.

The following example adds a watermark with the text "Draft", in bold, to all the pages in the resultant document.

**Example:** Adding a watermark using styled text

```xml
<PDF result="doc2">
  <Watermark>
    <StyledText>
      <p><b>Draft</b></p>
    </StyledText>
  </Watermark>
  <PDF source="doc1"/>
</PDF>
```

A `StyledText` element contains as children one or more `p` (paragraph) elements. The `p` element can contain any (zero or more) of the following items:
A text string

A built-in key (see “Built-in keys” on page 114) that generates a text string depending on the value of a system or document property.

The **b** (bold) element, as shown in the previous example.

The *i* (italic) element.

The **Space** element, which specifies a space between two styled text elements.

The **span** element, which contains inline text, to which formatting can be applied.

The **leader** element, which is used for table of contents entries and specifies a pattern used to fill a line.

Each of the *rich text* elements (**StyledText**, **p**, **b**, **i**, **span**, and **leader**) can have attributes that specify further information, as described in the next section.

**Note:** Remove any unnecessary white space, including line feed and tab characters, from DDX elements that contain text. Unnecessary white space can result in unexpected line feeds or spaces. The **p**, **b**, **i**, **span**, and **DatePattern** elements are used to display styled text. If only white space is required between two child elements, use one of the following:

- Nonbreaking space entity number &#160;
- Space element for the **DatePattern** element and rich text elements.

The &nbsp; entity reference is undefined.

### Style attributes

You can use the following attributes to specify attributes for the rich text elements which are based on rich text elements found in CSS. Most the attributes specify font information, which includes the name of the font, its size, style, and weight.

- **font-family**
- **font-weight**
- **font-size**
- **font-style**

If you do not specify font information, the Document Manipulation APIs uses default font characteristics, which correspond to these settings:

- **font-family**="Minion Pro"
- **font-weight**="12pt"
- **font-size**="normal"
- **font-style**="normal"

The following style attributes are also supported:

- **Color**
- **Text decoration** (for example, **strikethrough**)
- **Text margins, alignment, and indentation**
You can specify any of these attributes on any rich text element even if the attribute has no meaning for that element. All of the child elements inherit the attribute, even though the attribute applies only to certain elements.

Applying identifying labels

Bates numbering is a method of applying identifying labels to a batch of related documents. Consider, for example, legal documents associated with a court case. Each page in the document (or set of documents) is assigned a Bates number that uniquely identifies the page. The assigned Bates number also establishes each document’s relationship to other Bates numbered documents. A Bates number contains a sequentially incremented numeric value plus an optional prefix and suffix. The prefix + numeric + suffix is called a Bates pattern.

All result blocks within the `<DDX>` root element define the set of documents. The `start` attribute (if available) provides the first number in the sequence. If the `start` attribute is omitted, the DDX processor’s default value (1) provides the first number in the sequence. Any other patterns that are started without specifying a `start` value begin with a default value of 1. The `start` value for a given pattern is set once. Any further settings of the `start` value in subsequent `BatesNumber` elements in the DDX are ignored for that pattern.

The default and minimum number of digits for the numeric portion, `numberOfDigits` attribute, is 6. Therefore, the number 1 would appear as 000001.

The `BatesNumber` element can appear anywhere inside a `<Header>`, `<Footer>`, `<Watermark>`, `<Background>`, and `<TableOfContentsEntryPattern>`. The numeric portion of the number is incremented only once per page. If there are multiple uses per page, the numeric value on that page would remain unchanged.

**Note:** It is not recommended to place a `BatesNumber` within a `PageContent` element.

The `Header`, `Footer`, `Watermark`, or `Background` elements can be used to apply Bates numbers to the files in a PDF package or portfolio. If these elements specify identical Bates number patterns to sequential files in the package or portfolio, then the `start` value is ignored. The numeric portion increments sequentially from file to file.

**Note:** Bates numbers added by using the `Header`, `Footer`, `Watermark`, or `Background` elements cannot be removed with the `NoHeader`, `NoFooter`, `NoWatermark`, or `NoBackground` elements.

The following example applies an identifying label to pages within a document.

**Example:** Applying an identifying label to pages within a set of documents

```xml
<?xml version="1.0"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/"/>

  <PDF result="result1.pdf">
    <PDF source="input1.pdf"/>
    <Watermark verticalOffset="4.5in">
      <StyledText>
        <p><BatesNumber prefix="Ref ID = " start="1"/></p>
      </StyledText>
    </Watermark>
  </PDF>

  <PDF result="result2.pdf">
```
Notice there is one Bates pattern for the two result blocks so that the number sequence is carried through both results. That is, if the first document has 6 pages, the last Bates number is 6, and the stamp appears as Ref ID = 000006. The first page of the second document is numbered 7, and the stamp appears as Ref ID = 000007.

**Built-in keys**

You can use the Document Manipulation APIs built-in keys to insert variable information into headers, footers, watermarks, backgrounds, and TOC entries. DDX processors replace these keys with appropriate strings as.

You can use built-in keys in two ways:

- As element names within the styled text elements `p`, `span`, `i`, and `b` to represent variable text. For example, the `_Title` element in this example becomes the title of the resultant document.

```xml
<StyledText><p>_Title/></p></StyledText>
```

- As strings representing attribute values (or parts of values). The `bookmarkTitle` attribute of the `TableOfContents` or `PDF source` elements and the `prefix` attribute of the `PageLabel` element can use the following metadata keys:

  - `_Title`
  - `_SourceTitle`
  - `_Author`
  - `_SourceAuthor`
  - `_Subject`
  - `_SourceSubject`
  - `_Created`
  - `_Modified`

The `start` attribute of the `PageLabel` element can use the `_PageNumber` key only. For example, the following snippet generates a bookmark in the resultant document. The bookmark contains the name of the author of the source document.

```xml
<PDF source="doc1" bookmarkTitle="By _SourceAuthor"/>
```

If the author is "Adobe Systems Incorporated", the resulting bookmark is "By Adobe Systems Incorporated".

The following example uses two built-in keys as elements within a `StyledText` element to specify information in a header. The header shows the page number (which is different for each page) and the total number of pages (which is constant for the entire resultant document).
Example: Using built-in keys

```xml
<PDF result="doc3.pdf">
  <Header>
    <Center>
      <StyledText>
        <p>Page <_PageNumber/> of <_LastPageNumber/></p>
      </StyledText>
    </Center>
  </Header>
  <PDF source="doc1.pdf"/>
  <PDF source="doc2.pdf"/>
</PDF>
```

The resultant document contains a centered header on each page. For a resultant document with 30 pages, the headers are "Page 1 of 30", "Page 2 of 30", and so on.

 CREATED, _Modified, and _DateTime can have an optional styleReference attribute that references a DatePattern element defined within a StyleProfile element. See "Formatting dates" on page 117 for details.

This example uses two documents with the following metadata:

doc1.pdf: Title is History of Chocolate and Author is Charlie
doc2.pdf: Title is Chocolate Futures and Author is Willy

Example: Using built-in keys

```xml
<PDF result="doc3.pdf">
  <TableOfContents styleReference="myTOC"/>
  <PDF source="doc1.pdf" bookmarkTitle="Section 1: _SourceTitle"/>
  <PDF source="doc2.pdf" bookmarkTitle="Section 2: _SourceTitle"/>
  <Watermark>
    <StyledText>
      <p>This was created by <_Author/>.</p>
    </StyledText>
  </Watermark>
</PDF>
```

This example would result in the following table of content entries for doc3.pdf:

Section 1: History of Chocolate ..... 1
Section 2: Chocolate Futures .......200

Along with a watermark which contains the text "This was created by Charlie."

Using style profiles

A StyleProfile element can specify information about a package file, header, footer, watermark, background, table of contents, and date pattern. Other elements in a DDX document reference the named StyleProfile element to specify the corresponding characteristics.

Referencing StyleProfile element lets you create and maintain a set of named styles that can be used in a DDX document as needed. You can reference a style profile in multiple places in the same DDX document.
For example, to include the same header in two or more resultant documents, specify the following DDX elements:

Enclose a Header element describing the header within a StyleProfile element.
Set the name attribute of the StyleProfile element to an identifying name.
Use this name as the value of the styleReference attribute of all the Header elements to which you want to apply the style profile.

The following example places the same header in two different resultant documents.

**Example: Using a style profile in two resultant documents**

```
<PDF result="doc2.pdf">
  <PDF source="doc1.pdf">
    <Header styleReference="myProfile"/>
  </PDF>
</PDF>

<PDF result="doc4.pdf">
  <PDF source="doc3.pdf">
    <Header styleReference="myProfile"/>
  </PDF>
</PDF>
<StyleProfile name="myProfile">
  <Header>
    <Left> <!--styled text--></Left>
    <Center> <!--styled text--></Center>
    <Right> <!--styled text--></Right>
  </Header>
</StyleProfile>
```

Each **StyleProfile** can contain the following elements: Header, Footer, Watermark, Background, TableOfContents, or DatePattern. There can be at most one of each element. The exception is when distinguishing between odd and even pages, in which case there can be two (see “Odd and even pages” on page 25).

**Note:** In terms of scope, elements that reference definitions within **StyleProfile** elements are treated as if the definition appeared directly inline.

With one exception, any of those elements appearing elsewhere in the DDX can use the styleReference attribute to reference the description in the style profile.

The exception is DatePattern, which cannot appear anywhere other than in a **StyleProfile** element. It formats dates specified by the built-in keys _Created, _Modified, and _DateTime. Those keys can reference a DatePattern element in a style profile with a styleReference attribute.

A DDX document can contain any number of **StyleProfile** elements as children of the DDX root element. The following example shows the use of two different profiles.

**Example: Using two style profiles**

```
<PDF result="doc2.pdf">
  <PDF source="cover.pdf" pages="1">
    <Header styleReference="cover"/>
    <Footer styleReference="cover"/>
  </PDF>
</PDF>
```

For an example of using StyleProfiles for Package definitions, see "Creating a PDF package" on page 52.

Formatting dates

The built-in keys _DateTime, _Created and _Modified can be used to specify dates within StyledText elements. These keys have an optional styleReference attribute that references a StyleProfile element. This StyleProfile element can contain as a child a DatePattern element that specifies formatting for the dates.
DatePattern can contain a number of child elements that represent the building blocks of a date string, along with text. The elements have mostly self-explanatory names such as Second, Minute, Hour, Year, and TimeZone. They take the current system time when the DDX document is processed and specify how to format it.

If only white space is required between two child elements, use one of the following:

- Nonbreaking space entity number &amp;#160;
- Space element for the DatePattern element and rich text elements

The default format in situations where DatePattern is not specified is equivalent to what is specified in this example:

**Example: The default date pattern**

```xml
<DatePattern>
  <Year/>-<MonthNumber01/>-<DayNumber01/>T<Hour01/>:
  <Minute00/>:<Second00/><UTCOffset/>
</DatePattern>
```

For example, Jan 3, 2006 at 12:01am PST would be formatted as

2006-01-03T:00:01-0700

The following example specifies a header for all the pages in a source document. The Left element of the header specifies a formatted date string with a DateTime built-in key inside a StyledText element. The DateTime element references the StyleProfile whose name attribute is "greendate".

**Example: Using a date pattern**

```xml
<PDF result="doc2.pdf">
  <PageLabel prefix="page "/>
  <PDF source="doc1.pdf">
    <Header>
      <Left>
        <StyledText>
          <p color="green" font-weight="bold">
            <_DateTime styleReference="greendate"/>
          </p>
        </StyledText>
      </Left>
      <Center><StyledText><p>Confidential</p></StyledText></Center>
      <Right>
        <StyledText><p color="red"><_PageLabel/></p></StyledText>
      </Right>
    </Header>
  </PDF>
</PDF>

<StyleProfile name="greendate">
  <DatePattern>
    <DayNumber01/>&amp;#160;<ShortMonthName/><Space/><Year/> at
    <Hour24/>:<Minute00/>
  </DatePattern>
</StyleProfile>
```

This DDX would result in a page header with this appearance:
Transforming page content

The Transform element allows you to alter the page content of existing pages in the following ways:

Scaling: You can make the contents of the page larger or smaller by using the scale attribute. This attribute specifies a nonnegative number that can be expressed as a decimal or percentage. A value less than 1 scales the contents down, and a value greater than 1 scales them up.

Rotation: You can rotate the page contents in increments of 90 degrees by using the rotate90 attribute.

Translation: You can move the page contents horizontally or vertically by using the newX and newY attributes. These values are length specifiers (see "Specifying length" on page 26).

The default for all of the attributes is to do no transformation on the page contents.

The Transform element is categorized as a page content element rather than a page property element because it affects existing page content. It has no effect on the properties of the page itself. For example, rotating the contents of the page does not change the orientation of the page (portrait or landscape).

Note: Be careful when using this element. It is possible to move part or all of the page content outside the visible portions of the page.

This example applies the following changes to the pages in the resultant document:

- Shrinks the page content of doc1.pdf to one-quarter size
- Moves the origin from the page to 50 points horizontal and 80 points vertical
- Rotates the content by 90 degrees

The content of doc2.pdf is unchanged.

Example: Transforming page content

```xml
<PDF result="doc3.pdf">
  <PDF source="doc1.pdf">
    <Transform
      scale="0.25" newX="50" newY="80" rotate90="90"/>
  </PDF>
  <PDF source="doc2.pdf"/>
</PDF>
```
This figure shows how the page looks before and after the Transform element is applied.
Specifying Page Labels

Acrobat and Adobe Reader identify pages in PDF documents with one or both of the following:

- The **ordinal page number**. PDF documents are numbered consecutively starting with 1.
- The **page label**. An optional identifier that has the following form:
  - `prefix + page number`
  - Both `prefix` and `page number` are optional.

The following items are valid page labels:

- iii
- page 3
- I-A
- Chapter 5 - 7

Typically, labels are used to identify sections such as chapters. However, there is no necessary relationship between the ordinal page number and the page label. When a page label is present, Acrobat displays both the page number and the label in a form such as "IV-3 (42 of 109)".

**Note:** The internal data structures in a PDF file use zero (0) to indicate the first page in the document. The Communications - Document Manipulation APIs and viewer applications use 1 to indicate the first page.

In addition to `PageLabels` element, you can also use other elements to add page labels to page content. For example, these elements let you add page labels: `Header`, `Footer`, `Watermark`, `Background`, or `TableOfContents`. To use these elements to add page labels, specify the label within a `StyledText` element. See “Specifying styled text” on page 111 for details.

### About page labels

To specify the page label to apply to a group of pages, use the `PageLabel` element. To remove page labels, use the `NoPageLabels` element (see “Removing page labels” on page 127).

The parent element of the `PageLabel` element determines the range of pages to which it applies. For the scope of any `PageLabel` element, the pages are numbered continuously, starting with 1 by default. Pages within a lower scope have their own independent numbering. (See “Scope of elements that affect PDF or XDP properties” on page 24.)

**Note:** You can specify `PageLabel` or `NoPageLabels` only once for any given scope.

This example shows a document with a preface and two chapters. The `PageLabel` element specifies that the page labels in the resultant document use the decimal numbering style. All source documents within this scope (Chapter1 and Chapter2) use this style. However, you can also specify other numbering conventions such as lower Roman numbering (i, ii, iii...). The attributes are described further below.

**Example: Specifying two page label styles**

```xml
<PDF result="doc4">
  <PageLabel format="Decimal"/>
</PDF>
```
Page labeling is optional. However, when assembling documents using the Document Manipulation APIs, if any of the source documents have page labels, then all pages in the resultant document have them.

When page labels are not explicitly specified for a range of pages, the Document Manipulation APIs uses this default behavior:

**Pages with existing page labels.** When assembling a PDF document from existing documents, explicit page labels in source documents are preserved. For example, a page labeled "Chap 2 - 3" in the source has the same label in the result regardless of the ordinal page number.

**Pages that lack a label.** Pages that have not been explicitly labeled are given a *filler* label which is the ordinal page number with no prefix.

**Table of content pages or blank pages.** You can add new pages using elements such as TableOfContents or BlankPage. The new pages take the labeling scheme of the preceding pages in the resultant document, if any. If the new page is the first page in the assembly, it is given the filler page label (the ordinal page number). This assignment assumes that any other pages in the document contain page labels.

To ensure accurate page labels, provide PageLabel elements for all pages. Providing labels for all pages avoids the default behavior which can produce unintended results.

The PageLabel element's mode attribute lets you specify whether page labels from source documents are preserved, added, or modified. That attribute supports the following values:

- **Define** (the default) means that the other attributes of the PageLabel element are used to define the characteristics of the page labels for these pages.
- **Preserve** means that existing page labels from source pages remain the same in the result. Labels are not changed regardless of the ordinal page number.
- **Continue** means that the pages use the page label style from the previous page in the document. The pages are renumbered as they are assembled. If the previous page had no defined page label style, the current pages also have no defined page label style.

**Note:** Continue is not valid for the first page in an assembly.

The remaining attributes of PageLabel apply only when mode is set to Define; otherwise, they are ignored.

The prefix attribute specifies the prefix, which is a string that precedes the page number. The string can contain built-in keys (see "Built-in keys on page 114").

The format attribute specifies the style of the page number portion of the label:

- **Decimal:** 1, 2, 3, ...
- **LowerRoman:** i, ii, iii, ...
- **UpperRoman:** I, II, III, ...
- **LowerAlpha:** a, b, c, ...
- **UpperAlpha:** A, B, C, ...
None: No page numbers are included. Only the prefix (if defined) appears; otherwise, there is a blank page label.

The **start** attribute specifies the starting page number in the resultant document for the group of pages to which the `PageLabel` element applies. All other pages are numbered consecutively beginning with **start**. It can be a positive integer or the `PageNumber` built-in key. By default, the starting page number is 1.

**Note:** Page labels need not be unique. For example, it is permissible to have two pages labeled "3" in a document.

### Specifying page labels

This example assembles three PDF documents that have two pages each. Because the `PageLabel` element is specified separately for each source document, numbering begins at "1" for each document. This behavior is equivalent to the **start** attribute being set to 1, its default value.

**Example: Specifying the page label separately for each source document**

```xml
<PDF result="doc4">
  <!-- Each source provides a 2-page document. -->
  <PDF source="doc1">
    <PageLabel prefix="Chapter 1 - "/>
  </PDF>
  <PDF source="doc2">
    <PageLabel prefix="Chapter 2 - "/>
  </PDF>
  <PDF source="doc3">
    <PageLabel prefix="Appendix A - "/>
  </PDF>
</PDF>
```

The resulting page numbers and labels are shown in the following table.

<table>
<thead>
<tr>
<th>Ordinal page number</th>
<th>Page label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chapter 1 - 1</td>
</tr>
<tr>
<td>2</td>
<td>Chapter 1 - 2</td>
</tr>
<tr>
<td>3</td>
<td>Chapter 2 - 1</td>
</tr>
<tr>
<td>4</td>
<td>Chapter 2 - 2</td>
</tr>
<tr>
<td>5</td>
<td>Appendix A - 1</td>
</tr>
<tr>
<td>6</td>
<td>Appendix A - 2</td>
</tr>
</tbody>
</table>

In the next example, the `PageLabel` element is a child of the `PDF` result element. Therefore, the page label style, with the prefix "page ", applies to the entire resultant document. The page number starts with 1 and increments throughout the document.

**Example: Applying a page label style to an entire document**

```xml
<PDF result="doc4">
  <PageLabel prefix="page ">
  </PageLabel>
</PDF>
```
The resulting page numbers and labels are shown in the following table.

<table>
<thead>
<tr>
<th>Ordinal page number</th>
<th>Page label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>page 1</td>
</tr>
<tr>
<td>2</td>
<td>page 2</td>
</tr>
<tr>
<td>3</td>
<td>page 3</td>
</tr>
<tr>
<td>4</td>
<td>page 4</td>
</tr>
<tr>
<td>5</td>
<td>page 5</td>
</tr>
<tr>
<td>6</td>
<td>page 6</td>
</tr>
</tbody>
</table>

This example produces the same result as the previous one. It specifies the page label style for the first source document. It also continues the doc1 page label style for the documents in the PDFGroup element.

**Example: Continuing a page label style**

```xml
<PDF result="doc4">
    <!-- Each source provides a 2-page document. -->
    <PDF source="doc1">
        <PageLabel prefix="page ">
    </PDF>
    <PDFGroup>
        <PageLabel mode="Continue"/>
        <PDF source="doc2"/>
        <PDF source="doc3"/>
    </PDFGroup>
</PDF>
```

The following examples illustrate several page labeling options using the same assembled documents. The source documents are a title page, two content sections, and an advertising supplement inserted between the two sections.
Some of the documents contain pre-existing page labels that are applied by using the `PageLabel` element.

<table>
<thead>
<tr>
<th>Document</th>
<th>Number of pages</th>
<th>PageLabel attributes (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>1</td>
<td>format=&quot;None&quot; prefix=&quot;Title page&quot;</td>
</tr>
<tr>
<td>FirstHalf</td>
<td>2</td>
<td>None</td>
</tr>
<tr>
<td>AdSection</td>
<td>2</td>
<td>format=&quot;Decimal&quot; prefix=&quot;Ad - &quot;</td>
</tr>
<tr>
<td>SecondHalf</td>
<td>2</td>
<td>None</td>
</tr>
</tbody>
</table>

In the first example, the goal is to number FirstHalf and SecondHalf consecutively and number AdSection independently. The following example shows how to achieve that goal. Here is how the example uses page labels elements:

**Global page label format.** The `PageLabel` element that is a child of the PDF result defines a global page label format. The `mode="Define"` expression applies the page label format to all pages in the resultant document unless another page label property is specified. Those pages start with "1" (by default) and are numbered consecutively, ignoring any intervening pages in a lower scope.

**Document-specific page label override.** The `PageLabel` elements for the Title and AdSection documents preserve the original page labels in those documents.

**Example: Overriding a global page label format**

```xml
<PDF result="doc1">
  <PageLabel format="Decimal" mode="Define"/>
  <PDF source="Title">
    <PageLabel mode="Preserve"/>
  </PDF>
  <PDF source="FirstHalf"/>
  <PDF source="AdSection">
    <PageLabel mode="Preserve"/>
  </PDF>
  <PDF source="SecondHalf"/>
</PDF>
```

<table>
<thead>
<tr>
<th>Ordinal page number</th>
<th>Page label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Title page</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Ad - 1</td>
</tr>
<tr>
<td>5</td>
<td>Ad - 2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>
The next example uses the same documents and also preserves the page labels for Title and AdSection. However, there is no global page label format; therefore, the pages in FirstHalf and SecondHalf are given the ordinal page number as the page label.

**Example: Inheriting the default page label**

```xml
<PDF result="doc1">
    <PDF source="Title">
        <PageLabel mode="Preserve"/>
    </PDF>
    <PDF source="FirstHalf"/>
    <PDF source="AdSection">
        <PageLabel mode="Preserve"/>
    </PDF>
    <PDF source="SecondHalf"/>
</PDF>
```

<table>
<thead>
<tr>
<th>Ordinal page number</th>
<th>Page label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Title page</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Ad - 1</td>
</tr>
<tr>
<td>5</td>
<td>Ad - 2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

More complicated page labeling can require using intermediate results. For example, to have labeling similar to the previous example but starting the numbering for FirstHalf with “1”, use the following DDX.

**Example: Using an intermediate result for page labels**

```xml
<PDF result="TempDoc" return="false">
    <PDF source="FirstHalf"/>
    <PDF source="AdSection"/>
    <PDF source="SecondHalf"/>
</PDF>
<PDF result="doc3">
    <PDF source="Title"/>
    <PDF source="TempDoc"/>
</PDF>
```
In TempDoc, FirstHalf and SecondHalf are given default page labels starting with "1". Therefore, those page labels are preserved when assembling doc3.

<table>
<thead>
<tr>
<th>Ordinal page number</th>
<th>Page label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Title Page</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Ad - 1</td>
</tr>
<tr>
<td>5</td>
<td>Ad - 2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Removing page labels

If you use the `NoPageLabels` element as a child of a `PDF` result element, the resultant document omits page labels.

**Example: Removing page labels**

```
<PDF result="doc1">
  <NoPageLabels/>
  <PDF source="doc2"/>
  <PDF source="doc3"/>
</PDF>
```

If you use the `NoPageLabels` element as a child of a `PDF` source element, pre-existing labels are removed. However, if any other pages in the document have page labels, pages in the scope are given the filler page label (the ordinal page number).

In the following example, the EndMatter section is specified as having no page labels. However, because other pages in the result have page labels, the pages in EndMatter are given the default filler page label (the ordinal page number).

**Example: Using the default page label**

```
<PDF result="book">
  <PDF source="Preface">
    <PageLabel format="LowerRoman"/>
  </PDF>
  <PDF source="Body">
    <PageLabel start="_PageNumber" format="Decimal"/>
  </PDF>
  <PDF source="EndMatter">
    <NoPageLabels/>
  </PDF>
</PDF>
```

To ensure that the page labels are blank (that is, the empty string), use the `PageLabel` element rather than the `NoPageLabels` element, as in this example.
Example: Specifying an empty page label

```xml
<PDF result="book">
  <PDF source="Preface">
    <PageLabel format="LowerRoman"/>
  </PDF>
  <PDF source="Body">
    <PageLabel start="_PageNumber" format="Decimal"/>
  </PDF>
  <PDF source="EndMatter">
    <PageLabel format="None"/>
  </PDF>
</PDF>
```
PDF documents have various levels of security. For example, they can be encrypted so that only users providing passwords can open them. They can also contain digital signatures attesting to the validity of the document.

### Specifying passwords

When creating a document using the Communications - Document Manipulation APIs, you can specify passwords that are required to access the document. PDF supports two types of passwords: a *master password* and an *open password*.

A master password (also called an *owner password* or *permissions password*) controls the ability to change the permissions on a document. When you specify a master password, you can specify permissions that restrict the operations users can perform. Restricted operations include the ability to print a document, change its content, and extract its contents.

When opening a document that is encrypted with a master password, the following rules are applied:

- Users who provide a master password are considered the owners of the document. Such users are unconstrained by the permission settings in the document. They can also change the permissions.
- Users can open the document without supplying a password if the document does not also have an open password. Such users can perform only those actions allowed by the permissions settings in the document. They cannot change the permissions.

An open password (also called a *user password* or *document open password*) controls the ability to open a document. If this password is set for a document, a user is required to provide the password to open the document:

- If the document does not also have a master password, no restrictions are imposed on the user’s operations.
- If the document has a master password and the users provide the open password, the permission settings in the document limit their actions.

In DDX, you specify password encryption for a document by setting the `encryption` attribute of the `PDF` result element. The value of this attribute must match the `name` attribute of a `PasswordEncryptionProfile` element, which provides the encryption information.

**Note:** You can also specify encryption when disassembling a document using the `PDFsFromBookmarks` element. See "Disassembling PDF Documents" on page 61.

In this example, the `PasswordEncryptionProfile` element whose `name` attribute is "userProtect" specifies an open password that is used to encrypt the `PDF` result `doc2`.

**Example: Encrypting a document with an open password**

```xml
<PDF result="doc2" encryption="userProtect">
    <PDF source="doc"/>
</PDF>
```
If you specify None for the encryption attribute of the PDF result element, the document is not encrypted. Whether the base source document is encrypted has no bearing on this behavior. If you do not specify a value for encryption, the result is encrypted with the same settings as the base document. This encryption behavior applies only for documents that are saved incrementally (see “Saving PDF documents” on page 34).

A PasswordEncryptionProfile element must be at the root of the DDX document; that is, it must be a child of the DDX element. A DDX document can contain any number of PasswordEncryptionProfile elements. More than one PDF result element can reference the same PasswordEncryptionProfile element.

The PasswordEncryptionProfile element has two additional attributes that you can set:

- **compatibilityLevel** specifies backward compatibility with previous PDF versions. A value of Acrobat3 uses 40-bit RC4 encryption. Acrobat5 and later use 128-bit RC4 encryption. Acrobat6 allows metadata to be unencrypted in an encrypted document and Acrobat7 allows file attachments only to be encrypted (see encryptionLevel below).
- **encryptionLevel** allows you to do selective encryption on the document and depends on the value of compatibilityLevel. The default value, All, means that the entire document is encrypted. NotMetadata means that the document metadata remains unencrypted while the rest of the document is encrypted. OnlyFileAttachments means that the file attachments are encrypted while the rest of the document is unencrypted.

You specify the permission settings associated with the document by using the Permissions element. Its child element MasterPassword specifies the master password. The attributes copy, edit, print, and screenReading specify different categories of permissions and are available only when compatibilityLevel is Acrobat5 or greater. (See “Permissions” on page 250.)

The following example specifies both an open and master password. The permissions indicate that the document cannot be printed and copying of content is not allowed. Users can still fill in forms and add comments and digital signatures.

**Example: Specifying an open and master password**

```xml
<PDF result="doc" encryption="limit">
  <PDF source="doc1"/>
</PDF>

<PasswordEncryptionProfile name="limit">
  <OpenPassword>opensesame</OpenPassword>
  <Permissions print="No" edit="CommentsFormFillinSign" copy="No">
    <MasterPassword>docmaster</MasterPassword>
  </Permissions>
</PasswordEncryptionProfile>
```

This example encrypts a document and sets permissions but leaves the document metadata unencrypted.

**Example: Encrypting a document and leaving metadata unencrypted**

```xml
<PDF result="doc2.pdf" encryption="limit">
  <PDF source="doc1.pdf"/>
</PDF>
```
<PasswordEncryptionProfile name="limit" compatibilityLevel="Acrobat6"
    encryptionLevel="NotMetadata">
  <Permissions print="No" edit="CommentsFormFillinSign" copy="No">
    <MasterPassword>letmein</MasterPassword>
  </Permissions>
</PasswordEncryptionProfile>

**Accessing a password-protected document**

When you assemble documents, it is possible that one or more of your source documents is encrypted with a password. The password must be specified for the Document Manipulation APIs to be able to decrypt the data and work with it. To specify the password, supply the following attributes in the DDX:

- Specify a **PasswordAccessProfile** element as a child of the **DDX** element. This element must have one child element **Password** whose contents are the password itself. It must also have a **name** attribute that is a unique identifying string that references the profile from elsewhere in the DDX.
- Use the **PDF source element's access attribute** to reference a **PasswordAccessProfile** element.

This example shows the use of a **PasswordAccessProfile** element.

**Example: Providing a password to access a document**

```xml
<PDF result="doc2.pdf" encryption="None">
  <PDF source="doc1.pdf" pages="2-last" access="deptA"/>
</PDF>
<PasswordAccessProfile name="deptA">
  <Password>iamcy4jn</Password>
</PasswordAccessProfile>
```

If the password specified by the **PasswordAccessProfile** element is the master password, the Document Manipulation APIs can perform any operation on the document. If the password is the open password, the Document Manipulation APIs can only perform operations allowed by the permissions specified when the document was encrypted.

Consider the situation where doc1.pdf does not permit page extraction and the profile provides only the open password. In this situation, the Document Manipulation APIs cannot assemble the resultant document because the PDF source element for doc1.pdf removes one of the pages. This action is not permitted. Even though the example creates a new unencrypted document, the permissions on the source document are enforced.

To ensure successful assembly of the resultant document, provide the master password for the following encrypted source documents:

- **Non-base document in a PDF source element**
- **Source document in a PDFsFromBookmarks element**

**Digital signatures**

Digital signatures can be used in PDF documents to authenticate the identity of a user and the document’s contents. A signature stores information about the signer and the state of the document when it was signed. Acrobat users sign PDF documents in **signature fields**, which are a type of form field.

Signatures can have several types:
A certifying or author signature enables the author of a document to attest to its contents. It also specifies the types of changes permitted for the document to remain certified. It must be the first signature in the document.

Regular or ordinary signatures enable signers to attest to the contents of a document but do not specify permitted changes. All signatures in a certified document other than the first one are ordinary signatures, as are all signatures in non-certified documents.

Usage rights signatures are created when usage rights are added to a document by a product such as the Reader Extensions service. These rights enable users to perform operations in Adobe Reader, such as filling in form fields, that are not normally permitted. Individual users cannot create usage rights signatures.

You cannot create digital signatures using the Document Manipulation APIs. However, when you assemble documents, some operations affect the digital signatures that are present in the source documents:

When specifying save="Full" for a PDF result, all signatures become invalid and certifying signatures from non-base documents are removed. (See "Saving PDF documents" on page 34.)

Signatures become invalid when disassembling documents using the PDFsFromBookmarks element. (See "Disassembling PDF Documents" on page 61.)

Signatures become invalid when flattening form fields using the NoForms element. (See "Flattening forms" on page 36.)

Certification can only be retained for the resultant document if the base document is certified. Certification in non-base documents is never retained.

You can specify certification="None" on the resultant document to indicate that the resultant document is not certified. If you do not specify the certification attribute, certification is retained for the result when the base document is certified, except in the following cases:

If you specify save="Full" or save="FastWebView", the resultant document is reorganized and therefore the certification becomes invalid.

If you specify the NoForms element, signature fields are flattened along with other form fields (see "Flattening forms" on page 36). Therefore, the certifying signature is removed.

You can remove usage rights from a document by specifying a value of None for the readerUsageRights attribute of the PDF result element.
Querying Documents

The Communications - Document Manipulation APIs provides the ability to obtain information about PDF documents and the DDX processor itself. These capabilities are independent of specifying the features of documents. To obtain this information, you use one of the following elements: DocumentInformation, DocumentText, and About.

Each of these elements returns information in XML format. Typically, you process the result of such files programmatically in order to use the results in the construction of another document.

Getting document information

You can use the DocumentInformation DDX element to return an XML file containing information about a PDF document. The PDF document can be a source document (one that was provided as an input stream) or one that was created by using a PDF result element. (See “DocumentInformation” on page 177.)

In this example, information about doc1 is returned as an XML stream.

Example: Getting document information

```xml
<DDX>
    <PDF result="doc1">
        <PDF source="doc2"/>
    </PDF>
    <DocumentInformation result="info.xml" source="doc1"/>
</DDX>
```

Note: The DocumentInformation element must appear after the PDF result element referenced by its source attribute because the result document does not exist until the element that describes it has been interpreted.

The XML stream (info.xml in the example) conforms to a schema specified in docinfo.xsd. Its namespace is http://ns.adobe.com/DDX/DocInfo/1.0

The data returned by the DocumentInformation element contains the following information about the document:

- Metadata from the PDF document properties: title, author, subject, keywords, date created, date modified, creator application, PDF producer
- PDF version
- Number of pages in the document
- Page sizes, rotation angles, and page labels for all pages in the document

Getting the text of a document

You can use the DocumentText DDX element to return an XML file containing the words in one or more PDF documents. The documents are specified as child elements of the DocumentText element, which can be one or more PDF source or PDFGroup elements.
In this example, the words from doc1 are listed in the XML stream words.xml.

**Example: Getting the text of a document**

```xml
<DDX>
  <PDF result="doc1">
    <PDF source="doc2"/>
  </PDF>
  <DocumentText result="words.xml">
    <PDF source="doc1"/>
  </Text>
</DDX>
```

The XML stream conforms to a schema specified in doctext.xsd. Its namespace is

http://ns.adobe.com/DDX/DocText/1.0

When more than one source document are specified, the pages are aggregated and the text is returned as if it were a single document. In this example, words.xml contains the words from a subset of pages from two documents.

**Example: Getting the words from pages in two documents**

```xml
<DDX>
  <DocumentText result="words.xml">
    <PDF source="doc1" pages="1-10"/>
    <PDF source="doc2" pages="3-5"/>
  </DocumentText>
</DDX>
```

The result document looks like this:

```xml
<DocText xmlns="http://ns.adobe.com/DDX/DocText/1.0/">
  <TextPerPage>
    <Page pageNumber="1">
      It a re, uterest abuspiostam, C. Axim il hortam intiam tervisq uemorum ommodii fecte in sedii consulvid autea vehebem orurnum is.
    </Page>
    <Page pageNumber="2">
      Sample Text Sample Text Sample Text Sample Text Sample Text Sample Text Sample Text Sample Text
    </Page>
  </TextPerPage>
</DocText>
```

**Getting information about the DDX processor**

You can use the `About` element to return an XML file containing information about the DDX processor.

**Example: Getting information about the DDX processor**

```xml
<DDX>
  <About result="AboutDDX.xml"/>
</DDX>
```

The result document looks like this:

```xml
<About xmlns="http://ns.adobe.com/DDX/AboutDDX/1.0/">
```
<Processor>Adobe® Assembler</Processor>
-Version>7.2</Version>
<Build>7.2.1087.0.107773</Build>
<Copyright>
  Copyright 2005-2006 Adobe Systems Incorporated. All Rights Reserved.
</Copyright>
</About>
Part II: DDX Reference

This section describes the syntax and semantics of the DDX grammar.
DDX Concepts

Each job submitted to the Communications - Document Manipulation APIs includes a Document Description XML (DDX) document and a set of source PDF and XML documents. The DDX document provides instructions on how to use the source documents to produce a set of result documents. The set of result documents usually includes one or more PDF documents. The set can also include XML documents derived from the source documents and documents attached to the source documents.

Note: The Document Manipulation APIs is an implementation of a DDX processor.

Element relationships and roles

A number of DDX elements have syntax that varies depending on the context in which they are used. These elements are identified using the following terminology:

- A result element represents data being created. Result elements include a result attribute (except for the PDFsFromBookmarks element, which has a prefix attribute instead).
  - Result elements have no initial content but accumulate the content of their child elements. Examples of result elements are the PDF result element and the FileAttachments result element.

- A source element represents content to contribute to the result. It must be a child of a result element and must have a source attribute with one exception. The PDF source and PackageFile elements can instead have a sourceMatch attribute. Examples of source elements are the PDF source element and the FileAttachments source element.

- A filter element is like a result element in that it takes its content from source elements nested within it. A filter element is also like a source element in that it must be a child of a result element. Filter elements lack the result and source attributes but contain child elements of a certain type. Examples of filter elements are the Comment filter element and the Links filter element.

When using DDX, it is recommended that you understand the meaning of relationships between elements, such as parent, child, and sibling. It is also recommended that you understand the scope of elements and the inheritance of attributes.

Attributes, child elements, and text content

This reference guide describes the information conveyed in each DDX element and provides the syntax for those elements’ attributes and attribute values.

Attribute names, formatting, and possible values

As shown in the following declaration, attribute names appear immediately after the element name. The possible values for the attribute are shown to the right of the equal sign. Clicking the attribute name takes you to a description of the attribute (although this feature is not implemented in this example).

```xml
<Comments
    filter="Exclude" or "Include"
    afterDate="unspecified" or "YYYYMMDD"
...```
The syntax in the DDX Reference uses the following conventions.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Exclude&quot;</td>
<td>Indicates the default value</td>
</tr>
<tr>
<td>&quot;Include&quot;</td>
<td>An alternative value for the attribute</td>
</tr>
<tr>
<td>&quot;YYYYMMDD&quot;</td>
<td>Italics indicates that a value must conform to the indicated type. The attribute description explains the restrictions on such values. The following list shows such attribute value types:</td>
</tr>
<tr>
<td></td>
<td>- xs:integer</td>
</tr>
<tr>
<td></td>
<td>- xs:string</td>
</tr>
<tr>
<td></td>
<td>- color</td>
</tr>
<tr>
<td></td>
<td>- File description</td>
</tr>
<tr>
<td></td>
<td>- page range</td>
</tr>
<tr>
<td>unspecified</td>
<td>Indicates that the attribute is omitted.</td>
</tr>
<tr>
<td>or</td>
<td>Indicates a descriptive comment.</td>
</tr>
</tbody>
</table>

**Child elements**

As shown in the following declaration, child elements appear after attributes. Clicking the element name takes you to a description of the element (although this feature is not implemented in this example).

```xml
<SomeElement
    filter="Exclude" or "Include"
    afterDate="unspecified" or "YYYYMMDD"
    ...
>
  <PDF source> and/or <PDFGroup> [1..n]
  <Comments filter> [0..n]
</SomeElement>
```

The syntax uses the following conventions.
As shown in the following declaration, elements can include text values.

```xml
<OpenPassword>
  password
</OpenPassword>
```

As with attributes that take text values, the syntax used for text content uses the following convention.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>A text value is provided.</td>
</tr>
</tbody>
</table>

**Element categories**

The names of DDX elements represent their content type. Elements can be placed into the following categories, which relate to their use as PDF building blocks:

- “Document assembly” on page 140
- “Document components” on page 141
- “Document disassembly” on page 143
- “Document properties” on page 143
- “Page labels” on page 144
- “Page properties” on page 144
Document assembly

Document assembly elements are used for “PDF assembly” on page 140, “PDF package or portfolio assembly” on page 140, and “XDP assembly” on page 141.

PDF assembly

The following elements are in this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlankPage</td>
<td>Specifies the addition of an initially blank page to the document.</td>
</tr>
<tr>
<td>PDF result</td>
<td>PDF document that is returned to the client.</td>
</tr>
<tr>
<td>PDF source</td>
<td>One or more PDF documents that are used to construct the parent result PDF document.</td>
</tr>
<tr>
<td>PDFGroup</td>
<td>Specifies a grouping of source documents to which page properties, page content, and document components can be applied.</td>
</tr>
<tr>
<td>TableOfContents</td>
<td>Specifies a table of contents in the result document.</td>
</tr>
<tr>
<td>TableOfContentsEntryPattern</td>
<td>Specifies the style to apply to table of contents entries.</td>
</tr>
<tr>
<td>TableOfContentsPagePattern</td>
<td>Defines the style for table of contents pages.</td>
</tr>
</tbody>
</table>

PDF package or portfolio assembly

The following elements are in this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorScheme</td>
<td>Color settings used in the PDF Portfolio.</td>
</tr>
<tr>
<td>DisplayOrder</td>
<td>Determines the order in which the viewer displays the fields of a PDF package.</td>
</tr>
<tr>
<td>Header (portfolio navigation pane)</td>
<td>Resource that supplies a header used in a PDF Portfolio.</td>
</tr>
<tr>
<td>Field</td>
<td>Describe the fields and their attributes, including the type of data and the name used to identify the field in a PDF package or portfolio.</td>
</tr>
<tr>
<td>FieldData</td>
<td>References metadata for a package file in a PDF package or portfolio.</td>
</tr>
</tbody>
</table>
The following elements are in this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>Identifies the parent element, PDF result or PDF source, as being a PDF package. A <code>&lt;PackageFiles</code> on page 215&gt; element can either add new package files or select package files for modification.</td>
</tr>
<tr>
<td>NoPackage</td>
<td>Assembles a PDF package or portfolio into a single PDF that omits the PDF package navigational features.</td>
</tr>
<tr>
<td>Navigator</td>
<td>Specifies a navigator to use for a PDF Portfolio.</td>
</tr>
<tr>
<td>PackageFiles</td>
<td>Adds new package files to the PDF package or portfolio being assembled.</td>
</tr>
<tr>
<td>NoPackageFiles</td>
<td>The package files in the package are not included, but the package specification remains and the parent element remains a PDF package.</td>
</tr>
<tr>
<td>Portfolio</td>
<td>Extends PDF packages with additional navigational features introduced with Adobe Acrobat 8.2.</td>
</tr>
<tr>
<td>NoPortfolio</td>
<td>Assembles a PDF Portfolio into a single PDF that omits the PDF Portfolio navigational features.</td>
</tr>
<tr>
<td>Schema</td>
<td>Defines the characteristics of the metadata (Field) attributes associated with each package file.</td>
</tr>
<tr>
<td>SortOrder</td>
<td>Determines the order of the package files when assembling a single result PDF. This element also indicates the priority the viewer uses when sorting fields.</td>
</tr>
<tr>
<td>Schema</td>
<td>Defines the metadata (Field) attributes that can be associated with each package file.</td>
</tr>
<tr>
<td>String</td>
<td>Adds entries to the String name tree in a navigator dictionary.</td>
</tr>
<tr>
<td>WelcomePage</td>
<td>Specifies the Welcome Page used in a PDF Portfolio.</td>
</tr>
</tbody>
</table>

**XDP assembly**

The following elements are in this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDP</td>
<td>Specifies an XDP document assembled from other XDP documents. Also packages an XDP document as a PDF document.</td>
</tr>
<tr>
<td>XDPContent</td>
<td>Specifies XDP content to insert into the XDP source or result being specified.</td>
</tr>
</tbody>
</table>

**Document components**

Document component elements represent parts of PDF documents that can be imported and exported but are not pages or page content. These elements fall into the subcategories of navigation, comments, and file attachments.
Navigational content

The elements in this subcategory represent navigational content, which can be imported into or exported from PDF documents. The following elements fall into this subcategory.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bookmarks</strong> result</td>
<td>Exports bookmarks from the child elements to an XML document.</td>
</tr>
<tr>
<td><strong>Bookmarks</strong> source</td>
<td>Specifies that bookmarks contained in an XML document are imported into the parent element.</td>
</tr>
<tr>
<td><strong>Bookmarks</strong> filter</td>
<td>Specifies bookmarks from child source elements that can be imported into the parent element but are not returned as XML.</td>
</tr>
<tr>
<td><strong>LinkAlias</strong></td>
<td>Provides an alternative name for the parent document, for use as a link destination.</td>
</tr>
<tr>
<td><strong>Links</strong> result</td>
<td>Specifies that links be exported from the child elements as an XML Forms Data Format (XFDF) document.</td>
</tr>
<tr>
<td><strong>Links</strong> source</td>
<td>Imports links into the parent element. The links are provided in the source, which must be an XFDF document.</td>
</tr>
<tr>
<td><strong>Links</strong> filter</td>
<td>Specifies that links from child source elements be imported into the parent element.</td>
</tr>
<tr>
<td><strong>NoBookmarks</strong></td>
<td>Specifies removal of bookmarks from the parent element.</td>
</tr>
<tr>
<td><strong>NoLinks</strong></td>
<td>Specifies removal of links from the parent element.</td>
</tr>
<tr>
<td><strong>NoThumbnails</strong></td>
<td>Specifies removal of embedded thumbnails from the parent element.</td>
</tr>
</tbody>
</table>

Comments

The elements in this subcategory represent comments, which can be imported or exported from PDF documents. The following elements fall into this subcategory.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comments</strong> result</td>
<td>Exports the comments from the child elements as an XFDF or Forms Data Format (FDF) document.</td>
</tr>
<tr>
<td><strong>Comments</strong> source</td>
<td>Imports comments into the parent element. The comments are contained in an XFDF or FDF document.</td>
</tr>
<tr>
<td><strong>Comments</strong> filter</td>
<td>Specifies comments from child source elements that can be imported into the parent element but are not returned as FDF or XFDF.</td>
</tr>
<tr>
<td><strong>NoComments</strong></td>
<td>Specifies removal of annotations from the parent element.</td>
</tr>
</tbody>
</table>
File attachments

The elements in this subcategory represent file attachments, which can be imported or exported from PDF documents. The following elements fall into this subcategory.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttachmentAppearance</td>
<td>Defines the appearance and placement of a page-level file attachment</td>
</tr>
<tr>
<td>Description</td>
<td>Provides a description of a file attachment</td>
</tr>
<tr>
<td>File</td>
<td>Specifies a file attached to a PDF document as a document-level or page-level file attachment</td>
</tr>
<tr>
<td>FileAttachments result</td>
<td>Requests information about file attachments and optionally returns attachments as separate data streams</td>
</tr>
<tr>
<td>FileAttachments source</td>
<td>Specifies a file to be attached to the parent document</td>
</tr>
<tr>
<td>NoFileAttachments</td>
<td>Specifies removal of all file attachments in the parent element</td>
</tr>
</tbody>
</table>

Document disassembly

The document disassembly element splits a single document into multiple PDF documents.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDFsFromBookmarks</td>
<td>Splits a single PDF document into multiple PDF documents, based on top-level bookmarks</td>
</tr>
</tbody>
</table>

Document properties

Document property elements specify information that applies to a PDF document as a whole, as opposed to individual pages. The following elements fall into this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Provides a value for the author metadata in the result document.</td>
</tr>
<tr>
<td>&quot;JavaScript&quot; on page 196</td>
<td>A source element that specifies a document-level JavaScript to be added to the result PDF document.</td>
</tr>
<tr>
<td>&quot;NoJavaScripts&quot; on page 207</td>
<td>Specifies that the parent PDF result or source element contains no document-level JavaScript.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Provides a single keyword for use as metadata.</td>
</tr>
<tr>
<td>Keywords</td>
<td>Provides metadata keywords for the result document.</td>
</tr>
<tr>
<td>MasterPassword</td>
<td>Specifies a password that is used to change permissions for the result document.</td>
</tr>
<tr>
<td>Metadata result</td>
<td>Exports the metadata from the child document as XMP.</td>
</tr>
</tbody>
</table>
Page labels

Page label elements provide the format and page number starting point for a string in the form "text + page number". This string is displayed beneath the page and beneath thumbnails in a viewer to identify the page. Using the Document Manipulation APIs, this string can also be added to page content such as footers. The following elements fall into this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata source</td>
<td>Imports the metadata contained in an XMP stream into the parent document.</td>
</tr>
<tr>
<td>NoForms</td>
<td>Flattens all form fields in the parent document.</td>
</tr>
<tr>
<td>NoXFA</td>
<td>Flattens XFA-based forms in the parent element.</td>
</tr>
<tr>
<td>OpenPassword</td>
<td>Specifies a password that is used to open the result document.</td>
</tr>
<tr>
<td>Password</td>
<td>Provides an access password for the Document Manipulation APIs to use to open encrypted source documents.</td>
</tr>
<tr>
<td>Subject</td>
<td>Provides the value for the subject metadata in the result document.</td>
</tr>
<tr>
<td>Title</td>
<td>Specifies the value for the title metadata in the result document.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoPageLabels</td>
<td>Specifies removal of page labels from the parent element</td>
</tr>
<tr>
<td>PageLabel</td>
<td>Specifies the format and content of page labels</td>
</tr>
</tbody>
</table>

Page properties

Page property elements specify how pages are viewed or printed. The following elements fall into this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArtBox</td>
<td>Defines the extent of the page’s meaningful content as intended by the creator of the page</td>
</tr>
<tr>
<td>BleedBox</td>
<td>Defines the bounds to which the contents of the page are clipped when output in a production environment</td>
</tr>
<tr>
<td>PageMargins</td>
<td>Specifies margins for page content elements</td>
</tr>
<tr>
<td>PageRotation</td>
<td>Specifies the rotation angle of pages within the scope of the parent element</td>
</tr>
<tr>
<td>PageSize</td>
<td>Defines the page dimensions for purposes of display or print</td>
</tr>
<tr>
<td>TrimBox</td>
<td>Defines the intended dimensions of the finished page after printing and trimming</td>
</tr>
</tbody>
</table>
Page content

Page content elements specify new content that is added to pages in PDF documents. The following elements fall into this category.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>Provides styled text or graphics for placement behind existing page content</td>
</tr>
<tr>
<td>Center</td>
<td>Specifies a centered header or footer</td>
</tr>
<tr>
<td>DatePattern</td>
<td>Provides style information used to construct a string representing the date and time</td>
</tr>
<tr>
<td>Footer</td>
<td>Provides styled text or graphics for footer content</td>
</tr>
<tr>
<td>graphic</td>
<td>Inserts a graphic as if it were a character in a line of text.</td>
</tr>
<tr>
<td>Header</td>
<td>Provides styled text or graphics for header content</td>
</tr>
<tr>
<td>Left</td>
<td>Specifies a left-justified header or footer</td>
</tr>
<tr>
<td>NoBackgrounds</td>
<td>Specifies removal of backgrounds from the parent element</td>
</tr>
<tr>
<td>NoFooters</td>
<td>Specifies removal of footers from the parent element</td>
</tr>
<tr>
<td>NoHeaders</td>
<td>Specifies removal of headers from the parent element</td>
</tr>
<tr>
<td>NoWatermarks</td>
<td>Specifies removal of watermarks from the parent element</td>
</tr>
<tr>
<td>&quot;PageContent&quot; on page 223</td>
<td>Provides styled text or graphics for placement either over or under the existing page content. Allows for alternate text to be provided for screen reading.</td>
</tr>
<tr>
<td>PageOverlay</td>
<td>Provides content for placement over the current page content</td>
</tr>
<tr>
<td>PageUnderlay</td>
<td>Provides content for placement under the current page content</td>
</tr>
<tr>
<td>Right</td>
<td>Specifies a right-justified header or footer</td>
</tr>
<tr>
<td>StyledText</td>
<td>Provides a rich text expression</td>
</tr>
<tr>
<td>Transform</td>
<td>Specifies transformations applied to the page contents, including scaling, translation, and rotation</td>
</tr>
<tr>
<td>Watermark</td>
<td>Provides styled text or graphics for placement over the existing page content</td>
</tr>
</tbody>
</table>

Profile

Profile elements are top-level elements specified only as children of the DDX root element. They contain reusable specifications. Other elements reference them by name. Profile elements let you create and maintain a repository of named profiles for different types of assemblies that are included within a DDX document.
Query elements enable the return of XML documents containing specific types of information about PDF documents. These XML documents provide information about a PDF document, such as the words it contains, the bookmarks it contains, and its metadata and page-related information.

Some manual or programmatic examination may be required to use the results of the query in the construction of another document.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialViewProfile</td>
<td>Specifies information about how a document is viewed when it is opened in a viewer application.</td>
</tr>
<tr>
<td>PDFAProfile</td>
<td>Specifies settings for conversion to PDF/A.</td>
</tr>
<tr>
<td>StyleProfile</td>
<td>Specifies a named profile that contains style information about page contents or table of contents elements.</td>
</tr>
<tr>
<td>PasswordEncryptionProfile</td>
<td>Specifies a named profile containing password security settings.</td>
</tr>
<tr>
<td>PasswordAccessProfile</td>
<td>Specifies a named profile containing an access password.</td>
</tr>
</tbody>
</table>

**Built-in keys**

You can use DDX built-in keys to insert variable information into headers, footers, watermarks, backgrounds, and table of contents entries. The Document Manipulation APIs replaces these keys when it processes the DDX document.

The following table describes the Document Manipulation APIs built-in keys.

<table>
<thead>
<tr>
<th>Built-in key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PageLabel</td>
<td>Page label of the current page in the result document. If the document has no page labels, this key is set to the value of the PageNumber built-in key.</td>
</tr>
<tr>
<td>_LastPageLabel</td>
<td>Page label of the last page in the result document.</td>
</tr>
<tr>
<td>_PageNumber</td>
<td>Ordinal (1-based) page number of the current page in the result document.</td>
</tr>
<tr>
<td>_LastPageNumber</td>
<td>Total number of pages in the result document.</td>
</tr>
<tr>
<td>Built-in key</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>_Title</td>
<td>Title from the metadata of the result document. If no title is defined, this key is set to an empty string.</td>
</tr>
<tr>
<td>_SourceTitle</td>
<td>Title from the metadata of the source document. If no title is defined, this key is set to an empty string.</td>
</tr>
<tr>
<td>_Author</td>
<td>Author from the metadata of the result document. If the metadata specifies multiple authors, this key represents only the first author name. If no author is defined, this key is set to an empty string.</td>
</tr>
<tr>
<td>_SourceAuthor</td>
<td>Author from the metadata of the source document. If the metadata specifies multiple source authors, this key represents only the first author name. If no author is defined, this key is set to an empty string.</td>
</tr>
<tr>
<td>_Subject</td>
<td>Subject from the metadata of the result document. If no subject is defined, this key is set to an empty string.</td>
</tr>
<tr>
<td>_SourceSubject</td>
<td>Subject from the metadata of the source document. If no subject is defined, this key is set to an empty string.</td>
</tr>
<tr>
<td>_Created</td>
<td>Creation date from the metadata of the result document. When this key is used as the name of a rich text (styled text) element, it can have an optional styleReference attribute. The value of this attribute references a StyleProfile element, which in turn contains a DatePattern element. The formatting in the DatePattern element is applied to the value supplied by the built-in key.</td>
</tr>
<tr>
<td>_Modified</td>
<td>Modification date from the metadata of the result document. When this key is used as the name of a rich text (styled text) element, it can have an optional styleReference attribute. See the _Created built-in key.</td>
</tr>
<tr>
<td>_DateTime</td>
<td>Current system time stamp. When this key is used as the name of a rich text (styled text) element, it can have an optional styleReference attribute. See the _Created built-in key.</td>
</tr>
<tr>
<td>_BookmarkTitle</td>
<td>Title of the current bookmark. (Used only with the TableOfContentsEntryPattern element.)</td>
</tr>
<tr>
<td>_BookmarkPageCitation</td>
<td>Page reference of the current bookmark. (Used only with the TableOfContentsEntryPattern element.)</td>
</tr>
<tr>
<td>_AdobeCoverSheet</td>
<td>An input source that is stored internally and is not provided in the input map. The Adobe cover sheet is a single PDF file that is the default cover sheet for a PDF package. Its use is restricted to PDF source. See “_AdobeCoverSheet” on page 148 for more details and information about localization.</td>
</tr>
<tr>
<td>_Filename</td>
<td>Filename stored with a package file. Its use is restricted to the FieldData element, as the value of the name attribute.</td>
</tr>
</tbody>
</table>
If a basic cover sheet is needed, there is a built-in Adobe cover sheet available, which is one single PDF page. To include the built-in cover sheet, the PDF source attribute value is specified as "_AdobeCoverSheet". DDX processors use the TargetLocale element to select locale-specific versions of the basic cover sheets.

Specifying the built-in cover sheet indicates that the result is a PDF package, and a default package specification is provided if necessary.

If no cover sheet is specified for a PDF package result, the Adobe cover sheet is automatically included in an assembly.

If the most locally specified TargetLocale includes a supported language code, then a localized version of _AdobeCoverSheet is provided. If TargetLocale is not specified or is not supported, the version for the default locale is used.

The supported languages are listed in the following table, along with the locale value for TargetLocale:

<table>
<thead>
<tr>
<th>Language</th>
<th>TargetLocale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulgarian</td>
<td>bg_BG</td>
</tr>
<tr>
<td>Chinese (China)</td>
<td>zh_CN</td>
</tr>
<tr>
<td>Chinese (Taiwan)</td>
<td>zh_TW</td>
</tr>
<tr>
<td>Croatian</td>
<td>hr_HR</td>
</tr>
<tr>
<td>Czech</td>
<td>cs_CZ</td>
</tr>
<tr>
<td>Danish</td>
<td>da_DK</td>
</tr>
<tr>
<td>Dutch</td>
<td>nl_NL</td>
</tr>
<tr>
<td>English</td>
<td>en_US</td>
</tr>
<tr>
<td>Finnish</td>
<td>fi_FI</td>
</tr>
<tr>
<td>French</td>
<td>fr_FR</td>
</tr>
<tr>
<td>German</td>
<td>de_DE</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hu_HU</td>
</tr>
<tr>
<td>Italian</td>
<td>it_IT</td>
</tr>
<tr>
<td>Language</td>
<td>TargetLocale</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Japanese</td>
<td>ja_JP</td>
</tr>
<tr>
<td>Korean</td>
<td>ko_KR</td>
</tr>
<tr>
<td>Latvian</td>
<td>lv_LV</td>
</tr>
<tr>
<td>Lithuania</td>
<td>lt_LT</td>
</tr>
<tr>
<td>Norwegian</td>
<td>no_NO</td>
</tr>
<tr>
<td>Polish</td>
<td>pl_PL</td>
</tr>
<tr>
<td>Portuguese (Brazil)</td>
<td>pt_BR</td>
</tr>
<tr>
<td>Romanian</td>
<td>ro_RO</td>
</tr>
<tr>
<td>Russian</td>
<td>ru_RU</td>
</tr>
<tr>
<td>Slovak</td>
<td>sk_SK</td>
</tr>
<tr>
<td>Slovenian</td>
<td>sl_SL</td>
</tr>
<tr>
<td>Spanish</td>
<td>es_ES</td>
</tr>
<tr>
<td>Swedish</td>
<td>sv_SE</td>
</tr>
<tr>
<td>Turkish</td>
<td>tr_TR</td>
</tr>
<tr>
<td>Ukrainian</td>
<td>uk_UA</td>
</tr>
</tbody>
</table>
Special DDX attribute values

DDX uses special types to represent possible values of attributes.

Color-specifier

A color-specifier definition can have one of the values described in the following table.

<table>
<thead>
<tr>
<th>String or string pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#rrggbb</td>
<td>Hexadecimal representation of a color in the RGB color-space. (rr) is a placeholder for the red component, (gg) for the green component, and (bb) for the blue component. For example, the value #000000\ represents black and the value #FFFFFF\ represents white.</td>
</tr>
<tr>
<td>SVG color keyword name</td>
<td>One of the colors named in the Scalable Vector Graphics (SVG) Specification, version 1.1 (<a href="http://www.w3.org/TR/SVG/">http://www.w3.org/TR/SVG/</a>). The following examples are SVG color keyword names:</td>
</tr>
<tr>
<td></td>
<td>black represents #000000</td>
</tr>
<tr>
<td></td>
<td>aqua represents #00FFFF</td>
</tr>
<tr>
<td></td>
<td>blue represents #0000FF</td>
</tr>
<tr>
<td></td>
<td>white represents #FFFFFF</td>
</tr>
</tbody>
</table>

External Data URL

An external data URL can be used as a source of documents and strings. An external data URL can also be used as the destination of result documents. (See “Using External Data URLs for source and result values” on page 23.) Adobe Experience Manager Forms supports the following URL schemes:

- "Application URL" on page 151
- "Contentspace URL" on page 152
- "Process URL" on page 152
- "File URL" on page 152
- "File URL" on page 152
- "HTTP/HTTPS URL" on page 153
- "FTP URL" on page 153
- "Inputmap URL" on page 153

Any result attribute can be an URL, as long as the location the URL specifies is writable. Any source, import, or thumbnail attribute can also be a URL, as long as the location specifies a document.

Some attributes that take string values can also be a URL.
Attribute values that require the URL to resolve to a String value

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;Author&gt;</code></td>
<td>value</td>
</tr>
<tr>
<td><code>&lt;BatesNumber&gt;</code></td>
<td>prefix, suffix</td>
</tr>
<tr>
<td><code>&lt;File&gt;</code></td>
<td>filename</td>
</tr>
<tr>
<td><code>&lt;Folder&gt;</code></td>
<td>name</td>
</tr>
<tr>
<td><code>&lt;Keyword&gt;</code></td>
<td>value</td>
</tr>
<tr>
<td><code>&lt;Resource&gt;</code> (as child of <code>&lt;Navigator&gt;</code> and as child of <code>&lt;Portfolio&gt;</code>)</td>
<td>name</td>
</tr>
<tr>
<td><code>&lt;PackageFiles&gt;</code></td>
<td>sourceMatch</td>
</tr>
<tr>
<td><code>&lt;PageLabel&gt;</code></td>
<td>prefix</td>
</tr>
<tr>
<td><code>&lt;PDF&gt;</code></td>
<td>bookmarkTitle, sourceMatch</td>
</tr>
<tr>
<td><code>&lt;PDFGenerationSettings&gt;</code></td>
<td>conversionSettings, fileTypeSettings</td>
</tr>
<tr>
<td><code>&lt;ReaderRights&gt;</code></td>
<td>credentialAlias</td>
</tr>
<tr>
<td><code>&lt;String&gt;</code> (String element as a child of the <code>StyledText</code> element)</td>
<td>url</td>
</tr>
<tr>
<td><code>&lt;String&gt;</code> (String element as child of the <code>Navigator</code> element)</td>
<td>name</td>
</tr>
<tr>
<td><code>&lt;TargetLocale&gt;</code></td>
<td>locale</td>
</tr>
<tr>
<td><code>&lt;Subject&gt;</code></td>
<td>value</td>
</tr>
<tr>
<td><code>&lt;TableOfContents&gt;</code></td>
<td>bookmarkTitle</td>
</tr>
<tr>
<td><code>&lt;Title&gt;</code></td>
<td>value</td>
</tr>
<tr>
<td><code>&lt;XDP&gt;</code></td>
<td>fragment, sourceMatch, insertionPoint, retainInsertionPoints, removeInsertionPoints</td>
</tr>
<tr>
<td><code>&lt;XDPContent&gt;</code></td>
<td>n/a</td>
</tr>
</tbody>
</table>

Application URL

Application URLs provide access to data available in an application created in Workbench. DDX elements can use Process URLs to access data independent of the Document Manipulation APIs operation’s input map.
Assets such as documents and data can be contained in applications. An Application is identified name and version, for example myApp/1.0. The version can contain additional folders for resources and processes.

The following example shows the syntax of an application URL:
application:///myApp/1.0/myFiles/doc1.pdf

**Contentspace URL**

When Contentspace is installed and available (at http://localhost:8080/contentspace), assets can be referenced in the DDX by using the contentspace URL. The path comes from the path visible in Contentspace, for example:

Company Home > User Homes > Assembler Dev Tests > doc1.pdf

The DDX can reference the above document using a Contentspace URL:

```xml
<PDF
  source="contentspace:///Company Home/User Homes/Assembler Dev Tests/doc1.pdf"/>
```

**Process URL**

Process URLs provide access to data available in a process created in Workbench. DDX elements can use Process URLs to access process data independent of the Document Manipulation APIs operation's input map.

The following example shows the syntax of a Process URL:

```xml
process://host:port/lc-xpath-data?lc-xpath-function#name
```

Where:

- **host:port** can be null and defaults to localhost:8080. All other values are treated as null.
- **lc-xpath-data** is an XPath expression consisting of the following: /process_data/@inDoc
- **lc-xpath-function** is an XPath expression consisting of the following:
  - get-map-values(/process_data/mapOfDocuments)
- **lc-xpath-data takes precedence over any** lc-xpath-function
- **name** (optional) is a string used as the logical name for a document. The name can include an extension that identifies the file type. If the Process URL resolves to a list of Document objects, each document name is prefaced with an integer. The integer specifies the position of the Document in the list. For example, 0_doc.pdf, 1_doc.pdf.

Here are examples of process URLs.

- process://localhost:8080/process_data/@doc1#doc1.pdf
- process://process_data/@doc1#doc1.pdf
- process://?get-map-values(/process_data/assemInputs)#docZ.pdf

**File URL**

A File URL enables direct access to files on locally accessible disk storage. The following example shows the syntax of a File URL:

```xml
file:///dir-path/filename
```
where:
  
  * `dir-path` is a directory path
  * `filename` is the (optional) name of the repository file being referenced

If a File URL omits the `filename` part, the Document Manipulation APIs includes every file in the `dir-path` folder that satisfies the filtering criteria. The `sourceMatch` attribute (if present) specifies the filtering criteria.

**HTTP/HTTPS URL**

A HTTP/HTTPS URL enables access to files on intranets and the World Wide Web. Basically, any URL that works in a browser works with DDX. Your server settings determine whether to use https or http URLs.

**FTP URL**

An FTP URL enables access to files on FTP servers. As with HTTP/HTTPS URLs, some FTP servers require security credentials. Here is the syntax of an FTP URL, where optional parts are enclosed in brackets:

- `ftp://[userName:userPassword@]ftpServerAddress[:port]/myPath`
- `ftps://[userName:userPassword@]ftpServerAddress[:port]/myPath`

**Inputmap URL**

An inputmap URL enables access to strings specified by the input map. Here is the syntax of an inputmap URL:

- `inputmap:///String/String`

Usually, you use the input map to provide the documents used in the assembly. You can also use it to provide strings.

You can use an external data URLs to provide string values. For example, the following DDX expression defines a watermark that has a value supplied as an inputmap URL.

```
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <PDF result="Untitled 1">
    <PDF source="sourcePDF1"/>
    <Watermark>
      <StyledText>
        <p>
          <String url="inputmap:///companyStrings/watermarkString">Temporary string</String>
        </p>
      </StyledText>
      <Watermark>
    </PDF>
  </DDX>
```

**URL Examples**

Accessing the output value of a service such as Generate PDF by using a value in the DDX element. For example, consider the following:

```
<PDF source="process:///process_data/@pdfgResult"/>
instead of using a `SetValue` action to place the output document from the Generate PDF service into the input map for the Document Manipulation APIs.

Enables the document name to come from an XPath string variable. For example:

```xml
<File filename="process://process_data/@doc1_name"/>
```

Allows the content of `StyledText` to come from an XPath string variable using the `String` element. For example:

```xml
<StyledText>
  <p>
    <String url="process://process_data/@recipient"/>
  </p>
</StyledText>
```

### Page and document ranges

A page range or document range value is a comma-separated sequence of one or more of the following:

- Ordinal page number, such as 1, 5, or 99, where 1 specifies the first page in the document
- Keyword `last`, which specifies the last page or document
- Keyword `penultimate`, which specifies the next to last page
- Continuous and increasing range of pages, separated by a dash, such as 1-5, or 8-9
- Discontinuous range of pages, separated by a comma, such as 1, 3, 5

Here are some examples of allowable values:

```
1,4-99,penultimate,4,5,1-3
```

Even though the comma-separated values are specified in a mixed order, the result uses the increasing order of 1-5, penultimate.

The ordinal pages specified for this attribute are independent of page numbers appearing in page labels or as part of page content.

### length-specifier

A length-specifier is a positive or negative value specified along with a unit of measurement without any spaces. The units of measure allowed are "in" for inch, "cm" for centimeter, "mm" for millimeter, and "pt" for points (1/72 in).

Here are some examples of length values.

<table>
<thead>
<tr>
<th>Length value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;1.0in&quot;</td>
<td>1.0 in</td>
</tr>
<tr>
<td>&quot;-3.77cm&quot;</td>
<td>-3.77 cm</td>
</tr>
<tr>
<td>&quot;35pt&quot;</td>
<td>35 points (a point is 1/72 in)</td>
</tr>
<tr>
<td>&quot;0pt&quot;</td>
<td>0 points</td>
</tr>
</tbody>
</table>

### nonnegative-length-specifier

A nonnegative-length-specifier is a positive value, greater than or equal to zero, specified along with a unit of measurement without any spaces. The format is the same as for length-specifier.
**positive-length-specifier**

A positive-length-specifier is a positive value, greater than zero, specified along with a unit of measurement without any spaces. The format is the same as for length-specifier.
DDX Language Elements

Document Description XML (DDX) language is a declarative markup language for describing documents that are constructed from component PDF, XDP, and XML documents.

The namespace of the DDX language is http://ns.adobe.com/DDX/1.0/, and the root element is DDX.

About

Requests information about the Communications - Document Manipulation APIs.

```xml
<About
    result="xs:string"
/>
```

Can be contained in the DDX element, which is the DDX root.

This element requests that the Document Manipulation APIs produce an XML document that describes its version and release.

The format of the XML document produced for this query element is described in “About Language” on page 308.

Category

“Query” on page 149

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. Name of the stream containing the result of this query or an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>

ArtBox

Defines the extent of the page’s meaningful content as intended by the creator of the page.

```xml
<ArtBox
    left="0pt" or "length"
    top="0pt" or "length"
    right="0pt" or "length"
    bottom="0pt" or "length"
    alternation="None" or "OddPages" or "EvenPages"
/>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, and conversion elements, and BlankPage.
This element is intended for use by prepress professionals only.

**Category**

“Page properties” on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Optional. Width of the margin as measured from the left side of the page to the left side of the art box. In this case, the page is the visible page as viewed in Acrobat. Default: 0 pt</td>
</tr>
<tr>
<td>top</td>
<td>Optional. Width of the margin as measured from the top of the page to the top of the art box. In this case, the page is the visible page as viewed in Acrobat. Default: 0 pt</td>
</tr>
<tr>
<td>right</td>
<td>Optional. Width of the margin as measured from the right side of the page to the right side of the art box. In this case, the page is the visible page as viewed in Acrobat. Default: 0 pt</td>
</tr>
<tr>
<td>bottom</td>
<td>Optional. Width of the margin as measured from the bottom of the page to the bottom of the art box. In this case, the page is the visible page as viewed in Acrobat. Default: 0 pt</td>
</tr>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or to alternating pages. This attribute can have the following values: None (default) - Settings apply to all pages. OddPages - Settings apply to odd pages only. EvenPages - Settings apply to even pages only. Pages are considered odd or even depending on their ordinal page number in the resultant document.</td>
</tr>
</tbody>
</table>

**AttachmentAppearance**

Defines the appearance and placement of a page-level file attachment (also known as a file attachment annotation).

```xml
<AttachmentAppearance
    author="Author from document metadata" or "xs:string"
    color="blue" or "color"
    icon="Paperclip" or "Graph" or "Pushpin" or "Tag"
    opacity="100%" or "percentage or decimal"
    x="72pt" or "length"
    y="72pt" or "length"
/>
```
Can be contained in the FileAttachments source element.

**Category**

“Document components” on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icon</td>
<td>Optional. The name of an icon that represents the attachment on the page. Default: PushPin</td>
</tr>
<tr>
<td>color</td>
<td>Optional. The color of the icon as displayed on the page. Default: blue</td>
</tr>
</tbody>
</table>
| opacity | Optional. Controls the transparency of the icon associated with the attachment. The value of this attribute can have the following forms:  
  Decimal in the range of .0 to 1.0  
  Percentage in the range of 0% to 100%. In this case, the percentage sign (%) is required.  
  The default is 100%. |
| author | Optional. The name of the author to be associated with the comment. If omitted, the author (if any) from the result PDF document is used; if there are multiple authors, the first one is used. |
| x | Optional. Specifies the horizontal location of the icon on the page. It is measured as the horizontal distance from the lower left corner of the page to the upper left corner of the icon. These positions reflect any rotation and resizing. Default: 72pt (1 inch) |
| y | Optional. Specifies the vertical location of the icon on the page. It is measured as the vertical distance from the lower left corner of the page to the upper left corner of the icon. These positions reflect any rotation and resizing. Default: 72pt (1 inch) |

**Author**

Provides a value for the author metadata in the resultant document.

```xml
<Author
  value="xs:string"
/>
```

Can be contained in the PDF result element, and the PackageFiles filter, select, and conversion elements.

value can specify multiple authors, separated by commas or semicolons. (If there are multiple authors, the Author built-in key and the DocumentInformation element return only the first author.)
If specified as a sibling to a `Metadata` source element, the `Metadata` source is imported first and the value of `Author` overrides the imported metadata.

**Category**

"Document properties" on page 146

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Required. Specifies the author name. Any value, including an empty string, overrides the author name in the resultant document. The string value can be specified with an External Data URL. (See &quot;External Data URL&quot; on page 150.)</td>
</tr>
</tbody>
</table>

`Background`

Provides styled text or PDF content for placement behind the existing page content.

```xml
<Background
    alternation="None" or "OddPages" or "EvenPages"
    fitToPage="true" or "false"
    horizontalAnchor="Left" or "Center" or "Right"
    horizontalOffset="0pt" or "length"
    opacity="100%" or "percentage"
    rotation="0" or "xs:integer"
    scale="100%" or "percentage"
    showOnScreen="true" or "false"
    showWhenPrinting="true" or "false"
    verticalAnchor="Top" or "Middle" or "Bottom"
    verticalOffset="0pt" or "length"
    replaceExisting="true" or "false"
>
    <StyledText> or < PDF source> [1]
    <TargetLocale> [0..1]
</Background>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, PackageFiles filter, select, or conversion elements, TableOfContentsPagePattern, BlankPage, and StyleProfile.

The `Background` element applies a background to all pages in the scope of its parent element. If `replaceExisting` is omitted or is "true", then this element replaces any pre-existing backgrounds on those pages. The `replaceExisting` attribute lets you retain pre-existing backgrounds when adding one new background. If `replaceExisting` is "false", a new `Background` is added to the existing `Background`. Each page can contain at most one background. Transient results allow for the addition of multiple backgrounds.

If the background content is provided in a PDF source element, the first page from the source document is used for the background.

Use the `Watermark` element to place content over the existing page content.
### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or to alternating pages. This attribute can have the following values: &lt;br&gt;None (default) - Settings apply to all pages. &lt;br&gt;OddPages - Settings apply to odd pages only. &lt;br&gt;EvenPages - Settings apply to even pages only.&lt;br&gt;Pages are considered odd or even depending on their ordinal page number in the resultant document.</td>
</tr>
<tr>
<td>fitToPage</td>
<td>Optional. Specifies whether to scale the background content to fit the page. This attribute can have the following values: &lt;br&gt;false (default) - Background content is scaled according to the scale attribute. &lt;br&gt;true - Background content is forced to fit the boundaries of the page size, by either expanding or shrinking the text. The scale attribute is ignored.</td>
</tr>
<tr>
<td>horizontalAnchor</td>
<td>Optional. Specifies a horizontal anchor point, relative to the PageMargins element of the page on which the background is placed. The horizontalOffset attribute lets you modify the placement relative to this anchor. This attribute can have the following values: &lt;br&gt;Left - Left page margin &lt;br&gt;Center (default) - Midpoint between the left and right margins &lt;br&gt;Right - Right margin</td>
</tr>
<tr>
<td>horizontalOffset</td>
<td>Optional. Offset from the horizontal anchor point specified in the horizontalAnchor attribute. A positive value moves the background right, while a negative value moves it left.</td>
</tr>
<tr>
<td>opacity</td>
<td>Optional. Transparency of the background. The value can be a decimal in the range of .0 to 1.0 or a percentage in the range of 0% to 100%. In the latter case, the percentage sign (%) is required.</td>
</tr>
<tr>
<td>replaceExisting</td>
<td>Optional. If set to false, the specified background does not replace any pre-existing background.</td>
</tr>
<tr>
<td>rotation</td>
<td>Optional. Rotation of the background upon the page. The valid range is -360 to 360°.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional. Scaling of the background. The valid range is 8% to 3200%. The value can be a decimal in the range of .0 to 1.0 or a percentage in the range of 0% to 100%. In the latter case, the percentage sign (%) is required.</td>
</tr>
</tbody>
</table>
BlankPage

Specifies the addition of an initially blank page to the document.

```xml
<BlankPage
    forceEven="true" or "false"
>
    <ArtBox> [0..2, where 2 is allowed for alternating pages]
    <Background>
    or
    <NoBackgrounds> [0..2, where 2 is allowed for alternating pages]
    <BleedBox> [0..2, where 2 is allowed for alternating pages]
    <Footer>
    or <NoFooters> [0..2, where 2 is allowed for alternating pages]
    <Header>
    or <NoHeaders> [0..2, where 2 is allowed for alternating pages]
    <PageContent> [0..n]
    <PageLabel> [0..1]
    <PageMargins> [0..2, where 2 is allowed for alternating pages]
    <PageOverlay> [0..n]
    <PageRotation> [0..1]
    <PageSize> [0..1]
    <PageUnderlay> [0..n]
    <TargetLocale> [0..1]
    <TrimBox> [0..2, where 2 is allowed for alternating pages]
    <Watermark>
```
or `<NoWatermarks> [0..2, where 2 is allowed for alternating pages]`</BlankPage>

Can be contained in the elements PDF result and PDFGroup.

**Note:** The PageContent element adds only its content, not the value of its alternateText attribute. This limitation is because pages added via the BlankPage element are not tagged.

This element causes the Document Manipulation APIs to append a blank page to the assembled document in the following situations:

- `forceEven="false"
- `forceEven="true"` and there is an odd number of pages in the resultant document at the point where the BlankPage element appears.

Pages added by the BlankPage element inherit page properties and content specified for parent elements, as well as the page properties specified as children of BlankPage. If PageSize and PageRotation are not specified within the scope of the BlankPage element, their values are taken from the result PDF page just before the blank page. If there is no preceding page, their values are taken from the following page.

Similarly, if a page label is not specified within the scope of the blank page, the following results occur:

- If there are no other page labels in the document, the blank page has no page labels.
- If there are page labels in the document, the blank page takes its labeling style from the previous page in the assembly. This same behavior also occurs when `<PageLabel mode="Continue"/>` is specified. If no pages precede the blank page, it takes a filler label that is equal to the ordinal page number.

**Category**

"Document assembly" on page 143

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| forceEven | Optional. Specifies the conditions when a blank page is added. (See description for the BlankPage element.) This attribute can have the following values:  
  - false (default) - A blank page is always added.  
  - true - A blank page is added only if there is an odd number of pages in the resultant document up to that point. |

**BleedBox**

Defines the bounds to which the contents of the page are clipped when output in a production environment.

```
<BleedBox
  left="0pt" or "length"
  top="0pt" or "length"
  right="0pt" or "length"
  bottom="0pt" or "length"
  alternation="None" or "OddPages" or "EvenPages"
```
Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element is intended for use by prepress professionals only.

**Category**

“Page properties” on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Optional. Width of the margin as measured from the left side of the page to the left side of the bleed box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>top</td>
<td>Optional. Width of the margin as measured from the top of the page to the top of the bleed box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>right</td>
<td>Optional. Width of the margin as measured from the right side of the page to the right side of the bleed box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>bottom</td>
<td>Optional. Width of the margin as measured from the bottom of the page to the bottom of the bleed box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the margin settings apply to all pages or alternating pages. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>None (default) - The margin settings apply to all pages.</td>
</tr>
<tr>
<td></td>
<td>OddPages - The margin settings apply to odd pages only.</td>
</tr>
<tr>
<td></td>
<td>EvenPages - The margin settings apply to even pages only.</td>
</tr>
</tbody>
</table>

**Bookmarks**

Bookmarks elements enable the bookmarks contained in PDF documents to be exported, imported, and removed. Bookmarks elements can have the following varieties:

- **Bookmarks** result. Specifies an XML document containing bookmarks from its child elements.
- **Bookmarks** source. Specifies bookmarks contained in a Bookmarks XML document. The bookmarks are imported into all pages within the scope of the parent element.
- **Bookmarks** filter. Specifies the bookmarks from its child elements, which can be imported into all pages within the scope of the parent element.

Also see the NoBookmarks element, which specifies that bookmarks be removed from all pages within the scope of the parent element.

In PDF documents, bookmarks are a tree-structured hierarchy of outline items that provide a means of navigating the document. When a user in a viewer application clicks a bookmark, an action is triggered.
Typically, a bookmark action specifies a particular location in the document to which the viewer should navigate. However, bookmarks can also trigger actions such as opening web pages or running JavaScript code.

**Bookmarks result**

Specifies that bookmarks from the child elements be exported to an XML document.

```xml
<Bookmarks
    result="xs:string"
    return="true" or "false"
>
    <PDF source> and/or <PDFGroup> [1..n]
    <Bookmarks source> [0..n]
    <Bookmarks filter> [0..n]
    <TargetLocale> [0..1]
</Bookmarks>
```

Can be contained in the **DDX** element, which is the DDX root.

This element directs the Document Manipulation APIs to export the bookmarks in the element’s child elements. If the Bookmarks result element contains multiple **PDF** source documents, they are effectively assembled into a single PDF document from which bookmarks are exported. (This document is not returned to the user.)

Bookmarks are exported in an XML document that conforms to the Bookmarks XML language, which is a component of the XPDF language. (See “Bookmarks Language” on page 321.)

This element must contain at least one **PDF** source element, either as a direct child or within a child **PDFGroup** element.

**Category**

“Document components” on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. Specifies a name to be associated with the exported bookmarks. This name must be unique among all result elements in the DDX document. This name can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>
| return | Optional. Specifies whether the exported bookmarks are returned to the client. This attribute can have the following values:  
true (default) - Exported bookmarks are returned to the client.  
false - Exported bookmarks are transient data, which can be referenced as the source from a subsequent result element. The bookmarks are not returned to the client. |
Bookmarks source

Bookmarks contained in an XML document are imported into all pages within the scope of the parent element.

```xml
<Bookmarks source="xs:string"/>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

This element identifies an XML document representing PDF bookmarks. The bookmarks are aggregated with bookmarks in other sibling source elements, if any. The aggregation replaces any bookmarks found in any pages within the scope of the parent element.

**Category**

"Document components" on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. Identifies the name of the Bookmarks XML document. This name can be specified with an External Data URL. (See &quot;External Data URL&quot; on page 150 and &quot;Bookmarks Language&quot; on page 318.)</td>
</tr>
</tbody>
</table>

Bookmarks filter

Specifies that bookmarks from child source elements be imported into the pages within the scope of the parent element.

```xml
<Bookmarks>
  <PDF source> and/or <PDFGroup> [1..n]
  <Bookmarks source> [0..n]
  <TargetLocale> [0..1]
</Bookmarks>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

The Bookmarks filter element contributes bookmarks to the pages within the scope of the parent element. It is shorthand for a simple export/import operation. That is, the Bookmarks filter element is equivalent to using these expressions (in order):

1. Bookmarks result to produce an intermediate Bookmarks XML document. The intermediate Bookmarks XML document is not returned to the client.

2. Bookmarks source element to then import the bookmarks from that Bookmarks XML document into the parent document. (See "Bookmarks Language" on page 321.)

As with the Bookmarks source element, the bookmarks specified by this element are aggregated with bookmarks in other sibling source elements, if any. The aggregation replaces any bookmarks found in the pages within the scope of the parent element.
The filter element must contain at least one PDF source element, which can be a child or be embedded within a child PDFGroup element.

**Category**

“Document components” on page 144

---

**Center**

Specifies the center point of the page as the anchor point for a header or footer.

```
<Center> [0..1]
  <StyledText> or < PDF source> [1]
</Center>
```

Can be contained in the elements Footer and Header.

The content specified by the child elements is centered on the page. If the StyledText element includes a text-align attribute, that attribute is ignored.

If this element specifies a PDF source element as a child, the first page of the document provides the content.

This element does not support containment of the text within the middle third of the page. The text can go off the left and right sides of the page.

This element it does not support auto-wrapping of text. Use the styled text <p> element to wrap the text.

**Category**

“Page content” on page 148

---

**ColorScheme**

(Since 9.0) Color settings used in the PDF Portfolio.

```
<ColorScheme
  scheme="noScheme | blueScheme | brownScheme | darkblueScheme |
  darkbrownScheme | darkgreenScheme | lightblueScheme |
  lightgrayScheme | oliveScheme | orangeScheme | pinkScheme |
  purpleScheme | rustScheme | steeleScheme | taupeScheme | yellowScheme"
  primaryText="color-specifier"
  secondaryText="color-specifier"
  background="color-specifier"
  cardBackgroundColor="color-specifier"
  cardBorder="color-specifier"
/>
```

Can be contained in the Portfolio element.

The ColorScheme properties can be used for various components of a PDF Portfolio navigator, such as a built-in navigator, a navigator header, or portfolio welcome page. The named schemes match the colors that can be selected in Adobe Acrobat 9.0 and later.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheme</td>
<td>Optional. The name of a built-in color scheme. These schemes are based on the schemes available in Adobe Acrobat 9.0. A value of noScheme allows the viewing application to use its default colors.</td>
</tr>
<tr>
<td>primaryText</td>
<td>Optional. The color of the primary text in a navigator, such as filenames and links. This color overrides the corresponding value in the scheme attribute, if specified. The value is a hexadecimal representation of a color (for example #000000) or an SVG color keyword name (for example black). See “Color-specifier” on page 153.</td>
</tr>
<tr>
<td>secondaryText</td>
<td>Optional. The color of the text other than primary text in a navigator. Overrides the corresponding value in the scheme attribute, if specified.</td>
</tr>
<tr>
<td>background</td>
<td>Optional. The color of the navigator background. Overrides the corresponding value in the scheme attribute, if specified.</td>
</tr>
<tr>
<td>cardBackgroundColor</td>
<td>Optional. The color of the navigator card background. Overrides the corresponding value in the scheme attribute, if specified.</td>
</tr>
<tr>
<td>cardBorder</td>
<td>Optional. The color of the navigator card border or the &quot;selected&quot; card. Overrides the corresponding value in the scheme attribute, if specified.</td>
</tr>
</tbody>
</table>

Comments

PDF documents can contain *annotations* that appear on a page but are not considered to be part of the page content. Comments, in both the Document Manipulation APIs and Acrobat, are the equivalent of what the *PDF Reference* calls *markup annotations*. They include items such as text notes, highlights, lines, and circles.

Comments elements enable comments contained in PDF documents to be exported, imported, and removed. The Comments elements include the following formats:

- **Comments result**. Exports comments from the child elements as an FDF or XFDF document.
- **Comments source**. Imports comments contained in an XFDF or FDF document into the pages within the scope of the parent element.
- **Comments filter**. Represents the comments in its child elements, without exporting them as FDF or XFDF.

Also see the NoComments element, which specifies that comments be removed from the pages within the scope of the parent element.

Some operations that you can perform on comments also affect other annotation types, as noted in the individual elements.

Comments result

Specifies that the comments from the child elements be exported as an XFDF or FDF document.
Can be contained in the DDX element, which is the DDX root.

To select the comments to export, add attributes that specify the selection criteria. If multiple selection attributes are included in a Comments element, comments are selected if they satisfy any of the criterion, as in an "or" logical operation. If a Comments element is nested within another Comments element, comments selected by one of the elements must also satisfy the conditions in the other element. This behavior is similar to an "and" logical operation. (See “Selection Attributes” on page 169.) For an explanation of the use of the Comment element selection attributes, see the “Introducing Document Description XML” on page 15.

The comments are exported as FDF or XFDF.

The result element must contain at least one PDF source element, which can be a child or be embedded within a child PDFGroup element.

**Category**

“Document components” on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. Specifies a name to be associated with the exported comments. This name must be unique among all result elements in the DDX document. This name can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>
### Selection Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>afterDate</td>
<td>Optional. Select comments dated after this date. The date is specified in YYYYMMDD format, where YYYY is the year, MM is the month number, and DD is the day number.</td>
</tr>
<tr>
<td>beforeDate</td>
<td>Optional. Select comments dated before this date. The date is specified in YYYYMMDD format, where YYYY is the year, MM is the month number, and DD is the day number.</td>
</tr>
<tr>
<td>byAuthor</td>
<td>Optional. Select comments that match the name specified by this attribute.</td>
</tr>
</tbody>
</table>
| byCategory    | Optional. Select comments by category. Multiple categories can be selected by entering the names as a comma-separated list. This attribute can have any combination of the following values:  
  - Notes - Includes the comment type Text.  
  - DrawingMarkups - Includes the comment types Line, PolyLine, Square, Circle, Polygon, and Ink.  
  - TextEditingMarkups - Includes the comment types Highlight, Underline, Squiggly, StrikeOut, Caret, and FreeText.  
  - Attachments - Includes the comment types FileAttachment and Sound. FileAttachments are limited to page-level file attachments. That is, document-level file attachments are not represented as comments. (See “FileAttachments” on page 181.)  
  - All (default) - All comments are selected. If byCategory is set to this value and filter is set to Include, no comments are exported. |

**Note:** When All is specified and filter is set to Include, the Document Manipulation APIs removes other annotation types. The exception are comments from the pages within the scope of the parent element, including Movie, Screen, PrinterMark, TrapNet, 3D, and Watermark annotations. Link and Widget annotations are not removed.
Comments source

Comments contained in an XFDF or FDF document are imported into the pages within the scope of the parent element.

```xml
<Comments
    source="xs:string"
    format="PDF" or "XFDF"
/>```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

This element identifies an FDF or XFDF document representing PDF comments. The comments are aggregated with comments in other sibling source elements, if any. The aggregation replaces any comments found in the pages within the scope of the parent element.

**Category**

“Document components” on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. A string or an External Data URL that identifies the XFDF or FDF document containing comments. A string must specify the name of an input map entry, which must resolve to a document (idp.document) of type XFDF or FDF. An input map entry and an External Data URL must resolve to a document. (See “External Data URL” on page 150.)</td>
</tr>
<tr>
<td>format</td>
<td>Optional. Specifies the format of the imported comments. This attribute can have the values XFDF (default) or FDF.</td>
</tr>
</tbody>
</table>

Comments filter

Specifies comments from child source elements that can be imported into the pages within the scope of the parent element.

```xml
<Comments
    filter="Exclude" or "Include"
    afterDate="unspecified" or "YYYYMMDD"
    beforeDate="unspecified" or "YYYYMMDD"
/>```
byAuthor=unspecified or "Author name"
byCategory=unspecified or "Notes" or "DrawingMarkups"
   or "TextEditingMarkups" or "Stamps"
   or "Attachments" or "All"
byType=unspecified or "name1,name2,name3"

</Comments>

This element contributes selected comments to the aggregation of comments contained in the pages within the scope of the parent element. This element is shorthand for a simple export/import operation. That is, the Comments filter element is equivalent to using these elements (in order):

1. Comments result to produce an intermediate XFDF document. The intermediate XFDF document is not returned to the client.

2. Comments source element to then import the comments from that XFDF document into the parent document.

The Comments filter element is identical to the Comments source. The comments are aggregated with comments in other sibling source elements, if any. The aggregation replaces any comments found in the pages within the scope of the parent element.

The Comments filter element must contain at least one PDF source element, which can be a child or be embedded within a child PDFGroup element. A Comments filter element cannot contain a Comments source element because the filter only applies to a PDF document. Filtering previously extracted comments is not supported.

Category

“Document components” on page 144

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| filter | Optional. Specifies whether matching is inverted such that instead of selecting comments to include, the filter selects comments to exclude. This attribute can have the following values:  
Include (default) - Comments that match the criteria are included.  
Exclude - Comments that match the criteria are excluded. |

DatePattern

Provides information used to construct a string representing the date and time.

<DatePattern>
   <xs:string/>
   <Space/> [0..1]
   <Second/> [0..1]
   <Second00/> [0..1]
</DatePattern>
Must be contained in `<StyleProfile>` element.

The formatting described by the `DatePattern` element can be applied only to the built-in keys `_Created`, `_Modified`, and `_DateTime`.

The following table describes the formatting specified by the appearance of each formatting element. These elements have no attributes, child elements, or content.

<table>
<thead>
<tr>
<th>DatePattern children</th>
<th>Formatting applied to given built-in key</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xs:string</code></td>
<td>Optional. Typically, such text provides separators used between values or provides other information.</td>
</tr>
<tr>
<td>Space</td>
<td>Optional. Specifies a white space between two styled text elements.</td>
</tr>
<tr>
<td>Second</td>
<td>Optional. 1- or 2-digit (0-59) second of the minute.</td>
</tr>
<tr>
<td>Second00</td>
<td>Optional. 2-digit (00-59) second of the minute.</td>
</tr>
<tr>
<td>Minute</td>
<td>Optional. 1- or 2-digit (0-59) minute of the hour.</td>
</tr>
<tr>
<td>Minute00</td>
<td>Optional. 2-digit (00-59) minute of the hour.</td>
</tr>
<tr>
<td>Hour</td>
<td>Optional. 1- or 2-digit (1-12) hour of the meridian (AM/PM), expressed as a 12-hour clock. The AM/PM designator can be included using the <code>AmPm</code> element.</td>
</tr>
<tr>
<td>Hour01</td>
<td>Optional. 2-digit (01-12) hour of the meridian (AM/PM), expressed as a 12-hour clock. The meridian name can be added using the <code>AmPm</code> element.</td>
</tr>
<tr>
<td>Hour24</td>
<td>Optional. Zero-padded 2-digit (00-23) hour of the day, expressed as a 24-hour clock.</td>
</tr>
<tr>
<td>AmPm</td>
<td>Optional. Meridian name (AM or PM) of the prevailing locale.</td>
</tr>
<tr>
<td>DayNumber</td>
<td>Optional. 1- or 2-digit (1-31) day of the month.</td>
</tr>
<tr>
<td>DatePattern children</td>
<td>Formatting applied to given built-in key</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>NayNumber01</td>
<td>Optional. Zero-padded 2-digit (01-31) day of the month.</td>
</tr>
<tr>
<td>DayName</td>
<td>Optional. Full weekday name of the prevailing locale.</td>
</tr>
<tr>
<td>ShortDayName</td>
<td>Optional. Abbreviated weekday name of the prevailing locale.</td>
</tr>
<tr>
<td>MonthNumber</td>
<td>Optional. 1- or 2-digit (1-12) month of the year.</td>
</tr>
<tr>
<td>MonthNumber01</td>
<td>Optional. Zero-padded 2-digit (01-12) month of the year.</td>
</tr>
<tr>
<td>MonthName</td>
<td>Optional. Full month name of the prevailing locale.</td>
</tr>
<tr>
<td>ShortMonthName</td>
<td>Optional. Abbreviated month name of the prevailing locale.</td>
</tr>
<tr>
<td>Year</td>
<td>Optional. 4-digit year.</td>
</tr>
<tr>
<td>ShortYear</td>
<td>Optional. 2-digit year, where 00 = 2000, 29 = 2029, 30 = 1930, and 99 = 1999.</td>
</tr>
</tbody>
</table>
| UTCOffset                    | Optional. Time zone as a delta from Coordinated Universal Time (east or west of the Greenwich meridian) in RFC 822. The time zone format is Z, +HH[MM], or -HH[MM], where HH is a placeholder for a zero-padded 2-digit hour of the day, and MM is a placeholder for a zero-padded 2-digit minute of the hour. The acceptable values are further described below:  

Z - Indicates the time zone is 'zero meridian' or 'Zulu Time'.  

+HH[MM] or -HH[MM] - A time zone expressed as an offset of plus or minus states that the offset can be added to the time to indicate that the local time zone is HH hours and MM minutes ahead or behind. The plus or minus sign must be included. For example, PDT is "-0700".  

**Note:** The convention +HH[MM] and -HH[MM] conforms to RFC 822, which does not use a colon to separate hours and minutes. This convention differs from the XML schema xs:dateTime format, which requires a colon between the hours and minutes. |
| ShortTimeZone                | Optional. Abbreviated time zone name of the prevailing locale, such as GMT, EST, and GMT-00:30. |
| TimeZone                     | Optional. Full time zone name of the prevailing locale, for example "Pacific Standard Time." |
The default format is the following:

\[\text{<Year/>-<MonthNumber01/>-<DayNumber01/>T<Hour01/>:<Minute00/>:<Second00/>}<\text{<UTCOffset/>}\]

For example, Jan 3, 2006 at 12:01am PST would be formatted as follows:

2006-01-03T:00:01-0700

**Category**

“Page content” on page 148

---

**DDX**

The DDX element is the root element and there can only be one element of this type. Only result, setting, and profile elements exist at the root level. The name of each result element must be unique among other result elements in the same DDX document.

```xml
<DDX>
    <About> [0..1]
    <Bookmarks result> [0..n]
    <Comments result> [0..n]
    <DDXProcessorSetting> [0..1]
    <DocumentInformation> [0..n]
    <DocumentText> [0..n]
    <FileAttachments result> [0..n]
    <FilenameEncoding> [0..n]
    <FileSize> [0..1]
    <InitialViewProfile> [0..n]
    <Links result> [0..n]
    <Metadata result> [0..n]
    <Navigator> [0..n]
    <PackageFiles result> [0..n]
    <PasswordAccessProfile> [0..n]
    <PasswordEncryptionProfile> [0..n]
    <PDF result> [0..n]
    <PDFAProfile> [0..n]
    <PDFGenerationSettings> [0..n]
    <PDFsFromBookmarks> [0..1]
    <StyleProfile> [0..n]
    <TargetLocale> [0..1]
    <XDP result> [0..1]
    <XFAConversionSettings> [0..1]
</DDX>
```

The order in which child elements appear in the DDX element dictates the order in which those elements are interpreted. Any result element can contain source elements that reference a preceding result. In the following example, the PDF source element specifies the Metadata result (doc1.xmp). The doc1.xmp is created as an interim result before the PDF source element is interpreted.

```xml
<Metadata result="doc1.xmp" return="false">
    <PDF source="doc1.pdf"/>
</Metadata>

<PDF result="resultDoc.pdf">
```
DDXProcessorSetting

(Since 8.0.1) Provides generic tuning hints and configuration settings to the DDX processor.

```xml
<DDXProcessorSetting
    name="settingName"
    value="settingValue"
/>
```

Can be contained in the DDX or PDF result elements.

The name-value pairs supported can vary between DDX processors and releases. A DDX processor must ignore any settings that it does not support.

See “Operation checkpoints (DDXProcessorSetting)” on page 358.
Description

Provides a description for a file attachment.

<Description>
  A description of the file
</Description>

Can be contained in the FileAttachments source element.

The contents of the Description element is a text description of a file being attached to a source document. (When the resultant document is viewed with Acrobat, the attachment description is visible in the Attachments panel.)

Category

“Document components” on page 144

DisplayOrder

Order for displaying the PDF package or portfolio fields.

<DisplayOrder>
  <Field> [0..n]
</DisplayOrder>

Can be contained in the Package or Portfolio elements.

The order of the Field elements contained in the DisplayOrder element corresponds to the order in which the viewer application displays the fields. If a Field is present in the Schema and absent in the DisplayOrder, its order is undefined and dependent on the PDF viewer.

If DisplayOrder is missing from the Package or Portfolio element, then the display order is aggregated from PDF package and portfolio documents (if any) in the PDF result. (The PDF result is the parent of the Package or Portfolio element.) If these elements define multiple PDF package or portfolio documents, their display orders are aggregated in the following order:

1. PDF base document (if it defines a PDF package or portfolio)
2. Subsequent PDF source documents, in the order specified in the DDX document

Fields that are duplicated across multiple PDF packages or portfolios are represented only once in the aggregated display order. A Field is a duplicate if it has the same name value and type value. If the Field name does not exist in the Schema for the result package, it is not added.

If the DisplayOrder is not specified, it is left to the application to determine the order in which the Schema name values are displayed.

Category

“Document assembly” on page 143
DocumentInformation

Returns an XML document that contains information about a PDF document. The information can be the number of pages, page sizes, and labels, and metadata settings such as author and title.

```xml
<DocumentInformation
    result="xml output name"
    source="pdf input name">
    <PDFAValidation> [0..1]
</DocumentInformation>
```

Can be contained in the DDX element, which is the DDX root.

If the PDFAValidation child element is specified, then the information includes the results of checking the PDF document for conformance. The PDFAValidation element specifies conformance criteria.

The format of the XML document produced for this element is described in "Document Information Language" on page 310.

Category

"Query" on page 149

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. Name of the stream containing the resultant XML document. The source can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
<tr>
<td>source</td>
<td>Required. Name of the PDF document for which information is desired. The source can be specified with an External Data URL. (See “External Data URL” on page 150.) This attribute can specify either of the following designations: The name of a PDF stream specified in the input to the DDX. The name of a PDF result element that was created previously in the DDX. The DocumentInformation element must appear after the PDF result element.</td>
</tr>
</tbody>
</table>

DocumentText

Returns an XML document that describes the words used in the document, the positions on the page of each word, or the paragraphs per page.

```xml
<DocumentText
    mode attribute = "TextPerPage" or "WithQuads" or "ParagraphsPerPage"
    result="xml output name">
    <PDF source> and/or <PDFGroup> [1..n]
</DocumentText>
```

Can be contained in the DDX element, which is the DDX root.
The format of the XML document produced for this element is described in “Document Text Language” on page 340.

Category

“Query” on page 149

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Optional. Specifies whether the Document Text (DocText) XML document includes word positions. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>TextPerPage (default) - The DocText document provides the text that appears on each page, without providing the positions for individual words.</td>
</tr>
<tr>
<td></td>
<td>WithQuads - For each word on each page, the DocText document includes a bounding box that describes the word’s position on the page.</td>
</tr>
<tr>
<td></td>
<td>ParagraphsPerPage - (since 9.0) Groups the words into sentences and paragraphs per page. There is no overflow to the next page.</td>
</tr>
<tr>
<td>result</td>
<td>Required. Name of the output stream containing the DocText document produced in response to the DocumentInformation element. The result can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>

Field

The Field element defines the schema for the custom metadata for the PDF Portfolio or package files.

A Field element exists within Schema, DisplayOrder, and SortOrder elements (see “Assembling PDF Packages and Portfolios” on page 48).

Category

“Document assembly” on page 143

Field contained in Schema element

The Field elements contained in a Schema element describe the metadata, the Fields, and their attributes.

The field name specified through DDX is normalized. That is, the field name is stripped of leading and trailing white space, and sequences of white space are replaced with a single space character.

```
<Field
    name="xs:string"
    type="Text" or "Date" or "Number" or "Filename" or "Description" or "ModificationDate" or "CreationDate" or "Size" or "CompressedSize"
    visible="true" or "false"
    editable="true" or "false"
/>```

Can be contained in Schema.
Field contained in DisplayOrder element

The Field elements contained in a DisplayOrder element identify the order of the fields shown in the user interface of a PDF viewer. Any field names that do not exist in the resulting schema are not included.

```xml
<Field
    name="xs:string"
/>
```

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>editable</td>
<td>Optional. Indicates whether the PDF viewing application provide support for editing the field value. This attribute has no affect on whether the field is editable by a DDX processor.</td>
</tr>
<tr>
<td>name</td>
<td>Required. The textual field name that is displayed to the user by the PDF viewing application. The name is normalized. That is, it is stripped of leading and trailing white space, and sequences of white space are replaced with a single space character. To add spaces, use the entity number for a non-breaking space (&amp;#160;).</td>
</tr>
<tr>
<td>type</td>
<td>Required. Identifies the type of data that is stored in this field. For the types &quot;Text&quot;, &quot;Date&quot; and &quot;Number&quot;, the value must be provided using the FieldData element contained in a PackageFiles element. The other types are for basic metadata stored with all attached files: filename, description, modification date, and creation date. If the value is not provided, it comes from the File and Description elements specified when importing PackageFiles. The &quot;Size&quot; is the actual size, in bytes, of the attached file, and must not be modified. The field data for the CompressedSize type is the length of the embedded file stream. The Length entry in the embedded file stream dictionary specifies the file length. (See the “Embedded File Streams” chapter of the ISO 32000, Document management, Portable document format, PDF 1.7.) Support for the CompressedSize attribute is added.</td>
</tr>
<tr>
<td>visible</td>
<td>Optional. The initial visibility of the field in the PDF viewer. At least one field must be specified as visible, or an error is thrown.</td>
</tr>
</tbody>
</table>

Field contained in SortOrder element

The Field elements contained in a SortOrder element identify fields used by the PDF viewer to sort the data in the display. Any Field names that do not exist in the resulting schema are not included. The locale affects the sorting behavior. The TargetLocale element specifies the locale.

```xml
<Field
    name="xs:string"
/>
FieldData

Defines the metadata that corresponds to the related Field defined in the Schema.

When a field name is not defined in the resulting schema, the metadata entry is not included and a warning is logged. Similarly, if the resulting PDF document is not a PDF package or portfolio, the metadata entry is not included and a warning is logged.

It is possible to modify the basic metadata (filename, description, modification date, creation date, and size) by using built-in keys for the name. The build-in keys are _Filename, _Description, _ModificationDate, and _CreationDate.

```
<FieldData
    name="xs:string"
    prefix="xs:string"
>
    xs:string
</FieldData>
```

Can be contained in the PackageFiles and Folder elements.

While the DDX schema allows an xs:string value, the value must be convertible to the data type defined for that field in the Schema. Otherwise, the value is cleared and a warning is logged. More specifically, convert values for "Date", "ModificationDate", and "CreationDate" metadata to xs:dateTime. If no time zone information is present in the string, the TargetLocale in effect is applied. For "Number", the value must be convertible to xs:decimal.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. The name of a field as described in Schema.</td>
</tr>
<tr>
<td>prefix</td>
<td>Optional. Additional text that is prefixed to the value, but is not included when sorting on the values of this FieldData. As an attribute, the prefix is normalized. That is, it is stripped of leading and trailing white space, and sequences of white space are replaced with a single space character. To add spaces, use the entity number for a non-breaking space (&amp;#160;).</td>
</tr>
</tbody>
</table>
File

Specifies a document-level or page-level file attachment.

```xml
<File
    filename="xs:string"
    mimetype="unspecified" or "IANA MIME type"
    creationDate="current dateTime" or "xs:dateTime"
    modificationDate="current dateTime" or "xs:dateTime"
/>
```

Can be contained in the **FileAttachments** source element.

**Category**

“**Document components** on page 144”

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Required. The filename to be associated with the file attachment. If the filename does not have a valid extension, the file attachment may not be viewable from within Acrobat or Adobe Reader. (The attachment is a file in the resultant PDF document.) The string value for the filename can be specified with an External Data URL.</td>
</tr>
<tr>
<td>mimetype</td>
<td>Optional. The MIME type for the data being attached. MIME types are registered with IANA (<a href="http://iana.org/assignments/media-types/">http://iana.org/assignments/media-types/</a>).</td>
</tr>
<tr>
<td>creationDate</td>
<td>Optional. The creation date for the data being attached. If not specified, the current date, as known to the server, is used.</td>
</tr>
<tr>
<td>modificationDate</td>
<td>Optional. The modification date for the data being attached. If not specified, the current date, as known to the server, is used.</td>
</tr>
</tbody>
</table>

**FileAttachments**

PDF documents can include file attachments. A file attachment is data extracted from a named file and relocated into the PDF document. A file attachment can be associated with the entire PDF document (document-level file attachment) or with an individual page (page-level file attachment).

The **FileAttachments** element has a source and result version:

- The **FileAttachments** result element provides information in an XML document about one or more file attachments from a source document, optionally returning them as named streams.

- The **FileAttachments** source element attaches a single file to the result document.

**FileAttachments result**

Requests information about file attachments. For single PDF documents that are not PDF packages or portfolios, this element optionally returns attachments as separate data streams.
Can be contained in the `DDX` element, which is the DDX root.

The `FileAttachments` element specifies one or more file attachments, each of which is returned as a separate data stream if the `extract` attribute is `true`. An XML document is always returned, which contains information about the specified attachments, including the unique names mapped to the output data stream. (See "File Attachments Language" on page 345.)

The `FilenameEncoding` element is optional. In PDF version 1.7 or later, the encoding information for the filename is stored in the PDF document. However, for compatibility with older documents, it is recommended that you continue to provide the `FilenameEncoding` element. This element provides an encoding to use to decode the filename as it is extracted from the.

More than one character encoding name can be specified. The Document Manipulation APIs attempt to decode the filenames using each character encoding until it finds the one that works best.

**Category**

"Document components" on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. A name to be associated with the returned XML document. This name must be unique among all result elements in the DDX document. The result can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
<tr>
<td>extract</td>
<td>Optional. Specifies whether the attachments specified in the <code>nameKeys</code> attribute are returned to the client. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>true (default) - The file attachments are returned as separate data streams to the client.</td>
</tr>
<tr>
<td></td>
<td>false - Only the XML document describing the attachments is returned.</td>
</tr>
<tr>
<td>nameKeys</td>
<td>Optional. Identifies file attachments selected from the source document. Here are the supported values:</td>
</tr>
<tr>
<td></td>
<td>Unspecified (default). All page-level file attachments are extracted.</td>
</tr>
<tr>
<td></td>
<td>An asterisk (*). All page-level and document-level file attachments are extracted.</td>
</tr>
<tr>
<td></td>
<td>A string specifying a single name key or a comma-separated list of name keys representing document-level file attachments to extract.</td>
</tr>
<tr>
<td></td>
<td>The appearance of a <code>pages</code> attribute in the PDF source element specifies pages from the source document. If such a PDF source is a child of the <code>FileAttachments</code> <code>result</code> element, only the specified pages are included in the file attachment.</td>
</tr>
</tbody>
</table>
**FileAttachments document-level source**

Specifies a file to be attached to the parent document.

```xml
<FileAttachments
    source="input name"
    nameKey="name"
>
    <File> [1]
    <FilenameEncoding> [0..1]
    <Description> [0..1]
</FileAttachments>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDPGroup.

The FileAttachments source element attaches a single file to the parent document. It can attach the file as a document-level file attachment. If an AttachmentAppearance element is specified, the element attaches the file as a page-level file attachment. (See "FileAttachments page-level source" on page 183.)

**Note:** The FileAttachments document-level source element replaces an existing file attachment only if the nameKey value matches the name of the existing file attachment. If the FileAttachments element omits the nameKey attribute, then it is possible to attach multiple copies of the same files to a PDF document. To distinguish such identically named files, provide different filenames in the File elements.

**Category**

"Document components" on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. A name associated with the data to be attached to the parent document. The source can be specified with an External Data URL. (See &quot;External Data URL&quot; on page 150.)</td>
</tr>
<tr>
<td>nameKey</td>
<td>Optional. A name that identifies a document-level file attachment. If the value of this attribute is already used by another attachment in the PDF document, the new attachment replaces the existing attachment. Otherwise, the new attachment is assigned a unique nameKey. The nameKey value is based on the filename specified by the File element and (if relevant) on the Folder element.</td>
</tr>
</tbody>
</table>

**FileAttachments page-level source**

Specifies a file to be attached to a specific page of the PDF document. The page to which the file is attached is the first page of the document specified by the parent PDF element.

```xml
<FileAttachments
    source="xs:string"
>
    <File> [1]
    <FilenameEncoding> [0..1]
    <Description> [0..1]
</FileAttachments>
```
<AttachmentAppearance> [1]
</FileAttachments>

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

The FileAttachments source element attaches a single file to the parent document. It can attach the file as a page-level file attachment (a file attachment annotation). In this case, the file is attached to the first page of the pages within the scope of the parent element. The appearance of the AttachmentAppearance element distinguishes page-level file attachments from document-level file attachments. That element also specifies the appearance of the icon associated with the annotation.

**Note:** The FileAttachments page-level source element replaces an existing file attachment only if the nameKey value matches the name of the existing file attachment. If the FileAttachments element omits the nameKey attribute, then it is possible to attach multiple copies of the same files to a PDF document. To distinguish such identically named files, provide different filenames in the File elements.

**Category**

“Document components” on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. A name associated with the data to be attached to the parent document. The source can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>

**FilenameEncoding**

Specifies character encodings to use for encoding and decoding the names of files being attached or extracted.

```
<FilenameEncoding
  encoding= "character encoding name"
  useOnImport= "true" or "false"
/>
```

Can be contained in DDX, PDF result, PDF group, PackageFiles result, source, filter, select, and conversion elements, NoPackage, FileAttachments result, FileAttachments document-level source, and FileAttachments page-level source.

The FilenameEncoding element is optional. However, it can be useful if the filenames are not stored as Unicode strings, as was the case in PDF 1.6 or earlier. In such cases, the package files were document-level file attachments before the PDF to which they were attached became a PDF package or portfolio. If the original host encoding is unknown, the FilenameEncoding element is used to decode the bytes in the filename. If multiple FilenameEncoding elements are provided, the first encoding that successfully decodes the bytes in the filename is used. However, there is no guarantee that the result is the expected result.

In PDF version 1.7 or later, if the FilenameEncoding element is defined at the root DDX level, it applies to all elements that can contain it. In such cases, the most local FilenameEncoding specified is the one
used. The most local set of \texttt{FilenameEncoding} elements is used in the order specified to decode filenames when exporting or filtering package files or exporting file attachments. Only one \texttt{FilenameEncoding} element is used when importing package files or file attachments. This one is either the first element specified in the most local set, or the first one marked \texttt{useOnImport="true"} in the most local set.

\textbf{Category}

“Document components” on page 144

\textbf{Attributes}

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>Required. The Document Manipulation APIs supports the character encodings described in the following table. Depending on the character encodings available through your installation's Java Virtual Machine, the Document Manipulation APIs can also support additional character encodings. Examples of such additional character encodings are ISO-8859-1, ISO-10646-UCS-2, and ISO-2022. (See the Extensible Markup Language (XML) Specification, 1.0.) If this attribute is omitted, the DDX processor may employ a default strategy. For example, it may use UTF-8 as the default encoding.</td>
</tr>
<tr>
<td>useOnImport</td>
<td>Optional. Flags a \texttt{FilenameEncoding} element as the preferred one to use when importing package files or file attachments. If multiple \texttt{FilenameEncoding} elements are flagged in a set, the following occurs: Warning is logged First flagged \texttt{FilenameEncoding} element is used to encode the filename for compatibility with older viewers.</td>
</tr>
</tbody>
</table>

\textbf{Character encodings}

<table>
<thead>
<tr>
<th>Character encoding names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE64</td>
<td>The Base16, Base32, and Base64 Data Encodings, RFC 3548 (<a href="http://ietf.org/rfc/rfc3548.txt">http://ietf.org/rfc/rfc3548.txt</a>)</td>
</tr>
<tr>
<td>UTF-8</td>
<td>The Unicode Standard, Version 4.0, (<a href="http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404">http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404</a>)</td>
</tr>
<tr>
<td>UTF-16</td>
<td>The Unicode Standard, Version 4.0, (<a href="http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404">http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404</a>)</td>
</tr>
</tbody>
</table>
(Since 8.2) The `FileSize` element does not specify pages or add content to the resulting PDF document. Instead, it is a hint to reduce the size of the resulting PDF document, if possible. When files are incrementally saved (the default save mode), they are always larger than the original. This growth in size occurs even when removing pages because the original pages remain.

To attempt to reduce the file size, the PDF document must be saved for FastWebView. **Caution:** Saving a PDF document for FastWebView, breaks signatures and certification.

There are several effects of using the `FileSize` element:

Document assembly takes more time.

Setting the `useObjectStreams` attribute to `true` requires Acrobat 6 or later to view the resulting document. If the PDF document is version 1.4 or earlier, the version is changed to 1.5.

Disabling the `FileSize` element’s attributes (false) has the same effect as omitting the `FileSize` element. By default, all of the `FileSize` element’s attributes are enabled (true).

Setting the PDF result element’s `save` attribute to `Incremental` (the default setting) can disable the `FileSize` element’s hint.

A `FileSize` element in the DDX root element inherits into all PDF result elements. The exception is PDF result elements that contain their own `FileSize` element.

```xml
<FileSize
compressNonObjectStreams="true" or "false"
removeDuplicateResources="true" or "false"
useObjectStreams="true" or "false"
/>

Can be contained in the elements PDF result and DDX.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compressNonObjectStreams</td>
<td>Optional. If <code>true</code>, force compression of all non-object streams,</td>
</tr>
<tr>
<td></td>
<td>such as content streams, using a flate filter. The default is <code>true</code>.</td>
</tr>
<tr>
<td>removeDuplicateResources</td>
<td>Optional. If <code>true</code>, remove duplicate resources. The default is <code>true</code>.</td>
</tr>
<tr>
<td>useObjectStreams</td>
<td>Optional. If <code>true</code>, use object streams for objects and the</td>
</tr>
<tr>
<td></td>
<td>crossreference table, and compress them. This requires the PDF</td>
</tr>
<tr>
<td></td>
<td>document version to be a minimum of 1.5 and for the save mode to</td>
</tr>
<tr>
<td></td>
<td>be FastWebView. The default is <code>true</code>.</td>
</tr>
</tbody>
</table>

Example

The following DDX applies all the default `FileSize` settings to `FileSizeAll.pdf`, but removes only duplicate resources from RemoveDuplicates.pdf.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
<FileSize/>
<PDF result="FileSizeAll.pdf" save="FastWebView">  
<PDF source="largeFileSize.pdf"/>
```
<PDF>
<Folder>
  <name>xs:string</name>
  <thumbnail>xs:string</thumbnail>
  <Description>[0..1]</Description>
  <FieldData>[0..1]</FieldData>
  <Folder>[0..n]</Folder>
  <PackageFiles>[0..n]</PackageFiles>
</Folder>
</PDF>
</DDX>

Folder

(Since 9.0) Identifies a folder for the package files in a PDF Portfolio.

Folders in a PDF Portfolio are similar to directories in a hierarchical file system. They allow files to be logically grouped. It follows that folder names must be unique among other folders at the same level in the hierarchy. Similarly, filenames must be unique within a folder.

To add a nested file structure to a PDF Portfolio, use the PackageFiles element rather than this element. Set the PackageFiles element's source attribute to a URL that specifies the root of the file structure. By default, that element adds all files and subfolders to the portfolio and retains the folder structure.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| name      | Required. The name of the folder. This name can be specified with an External Data URL. (See “External Data URL” on page 153.) The value of name has these restrictions:  
Cannot contain any embedded NULL characters  
Must not be longer than 255 characters  
Must not contain any of the eight special characters " / : * " < > | (forward slash, back slash, colon, asterisk, double quote, left angle, right angle, bar)  
Must not have period (\.) as the last character |
| thumbnail | Optional. The input map key or External Data URL mapped to a document containing an image of type JPG, GIF, or PNG. It is recommended that this image have a max resolution of 170x90 (width*height).  
Some viewing applications (such as Acrobat 9) do not display thumbnails for folders.  
The NoThumbnails element removes thumbnails in the resultant document. |

Footer

Characteristics of footer content placed on a page.

```xml
<Footer
    shrinkContentToFit="true" or "false"
    padding="0pt" or "nonnegative length"
    alternation="None" or "OddPages" or "EvenPages"
    whiteout="true" or "false"
    styleReference="Name of style element"
    replaceExisting="true" or "false"
    background-color="transparent" or "color-specifier"
    margin="margin-shorthand-specifier"
    margin-top="0pt or nonnegative-length-specifier"
    margin-right="0pt or nonnegative-length-specifier"
    margin-bottom="0pt or nonnegative-length-specifier"
    margin-left="0pt or nonnegative-length-specifier"
>
    <Left/> or <Center/> or <Right/> [1..3]
    <TargetLocale/> [0..1]
</Footer>
```

Can be contained in the elements PDF result, PDF Source, PDFGroup, TableOfContents, TableOfContentsPagePattern, BlankPage, PackageFiles filter, select, or conversion elements, and StyleProfile.
At least one of the three child elements Center, Left, and Right must be present. These elements indicate footer fields that are centered, left justified, and right justified, respectively.

This element specifies removal of any preexisting footers on the pages to which it applies, even if the contents of the footer components are blank. It also supports multiple footers (if replaceExisting is "false").

The `background-color` and `margin-*` attributes on the `Footer` element define the position and extent of the background area occupied by the footer.

For both headers and footers, the left border of the background area is drawn inside the left `PageMargin`. Additionally, the right border is drawn inside the right `PageMargin`.

For headers, the top border is drawn inside the top `PageMargin` and the bottom is drawn at the upper edge of the header's padding area. This bottom edge position is the tallest height of the `<Left>`, `<Center>`, or `<Right>` zones present in the header.

For footers, the bottom border is drawn inside the bottom `PageMargin` and the top border is drawn at the lower edge of the footer's padding area. This top edge position is the tallest of the `<Left>`, `<Center>`, or `<Right>` zones present in the footer.

The content or layout area of the header or footer is inside the border on each of the four sides by the corresponding side's `margin-*` setting.

The left, bottom, and right margins for the footer come from the `PageMargins` element in effect for the scope of the `Footer` element. The `padding` attribute specifies the top margin, between the footer and the body content. The `Footer` element content plus page margins and footer padding defines the bounding box for the footer. The `margin-*` attributes (for example, `margin-top`) further refine the bounding box position. The `margin-*` attributes are relative to the `PageMargin`.

The `Footer` element can specify the appearance and content of the footer or it can reference a `StyleProfile` element that itself contains a `Footer` element.

**Category**

"Page content" on page 148

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shrinkContentToFit</code></td>
<td>Optional. Specifies whether the original page content is scaled down if necessary so that it fits above the footer being added to the page. This attribute can have the following values: false (default) - Original page content is not scaled down. true - Original page content is scaled so that it fits in its entirety between the header and footer.</td>
</tr>
<tr>
<td><code>padding</code></td>
<td>Optional. A nonnegative length value specifying how much white space is added above the footer, between it and the page body content. This padding is included in the footer area when applying <code>shrinkContentToFit</code>.</td>
</tr>
<tr>
<td><code>replaceExisting</code></td>
<td>Optional. If it is set to &quot;false&quot;, pre-existing footers in the source PDF document remain. That is, the <code>Footer</code> element does not replace the existing footer.</td>
</tr>
</tbody>
</table>
Header

Characteristics of header content placed on a page.

```xml
<Header>
  
  shrinkContentToFit="true" or "false"
  padding="0pt" or "nonnegative length"
  alternation="None" or "OddPages" or "EvenPages"
  whiteout="true" or "false"

  
</Header>
```
styleReference="Name of style element"
replaceExisting="true" or "false"
background-color="transparent" or "color-specifier"
margin="margin-shorthand-specifier"
margin-top="0pt or nonnegative-length-specifier"
margin-right="0pt or nonnegative-length-specifier"
margin-bottom="0pt or nonnegative-length-specifier"
margin-left="0pt or nonnegative-length-specifier"

<Left/> or <Center/> or <Right/> [1..3]
<TargetLocale/> [0..1]
</Header>

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, BlankPage, PackageFiles filter, select, or conversion elements, and StyleProfile.

At least one of the three child elements Center, Left, and Right must be present. These elements indicate header fields that are centered, left justified, and right justified, respectively.

This element specifies removal of any preexisting headers on the pages to which it applies, even if the contents of the header components are blank. It also supports multiple headers (if replaceExisting is "false").

The background-color and margin-* attributes on the Header element define the position and extent of the background area occupied by the header.

For both headers and footers, the left border of the background area is drawn inside the left PageMargin. Additionally, the right border is drawn inside the right PageMargin.

For headers, the top border is drawn inside the top PageMargin and the bottom is drawn at the upper edge of the header’s padding area. This bottom edge position is the tallest height of the <Left>, <Center>, or <Right> zones present in the header.

For footers, the bottom border is drawn inside the bottom PageMargin and the top border is drawn at the lower edge of the footer’s padding area. This top edge position is the tallest of the <Left>, <Center>, or <Right> zones present in the footer.

The content or layout area of the header or footer is inside the border on each of the four sides by the corresponding side’s margin-* setting.

The left, bottom, and right margins for the header come from the PageMargins element in effect for the scope of the Header element. The padding attribute specifies the bottom margin, between the header and the body content. The Header element content plus page margins and padding defines the bounding box for the header. The margin-* attributes (for example, margin-top) further refine the bounding box position for the header. The margin-* attributes are relative to the PageMargin.

The Header element can specify the appearance and content of the header or it can reference a StyleProfile element that itself contains a Header element.

Category

"Page content" on page 148
## Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shrinkContentToFit</td>
<td>Optional. Specifies whether the original page content is scaled down if</td>
</tr>
<tr>
<td></td>
<td>necessary so that it fits below the header being added to the page.</td>
</tr>
<tr>
<td></td>
<td>This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>false (default) - Original page content is not scaled down.</td>
</tr>
<tr>
<td></td>
<td>true - Original page content is scaled so that it fits in its entirety</td>
</tr>
<tr>
<td></td>
<td>between the header and footer.</td>
</tr>
<tr>
<td>padding</td>
<td>Optional. A nonnegative length value specifying how much white space is</td>
</tr>
<tr>
<td></td>
<td>added below the header, between it and the page body content. This padding</td>
</tr>
<tr>
<td></td>
<td>is included in the header area when applying shrinkContentToFit.</td>
</tr>
<tr>
<td>replaceExisting</td>
<td>Optional. If it is set to &quot;false&quot;, pre-existing headers in the source PDF</td>
</tr>
<tr>
<td></td>
<td>document remain. That is, the Header element does not replace the existing</td>
</tr>
<tr>
<td></td>
<td>header.</td>
</tr>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or</td>
</tr>
<tr>
<td></td>
<td>to alternating pages. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>None (default) - Settings apply to all pages.</td>
</tr>
<tr>
<td></td>
<td>OddPages - Settings apply to odd pages only.</td>
</tr>
<tr>
<td></td>
<td>EvenPages - Settings apply to even pages only.</td>
</tr>
<tr>
<td></td>
<td>Pages are considered odd or even depending on their ordinal page number in</td>
</tr>
<tr>
<td></td>
<td>the result document.</td>
</tr>
<tr>
<td>whiteout</td>
<td>Optional. Specifies whether the header area (including padding) obscures</td>
</tr>
<tr>
<td></td>
<td>the underlying page content. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>false (default) - Header area does not obscure underlying page content.</td>
</tr>
<tr>
<td></td>
<td>true - Header area uses a white background to obscure any underlying</td>
</tr>
<tr>
<td></td>
<td>page body content.</td>
</tr>
<tr>
<td>styleReference</td>
<td>Optional. The name of a StyleProfile element that contains a Header element</td>
</tr>
<tr>
<td></td>
<td>describing the header. If this attribute is present, the other attributes</td>
</tr>
<tr>
<td></td>
<td>must not be present.</td>
</tr>
<tr>
<td>background-color</td>
<td>Optional. The color of the background area for the header or footer body</td>
</tr>
<tr>
<td></td>
<td>region. The value is a hexadecimal representation of a color (for example</td>
</tr>
<tr>
<td></td>
<td>#000000) or an SVG color keyword name (for example black). See “Color-specifier” on page 150.</td>
</tr>
<tr>
<td></td>
<td>This attribute is not inheritable.</td>
</tr>
<tr>
<td>margin</td>
<td>Optional. A shorthand CSS margin property of the form. Use XML escape</td>
</tr>
<tr>
<td></td>
<td>characters (for example, use &quot;&lt;&quot; for &quot;&lt;&quot;).</td>
</tr>
<tr>
<td>margin-top</td>
<td>Optional. Sets the top margin of the content/layout area. Must not be less</td>
</tr>
<tr>
<td></td>
<td>than 0. This attribute is not inheritable.</td>
</tr>
</tbody>
</table>
Header (portfolio navigation pane)

(Since 9.0) Resource that supplies a navigation header used in a PDF Portfolio.

```xml
<Header source="xs:string">
  <Resource [0..n]
</Header>
```

Can be contained in the `<Portfolio>` element.

You can use Acrobat 9 to design a portfolio header.

The Header itself is a resource with a name of "header/model.xml". If the header source is specified as a PDF document, then all resources excluding the Welcome Page ("welcome/model.xml") in the PDF source are specified. Only resources referenced by a "header/model.xml" or "welcome/model.xml" are visible in the portfolio navigation pane. If the header source is specified as XML, then any resources referenced by the XML must be provided as children. (See the `<WelcomePage>` element.)

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin-right</td>
<td>Optional. Sets the right margin of the content/layout area. Must not be less than 0. This attribute is not inheritable.</td>
</tr>
<tr>
<td>margin-bottom</td>
<td>Optional. Sets the bottom margin of the content/layout area. This attribute is not inheritable.</td>
</tr>
<tr>
<td>margin-left</td>
<td>Optional. Sets the left margin of the content/layout area. Must not be less than 0. This attribute is not inheritable.</td>
</tr>
<tr>
<td>source</td>
<td>Optional. Input map key or URL mapped to either a PDF document which contains a Welcome Page or to an XML document. If the source is a PDF, then more resources than are necessary can be included. If the source attribute is not specified, then it defaults to the identified base document for the <code>&lt;PDF&gt;</code> result.</td>
</tr>
</tbody>
</table>

InitialViewProfile

Specifies how a document is displayed when it is initially opened in a viewer application.

```xml
<InitialViewProfile
  name="xs:string"
  display="FileName" or "DocumentTitle"
  magnification="Default" or "percentage" or "FitPage"
  or "FitVisible" or "FitWidth"
  openToPage="1" or "xs:positiveInteger"
  pageLayout="Default" or "SinglePage" or "Continuous" or "Facing"
  or "ContinuousFacing" or "ContinuousFacingRight" or "ContinuousFacingLeft" or "FacingRight" or "FacingLeft"
  show="PageOnly" or "BookmarksPanel" or "PagesPanel" or "AttachmentPanel" or "LayersPanel"
```
userInterfaceOptions="empty-string"
    or "HideMenuBar,HideToolBars,HideWindowControls"
windowOptions="empty-string"
    or "ResizeToInitialPage,CenterOnScreen,FullScreenMode"
packageUIPane = "Left" or "Top" or "Minimized" or "SplitHorizontal"
    or "SplitVertical"
splitterBarPosition="percentage-specifier"
packageInitialDocument = "CoverSheet" or "FirstSortedDocument"

Can be contained in the DDX element, which is the DDX root.

The settings in this element are equivalent to what can be specified in the Acrobat user interface by selecting File > Properties > Initial View.

PDF result elements can specify the profile to apply with their initialView attribute, which must match the name attribute of the InitialViewProfile element.

Category

“Profile” on page 148

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. Name of the initial view profile. The value of this attribute must be unique among other InitialViewProfile elements in the same DDX document.</td>
</tr>
<tr>
<td>display</td>
<td>Optional. Controls how the document is identified in the window title bar. This attribute can have the following values: FileName (default) - The PDF filename is displayed in the window title bar. DocumentTitle - The Title from the document metadata is displayed in the window title bar.</td>
</tr>
<tr>
<td>magnification</td>
<td>Optional. Controls the magnification level of the pages when the document is opened. This attribute can have the following values: Default (default) - The default magnification as set by the user's preference. percentage - Magnification expressed as a percentage or decimal. Values must be between 0% and 6400% (0 - 64.0). FitPage - Fit the entire page within the application window. FitVisible - Fit the visible content of the page within the width of the application window. FitWidth - Fit the entire width of the page within the width of the application window.</td>
</tr>
<tr>
<td>openToPage</td>
<td>Optional. Specifies the page displayed when the document is opened. The value must be an integer between 1 and the last ordinal page number.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>packageInitialDocument</td>
<td>Optional. A string that identifies whether the cover sheet or the first package file (according to SortOrder) is initially shown in the viewer pane. The cover sheet is also known as the root PDF document that the files in the package are attached to. With viewing applications that support PDF Portfolios, the portfolio navigator can override this setting. When a package specification is included from PDF source documents, the packageInitialDocument attribute remains unchanged if it is not specified.</td>
</tr>
<tr>
<td>packageUIPane</td>
<td>Optional. Instructions to the viewer.</td>
</tr>
<tr>
<td></td>
<td>Top (default)- The viewer presents all information in the schema in a multicolumn format docked at the top of the viewer pane. The behavior of this setting and SplitHorizontal are similar except this setting adds a fixed splitter bar position.</td>
</tr>
<tr>
<td></td>
<td>Left - The viewer presents a subset of information from the schema, docked to the left of the viewer pane. The behavior of this setting and SplitHorizontal are similar except this setting adds a fixed splitter bar position.</td>
</tr>
<tr>
<td></td>
<td>Minimized - The viewer minimizes the information.</td>
</tr>
<tr>
<td></td>
<td>SplitHorizontal - The viewer presents a split horizontal view. When one of the package files is opened, the viewer continues to display the detailed file list.</td>
</tr>
<tr>
<td></td>
<td>SplitVertical - The viewer presents a split vertical view. When one of the package files is opened, the viewer continues to display the detailed file list.</td>
</tr>
<tr>
<td>splitterBarPosition</td>
<td>Optional. Relative position of the splitter bar in the vertical or horizontal window. Relevant only if packageUIPane has a value of SplitHorizontal or SplitVertical. If this attribute is not specified and a split view is specified, the splitter bar position is viewer-dependent.</td>
</tr>
</tbody>
</table>
The **JavaScript** element specifies a document-level script that is added to the resultant PDF document. When the PDF document is opened, all document-level scripts are executed.

```xml
<JavaScript source="unspecified" or "xs:string"
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageLayout</td>
<td>Optional. Controls paging through the document. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>Default (default)</td>
</tr>
<tr>
<td></td>
<td>SinglePage</td>
</tr>
<tr>
<td></td>
<td>Continuous</td>
</tr>
<tr>
<td></td>
<td>Facing (same as FacingRight with odd-numbered pages on the right)</td>
</tr>
<tr>
<td></td>
<td>ContinuousFacing (same as ContinuousFacingRight with odd-numbered pages on the right)</td>
</tr>
<tr>
<td></td>
<td>ContinuousFacingRight</td>
</tr>
<tr>
<td></td>
<td>ContinuousFacingLeft</td>
</tr>
<tr>
<td></td>
<td>FacingRight</td>
</tr>
<tr>
<td></td>
<td>FacingLeft</td>
</tr>
<tr>
<td></td>
<td>If the <code>windowOptions</code> attribute is set to <code>FullScreenMode</code>, this attribute is given the value <code>SinglePage</code>.</td>
</tr>
<tr>
<td>show</td>
<td>Optional. Specifies the panel, if any, displayed along with the page. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>PageOnly - Displays the page and no panels.</td>
</tr>
<tr>
<td></td>
<td>BookmarksPanel (default)</td>
</tr>
<tr>
<td></td>
<td>PagesPanel</td>
</tr>
<tr>
<td></td>
<td>AttachmentPanel</td>
</tr>
<tr>
<td></td>
<td>LayersPanel</td>
</tr>
<tr>
<td>userInterfaceOptions</td>
<td>Optional. Controls the interface options displayed with the page. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td><code>empty-string</code> (default) - No user interface options are selected.</td>
</tr>
<tr>
<td></td>
<td>HideMenuBar, HideToolBar, HideWindowControls - One or more options can be specified in a comma-separated string.</td>
</tr>
<tr>
<td>windowOptions</td>
<td>Optional. Controls the appearance of the window containing the viewer. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td><code>empty-string</code> (default) - No window options are selected.</td>
</tr>
<tr>
<td></td>
<td>ResizeToInitialPage, CenterOnScreen, FullScreenMode - One or more options can be specified in a comma-separated string.</td>
</tr>
</tbody>
</table>
Can be contained in PDF result, and PackageFiles filter, select, or conversion elements.

The name of the script must be unique within a PDF document. DDX processors resolve name conflicts as follows:

**Conflict with imported script.** If a PDF source document and the JavaScript element contain an identically named script, then the script in the JavaScript element prevails. That is, the script in the JavaScript element is used in the resultant document.

**Conflict among PDF sources.** If an identically named script exists in multiple PDF sources and the JavaScript element is omitted, the script from the base document prevails. That is, the script in the base document is used in the resultant document.

**Note:** Document-level scripts are not included from non-baseDocument PDF sources that specify only some of the pages from the source document.

The NoJavaScripts element removes any JavaScript in the resultant document. The JavaScript and NoJavaScripts elements cannot be siblings.

**Category**

“Document properties” on page 146

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. A unique name within the PDF document associated with the script.</td>
</tr>
<tr>
<td>source</td>
<td>Required. A logical name, associated with an input data stream or an ordered list of data streams, containing JavaScript content. If the source is not provided, the document-level JavaScript with that name is not included in the result PDF document. The source can be specified with an External Data URL. (See “Source elements” on page 18.)</td>
</tr>
</tbody>
</table>

**Keyword**

Provides a single keyword for use as metadata.

```xml
<Keyword
.value="xs:string"
/>
```

Can be contained in the Keywords element.

**Category**

“Document properties” on page 146
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of this attribute can be specified as a string or with an External Data URL that returns a string.</td>
</tr>
</tbody>
</table>

Keywords

Provides metadata keywords for the result document.

```xml
<Keywords
  mode="Set" or "Append">
  <Keyword/> [0..n]
</Keywords>
```

Can be contained in the PDF result element, and PackageFiles filter, select, or conversion elements.

This element specifies a set of keywords for the metadata of the result document. Each keyword is specified in a separate Keyword child element.

The keywords specified here replace or supplement existing keywords, depending on the value of the mode attribute. If mode is defined as Set and there are no child Keyword elements, all existing keywords are removed from the document metadata.

If the Keywords and Metadata source elements are siblings, the settings in the Keywords element are evaluated after the settings in the Metadata source element. As a result, a Keywords element having its mode attribute defined as Set overrides keywords imported by the Metadata source element.

Category

"Document properties" on page 146

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| mode | Optional. Specifies whether existing keywords in the result document are retained. This attribute can have the following values:
  Set (default) - Keywords from this element replace keywords in the result document.
  Append - Keywords from this element supplement keywords in the result document. |

Left

Specifies the left edge of the page as the anchor point for a header or footer.

```xml
<Left>
  <StyledText> or <PDF source> [1]
</Left>
```
Can be contained in the elements Footer and Header.

The content specified by the child elements is aligned with the left margin (specified by the left attribute PageMargins element). If the StyledText element includes a text-align attribute, that attribute is ignored.

**Note:** There is no containment of the text within the left third of the page and there is no auto-wrapping of text. The text can go off the right side of the page. Use the <p> element to wrap the text.

If this element specifies a PDF source element as a child, the first page of the document provides the content.

**Category**

“Page content” on page 148

**LinkAlias**

Provides an alternative name for the parent document, for use as a link destination.

```xml
<LinkAlias>
  text
</LinkAlias>
```

Can be contained in the PDF source element.

This element is used for resolving cross-document links in an assembly of multiple PDF documents. Consider a cross-document link in another source document that references the file provided by the current source document. This link is resolved correctly if the filename specified in the link matches the name specified by this element.

**Note:** Leading and trailing spaces in the link alias are processed as part of the alias and, therefore, must be avoided.

**Category**

“Document components” on page 144

**Links**

Links elements enable the links contained in PDF documents to be exported, imported, and removed. Links elements can have the following varieties:

- **Links** result. Specifies that links be exported from the child elements as an XFDF document.
- **Links** source. Specifies links contained in an XFDF document be imported into the pages within the scope of the parent element.
- **Links** filter. Specifies the links from its child elements, which can be imported into the pages within the scope of the parent element.

Also see the NoLinks element, which specifies removal of links from the pages within the scope of the parent element.
Links result

Specifies that links be exported from the child elements as an XFDF document.

```xml
<Links>
  <result>xs:string</result>
  <return>true</return>
</Links>
```

Can be contained in the `DX` element, which is the DDX root.

The `Links result` element must contain at least one `PDF` source element, which can be a child or be embedded within a child `PDFGroup` element. If this element contains multiple `PDF` source elements, all children are assembled into one PDF document from which the link information is exported.

**Category**

"Document components" on page 144

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. Specifies a name to be associated with the XFDF stream. This name must be unique among all result elements in the DDX document. The result can be specified as an External Data URL.</td>
</tr>
<tr>
<td>return</td>
<td>Optional. Specifies whether the XFDF stream is returned to the client. This attribute can have the following values: true (default) - The XFDF stream is returned to the client. false - The XFDF stream is not returned to the client but can be referenced from within a subsequent result element.</td>
</tr>
</tbody>
</table>

Links source

Links contained in an XFDF document are imported into the pages within the scope of the parent element.

```xml
<Links>
  <source>xs:string</source>
</Links>
```

Can be contained in the elements `PDF` result, `PDF` source, `PackageFiles` filter, select, or conversion elements, and `PDFGroup`.

The `Links source` element identifies an XFDF document representing PDF links. The links are aggregated with the links in other sibling source elements, if any. The aggregation replaces any links found in the pages within the scope of the parent element.
Category

“Document components” on page 144

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. The name of the XFDF document containing links. This name can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>

Links filter

Specifies the links from its child elements, which can be imported into the pages within the scope of the parent element.

```
<Links>
  <PDF source> and/or <PDFGroup> [1..n]
  <Links source> [0..n]
  <TargetLocale> [0..1]
</Links>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

The Links filter element contributes links to the aggregation contained by the pages within the scope of the parent element. It is shorthand for a simple export/import operation. That is, the Links filter element is equivalent to this sequence of elements (in order):

1. Links result to produce an intermediate XFDF document
2. Links source element that imports the links into the parent document.

The Links filter element cannot return the links to the client as an XFDF data stream. The Links result element can return them.

The Links filter element has the same aggregating behavior as the Links source element.

The Links filter element must contain at least one PDF source element, which can be a child or can be embedded within a child PDFGroup element.

Category

“Document components” on page 144

MasterPassword

Password that is required to change permissions for the resultant document.

```
<MasterPassword>
  xs:string
</MasterPassword>
```

Must be contained in the Permissions element.
The password specified in this element corresponds to the owner password described in the PDF Reference. A person who successfully provides the password specified in the MasterPassword element has full (owner) access to the document. This unlimited access includes the ability to change the document’s passwords and access permissions.

If the MasterPassword element is omitted, then there are no permission restrictions on the document. That is, full access is granted to everyone who can open the document. The open password can restrict who can open the document. If both the MasterPassword element and the OpenPassword element are specified, they cannot be the same.

**Category**

“Document properties” on page 146

---

**Metadata**

PDF documents contain metadata (information about the document) in an XML format called Extensible Metadata Platform (XMP). PDF metadata includes properties such as the title, author, and date created.

Metadata elements enable the metadata contained in PDF documents to be exported, imported, and removed. The Metadata elements include the following varieties:

- **Metadata result.** Specifies that metadata specified by child elements be exported as an XMP document.
- **Metadata source.** Specifies an XMP metadata stream imported into the pages within the scope of the parent element.

**Metadata result**

Metadata from the child document is exported as an Extensible Metadata Platform (XMP) document.

```xml
<Metadata
   result="xmp output name"
   return="true" or "false"
>
< PDF source> [1]
</Metadata>
```

Can be contained in the DDX element, which is the DDX root.

**Note:** XMP provides a standard format for the creation, processing, and interchange of metadata. Metadata is data that describes the characteristics or properties of a document. It can be distinguished from the main contents of a document. For example, for a word-processing document, the contents include the actual text data and formatting information. In contrast, the metadata can include properties such as author, modification date, or copyright status. The XMP Specification can be obtained from [http://www.adobe.com/products/xmp/index.html](http://www.adobe.com/products/xmp/index.html).

**Category**

“Document properties” on page 146
**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. A name to be associated with the exported metadata. This name must be unique among all result elements in the DDX document. The result can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
<tr>
<td>return</td>
<td>Optional. Specifies whether the metadata is returned to the client. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>- true (default) - The XMP data is returned to the client as a named stream.</td>
</tr>
<tr>
<td></td>
<td>- false - The XMP data stream is not returned to the client but can be referenced as the source from within a subsequent result document.</td>
</tr>
</tbody>
</table>

**Metadata source**

Metadata contained in an XMP stream that is imported into the parent document.

```xml
<Metadata
  source="xmp input name"
/>```

Can be contained in the PDF result element, and the PackageFiles filter, select, or conversion elements.

The XMP content specified by this element becomes the metadata for the result document, replacing any preexisting metadata in that document.

**Category**

“Document properties” on page 146

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. The name of an input data stream containing XMP metadata. The source can be specified with an External Data URL. (See “External Data URL” on page 150.)</td>
</tr>
</tbody>
</table>

**MetadataSchemaExtension**

(Since 9.0) Metadata schema for use with PDF/A conformance.

```xml
<MetadataSchemaExtension source = "xs:string"/>
```

Can be contained in the `<PDFAPProfile>` element.

PDF/A conformance requires that all metadata used in the PDF/A document be associated with a schema. The schema can be published in the XMP specification or defined in the PDF/A document using an extension schema.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. The source represents an input document that contains RDF compliant metadata extensions. The source can be specified by using an External Data URL. (See &quot;External Data URL&quot; on page 153.)</td>
</tr>
</tbody>
</table>

Navigator

(Since 9.0) Specifies a navigator to use for a PDF Portfolio.

```xml
<Navigator source="xs:string">
  <Resource> [0..n]
  <String> [0..n]
</Navigator>
```

Can be contained in the Portfolio element.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>(Optional) Input map key or External Data URL mapped to a document which returns a NAV file or a PDF document containing a navigator. If the source attribute is not specified, then it defaults to the identified base document for the PDF result document.</td>
</tr>
</tbody>
</table>

**Note:** NAV files and other resources can come from the repository, content services (deprecated), and a Adobe Experience Manager Forms resource-only application.

NoBackgrounds

 Specifies removal of backgrounds from the pages within the scope of the parent element.

```xml
<NoBackgrounds
  alternation="None" or "OddPages" or "EvenPages"
/>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element removes only backgrounds added with Acrobat 8 or earlier. Beginning with Acrobat 9, backgrounds are identified only as watermarks. Also, this element does not remove backgrounds that contain Bates numbers.

Category

"Page content" on page 148
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or to alternating pages. This attribute can have the following values: None (default) - Settings apply to all pages. OddPages - Settings apply to odd pages only. EvenPages - Settings apply to even pages only. Pages are considered odd or even depending on their ordinal page number in the result document.</td>
</tr>
</tbody>
</table>

**NoBookmarks**

Specifies removal of bookmarks from the pages within the scope of the parent element.

```xml
<NoBookmarks/>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

**Category**

“Document components” on page 144

**NoComments**

Specifies removal of comments from the pages within the scope of the parent element.

```xml
<NoComments/>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

The NoComments element removes annotation types other than comments. It removes such annotations from the pages within the scope of the parent element. For example, the element removes Movie, Screen, PrinterMark, TrapNet, 3D, and Watermark annotations. It does not remove Link and Widget annotations.

**Category**

“Document components” on page 144

**NoFileAttachments**

Specifies removal of all file attachments in the pages within the scope of the parent element. The NoFileAttachments element does not remove package files from a PDF package or portfolio. The elements NoFileAttachments and FileAttachments cannot be siblings.

```xml
<NoFileAttachments/>
```
Can be contained in the elements **PDF** result, **PDF** source, **PackageFiles** filter, select, or conversion elements, and **PDFGroup**.

**Category**

"Document components" on page 144

---

**NoFooters**

Specifies removal of footers from the pages within the scope of the parent element.

```xml
<NoFooters
  alternation="None" or "OddPages" or "EvenPages"
  />
```

Can be contained in the elements **PDF** result, **PDF** source, **PDFGroup**, **TableOfContents**, **TableOfContentsPagePattern**, **PackageFiles** filter, select, or conversion elements, and **BlankPage**.

This element removes only footers added with Acrobat 8 or earlier. It cannot remove footers added with later versions. Acrobat 9 and later does not distinguish between watermarks, backgrounds, headers, and footers. Also, this element does not remove footers that contain Bates numbers.

**Category**

"Page content" on page 148

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or to alternating pages. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>None (default) - Settings apply to all pages.</td>
</tr>
<tr>
<td></td>
<td>OddPages - Settings apply to odd pages only.</td>
</tr>
<tr>
<td></td>
<td>EvenPages - Settings apply to even pages only.</td>
</tr>
<tr>
<td></td>
<td>Pages are considered odd or even depending on their ordinal page number in the result document.</td>
</tr>
</tbody>
</table>

---

**NoForms**

All Acrobat (Acroform) and XFA-based form fields in the parent document are flattened.

```xml
<NoForms/>
```

Can be contained in the elements **PDF** result, **PDF** source, **PackageFiles** filter, select, or source elements, and **PDFGroup**.

This element flattens all form fields in the parent document. Such form fields include signature fields and data fields. This flattening invalidates any digital signatures in the parent document. The form fields in the resultant document retain their graphical appearance but are no longer interactive.
**Note:** If the PDF document is a dynamic XFA form, the Document Manipulation APIs uses the Output service to flatten the form. If that service is unavailable, an exception is thrown.

If the PDF does not contain any forms, the document is unmodified.

For XFA forms, the XFA stream is not included in the resultant PDF document, therefore the parent result element must not specify the XDP format.

**Category**

"Document properties" on page 146

---

**NoHeaders**

Specifies removal of headers from the pages within the scope of the parent element.

```xml
<NoHeaders
    alternation="None" or "OddPages" or "EvenPages"
/>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element removes only headers added with Acrobat 8 or earlier. Beginning with Acrobat 9, headers are identified only as watermarks. Also, this element does not remove headers that contain Bates numbers.

This element removes only headers added with Acrobat 8 or earlier. It cannot remove headers added with later versions. Acrobat 9 and later does not distinguish between watermarks, backgrounds, headers, and footers. Also, this element does not remove headers that contain Bates numbers.

**Category**

"Page content" on page 148

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or to alternating pages. This attribute can have the following values: None (default) - Settings apply to all pages. OddPages - Settings apply to odd pages only. EvenPages - Settings apply to even pages only. Pages are considered odd or even depending on their ordinal page number in the result document.</td>
</tr>
</tbody>
</table>

---

**NoJavaScripts**

The *NoJavaScripts* element specifies that the parent PDF result or source element contains no document-level scripts. If the *NoJavaScripts* element is specified as the child of a PDFGroup element,
it applies to all PDF sources contained in that PDFGroup. The JavaScript and NoJavaScripts elements cannot be siblings.

<NoJavaScripts/>

Can be contained in PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

**Category**

"Document properties" on page 146

---

**NoLinks**

Specifies removal of links from the pages within the scope of the parent element.

<NoLinks/>

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

**Category**

"Document components" on page 144

---

**NoPackage**

Inclusion of a NoPackage element specifies that the PDF parent is not a PDF package or portfolio. Instead the PDF parent is a single PDF document. Any potential package files are included instead as pages in that single PDF document, in the order specified by the package's sort order.

The following types of package files are included as document-level file attachments:

- Non-PDF documents
  - PDF documents that cannot be assembled with other PDF documents, for example, PDF documents that contain XFA-based forms or encrypted documents.

The fields in the package files are removed as document-level file attachments contain only basic metadata.

NoPackage cannot be a sibling to the Package or Portfolio elements.

<NoPackage
  bookmarkPackageFiles="true" or "false"
>
  <FilenameEncoding/> [0..n]
</NoPackage>

Can be contained in PDF result, PDF source.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookmarkPackageFiles</td>
<td>Optional. Default is true. Determines whether to add a top-level bookmark for the pages from the Package file in the resulting &quot;flattened&quot; PDF document. Any bookmarks within the Package file are included automatically and appear under this bookmark. The title of the bookmark is the filename of the Package file. FilenameEncoding is used when necessary to decode the filename.</td>
</tr>
</tbody>
</table>

Category

“Document assembly” on page 143

NoPackageFiles

Inclusion of a NoPackageFiles element specifies that the PDF parent does not contain any package files.

```xml
<NoPackageFiles/>
```

Can be contained in PDF result and PDF source.

If `<NoPackageFiles/>` is specified for a single PDF, document-level file attachments are not removed.

The NoPackageFiles and NoPackage elements can be siblings. Such an appearance indicates that the parent PDF element does not have any package files and is not a PDF package or portfolio.

Category

“Document assembly” on page 143

NoPageLabels

Specifies removal of page labels from the pages within the scope of the parent element.

```xml
<NoPageLabels/>
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

If the NoPageLabels element appears as a child of the PDF result element, all page labels are removed from the result document.

If a NoPageLabels element appears as a child of a PDF source or PDFGroup element, the pages in the parent element have no page labels. However, if the result document contains page labels anywhere else, a default page label is defined for all pages that contain no page labels. Page labels in the result document are considered regardless of their origin. The default page label contains only the ordinal page number.

Category

“Page labels” on page 147
NoPortfolio

(Since 9.0) The PDF parent element is a single PDF document rather than a PDF Portfolio. This element is interchangeable with the NoPackage element.

```xml
<NoPortfolio
  bookmarkPackageFiles="true" or "false"
>
  <FilenameEncoding/> [0..n]
</NoPortfolio>
```

Can be contained in the elements PDF result, PDF source.

The NoPortfolio element cannot be a sibling to the Portfolio or Package elements.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookmarkPackageFiles</td>
<td>Optional. Default is true. Determines whether to add a top-level bookmark for the pages from the Package file in the resulting &quot;flattened&quot; PDF document. Any bookmarks within the Package file are automatically included and appear under this bookmark. The title of the bookmark is the filename of the Package file. The FilenameEncoding element is used when necessary to decode the filename.</td>
</tr>
</tbody>
</table>

Category

"Document assembly" on page 143

NoThumbnails

Specifies removal of embedded thumbnails from the pages, package files, and folders within the scope of the parent element.

```xml
<NoThumbnails/>
```

Can appear in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

Thumbnails are small images of the page that can be embedded in the document. If thumbnails are not present, the viewer application generates them. Eliminating embedded thumbnails can reduce file size.

Category

"Document components" on page 144

NoWatermarks

Specifies removal of watermarks from the pages within the scope of the parent element.

```xml
<NoWatermarks/>
```
Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element removes only watermarks added with Acrobat 8 or earlier. It cannot remove watermarks added with later versions. Acrobat 9 and later does not distinguish between watermarks, backgrounds, headers, and footers. Also, this element does not remove watermarks that contain Bates numbers.

**Category**

“Page content” on page 148
NoXFA

Specifies that XFA-based forms in the pages within the scope of the parent element be flattened. If the document does not contain XFA-based forms, it is unmodified.

```xml
<NoXFA
    flatten="true" or "false"
/>```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or source elements, and PDFGroup.

A flattened PDF lacks an XML form template (XFA) and non-signature elements.

When non-base documents are assembled with other documents, those non-base documents cannot contain XFA-based forms. This element lets you remove the XFA content from non-base document.

If the DDX processor cannot flatten the form, it throws an exception. To avoid the exception, set the flatten attribute to false. This setting causes the DDX processor to simply remove the XFA element from the PDF document, which allows it to be assembled. However, data may be lost.

**Note:** If the PDF document is a dynamic XML form template and the flatten attribute is true, the Document Manipulation APIs uses the Output service to flatten the form. If that service is unavailable, an exception is thrown.

**See also**

“Flattening forms” on page 37

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten</td>
<td>Optional. Specifies whether full flattening is required. If true, the form is flattened. However, an error is raised if the Document Manipulation APIs is unable to flatten the form. If false, the only XFA entry in the parent result element is removed. The data is not merged with the form before flattening. As a result, some data may be lost. If flatten is false, the Output service is not used.</td>
</tr>
</tbody>
</table>

**Category**

“Document properties” on page 146

OpenPassword

User password that is required to open the resultant document.

```xml
<OpenPassword>
    password
</OpenPassword>
```

Can be contained in the PasswordEncryptionProfile element.
The password specified in this element corresponds to the user password described in the PDF Reference. A person who successfully provides the password specified in the OpenPassword element can open the document. However, they are restricted in their activities according to the user access permissions specified when the owner secured the document.

If both the MasterPassword element and the OpenPassword element are specified, their values cannot be the same.

If the OpenPassword element is unspecified, the document is not password-protected at the user level. Anyone can open such a document.

**Category**

“Document properties” on page 146

---

**OutputIntent**

Specifies color settings for PDF/A conversion.

```xml
<OutputIntent
colorSpace="sRGB or CoatedFOGRA27 or JapanColorCoated or SWOP"/>
</OutputIntent>
```

Can be contained in PDFAProfile.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorSpace</td>
<td>Optional. Name of color space to use during conversion. Default is sRGB.</td>
</tr>
</tbody>
</table>

---

**Package**

Specifies that the parent PDF is a PDF package or portfolio. The Package element can contain the package specification or allow the specification to be aggregated from any PDF sources that are packages in the assembly. If no sources are packages, the empty <Package/> element specifies a default package specification.

The package specification aggregates the package specifications from PDF sources. If the baseDocument is a package, then its package specification becomes the starting point for the resulting package specification. The package specifications from other PDF sources are included in the order in which the packages are assembled. The first package contributes the packageInitialDocument and packageUIPane, unless specified in InitialViewProfile. See Schema, DisplayOrder, and SortOrder for a description of how package specifications are aggregated. See the InitialViewProfile element.

**Category**

“Document assembly” on page 143
Package defining element

Defines a package specification that can be used in a PDF package or portfolio.

```xml
<Package>
  <Schema>[0..1]
  <DisplayOrder> [0..1]
  <SortOrder> [0..1]
  <TargetLocale> [0..1]
</Package>
```

Can be contained in PDF result, PDF source, and StyleProfile.

If the Package defining element omits the Schema element, the DDX processor aggregates a schema from the PDF source documents. It aggregates the schemas from the PDF packages or portfolios in the assembled PDF source elements.

If the Package defining element omits the DisplayOrder or SortOrder elements, the DDX processor aggregates corresponding replacements from the PDF source documents. This aggregation is the same as for an omitted Schema element.

Package filter element

As a filter element, the package specification comes entirely from the package specification contained within the child PDF source element. If the PDF source is not a package, then it is as if the Package element were specified as the empty `<Package/>` element.

```xml
<Package>
  <PDF source [1]
  <TargetLocale> [0..1]
</Package>
```

Can be contained in PDF result, PDF source, and StyleProfile.

**Category**

“Document assembly” on page 143

Referencing a package or portfolio contained in a StyleProfile element

You can use the styleReference attribute to reference a Package or Portfolio element contained within a named StyleProfile element. The Package or Portfolio in the referenced StyleProfile is used.

```xml
<Package styleReference="xs:string"/>
<Portfolio styleReference="xs:string"/>
```

Can be contained in PDF result and PDF source.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>styleReference</td>
<td>Required. The name of a StyleProfile element that contains a specification for a PDF package.</td>
</tr>
</tbody>
</table>
The **PackageFiles** element performs various tasks related to the package files used in a PDF package or portfolio. Package files are also called component files or portfolio files. Here are some of the tasks it specifies:

- Add package files to a PDF package or portfolio (**PackageFiles** source). More specific versions of this element can also modify, filter, and select the documents added to the PDF package or portfolio. (See the elements **PackageFiles** modifying, **PackageFiles** filter, and **PackageFiles** filter.)
- Export an XML file containing information about the files in an existing PDF package or portfolio (**PackageFiles** result). You can also use the **PackageFiles** result element to export the package files from an existing PDF package or portfolio.
- Import information from another PDF package or portfolio (**PackageFiles** import)
- Select packages files. The **PackageFiles** element’s nameKeys attribute selects package files from the source document.

The above versions of the **PackageFiles** element allow the metadata associated with package files to be specified or modified. They also allow the contents of a package file to be modified if it is a modifiable PDF document. One such sequence is described below:

1. Use the **PackageFiles** result element to export information about the package files and to export the package files themselves.
2. Modify the exported files in another service.
3. Use the **PackageFiles** import element to import the modified files, using a modified version of the descriptive XML created when the package files were exported.

For example, you could export the documents, digitally sign them, and reimport them into the package.

**PackageFiles** does not include the cover sheet from any packages contained within the **PackageFiles** filter element.

**Category**

“Document assembly” on page 143

**PackageFiles modifying elements**

Modifies characteristics of the package files. The filter, select, and conversion elements allow a number of child elements. The following elements are allowed:

- `<Bookmarks` source and/or `<Bookmarks` filter [0..n]) or `<NoBookmarks` [0..1]
- `<Links` source and/or `<Links` filter [0..n]) or `<NoLinks` [0..1]
- `<NoThumbnails` [0..1]
- `<Comments` source and/or `<Comments` filter [0..n]) or `<NoComments` [0..1]
- `<NoFileAttachments` [0..1]
- `<Metadata` source [0..1]
- `<Author` [0..1]
- `<Title` [0..1]
- `<Subject` [0..1]
- `<Keywords` [0..1]
<PageContent> [0..n]
<PageSize> [0..1]
<PageRotation> [0..1]
<PageMargins> [0..2, where 2 is allowed for alternating pages]
<PageLabel> [0..1] or <NoPageLabels> [0..1]
<ArtBox> [0..2, where 2 is allowed for alternating pages]
<BleedBox> [0..2, where 2 is allowed for alternating pages]
<TrimBox> [0..2, where 2 is allowed for alternating pages]
<Transform> [0..1]
<Header> [0..2, where 2 is allowed for alternating pages] or <NoHeaders> [0..2, where 2 is allowed for alternating pages]
<Footer> [0..2, where 2 is allowed for alternating pages] or <NoFooters> [0..2, where 2 is allowed for alternating pages]
<JavaScript> [0..n] or <NoJavaScripts> [0..1]
<Watermark> [0..1] or <NoWatermarks> [0..2, where 2 is allowed for alternating pages]
<Background> [0..2, where 2 is allowed for alternating pages] or <NoBackgrounds> [0..2, where 2 is allowed for alternating pages]
(PageOverlay> [0..n]
(PageUnderlay> [0..n]
>NoForms> [0..1]
>NoXFA> [0..1]
<FilenameEncoding> [0..n]
<TargetLocale> [0..1]

Only selected package files that are modifiable PDF documents are modified. Here are examples of this syntax:

- **Header** adds headers to the package files identified by the PackageFiles element
- **Footer** adds footers to the package files identified by the PackageFiles element.

If **PageLabel** is specified for the PDF document, a mode of "Continue" is ignored and a warning is logged. The **FileAttachments** element attaches a file to the PDF document's pages. If a package file is not a modifiable PDF document, then a warning is logged.

**PackageFiles source elements**

Adds documents to a package and assigns metadata to those documents.

```xml
<PackageFiles
    access="unspecified" or xs:string"
    includeSubFolders="true" or "false"
    matchMode="Include" or Exclude"
    nameKey="xs:string"
    required="true" or false"
    source="xs:string"
    sourceMatch="unspecified" or regular-expression"
/>
<Description> [0..1]
<FieldData> [0..n]
<File> [0..1]
<FilenameEncoding> [0..1]
... PackageFiles modifying elements ...
</PackageFiles>
```
Can be contained in PDF result element.

As a source element, PackageFiles adds any type of document to a PDF package or portfolio. However, the best viewing experience is when PDF documents are added as package files. The source attribute specifies the stream that contains the contents of the document that is attached. Optionally, the nameKey attribute specifies a unique identifier for the package file.

The nameKey attribute is similar to document-level FileAttachments. If the nameKey attribute specifies the full filename and the PDF result already contains a PackageFile with that nameKey, then the existing file is replaced.

A viewing application may assign a name to the root folder, such as Home or Main. To accommodate these different names, the nameKeys attribute specifies the root folder as a backslash (/). For example, if you open a PDF Portfolio and display a detailed list view, the top-level folder is the first name in the path. In the nameKeys attribute, the backslash (/) precedes the first name. For example, consider a file with the following folder hierarchy:

- Reports is the top-level folder
- Reports contains a subfolder named January
- January contains the file Expenses.pdf

The nameKey value for file in the above folder hierarchy is /Reports/January/Expenses.pdf. However, the viewing application may display Main > Reports > January across the top of the navigation pane.

Any modifying elements contained as children of the PackageFiles source element modify the document if it is a modifiable PDF document. If the document being added is not a PDF document or is not modifiable, the document is left unchanged and warning is logged.

The PackageFiles source element does not contribute a package specification to any aggregation of package specifications.

A FieldData can redefine the Description or File values. For more information on the FieldData element, see “FieldData” on page 180.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access</td>
<td>Optional. If the source is a password encrypted PDF document, this attribute names the PasswordAccessProfile to apply if the document is opened or modified.</td>
</tr>
<tr>
<td>includeSubFolders</td>
<td>Optional. If this attribute is true and the source attribute specifies a URL that references a folder, then all files in the folder and subfolders are included. The included files retain the structure of the subfolders in the PDF Portfolio being created. No part of the source attribute value appears as a folder name in the resulting PDF Portfolio. This behavior is analogous to drag-and-drop from the file system into Acrobat when viewing a PDF Portfolio. The default value is true. If false, then only the files specified in the folder are included in the PDF Portfolio.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| matchMode    | Optional. Specifies whether to include the matched results in the resultant document. This attribute can have the following values:  
  - Include (default) - Includes the matched data streams.  
  - Exclude - Excludes the matched data streams. |
| nameKey      | Optional. A unique, internal identifier for a package file. If the nameKey attribute is omitted, the DDX processor generates a unique nameKey based on the filename.  
  Similar to a file system, the nameKey must be unique. Thus, if the filename is report.pdf, then the nameKey for the document is report.pdf. If a document is added to a PDF package or portfolio and that filename exists, then a number is added to the filename, such as report_0001.pdf. The modified filename is the nameKey.  
  The PackageFiles export element returns an XML file that specifies the nameKey attributes assigned to each document. |
| required     | Optional. A value of true (default) requires the PackageFile source element actually to add a package file to the PDF package or portfolio.  
  A value of false, eliminates this requirement. That is, if no data streams are identified in the input map, then no package files are added and no error occurs. |
| source       | Optional. The name of the input data stream provided by the client. This name can be specified with an External Data URL. (See “External Data URL” on page 150.)  
  This stream maps to the data stream included in the PDF package or portfolio. The data stream can contain content of any type. |
| sourceMatch  | Optional, but required if the source attribute is not specified. The value is a regular expression pattern that selects source names and their associated data streams from the input map or URL.  
  - **Source specifies an input map.** If source specifies a non-URL name and sourceMatch is specified, sourceMatch is used only when the source attribute does not match an entry in the input map or URL.  
  - **Source specifies a URL.** If the source attribute specifies a URL that references a folder of files, then sourceMatch can select specific files from the folder.  
  The regular expression syntax is a standard regular expression syntax as implemented in the java.util.regex class for Java.  
  Depending on the matchMode attribute, the matched documents are either included or excluded in the assembled document. If more than one name matches, the names are sorted, as specified in the sortOrder and sortLocale attributes.  
  The string value can be specified with an External Data URL. |

**See also**  
“External Data URL” on page 153  
“Specifying multiple input streams” on page 32
PackageFiles filter elements

As a filter element, PackageFiles allows only certain package files and their package specifications to be included.

```xml
<PackageFiles
    nameKeys="unspecified" or "xs:string"
>
    <PDF> source [1..n]
    <FieldData> [0..n]
    <TargetLocale> [0..1]
    ...
</PackageFiles>
```

Can be contained in the PDF result element.

If the PDF source element provides a single PDF document, then the PackageFiles filter element converts that file into a package file in the resultant document. Such conversion occurs only when the document is a single PDF document. Additionally, the DDX processor generates a unique nameKey for the file.

If the PDF source element provides a PDF package or portfolio, all its package files are included as package files in the resultant document. If the nameKeys attribute is provided, only those package files that match an entry in the nameKeys attribute are included.

A PDF package or portfolio document within a PackageFiles element contributes only the package files to the resultant document. It does not contribute cover sheets.

To specify a filename, use the FieldData element and provide a value for the _Filename built-in key.

For each file added to the package files in the resultant document, the DDX processor creates a unique nameKey property. To find out what these nameKey values are, use a PackageFiles result element.

**Note:** If you use the PackageFiles filter element to add an encrypted file, provide the file’s open password. (See the OpenPassword element.) If you omit the open password, the DDX processor throws an exception and the resultant document is not created. The open password is required because the document must be opened to determine whether it is a single PDF document or a PDF package or portfolio. The alternative is to use a PackageFiles source element to blindly add the file as a package file. The PackageFiles source element adds PDF and non-PDF documents to a PDF Portfolio or package.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameKeys</td>
<td>Optional. The value is a single nameKey or a comma-separated list of nameKeys. The entries in this attribute select package files from the PDF package or portfolio in the child PDF source element. Each package file in a PDF package or portfolio has a unique nameKey value. This value can be the filename or a variation of the filename. To determine the values to use in this nameKeys attribute, use the PackageFiles result element to export information about the package files.</td>
</tr>
</tbody>
</table>
PackageFiles select elements

As a select element, PackageFiles allows only certain package files and their package specifications to be modified. It never adds additional package files nor does it turn a single PDF document into a PDF package or portfolio.

When modifying the package file, the select element functions as an edit-in-place feature that works on package files in the parent PDF package or portfolio. It is the only `<PackageFiles>` variant that can be a child of the PDF source element.

The nameKeys attribute is used to select specific package files from the parent PDF package or portfolio. Each package file in a PDF package or portfolio has a unique nameKey value. This value can be the filename or a variation of the filename. To determine the values to use in this nameKeys attribute, use the PackageFiles result element to export information about the package files.

A PackageFiles select element is different from a PackageFiles filter element in that the select element does not contain a PDF source from which to filter. Instead, the package files are selected from package files contained in the parent element. If the parent element is not a PDF package or portfolio, then nothing is selected and a warning is logged.

If selected package files are not PDF documents, or are not modifiable PDF documents, then they are left unchanged and a warning is logged.

```xml
<PackageFiles
    nameKeys="unspecified" or "xs:string"
>
    <FieldData> [0..n]
        ... PackageFiles modifying elements ... 
    </FieldData>
</PackageFiles>
```

For more information on the FieldData element, see “FieldData” on page 180.

Can be contained in PDF result and PDF source.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nameKeys</td>
<td>Optional. The value is a single nameKey or a comma-separated list of nameKeys. The entries in this attribute select package files from the package or portfolio assembled for the PDF result element. Each package file in a PDF package or portfolio has a unique nameKey value. This value can be the filename or a variation of the filename. To determine the values to use in this nameKeys attribute, use the PackageFiles result element to export information about the package files.</td>
</tr>
</tbody>
</table>

PackageFiles result elements

Returns an XML file containing information about the package files. Can also return the package files.

```xml
<PackageFiles
    result="xs:string"
    extract="true" or "false"
    nameKeys="unspecified" or "xs:string"
>
```
Can be contained in the **DDX** element, which is the DDX root.

The **PackageFiles** result element returns an XML document that provides information about the package files. The information includes the files' metadata and the unique name (nameKey) mapped to the output data stream. The XML document conforms to the PackageFiles schema. (See “**PackageFiles Language**” on page 350.)

**Note:** If the **PDF** source element contains a simple PDF document, then the root element in the resultant XML PackageFiles document is empty. File attachments are not considered package files. If the **PDF** source element contains a PDF package or portfolio but no package files are selected, then the root element contains only the package specification.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>extract</strong></td>
<td>Optional. If its value is &quot;true&quot;, all package files specified are returned as separate data streams to the client. If its value is &quot;false&quot;, then only the result XML data, containing information about the package files specified, is returned.</td>
</tr>
</tbody>
</table>
| **result** | Required. A name to be associated with the returned data stream. The stream contains the XML data that provides the mapping of the data stream name to information stored with the package file. That information includes the nameKey, filename, creation date, MIME type, and any other custom metadata (see “**PackageFiles Language**” on page 350).

The value of this attribute must be unique among other result elements in the same DDX document. The result can be specified with an External Data URL. (See “**External Data URL**” on page 153.)

The name is not a filename and should not be treated as such by the client. |
| **nameKeys** | Optional. The nameKeys attribute identifies package files to include in the result XML and extracted if extract="true". The value is either a single nameKey or a comma-separated list of nameKeys. The default identifies all package files.

You can discover the names in a PDF package or portfolio by using the **PackageFiles** result element with the nameKeys attribute omitted. |

**PackageFiles import elements**

Imports package files provided as a string that represents an XML version of the package files.

```xml
<PackageFiles import="xs:string" />
```

Can be contained in the **PDF** result element.

The value of the import attribute is an XML PackageFiles document. Typically, using the XML PackageFiles document involves these steps:
1. **PackageFiles** result element produces an XML PackageFiles document. This file represents the package files for a PDF package or portfolio. The **PackageFiles** result element can optionally return the package files.

2. An external process modifies the XML PackageFiles document.

3. **PackageFiles** import element updates the package file with the changes. All package files listed in that XML document must be provided as inputs. Each input document must correspond to a unique name (nameKey) described in the XML document.

To prevent imported package files from replacing existing package files, modify the XML to remove the nameKeys. If no nameKey is present in the XML file for a package file, a new, unique nameKey is automatically generated.

**Note:** The automatically generated nameKey values are unique only within a document. Because the DDX processor automatically generates nameKey values where needed, different PDF documents can have package files with the same nameKeys. The nameKey values can be the same, even though there is no relationship between the files they identify.

The DDX processor merges the data in the XML PackageFiles document with the existing PDF package or portfolio. The major parts of the XML PackageFiles document are package specification, package files, and folders.

**Package specifications.** The specification is merged with the existing package specification. (See “Creating a package or portfolio specification by aggregating existing ones” on page 57.)

**Package files.** This category includes file identifiers and FieldData (metadata). The XML file can contain FieldData elements for each file. Those FieldData elements provide metadata for each package file. When you import such an XML file, the FieldData elements are imported into the package file FieldData elements. The exception is that the DDX processor omits metadata that is not defined in the schema in the package file schema. That schema is aggregated with the imported XML PackageFile.

**Folders.** Folders are merged with folders in the existing package specification.

**Note:** If all package files listed in the import XML are missing from the input map, then an error is thrown. If only some package files listed are missing, a warning is logged for each missing package file, while the rest are imported without error. Also, if multiple **PackageFiles** import elements are specified, an exception is thrown.

**See also**

“**PackageFiles**” on page 215

“**Modifying the package files in a PDF package or portfolio**” on page 59

“**PackageFiles Language**” on page 350
PageContent

(Since 8.2) Adds content to a page similar to Watermark but with alternate text and style profiles.

```xml
<PageContent
   alternateText="xs:string"
   alternation="None or OddPages or EvenPages"
   appears="Behind or OnTop"
   fitToPage="true or false"
   horizontalAnchor="Left or Center or Right"
   horizontalOffset="0pt or length-specifier"
   opacity="100% or percentage-specifier"
   rotation="0 or xs:integer"
   scale="100% or percentage-specifier"
   showOnScreen="true or false"
   showWhenPrinting="true or false"
   verticalAnchor="Top or Middle or Bottom"
   verticalOffset="0pt or length-specifier"
>
   <StyledText> or <PDF> source [1]
   <TargetLocale> [0..1]
</PageContent>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, BlankPage, and StyleProfile.

The PageContent element is similar to the Watermark element. It differs from the Watermark element in the following ways:

There is no replacement or removal of PageContent content.

Alternate text can be provided for screen readers to read for pages in the document that are already tagged. If a document is tagged, adding content to it does not cause screen readers to read the content. Hence, it is recommended that alternate text be provided for text and graphics added with the PageContent element.

Multiple PageContent elements can be specified per page.

With tagged PDF documents, screen readers receive the string values of non-empty alternateText attributes in place of the content. It is recommended that the text provided in the alternateText attribute match the content, with the addition of text to represent any graphical content. This recommendation does not apply if the content is purely graphical.
The anchor points and offset attributes describe the placement of the PageContent.

**Note:** Adding a PageContent element with alternateText does not promote untagged pages to tagged pages, even if assembled with a structured PDF. However, adding pages from unstructured PDF documents to structured PDFs does promote the new pages to structured PDF. Also, using the BlankPage and TableOfContents elements to add content to a structured PDF promotes the new pages to structured PDF.

The PageContent element can contain either one StyledText or one PDF source element, but not both. The first page from the pages specified by the PDF source is used for the page content.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternateText</td>
<td>Optional. If a non-empty string is provided, and the PDF is tagged, then this string is passed to a screen reader in place of the content. A structured PDF document is tagged.</td>
</tr>
<tr>
<td>alternation</td>
<td>Optional. If None, applies to all pages. If OddPages, applies to odd pages only. If EvenPages, applies to even pages only.</td>
</tr>
<tr>
<td>appears</td>
<td>Optional. Determines whether the new content appears behind or on top of the current page content. For tagged PDF documents, this attribute can affect whether the alternateText (if present) is spoken before existing page content or after. The default value (Behind) can cause screen readers to read the alternateText before the existing page content. There is no guarantee of the order in which the alternateText and existing page content is read.</td>
</tr>
<tr>
<td>fitToPage</td>
<td>Optional. If true, the scale attribute is ignored and the PageContent is guaranteed to fit to the boundaries of the PageSize element. The fit is accomplished by expanding or shrinking the text.</td>
</tr>
<tr>
<td>horizontalAnchor</td>
<td>Optional. The horizontalOffset is relative to the horizontalAnchor. Left is the left PageMargin. Center is the center of the page. Right is the right PageMargin.</td>
</tr>
<tr>
<td>horizontalOffset</td>
<td>Optional. The offset from the horizontalAnchor point. A positive value moves right, while a negative value moves left.</td>
</tr>
<tr>
<td>opacity</td>
<td>Optional. Controls the transparency of the PageContent text. The value of this attribute can have the following forms:</td>
</tr>
<tr>
<td></td>
<td>Decimal in the range of .0 to 1.0</td>
</tr>
<tr>
<td></td>
<td>Percentage in the range of 0% to 100%. In this case, the percentage sign (%) is required.</td>
</tr>
<tr>
<td></td>
<td>The default is 100%.</td>
</tr>
<tr>
<td>rotation</td>
<td>rotation Optional. The valid range is -360 to 360 degrees.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional. The valid range is 8 to 3200 percent.</td>
</tr>
<tr>
<td>showOnScreen</td>
<td>Optional. A Boolean value that controls whether the PageContent is displayed when pages are viewed within an application such as Acrobat.</td>
</tr>
</tbody>
</table>
The following example shows how to reference a `PageContent` element contained within a named `StyleProfile` element:

```xml
<PageContent styleReference="xs:string"/>
```

In the following example, `doc1` is a tagged PDF document.

```xml
<PDF result="doc2">
  <PageContent appears="Behind" alternateText="This is highly Confidential.">
    <StyledText><p><This is highly <graphic source="AdobeConfLogo.pdf"/></p></StyledText>
  </PageContent>
  <PDF source="doc1"/>
</PDF>
```

### PageLabel

Specifies the format and content of page labels, where the labels show the page number preceded by a prefix.

```xml
<PageLabel
  mode="Define" or "Preserve" or "Continue"
  start="1" or "ordinal-specifier" or "_PageNumber"
  format="None" or "Decimal" or "LowerRoman" or "UpperRoman"
  or "LowerAlpha" or "UpperAlpha"
  prefix="empty-string" or "xs:string"
/>
```

Can be contained in the elements `PDF` result, `PDF` source, `PDFGroup`, `TableOfContents`, `TableOfContentsPagePattern`, `PackageFiles` filter, select, or conversion elements, and `BlankPage`.

The `PageLabel` element defines the page labels for the scope of the parent element.

A page label is an optional identifier for a page that has the form "prefix + page number". In Acrobat and Adobe Reader, page labels are displayed in the banner beneath the page and beneath thumbnails. Labels have the following form, where `prefix` is optional.

```
prefix + page number
```

Page label can be directly related to the ordinal page number, which starts with 1 and ends with the number of pages. Alternatively, page labels can be independent of the ordinal page number.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>showWhenPrinting</code></td>
<td>Optional. A Boolean value that controls whether the <code>PageContent</code> appears on the page when printed.</td>
</tr>
<tr>
<td><code>verticalAnchor</code></td>
<td>Optional. The <code>verticalOffset</code> is relative to the <code>verticalAnchor</code>. Top is the top PageMargin. Middle is the middle of the page and Bottom is the bottom PageMargin</td>
</tr>
<tr>
<td><code>verticalOffset</code></td>
<td>Optional. The offset from the <code>verticalAnchor</code> point. A positive value moves up, while a negative value moves down.</td>
</tr>
</tbody>
</table>
Preexisting page labels in the source document are replaced with the page labels defined in this element. Page labels not within the scope of a `PageLabel` element are left unchanged.

**Category**

“Page labels” on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| mode   | Optional. Specifies the source of page label characteristics. This attribute can have the following values:  
  Define (default) - Indicates that the other attributes in this element define the page label style.  
  Preserve - Describes the source document as keeping the existing page label style; pages are not renumbered as they are assembled.  
  Continue - Uses the page label style from the previous source document and renumbers the pages from the current source document as they are assembled.  
  If the previous document has no defined page label style, no page labels are generated for these pages.  
  A value of Continue is not valid for the first document in the assembly. |
| start  | Optional. Specifies the page number for the first page the source document contributes to the result document. The value can be an ordinal page number or the value of the built-in key `PageNumber`. This built-in key specifies the current page’s ordinal page number. (See “Built-in keys” on page 149.)  
If the `mode` attribute has a value other than Define, this attribute is ignored. |
| format | Optional. Specifies a page number format. This attribute can have the following values:  
  None - No page numbers are included in page label, even if a prefix is defined.  
  Decimal (default) - For example, 1, 2, 3, …  
  LowerRoman - For example, i, ii, iii, …  
  UpperRoman - For example, I, II, III, …  
  LowerAlpha - For example, a, b, c, …  
  UpperAlpha - For example, A, B, C, …  
If the `mode` attribute has a value other than Define, this attribute is ignored. |
| prefix | Text displayed before the number. The string value can be specified with an External Data URL.  
If the `mode` attribute has a value other than Define, this attribute is ignored. |

**PageMargins**

Specifies margins for page content elements being added to a page.

```xml
<PageMargins
  left="36pt" or "nonnegative length"
```
Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

The margins specified by this element affect only content being added to a page in the resulting PDF document through the page content elements. These elements include Watermark, Background, Header, Footer, PageContent, and TableOfContents.

The PageMargins element has no affect on the preexisting content of the page. Margins can be specified differently for odd pages and even pages through the alternation attribute. There can be only one PageMargins element specified per page side. This limitation means that there can be two PageMargins elements for a document only if one is for odd pages and one is for even pages.

The PageMargins element defines a reference for initial placement of text. It does not provide a boundary to which text is restricted. In other words, the text is not clipped to the margin. As a result, content provided in the page content elements Watermark, Background, Header, Footer, and TableOfContents can exceed the margins specified in this element. For example, excessive text in the Left element of a Header element can overflow the right margin. In another example, excessive lines in a Center element of a Header element can cause the header to exceed the bottom margin.

This element’s attributes specify margins relative to the page size (provided in the PageSize element).

**Category**

“Page properties” on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Optional. Width of left margin, which is the distance between the left side of the page and the left side of the page contents. A negative value defaults to 0. Default is 36 points (0.5 inches).</td>
</tr>
<tr>
<td>top</td>
<td>Optional. Width of top margin, which is the distance between the top of the page and the top of the page content. A negative value defaults to 0. Default is 36 points (0.5 inches).</td>
</tr>
<tr>
<td>right</td>
<td>Optional. Width of right margin, which is the distance between the right side of the page and the right side of the page content. A negative value defaults to 0. Default is 36 points (0.5 inches).</td>
</tr>
</tbody>
</table>
PageOverlay

Provides content for placement over the current page content.

```xml
<PageOverlay
    embedFormsAndAnnots="true" or "false"
    opacity="100%" or "percentage"
    scale="1.0" or "percentage"
    newX="0pt" or "length"
    newY="0pt" or "length"
    rotate90= "degrees in increments of 90"
>
  < PDF source> [1]
</PageOverlay>
```

Can be contained in elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element, the PageContent element, and the PageUnderlay element differ from other page content elements in that the content cannot be distinguished from other page content. Unlike content provided by Header or Watermark elements, there is no way to specify the removal of an overlay or underlay in DDX. There can be multiple overlays or underlays and all are applied within their scope.

A parent document can contain multiple overlays and underlays, all of which are applied to each page in the parent document.

Only page content is used. That is, bookmarks, page properties, and page labels from the source are not incorporated into the parent document.

Forms and annotations (including links) are included only when the embedFormsAndAnnots attribute is set to true. When form fields are copied, form-level JavaScript segments associated with form fields are preserved. Document-level JavaScript segments in the source are not preserved.

The page overlay's visible page size defines its visible content in the resultant document. The orientation of the overlay is relative to the lower left corner of the PageSize of both the overlay source page and the destination page.
The **PageOverlay** and **PageUnderlay** elements contain a source **PDF** element from which the overlay or underlay pages are obtained. The pages to overlay or underlay are applied sequentially to all the pages in the destination page set. One successive overlay or underlay page (source pages) is applied per successive destination page. Mismatches can occur between the number of pages in the source pages and the number of pages in the destination document. Here are the considerations DDX processors use to resolve such mismatches:

**A single-page overlay.** In this case, the overlay is repeatedly applied to all pages in the set of destination pages. A single-page overlay can be a single-page PDF source document. It can also be a multi-page PDF source document that includes a **pages** attribute that specifies a single page.

**A multi-page overlay that provides more pages than the destination pages.** In this case, the overlay pages are repeated in full or in part.

**A multi-page overlay that provides fewer pages than the destination pages.** In this case, only the needed overlay pages are applied. The unneeded pages are skipped.

**Category**

*[Page content* on page 148]

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>embedFormsAndAnnots</code></td>
<td>Optional. Specifies whether to include any existing form fields and annotations (including links) from the source page when overlaying or underlaying on the destination pages.</td>
</tr>
<tr>
<td><code>newX</code></td>
<td>Optional. The value, in user space, for shifting the coordinates of the origin from (0,0) (the lower left corner of the visible page). A positive value shifts the origin to the right, while a negative value shifts the origin to the left.</td>
</tr>
<tr>
<td><code>newY</code></td>
<td>Optional. The value, in user space, for shifting the coordinates of the origin from (0,0) (the lower left corner of the visible page). A positive value shifts the origin up, while a negative value shifts the origin down.</td>
</tr>
</tbody>
</table>
| `opacity`                 | Optional. Controls the transparency of the source page before overlaying or underlaying the destination pages. The value of this attribute can have these forms:  
  - Decimal in the range of .0 to 1.0  
  - Percentage in the range of 0% to 100%. In this case, the percentage sign (%) is required.  
  The default value is 100%. |
| `rotate90`                | Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number rotates clockwise and a negative number rotates counterclockwise. |
| `scale`                   | Optional. The factor by which to scale the page. |
PageRotation

Specifies the rotation of the pages within the scope of the parent element.

```xml
<Rotation
   rotate90="degrees in increments of 90"
/>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, PackageFiles filter, select, or conversion elements, and BlankPage.

The PageRotation element specifies the rotation angle of the pages within the scope of the parent element. The rotation specified in this element overrides any existing page rotation, either from a preceding DDX element or from preexisting settings in the page itself. That is, page rotation settings are not additive.

The PageRotation element determines the viewer orientation of the page. It does not affect the relationship of the page content to the media. For example, consider a page whose media has a landscape orientation and whose text is placed according to that orientation. If that page is rotated 90 degrees, the resulting viewed page has a portrait orientation and its text appears sideways. To rotate the page content, see the Transform element.

The PageRotation element affects how new page content is placed on the page. The view of the page after applying the settings in this element determines the page's upper edge for adding a Header, Watermark, or other page content.

The DocumentInformation query element can be used to obtain the PageRotation values for all pages in a document.

**Category**

"Page properties" on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate90</td>
<td>Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number is clockwise, and a negative number is counterclockwise.</td>
</tr>
</tbody>
</table>

PageSize

Defines the page dimensions for purposes of display or print.

```xml
<PageSize
   width="612pt" or "positive page width"
   height="792pt" or "positive page height"
   horizontalAnchor = "Left" or "Center" or "Right"
   scaleDown = "false" or "true"
   scaleUp = "false" or "true"
   select="Auto" or "All" or "Landscape"
   verticalAnchor = "Top" or "Middle" or "Bottom"
/>
```
Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

The PageSize element sets the width and height of the page, as viewed in Acrobat. If the page size changes, the page content can be scaled or cropped, or white space can be added. If scaling is called for with the scaleUp or scaleDown attributes, the aspect ratio of the page contents is retained. That is, scaling is performed equally in the horizontal and vertical dimensions, and therefore the smaller dimension constrains scaleUp.

The anchor attributes taken together specify the point in the original page that stays constant. For example, if the following conditions occur, the page is cropped or expanded equally around its midpoint:

- Scaling is not specified.
- Horizontal and vertical anchors have respective values of Center and Middle.

More specifically, if the new page size is smaller than the existing page’s size, the page is cropped equally about its midpoint. If it is larger, the page is expanded equally in each direction about its midpoint. (See When the PageSize element is a child of a TableOfContents element, it serves only to describe the size of the table of contents.

The Document Manipulation APIs can swap the values of the width and height attributes to be consistent with the page orientation specified in the sibling PageRotation element. (See “Page size and rotation” on page 97.)

The DocumentInformation query element can be used to obtain the PageSize values for all pages in a document.

**Category**

“Page properties” on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Optional. Specifies the horizontal boundary of the page content, which correlates to the width of the target media. Default is letter size width: 612 pts (8.5 inches). The value of width must be greater than 0.</td>
</tr>
<tr>
<td>height</td>
<td>Optional. Specifies the vertical boundary of the page content, which correlates to the height of the target media. Default is letter size height: 792 pts (11.0 inches). The value of height must be greater than 0.</td>
</tr>
</tbody>
</table>
| scaleUp  | Optional. Specifies whether the page’s contents are scaled if both dimensions of the specified page size are larger than the corresponding dimensions of the current page size. This attribute can have the following values:  

  - false (default) - White space is added around the existing page content to accommodate increases in page size.  
  - true - The page’s contents are scaled up until the new page size is filled along either dimension. The page content’s aspect ratio is retained. |
### PageUnderlay

Provides content for placement under the current page content.

```xml
<PageUnderlay
    embedFormsAndAnnots="true" or "false"
    opacity="100%" or "percentage"
    scale="1.0" or "percentage"
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| scaleDown  | Optional. Specifies whether the page’s contents are scaled if either dimension of the specified page size is smaller than the corresponding dimension of the existing page size. This attribute can have the following values:  
  - false (default) - The page’s contents are cropped to the new page’s size.  
  - true - The page’s contents are scaled down to fit within the new page size. The page content’s aspect ratio is retained. |
| select     | Optional. Selects which pages are resized as specified by this element’s width and height attributes. This attribute can have the following values:  
  - Auto (default). The width and height values are applied while maintaining each page’s orientation as portrait or landscape, even if that means swapping the specified width and height.  
  - All. All the pages are set to the specified width and height regardless of their original width and height value. As a result, one edge of a page can be clipped while another is expanded.  
  - Portrait. Only pages whose original width is less than or equal to its height have their width and height values set to the specified values. As a result, one edge of a page can be clipped while another is expanded.  
  - Landscape. Only pages whose original height is less than its width have their width and height values set to the specified values. As a result, one edge of a page can be clipped while another is expanded. |
| horizontalAnchor | Optional. Specifies which portion of the page stays anchored in the horizontal direction when the content is scaled, as specified in scaleUp and scaleDown. This attribute can have the following values:  
  - Left  
  - Center (default)  
  - Right |
| verticalAnchor | Optional. Specifies which portion of the page stays anchored in the vertical direction when the content is scaled, as specified in scaleUp and scaleDown. This attribute can have the following values:  
  - Top  
  - Middle (default)  
  - Bottom |
Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element and the PageOverlay element differ from other page content elements in that the content cannot be distinguished from other page content. Unlike content provided by Header or Watermark elements, there is no way specify the removal of an overlay or underlay in the DDX document.

A parent document can contain multiple overlays and underlays, all of which are applied to each page in the parent document.

Only page content from the source page is used. That is, bookmarks, page properties, and page labels are not incorporated into the parent document.

Forms and annotations (including links) are included only when the embedFormsAndAnnots attribute is set to true. When form fields are copied, form-level JavaScript segments associated with form fields are preserved. Document-level JavaScript segments in the source are not preserved.

The page underlay's visible page size defines its visible content in the resultant document. The orientation of the underlay is relative to the lower left corner of the PageSize of both the underlay source page and the destination page.

The PageOverlay and PageUnderlay elements contain a source PDF element from which the overlay or underlay pages are obtained. The pages to overlay or underlay are applied sequentially to all the pages in the destination page set. One successive overlay or underlay page (source pages) is applied per successive destination page. Mismatches can occur between the number of pages in the source pages and the number of pages in the destination document. Here are the considerations DDX processors use to resolve such mismatches:

**A single-page underlay.** In this case, the underlay is repeatedly applied to all pages in the set of destination pages. A single-page underlay can be a single-page PDF source document. It can also be a multi-page PDF source document that includes a pages attribute that specifies a single page.

**A multi-page underlay that provides more pages than the destination pages.** In this case, the underlay pages are repeated in full or in part.

**A multi-page underlay that provides fewer pages than the destination pages.** In this case, only the needed underlay pages are applied. The unneeded pages are skipped.

**Category**

"Page content" on page 148
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>embedFormsAndAnnots</td>
<td>Optional. Specifies whether to include any existing form fields and annotations (including links) from the source page when overlaying or underlaying on the destination pages.</td>
</tr>
<tr>
<td>newX</td>
<td>Optional. The value, in user space, for shifting the coordinates of the origin from (0,0) (the lower left corner of the visible page). A positive value shifts the origin to the right, while a negative value shifts the origin to the left.</td>
</tr>
<tr>
<td>newY</td>
<td>Optional. The value, in user space, for shifting the coordinates of the origin from (0,0) (the lower left corner of the visible page). A positive value shifts the origin up, while a negative value shifts the origin down.</td>
</tr>
</tbody>
</table>
| opacity                     | Optional. Controls the transparency of the source page before overlaying or underlaying on the destination pages. The value of this attribute can have the following forms:  
  - Decimal in the range of .0 to 1.0  
  - Percentage in the range of 0% to 100%. In this case, the percentage sign (%) is required.  
  The default is 100%. |
| rotate90                    | Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number rotates clockwise and a negative number rotates counterclockwise. |
| scale                       | Optional. The factor by which to scale the page. |

Password

Provides an access password for the Document Manipulation APIs to use to open encrypted source documents. You can specify either the open or master password, depending on which value you know and your requirements.

```xml
<Password>password</Password>
```

Can be contained in the `PasswordAccessProfile` element.

Category

“Page properties” on page 147

PasswordAccessProfile

Specifies a named profile containing an access password.

```xml
>PasswordAccessProfile
   name="xs:string"
>
   <Password>
```
Can be contained in the DDX element, which is the DDX root.

The password specified in this element allows the Document Manipulation APIs to open password-protected source documents. Such documents can be opened only after an access password is supplied.

The password supplied in this profile must be the owner password (master password) used in those source documents. This password enables the Document Manipulation APIs to have unrestricted access to those documents.

The password specified in this element does not change the password protection specified in the result document.

**Category**

"Profile" on page 148

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. The name of the PasswordAccessProfile element. The PDF source element contains an access attribute that can reference the password contained in this element.</td>
</tr>
</tbody>
</table>

**PasswordEncryptionProfile**

Specifies a named profile containing password security settings for the result document.

```
<PasswordEncryptionProfile
  name="xs:string"
  compatibilityLevel="Acrobat3 or Acrobat5 or Acrobat6 or Acrobat7 or Acrobat9"
  encryptionLevel="All or NotMetadata or OnlyFileAttachments"
>
  <OpenPassword>[0..1]
  <Permissions> [0..1]
</PasswordEncryptionProfile>
```

The default for the compatibilityLevel is Acrobat5.

Can be contained in the DDX element, which is the root element.

Password protection affects access to PDF documents by limiting what users can do with the file, depending on the password they provide. If an open password is specified for a document, a user must provide a password to open the document. The user can perform only those tasks allowed in the document permissions. The OpenPassword element specifies an open password.

If the user provides the owner password (specified in the MasterPassword supplement of the Permissions element), the user can change the permissions. If the PasswordEncryptionProfile element applies only an owner password, a user can open the document without providing a password. However, that user can perform only those tasks allowed in the document permissions.
The PDF element is used to describe PDF documents, as described below:

A **PDF** result element has a **result** attribute. It describes the resultant document.

A **PDF** source element has a **source** or **sourceMatch** attribute. It describes the content to be assembled into a **PDF** result element.
PDF result

Describes the resultant PDF document in terms of other documents and content. When the result is a PDF package or portfolio, any child elements of PDF source affecting the content only apply to the resultant document’s cover sheet. Those child elements not apply to the package files. The PackageFiles element can be used to affect the content of a package file when it is a modifiable PDF document. PackageFiles specified with NoPackage or NoPortfolio can supply the required PDF source when at least one of the package files is a PDF document. In all cases, the PDF result must contain PDF pages.

A PDF result element contains elements and attributes that describe the contents of a resultant PDF document. A PDF result element cannot be included within any other result element and must contain at least one PDF source element. The constituent PDF source elements can be nested within a PDFGroup or Folder element.

```xml
<PDF
    result="pdf output name"
    certification="unspecified" or "None"
    encryption="unspecified" or "None" or "PasswordEncryptionProfile_name"
    format="PDF" or "XDP"
    initialView="unspecified" or "InitialViewProfile_name"
    mergeLayers="true" or "false"
    pdfa = "xs:string"
    readerUsageRights="unspecified" or "None"
    return="true" or "false"
    save="unspecified" or "Incremental" or "Full" or "FastWebView"
    sortBookmarks="true" or "false"
    optimizeFont="true" or "false"
/>
```

```xml
<ArtBox> [0..2, where 2 is allowed for alternating pages]
<Author> [0..1]
<Background>
    or <NoBackgrounds> [0..2, where 2 is allowed for alternating pages]
<BlankPage> [0..n]
<BleedBox> [0..2, where 2 is allowed for alternating pages])
<Bookmarks source> and/or <Bookmarks filter>)[0..n] or <NoBookmarks> [0..1]
<Comments source> and/or <Comments filter>)[0..n] or <NoComments> [0..1]
<DDXProcessorSetting> [0..1]
<FileAttachments source> [0..n] or <NoFileAttachments> [0..1]
<FilenameEncoding> [0..n]
<FileSize> [0..1]
<Folder> [0..1]
<Footer> or <NoFooters> [0..2, where 2 is allowed for alternating pages]
<JavaScript> [0..n] or <NoJavaScripts> [0..1]
<Keywords> [0..1]
(Links source and/or <Links filter>) [0..n] or <NoLinks> [0..1]
_Metadata source> [0..1]
<NoForms> [0..1]
    <NoThumbnails> [0..1]
<NoXFA> [0..1]
.Package> or <NoPackage> [0..n]
<PackageFiles> [0..n] or <NoPackageFiles> [0..1]
<PageContent> [0..n]
<PageLabel> [0..1] or <NoPageLabels> [0..1]
<PageMargins> [0..2, where 2 is allowed for alternating pages]  
<PageOverlay> [0..n]  
<PageRotation> [0..1]  
<PageSize> [0..1]  
<PageUnderlay> [0..n]  
<PDF source> and/or <PDFGroup> [1..n]  
<PDFGenerationSettings> [0..1]  
<RichMedia source> [0..1]  
<Portfolio> [0..1] or <NoPortfolio> [0..1]  
<ReaderRights> [0..1]  
<Subject> [0..1]  
<table of contents> [0..1]  
>Title> [0..1]  
<Transform> [0..1]  
<TrimBox> [0..2, where 2 is allowed for alternating pages]  
<Watermark> or <NoWatermarks> [0..2, where 2 is allowed for alternating pages]  
<XDP> [0..1]  
<XFAConversionSettings> [0..1]  
</PDF>

Can be contained in the DDX element, which is the root element.

Category

"Document assembly" on page 143

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>Required. A name to be associated with the assembled PDF document. This name must be unique among all result elements in the DDX document. The result can be specified with an External Data URL. (See “External Data URL” on page 153.)</td>
</tr>
<tr>
<td>format</td>
<td>Optional. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>PDF (default) - Result document is returned as a PDF document.</td>
</tr>
<tr>
<td></td>
<td>XDP - Result document is returned as an XML Data Packaging (XDP) document. This value can be used only if the base document is an XFA-based PDF document.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>return</td>
<td>Optional. Specifies whether the assembled PDF document is returned. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>true (default) - The assembled PDF document is returned to the client as a stream.</td>
</tr>
<tr>
<td></td>
<td>false - The assembled PDF document is available as transient data, which can be referenced as source from within a subsequent PDF result element. In this case, the assembled PDF document is not returned to the client.</td>
</tr>
<tr>
<td>initialView</td>
<td>Optional. Specifies the name of an InitialViewProfile element. If this attribute is not specified, the initial view from the base document is preserved in the result. This attribute is ignored for XFA-based documents.</td>
</tr>
<tr>
<td>save</td>
<td>Optional. Specifies how Acrobat save the result document. If specified, this attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>Incremental - Performs an incremental save. This means that changes to the document are placed at the end of the file, and the bytes corresponding to the original file are unchanged. Incremental saves are relative to the base document. That is, the bytes in the result stream begin with the original bytes of the base document, followed by updates.</td>
</tr>
<tr>
<td></td>
<td>Full - A full save is performed, overwriting any existing incremental saves. This option does not result in optimization for fast web viewing.</td>
</tr>
<tr>
<td></td>
<td>FastWebView - The PDF document is optimized and restructured for page-at-a-time downloading from web servers. This save results in the removal of any existing incremental saves.</td>
</tr>
<tr>
<td></td>
<td>If this attribute is not specified, the save characteristics depend on the PDF level. For PDF 1.4 and later, the DDX processor performs an incremental save, which is relative to the base document. For PDF 1.3 and earlier, the Document Manipulation APIs performs a full save.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> The removal of existing incremental saves invalidates any signatures, certification, or reader-enabled rights set on the base document.</td>
</tr>
<tr>
<td>encryption</td>
<td>Optional. If specified, this attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>PasswordEncryptionProfile name - This value is the name of the PasswordEncryptionProfile element with which to encrypt the result.</td>
</tr>
<tr>
<td></td>
<td>None - Specifies that the output must have no encryption regardless of whether the source base document was encrypted.</td>
</tr>
<tr>
<td></td>
<td>If this attribute is not specified and the baseDocument is encrypted, then the resultant document has the same encryption.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>readerUsageRights</td>
<td>Optional. If present, the value must be None, indicating that the document must have all Adobe Reader usage rights removed. If this attribute is not specified, any existing Adobe Reader usage rights are retained unless other changes to the document would invalidate its digital signatures. If the <code>&lt;ReaderRights&gt;</code> element is present and a Reader Extension service is available, then the specified Reader Rights overrides the value of None.</td>
</tr>
<tr>
<td>pdfa</td>
<td>Optional. Name of a PDFAProfile that specifies the type of PDF archive to which the resultant document conforms. A value of Default specifies that the default PDF/A profile is used. If not specified, the output does not conform to PDF/A.</td>
</tr>
</tbody>
</table>
| certification   | Optional. If specified, this attribute can have the following value:  
None - Indicates that the result must not contain a certifying (author) signature.  
If this attribute is not specified, the base document’s certifying signature is retained, if possible. The base document’s certifying signature cannot be retained in the following situations:  
If the certifying signature appears in a non-base document in an assembly. Also, the base document’s certification becomes invalid.  
If forms are removed in the result document, by specifying the NoForms element.  
The base document’s certifying signature is lost later in the document’s life if a user performs a full save on the document. (See the save attribute.) |
| mergeLayers     | Optional. Specifies how layers from different source documents are treated.  
This attribute can have the following values:  
false (default) - Layers from different source documents are kept distinct in the assembled document. If the layerLabel attribute in the source document is specified, it modifies the layer names. Otherwise, the layers names are unmodified.  
true - If there are name conflicts in layer names, layers with the same name are merged to make a single layer with that name. |
Identifies a source of PDF content from one or more input data streams. A PDF source element can be a single PDF document or a PDF package or portfolio. It can contain other elements that describe content and properties to replace corresponding items in the source content. A PDF source element must be contained within a PDF result element and cannot contain another PDF source element.

When the source is a PDF package or portfolio, the cover page and the package files are included. Any child elements of the PDF source affecting the content only apply to the cover sheet and not to the package files. The PackageFiles element can be used to affect the content of a package file if it is a modifiable PDF document.

A PDF source element can be an XDP document that contains PDF content. The XDP format is described in the XFA Specification, version 2.2 at http://www.adobe.com/go/learn_lc_XFA.

```
<PDF
    source="input name"
    sourceMatch="unspecified" or "regular expression"
    pages="1-last" or "page range"
    access="unspecified" or "PasswordAccessProfile name"
    baseDocument="true" or "false"
    bookmarkTitle="unspecified" or "xs:string"
    includeInTOC="true" [if document is after TOC in result document] or "false"
    includeSubFolders="true" or "false"
    layerLabel="unspecified" or "xs:string"
    matchMode="Include" or "Exclude"
    required="true" or "false"
    select="1-last" or "range specifier"
    sortLocale="xs:string"
    sortOrder="Ascending" or "Descending"
>
    <ArtBox> [0..2, where 2 is allowed for alternating pages]
    <Background>
        or <NoBackgrounds> [0..2, where 2 is allowed for alternating pages]
    <BleedBox> [0..2, where 2 is allowed for alternating pages]
    <Bookmarks source> and/or <Bookmarks filter> [0..n]) or
```
<NoBookmarks> [0..1]
<Comments source> and/or <Comments filter>) [0..n] or
<NoComments> [0..1]
<FileAttachments source> [0..n] or <NoFileAttachments> [0..1]
<Footer> or <NoFooters> [0..2, where 2 is allowed for alternating pages]
<Header>
or <NoHeaders> [0..2, where 2 is allowed for alternating pages]
<LinkAlias> [0..n]
<Links source> and/or <Links filter>) [0..n] or <NoLinks> [0..1]
<NoForms> [0..1]
<NoJavaScripts> [0..1]
<NoThumbnails> [0..1]
<XFA> [0..1]
-package> or <NoPackage> [0..n]
<PackageFiles> [0..n] or <NoPackageFiles> [0..1]
<PageContent> [0..n]
<PageLabel> [0..1] or <NoPageLabels> [0..1]
<PageMargins> [0..2, where 2 is allowed for alternating pages]
<PageOverlay> [0..n]
<PageRotation> [0..1]
<PageSize> [0..1]
<PageUnderlay> [0..n]
<PDFGenerationSettings> [0..1]
<Portfolio> or <NoPortfolio> [0..1]
<RichMedia> [0..1]
<TargetLocale> [0..1]
<Transform> [0..1]
<TrimBox> [0..2, where 2 is allowed for alternating pages]
<br> or <NoWatermarks> [0..2, where 2 is allowed for alternating pages]
<XFAConversionSettings> [0..1]
<XFAData> [0..1]
</PDF>

Can be contained in any of the following elements: PDF result, PackageFiles filter, conversion, or result, PDFGroup, Bookmarks result, Bookmarks filter, Links result, Links filter, Comments result, Comments filter, FileAttachments result, Left, Right, Center, Watermark, Background, PageOverlay, PageUnderlay, Metadata result, PDFsFromBookmarks, PageContent, DocumentText, and Portfolio.

**Category**

"Document assembly" on page 143
### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| source   | Optional, but required if the `sourceMatch` attribute is not specified. A logical name associated with a single input data stream, or an ordered list of data streams. The source can be specified with an External Data URL. See “External Data URL” on page 153.  
  
  **Note:** DDX processors, such as the Document Manipulation APIs, attempt to convert non-PDF input streams to PDF content.  
  
  If both source and sourceMatch attributes are specified, this attribute is used only if it matches a name entry in the input map. |
| sourceMatch | Optional, but required if the source attribute is not specified. The value is a regular expression pattern that selects source names and their associated data streams from the input map or URL.  
  
  **Source specifies an input map.** If source specifies a non-URL name and sourceMatch is specified, sourceMatch is used only when the source attribute does not match an entry in the input map or URL.  
  
  **Source specifies a URL.** If the source attribute specifies a URL that references a folder of files, then sourceMatch can select specific files from the folder.  
  
  The regular expression syntax is a standard regular expression syntax as implemented in the `java.util.regex` class for Java.  
  
  Depending on the `matchMode` attribute, the matched documents are either included or excluded in the assembled document. If more than one name matches, the names are sorted, as specified in the `sortOrder` and `sortLocale` attributes.  
  
  The string value can be specified with an External Data URL.  
  
  **See also**  
  “External Data URL” on page 153  
  “Specifying multiple input streams” on page 32 |
| pages    | Optional. Specifies which pages from the source document to include in the result document. The default value is 1-last, which signifies the entire document is included. (See “Page and document ranges” on page 157.)  
  
  **Note:** Document-level file attachments are assembled from a non-base document when the entire PDF document is part of the assembly. If only some pages from a non-base document are assembled, then the document-level file attachments for that PDF are not included. |
| access   | Optional. Specifies the name of the `PasswordAccessProfile` element to apply when opening the document. This attribute is relevant if the source document is encrypted. |
## baseDocument

Optional. Identifies the base document and provides the initial structure that the Document Manipulation APIs uses to set certain document-level properties of the result PDF document. These include document properties, form data, document-level JavaScript code, and viewer preferences. The result document can contain only one source identified as a base document. Documents other than the base document contribute only the following content and properties to the resultant document:

- Pages
- Document components (such as bookmarks, links, file attachments)
- Page labels
- Page content
- Page properties

Document-level components, such as file attachments, are only included from a document once, even if the document is specified multiple times.

A file mapped to a `baseDocument` is always required in the input map for the Document Manipulation APIs, even if the `required` attribute is set to "false".

This attribute can have the following values:

- **true** - Identifies the parent PDF source element as the base document.
- **false** (default) - Does not identify the parent PDF source element as the base document, though a base document is always required. If none of the source documents in a PDF result are specified as the base document, the Document Manipulation APIs determines the base document.

## bookmarkTitle

Optional. Structures the appearance of bookmarks in the result document by specifying a bookmark title that identifies the source document bookmarks. The string can contain built-in keys. The string value can be specified with an External Data URL that resolves to a string. See "External Data URL" on page 153.

If this attribute is omitted, no bookmark title precedes the source document bookmarks. If the `bookmarkTitle` attribute value is a URL, it is assumed to return a string attribute. The content and its length are not checked.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| includeInTOC     | Optional. Controls whether the source document is included in a table of contents created in the result document. This attribute is ignored if the PDF result element lacks a TableOfContents element. This attribute can have the following values:  
|                  | **true** (default) - Source document bookmarks are included in the table of contents.  
|                  | **false** (default) - Source document bookmarks are not included.  
|                  | **default value** - The default value of this attribute depends upon the location of the TableOfContents element within the PDF result element. All PDF source elements that appear before the TableOfContents element are assigned a default value of false as for this attribute. All PDF source elements that appear after use true as the default value. |
| includeSubFolders| Optional. If true, all files in the folder and subfolders are included. This results in a list of documents for the PDF source element that maintains the original folder structure. If false, only the files in the specified folder are included. |
| layerLabel        | Optional. Specifies the name for a top-level label under which all the layers of the source document are grouped. This label enables users of the document to distinguish between layers from different source documents and to avoid name conflicts. |
| matchMode         | Optional. Specifies whether the match results in the source document being included or excluded from the document assembly. This attribute can have the following values:  
|                  | **Include** (default) - Includes the matched data streams.  
|                  | **Exclude** - Excludes the matched data streams. |
| required          | Optional. The default (true) requires that the PDF source element add PDF content to the assembly. If it does not, the DDX processor declares an error.  
|                  | If set to false and no data streams are identified for this PDF source element, the PDF source elements adds no PDF content to the assembly. No error is declared. |
| select            | Optional. Determines which documents are selected when an ordered list of input streams is provided. The default value is 1-last, indicating that all streams be selected. For the syntax of specifying ranges, see "Page and document ranges" on page 157.
PDFGroup

Specifies a grouping of source documents to which page properties, page content, and document components can be applied.

<PDFGroup>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortLocale</td>
<td>Optional. Specifies the locale to use for sorting, according to sortOrder, names matched by the sourceMatch attribute. The value of this attribute must contain a valid 2-character ISO language code (see ISO 639). Any locale passes schema validation; however, if the requested locale is not available, a ValidationException is thrown. The default value for this attribute is obtained from the TargetLocale element.</td>
</tr>
<tr>
<td>sortOrder</td>
<td>Optional. If the regular expression specified in the sourceMatch attribute matches multiple documents, this attribute specifies the order in which those documents are sorted. The sort order is used to create an ordered list of documents. This attribute is not used if the source attribute matches an entry in the input map. This attribute can have the following values: Ascending (default) - Matched documents are sorted in ascending order: A-Z. Descending - Matched documents are sorted in descending order: Z-A.</td>
</tr>
</tbody>
</table>

sortLocale

Optional. Specifies the locale to use for sorting, according to sortOrder, names matched by the sourceMatch attribute. The value of this attribute must contain a valid 2-character ISO language code (see ISO 639). Any locale passes schema validation; however, if the requested locale is not available, a ValidationException is thrown. The default value for this attribute is obtained from the TargetLocale element.

goalOrder

Optional. If the regular expression specified in the sourceMatch attribute matches multiple documents, this attribute specifies the order in which those documents are sorted. The sort order is used to create an ordered list of documents. This attribute is not used if the source attribute matches an entry in the input map. This attribute can have the following values: Ascending (default) - Matched documents are sorted in ascending order: A-Z. Descending - Matched documents are sorted in descending order: Z-A.
PDFsFromBookmarks

Single PDF document is split into multiple PDF documents, based on top-level bookmarks.

<PDFsFromBookmarks
    encryption="None" or "xs:string"
    prefix="xs:string"
    save="Full" or "FastWebView"
>
    < PDF source> [1]
</PDFsFromBookmarks>

Can be contained in the DDX element, which is the DDX root.

The Document Manipulation APIs breaks the source document into multiple result documents. It uses the level-one bookmarks in the source document to determine where to split the source document. Only the first level-one bookmark on a page is considered.

If the source document is encrypted, the master password must be provided using the PasswordEncryptionProfile element.

The PDF documents produced for this element are named by concatenating the following text:

(value of the prefix attribute)+(auto-generated 6-digit sequence number)+(bookmark title)+(pdf)

If any of the characters used to construct a PDF document contain invalid file path characters, the Document Manipulation APIs replaces these characters with an underscore (_). The following characters are invalid file path characters:
Slash "/\nBackslash "\"
Colon ":" 
Ampersand ( * )
Question mark (?)
Quote ( " )
Left angle bracket (<)
Right angle bracket (>)
Bar ( | )

Any certifying signatures or reader-enabled rights in the source document are invalidated in the result PDF documents.

**Category**
"Document properties" on page 146

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encryption</td>
<td>Optional. Name of the PasswordEncryptionProfile element to use for encrypting the result document. This attribute can have the following values:</td>
</tr>
<tr>
<td>prefix</td>
<td>Required. Prefix to use in the names of the result documents. The DDX processor modifies any invalid characters in the name.</td>
</tr>
<tr>
<td>save</td>
<td>Optional. Specifies how the result documents are saved. This attribute can have the following values:</td>
</tr>
</tbody>
</table>

**PDFAProfile**

(Since 8.2) Provides settings for conversion to a PDF/A document. The PDF result element’s pdfa attribute references this element.

```xml
<PDFAProfile
  compliance="PDF/A-1b, PDF/A-2b, or PDF/A-3b"
  name="xs:string"
  resultLevel="PassFail or Summary or Detailed"
  signatures="ArchiveAsNeeded or AlwaysArchive">
  optionalContent = "Visible or All"
  verify="true or false"
</PDFAProfile>
```
Can be contained in the **DDX** element, which is the DDX root.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compliance</td>
<td>Optional. Compliance level for PDF/A output as either PDF/A-1b, PDF/A-2b, and PDF/A-3b. The default is PDF/A-1b.</td>
</tr>
<tr>
<td>name</td>
<td>Required. The name of the PDF/A profile.</td>
</tr>
<tr>
<td>resultLevel</td>
<td>Optional. Specifies the reporting level from the conversion effort. A value of <em>PassFail</em> reports whether the conversion was successful for the compliance and options that were specified. A value of <em>Summary</em> is similar to Acrobat's PDF/A Preflight results in how it reports a count of each error encountered. Detailed attempts to provide enough information to find constructs in the PDF source that cause violations. There is no ordering of the detailed violations. The default is PassFail.</td>
</tr>
<tr>
<td>optionalContent</td>
<td>Indicates how the optional content within the PDF is handled. A PDF/A document does not allow optional content, so it must either be removed or converted to content. A value of <em>Visible</em> converts all visible optional content to content and remove the remaining optional content. A value of <em>All</em> converts all optional content to visible content regardless of the original optional content's visibility. The default is Visible.</td>
</tr>
<tr>
<td>signatures</td>
<td>Optional. Specifies whether to archive signatures. The default is <em>ArchiveAsNeeded</em>. This setting specifies that signatures are left intact unless it causes the conversion to fail. In cases where an incremental save is possible, it is also possible to leave the signatures intact.</td>
</tr>
<tr>
<td>verify</td>
<td>Optional. If <em>verify</em> is true, the conversion saves the converted PDF/A document and then runs the validation against it. If false, the conversion reports the success of the conversion based only on unfixable errors found. The default is true.</td>
</tr>
</tbody>
</table>

### PDFAValidation

Used to validate PDF/A conformance.

```xml
<PDFAValidation>
  <allowCertificationSignatures>true or false</allowCertificationSignatures>
  <compliance("PDF/A-1b", PDF/A-2b, or PDF/A-3b" ignoreUnusedResources="true or false"
```
resultLevel="PassFail or Summary or Detailed"

Reports whether an input PDF document is PDF/A compliant. This element can have the following attribute:

Can be contained in the DocumentInformation element.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowCertificationSignatures</td>
<td>Determines whether the PDF document is PDF/A compliant. Valid values are true, false, and notvalidated. Default value: true</td>
</tr>
<tr>
<td>compliance</td>
<td>Specifies the PDF/A compliance level. Valid values are PDF/A-1b, PDF/A-2b, or PDF/A-3b. Default value: PDF/A-1b</td>
</tr>
<tr>
<td>ignoreUnusedResources</td>
<td>Specifies whether to ignore resources that are not used. Valid values are true or false. Default value: true</td>
</tr>
<tr>
<td>resultLevel</td>
<td>Optional. Specifies the reporting level from a hypothetical conversion effort. PassFail reports whether the conversion would be possible for the compliance and options that were specified. Summary is similar to Acrobat’s PDF/A Preflight results in how it reports a count of each error encountered. Detailed attempts to provide enough information to find the construct in the PDF which is causing a violation. Note there is no ordering of the detailed violations. Default value: PassFail.</td>
</tr>
</tbody>
</table>

Permissions

Specifies document permissions and a master password that enables those permissions to be changed.

<Permissions
   copy="Yes" or "No"
   edit="All" or "No" or "NotExtract" or "DocAssembly" or "FormFillinSign" or "CommentsFormFillinSign"
   <MasterPassword> [1]
   print="No" or "HighResolution" or "LowResolution"
   screenReading="Yes" or "No"
>
  </Permissions>

Can be contained in the PasswordEncryptionProfile element.

Note: The All value for the edit attribute has been deprecated. If a permission (master) password has been set on the document, then the least restrictive edit permission is NotExtract, which means any changes except extracting pages.
The Permissions element describes the actions that can be performed on the document. It also specifies a master password that allows the permissions to be changed.

Category

“Document properties” on page 146

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| print | Optional. Specifies the quality at which the document can be printed. This attribute can have the following values:  
  - HighResolution (default) - The document can be printed at any resolution.  
  - LowResolution - The document can be printed at no higher than 150 dpi resolution (each page is printed as a bitmap image). This setting is available only if the parent PasswordEncryptionProfile element’s compatibilityLevel attribute has a value of Acrobat5 or later.  
  - No - The document cannot be printed. |
| edit | Optional. Specifies the permissions for editing the document. This attribute can have the following values:  
  - NotExtract (default) - Users can perform all editing operations except extracting pages. All is now the same as NotExtract.  
  - DocAssembly - Users can insert, delete, and rotate pages, as well as create bookmarks and thumbnail pages. This value is available only if the parent PasswordEncryptionProfile element’s compatibilityLevel attribute has a value of Acrobat5 or later.  
  - FormFillinSign - Users can fill forms and add digital signatures in addition to performing the tasks allowed by this attribute’s DocAssembly value. This value is available only if the parent PasswordEncryptionProfile element’s compatibilityLevel attribute has a value of Acrobat5 or later.  
  - CommentsFormFillinSign - Users can fill forms and add digital signatures and comments. |
Portfolio

(Since 9.0) Extends the `<Package>` element for PDF Portfolios.

Portfolio filter element

As a filter element, the package specification comes entirely from the package specification contained within the child PDF source element. If the PDF source is not a package, then it is as if the Portfolio element were specified as the empty `<Portfolio/>` element.

```
<Portfolio>
  <PDF source> [0..1]
  <TargetLocale> [0..1]
</Portfolio>
```

Can be contained in the elements PDF result and StyledText elements.

A PDF result or source can have either a Portfolio subelement or a Package subelement. It cannot have both.

Portfolio defining element

The portfolio defining element’s Schema, DisplayOrder, and SortOrder elements define a package specification.
<Portfolio
    styleReference="xs:string">
    <ColorScheme> [0..1]
    <DisplayOrder> [0..1]
    <Header> (portfolio navigation pane) [0..1]
    <Navigator> [0..1]
    <Schema> [0..1]
    <SortOrder> [0..1]
    <TargetLocale> [0..1]
    <WelcomePage> [0..1]
</Portfolio>

Can be contained in PDF result, PDF source, and StyleProfile.

If Schema is specified within Portfolio, then that is the schema. The same is true, individually, for DisplayOrder and SortOrder, meaning if DisplayOrder is specified in Portfolio. If the Schema element is missing from the Portfolio element, then the Schema element value aggregates the schemas in any PDF package or portfolio sources.

If the Portfolio defining element omits the DisplayOrder or SortOrder elements, the DDX processor aggregates corresponding replacements from the PDF source documents. This aggregation is the same as for an omitted Schema element.

**Category**

"Document assembly" on page 143

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>styleReference</td>
<td>Required. The name of a StyleProfile element that contains a specification</td>
</tr>
<tr>
<td></td>
<td>for a PDF Portfolio.</td>
</tr>
</tbody>
</table>

**Resource**

(Since 9.0) A resource for a PDF Portfolio collection header or welcome page (which use the NavigatorTemplate format) or for a navigator.

<Resource
    name="xs:string"
    source="xs:string"
>
</Resource>

Can be contained in the elements Header (portfolio navigation pane) and Portfolio.
RichMedia

(Since 9.0) Attaches a compiled ActionScript program (SWF file) to a page.

```xml
<RichMedia
    source="xs:string"
    page="1" or "page number"
    left="72pt" or "length-specifier"
    top="372pt" or "length-specifier"
    width="400pt" or "length-specifier"
    height="300pt" or "length-specifier"
/>
</RichMedia>
```

Can be contained in the elements `<PDF>` result and `<PDF>` source.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>(Optional) A relative reference to the resource from the Navigator (SWF content) or Navigator Template XML. If not specified, the name of the source document is used. The string can be specified with an External Data URL.</td>
</tr>
<tr>
<td>source</td>
<td>(Required) An input map key or External Data URL which specifies the document to add as a resource.</td>
</tr>
</tbody>
</table>

Right

Specifies the right edge of the page as the anchor point for a header or footer. The `PageMargins` element defines the right edge.

```xml
<Right> [0..1]
    <StyledText> or <PDF source> [1]
</Right>
```
Can be contained in the elements Footer and Header.

The content specified by the child elements is aligned with the right margin (specified by the PageMargins element). If the StyledText element includes a text-align attribute, that attribute is ignored.

**Note:** There is no containment of the text within the right third of the page and there is no auto-wrapping of text. The text can go off the left side of the page. To ensure text containment and wrapping, place the text in the rich text p element.

If this element specifies a PDF source element as a child, the first page of the document provides the content.

**Category**

“Page content” on page 148

**Schema**

The Schema element is part of the package specification and defines the metadata (Field) attributes that can be associated with each package file. There is only one Schema per PDF package or portfolio. The individual values for the metadata are specified for each package file as described in the PackageFiles element.

The Schema element can only be a child of a Package element. However, if Schema is not specified, then it is aggregated from any PDF package sources being included in the assembly.

DDX processors can aggregate a schema from multiple PDF package or portfolio source’s package specification. In this case, the first schema is from the PDF base document if it is a package. Otherwise, it is from the first PDF package or portfolio source specified. From that starting point, the schema is merged in from each PDF package or portfolio source in the order specified in the DDX document.

When DDX processors aggregate schemas, name conflicts occur when multiple schemas contain a Field with the same name but with different types. When such a conflict occurs, the Field is added and a warning is logged. The name remains unchanged. If both Fields are marked as being visible, both appear in the user interface without any distinction. Any element within DisplayOrder and SortOrder that references this name includes both Fields. However, the order of such identically named Fields is undefined. Any FieldData element specified within PackageFiles for adding metadata attempts to use the most appropriate type, but the result is undefined.

DDX processors apply special considerations when aggregating differently named Fields of the following types. In particular, only the first one specified is kept in the schema. Others of the duplicate type are discarded.

- Filename
- Description
- ModificationDate
- CreationDate
- Size
The name stored in a PDF source, which is a package, can contain any sequence of white space characters that the creating application allows. However, the name, when specified through DDX, is normalized as follows:

- Leading and trailing white space is stripped.
- Sequences of white space are replaced with a single space character.

Therefore, all comparisons on the name are made using a normalized copy of the name, without modifying pre-existing names in a package.

As an empty element, the Schema element specifies the default schema.

```xml
<Schema/>
```

which is the equivalent of specifying the following, given the inherited TargetLocale is "en":

```xml
<Schema>
  <Field name="Index" type="Number" visible="false"/>
  <Field name="Name" type="Filename" editable="true"/>
  <Field name="Description" type="Description" editable="true"/>
  <Field name="Modified" type="ModificationDate" />
  <Field name="Size" type="Size" />
</Schema>
```

For supported locales, the name Field is localized. See “AdobeCoverSheet” on page 151 for such a list.

As a defining element, the Schema element’s child Field elements specify the schema. If Schema is empty, then it specifies the default schema. If it is missing from the Package element, the Schema is aggregated from PDF sources in the assembly that are packages.

It is highly recommended that the DisplayOrder and SortOrder be specified when `<Schema>` is specified.

```xml
<Schema>
  <Field> [0..n]
</Schema>
```

For more information, see “Field contained in Schema element” on page 178.

Can be contained in Package.

**Category**

“Document assembly” on page 143

**SortOrder**

The order of the Field elements contained in the SortOrder element corresponds to the order of the package files when assembling into a single PDF document. The order also corresponds to the priority viewing applications apply to the fields when sorting. The first Field specified is the main sorting value. Additional fields, applied in order, are used when the previous Field does not result in unique values.

The SortOrder element can only be a child of Package. However, if SortOrder is not specified, then the SortOrder is aggregated from any PDF package or portfolio sources’ package specifications being included in the assembly.
Any Field listed in the SortOrder that does not exist in the Schema is not included and a warning is logged.

DDX processors aggregate the SortOrder elements from multiple PDF package or portfolio package specifications in the following steps:

- **Obtain base SortOrder element.** If the base PDF source document is a PDF package or portfolio, then its SortOrder is used as the base SortOrder. Otherwise, the base SortOrder is obtained from the first PDF source element that contains a PDF package or portfolio.

- **Merge subsequent SortOrder elements.** For the other PDF source files that contain PDF packages or portfolios, their SortOrder is merged into the base SortOrder.

If the SortOrder being merged in is a duplicate (meaning that it has the same name value), then it is not added. If the Field does not exist in the Schema for the result package, then it is not added.

If the SortOrder is not specified, then it is left to the application as to the order in which the metadata is sorted.

The locale affects the results of sorting the Field values. The inherited TargetLocale element sets the locale. If no locale is specified, the default is "en".

If SortOrder is empty, then the SortOrder is unspecified. Only if it is missing from the Package element is the SortOrder aggregated from PDF sources in the assembly that are packages.

```xml
<SortOrder>
  <Field> [0..n]
</SortOrder>
```

For more information, see “Field contained in SortOrder element” on page 179.

Can be contained in Package.

**Category**

“Document assembly” on page 143

---

**String**

(Since 9.0) Adds entries to the String name tree in a navigator dictionary.

```xml
<String
  name="xs:string"
  url="URL"
/>
```

Can be contained in the Navigator source element.

This property has limited usefulness for localization and personalization. The Acrobat Navigator API does not currently use these properties. This element is similar to the Resource element.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>(Required) Name of the resource string. The name can be provided with an External Data URL.</td>
</tr>
<tr>
<td>url</td>
<td>(Required) An input map key or External Data URL that provides the string to add.</td>
</tr>
</tbody>
</table>

**StyledText**

Describes styled text (rich text) content added to the page.

Can be contained in the elements TableOfContentsEntryPattern, Footer, Header, Watermark, Background, PageContent.

**Category**

“Page content” on page 148

This expanded section lists the elements that are supported in rich text strings in the DDX language.

The StyledText element describes styled text (rich text strings) to added to the page using content elements, such as the following:

- Left, Center, and Right elements, which are children of the Header and Footer elements
- Watermark element
- Background element
- PageContent element
- TableOfContentsEntryPattern element

The rich text strings are XML expressions that conform to a subset of the XHTML 1.0 specification. They must also conform to a limited set of style properties taken from the specification Cascading Style Sheets Specification, version 2 (www.w3.org/TR/CSS2). This document uses the term CSS to mean the Cascading Style Sheets Specification, version 2.

**Attributes used in the rich text elements**

**Inheritance**

CSS inheritance rules apply to the elements contained in the StyledText element. Under some circumstances, an attribute can be specified on an element even though it has no meaning on that element. The attribute is then propagated downward through the descendent elements to the node where it is used. Such attributes are called inheritable attributes.

Many of the attributes shown in the StyledText element attribute table are inheritable attributes. The attribute descriptions in the next section indicate which attributes are inheritable.
Attributes

The following table describes the attributes used in the `StyledText` element or its children. The syntax declarations for each element include relevant attributes, although any of the attributes can be used in any of the elements.
### Attributes used in the rich text elements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Optional and inheritable. Color of the enclosed text, leaders, and text decorations.</td>
</tr>
</tbody>
</table>
| font | Deprecated. Instead of using the `font` attribute, use the following individual font attributes:  
  - `font-family`  
  - `font-weight`  
  - `font-size`  
  - `font-style`  
  Optional and inheritable. A shorthand font property of the following form, where each font characteristic is optional. The exception is the `line-height` characteristic must be preceded by the `font-size` characteristic and a slash. Unspecified components of this attribute value are set to their default values. The descriptions of the individual properties explain those defaults.  
  ```
  font-style font-weight font-size/line-height
  font-family
  ```  
  The values in the following examples are delineated in the table. That is, the first example below is explained in the first row in the table, and the second example in the second row.  
  ```
  font="italic 25pt/50pt Helvetica"
  font="200 sans-serif"
  font="oblique normal normal/normal 'Times New Roman'"
  ```  
  **Note:** A font-family name in the font attribute must be delimited with single quotes, if there are spaces within its name. For example, `'Times New Roman'`. |

<table>
<thead>
<tr>
<th>Style</th>
<th>Weight</th>
<th>Size</th>
<th>Line height</th>
<th>Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>italic</td>
<td>normal* (400)</td>
<td>25pt</td>
<td>50pt</td>
<td>Helvetica</td>
</tr>
<tr>
<td>normal*</td>
<td>200</td>
<td>normal* (12pt)</td>
<td>normal* (1.2x font size)</td>
<td>sans-serif (Myriad Pro used)</td>
</tr>
<tr>
<td>oblique</td>
<td>normal (400)</td>
<td>normal (12pt)</td>
<td>normal (1.2x font size)</td>
<td>Times New Roman</td>
</tr>
</tbody>
</table>

**Note:** * default value

In any given element, you can specify the `font` attribute or the individual attributes that describe a font (`font-style`, `font-weight`, `font-size`, `line-height`, and `font-family`). If the `font` attribute is present in an element, do not specify any of the individual attributes that describe a font.
Font-family

Optional and inheritable. Provides a font name, a generic font type, or a comma-separated list of font names or generic font types. The font characteristics are used to display the enclosed text. The following list shows how the Document Manipulation APIs maps generic font types:

- serif to Minion Pro
- sans-serif to Myriad Pro
- cursive to Minion Pro
- fantasy to Minion Pro
- monospace to Courier Std

If a list of fonts is provided, the first font containing glyphs for the specified text is used. If no font family is specified, the Document Manipulation APIs environment variables specify a default font family.

Font names that contain a space must be enclosed in single quotes, as shown below:

```html
font-family="Helvetica, 'Times New Roman', Courier"
```

Font family name matching requires the exact presence or absence of spaces. Omit this attribute if it duplicates information in a font attribute that appears in the same element.

(See “Font-family naming issues” on page 265.)

Font-size

Optional and inheritable. The font size of the enclosed text. The following list shows supported values:

- 12pt (default)
- positive length - A length value greater than 0.

Omit this attribute if it duplicates information in a font attribute that appears in the same element.

Font-stretch

Optional and inheritable. Specifies the stretch values for a font family. Stretch values are particular to a font family. The following list shows supported values, in order of narrowest to widest stretch:

- ultra-condensed
- extra-condensed
- condensed
- semi-condensed
- normal (default)
- semi-expanded
- expanded
- extra-expanded
- ultra-expanded
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| font-style    | Optional and inheritable. Specifies whether the enclosed text is displayed using a normal or italic (oblique) font. Style values are particular to a font family. The following list shows supported values:  
  * normal (default)  
  * italic  
  * oblique  
  Omit this attribute if it duplicates information in a font attribute that appears in the same element. |
| font-weight   | Optional and inheritable. Specifies weight of the font for the enclosed text. Normal is equivalent to 400 and bold is equivalent to 700. The following list shows supported values:  
  * normal (default)  
  * bold  
  * any of the values "100" or "200" or "300" or "400" or "500" or "600" or "700" or "800" or "900". If a numeric value is specified, it must be a multiple of 100.  
  Omit this attribute if it duplicates information in a font attribute that appears in the same element. |
| leader-pattern| Optional and inheritable. Specifies the pattern of spaces, dashes, dots, and lines used in a leader. A leader is a repetitious pattern of characters that fill a line. For example, a leader-pattern is typically used between the bookmark title and bookmark page reference in a table of contents entry. The following list shows the supported values:  
  * space (default)  
  * dashed  
  * double-dashed  
  * triple-dashed  
  * solid  
  * double  
  * triple  
  * dotted  
  * double-dotted  
  * triple-dotted |
### line-height

Optional and inheritable. Specifies the line-height of the enclosed text, which is the same as the distance between the current baseline and the one just above. When a numerical value is specified, the line height is the font size of the current element multiplied with the numerical value. DDX processors handle inheritance differently between numerical values and percentage values. In particular, when a numerical value is specified, child elements inherit the factor itself, not the resultant value.

The following values are supported:

- **normal (normal)** - This value is equivalent to 1.2 x font-size.
- **length** - Line height as a length that must not be less than 0.

### margin

Optional. A shorthand CSS margin property that includes a space-separated, sequential set of margin values. This attribute uses the following format:

```
margin-top margin-right margin-bottom margin-left
```

If margin values are omitted from this sequence, the Document Manipulation APIs determines default values as described in the [Cascading Style Sheets Specification, version 2](http://www.w3.org/TR/CSS2).

This attribute is ignored in the elements `StyledText`, `span`, `b`, `i`, and `leader`.

This attribute is not inheritable.

### margin-top

Optional. Sets the top margin of a `p` element. The attribute value must not be less than 0.

This attribute is ignored in the elements `StyledText`, `span`, `b`, `i`, and `leader`.

This attribute is not inheritable.

### margin-right

Optional. Sets the right margin of a `p` element. The attribute value must not be less than 0.

This attribute is ignored in the elements `StyledText`, `span`, `b`, `i`, and `leader`.

This attribute is not inheritable.

### margin-bottom

Optional. Sets the bottom margin of a `p` element. The attribute value must not be less than 0.

This attribute is ignored in the elements `StyledText`, `span`, `b`, `i`, and `leader`.

This attribute is not inheritable.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin-left</td>
<td>Optional. Sets the left margin of a <code>p</code> element. The attribute value must not be less than 0. This attribute is ignored in the elements <code>StyledText</code>, <code>span</code>, <code>b</code>, <code>i</code>, and <code>leader</code>. This attribute is not inheritable.</td>
</tr>
<tr>
<td>text-align</td>
<td>Optional and inheritable. Horizontal alignment of lines in a paragraph. The following values are supported: left (default) right center justify justify-all. This attribute is ignored in the elements <code>span</code>, <code>b</code>, <code>i</code>, and <code>leader</code>. This attribute is not supported in the DDX Header and Footer elements, which take alignment instructions from their child elements <code>Left</code>, <code>Center</code> and <code>Right</code>. This attribute is inheritable if specified in the <code>p</code> or <code>StyledText</code> element.</td>
</tr>
<tr>
<td>text-decoration</td>
<td>Optional and inheritable. Specifies special characteristics applied to the enclosed text. The following values are supported: none (default) underline overline line-through.</td>
</tr>
<tr>
<td>text-indent</td>
<td>Optional and inheritable. Specifies additional indentation of the first line of a paragraph. A negative value cannot be greater than the paragraph’s <code>margin-left</code> value. This attribute is ignored in the elements <code>span</code>, <code>b</code>, <code>i</code>, and <code>leader</code>. This attribute is inheritable if specified in the <code>p</code> or <code>StyledText</code> element.</td>
</tr>
</tbody>
</table>
Font-family naming issues

Shortcomings in the way CSS specifies font-family names can result in font name collisions. Such collisions result in the incorrect font being selected to render the associated text. The following table summarizes the symptoms of such font name collisions and their workarounds.

<table>
<thead>
<tr>
<th>Desired font</th>
<th>Selected font</th>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain</td>
<td>Decorative variation, extension, or subset, as described in ”Font variations, extensions, and subsets” on page 266</td>
<td>Use the Windows font family name, as described in ”Workaround when a font variation is erroneously used instead of the plain font face” on page 265</td>
</tr>
<tr>
<td>Decorative variation, extension, or subset</td>
<td>Plain</td>
<td>Use the OTF preferred font family name, as described in ”Workaround when a plain font face is used instead of a font variation” on page 266</td>
</tr>
</tbody>
</table>

**Caution:** Font family name matching requires the exact presence or absence of spaces that the system uses.

**Workaround when a font variation is erroneously used instead of the plain font face**

If you use the OTF preferred font family name, collisions can occur on fonts that have decorative variations, or that have special extensions or subsets. When such collisions occur, the decorative, extension, or subset form can be chosen over the plain form. The choice depends on the order fonts are inserted into the Font
Manager Module font set. The Mac OS font family name is generally the same as the OTF family name for Adobe OTF fonts.

To avoid improperly selected variations, specify the Windows font family name rather than the OTF preferred font family name. Use this workaround only for the improperly selected variations.

Consider the situation when HelveticaNeueLTStd-Bd.otf (plain) and HelveticaNeueLTStd-BdOu.otf (outline) are installed together on the Font Manager Module. The OTF preferred font family name is Helvetica Neue LT Std, and the Windows font family name is HelveticaNeueLT Std.

**Example: OTF preferred font family yields outline font**

```html
<StyledText font-size="14pt" font-family="Helvetica Neue LT Std">
    <p font-weight="600">This is erroneously displayed as an outline font.</p>
</StyledText>
```

**Workaround when a plain font face is used instead of a font variation**

You can have the opposite situation, where you use the Windows font family name with a variation, extension, or subset but see the plain font instead. To avoid choosing the wrong font variation in this situation, use the OTF preferred font family name (Mac OS font family name). Use this workaround only for improperly selected variations.

If you are using the Windows font family name, you are limited to four or less weight-italic combinations.

**Font variations, extensions, and subsets**

Specific names are used for font variations, extensions, and subsets.

**Decorative variations that may have font family name collisions**

The following list of decorative variations may result in collisions within certain font families. These variation names are used with font family names.

- Borders
- Calm
- Active
- ExtraActive
- Caption
- Display
- Headline
- Cursive
- Decorated
- Gothic
- Grime
- Informal
- Inline
- Ornament[ed,s]
- Outline
Fill[ed]
Shadow[ed]
Stencil
Contour
Open
Solid
Dots
Squiggles
Sketch
Tilt
ZigZag

Extensions or subsets that may have family name collisions

The following list of extensions or subsets may result in collisions within certain font families. These extension or subset names are used with font family names.

Address
NameNum
SubC apt
Extension
Fractions [Fra]
Initial(s)
Ligatures
Phonetic
Poetica-*
Pi extensions: -1...-4
GreekwMathPi
NewswithCommPi
Ding[bats]
Thangs
ChessDraughts
DiceDominoes
EnglishCards
FrenchCards
SmallCaps
29AB
29BC
30AB
30BC
Rich text elements

The `StyledText` element is the root of a rich text expression. (See also “Attributes used in the rich text elements” on page 258.)

**b**

Specifies that the bold typeface of a font is applied to an isolated string of text (`font-weight="700"`).

```xml
<xs:string>
  <Built-in key> [0..n]
  <span> [0..n]
  <i> [0..n]
  <b> [0..n]
  <graphic> [0..n]
  <sub> [0..n]
  <sup> [0..n]
  <leader> [0..n]
  <BatesNumber> [0..n]
  <TargetLocale> [0..1]
  <String> [0..n]
</xs:string>
```

Can be contained in the elements `p`, `i`, `b`, and `span` and in the element contents described by `xs:string`.

**BatesNumber**

Provides a means to apply identifying labels to a batch of related documents. (See “Applying identifying labels” on page 115.) For example, legal documents associated with a court case is an example of a batch of related documents. Each page in the document, or set of documents, is assigned a Bates number. This
number uniquely identifies that page and establishes its relationship to other Bates numbered documents.

A Bates number contains a sequentially incremented numeric value plus an optional prefix and suffix. The prefix + numeric + suffix is known as a Bates pattern. For example, <BatesNumber prefix="Chapter 1 " numberOfDigits=10 suffix=""/> has a Bates pattern of "Chapter 1 10".

```
<BatesNumber
    start="unspecified or xs:positiveInteger"
    numberOfDigits="6 or 6 <= xs:positiveInteger <= 15"
    prefix="xs:string"
    suffix="xs:string"
/>`

Can be contained in the elements p, i, b, and span.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberOfDigits</td>
<td>Optional. The number of digits which make up the bates number. The default and minimum value is 6, which produces Bates numbers such as 000001, 000002, and so on. The value is set to the minimum if a smaller number is entered. The maximum value is 15. Values greater than 15 default to 15.</td>
</tr>
<tr>
<td>prefix</td>
<td>Optional. Any text that appears just before the bates number.</td>
</tr>
<tr>
<td>start</td>
<td>Optional. If specified, it is the number that is assigned to the first bates number for that bates pattern. This value must be greater than or equal to 1. If not specified, the DDX processor provides a mechanism for setting the default, else the default is 1. The start value is set once for a given pattern. Additional usage of the same BatesNumber element with the same pattern continues to increment sequentially.</td>
</tr>
<tr>
<td>suffix</td>
<td>Optional. Any text that is to appear just after the Bates number.</td>
</tr>
</tbody>
</table>

### Built-in key

Provides the value of a built-in key, where the name of the element reflects the name of a built-in key.

```
<Built-in key/>
```

or

```
<Built-in key
    styleReference="reference to named style profile"
/>`

Can be contained in the elements p, i, b, and span.

(See "Built-in keys" on page 149.)
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>styleReference</td>
<td>Optional. The name of a StyleProfile element that itself contains a DatePattern element as appropriate.</td>
</tr>
</tbody>
</table>

**Example: _Title built-in key replaced by document title from XMP**

```
<StyledText><p><_Title/></p></StyledText>
```

**Example: _DateTime key replaced by date and time styled as specified by the greendate style**

```
<p color="green" font-weight="bold">
  <_DateTime styleReference="greendate"/>
</p>
```

graphic

Inserts a graphic as if it were a character in a line of text.

```
<graphic
  source="xs:string"
  width="Auto" or "positive-length"
  height="Auto" or "positive-length"
  block-align="top" or "middle" or "bottom"
  inline-align="left" or "center" or "right"
  rotate90="0" or "degrees of rotation in intervals of 90"
  scale-to-fit="fit-maximum" or "fit-minimum" or "fit-width" or "fit-height"
  or "fit-both" or "do-not-scale"
  vertical-align="baseline" or "base" or "sub" or "subscript" or "sup"
  or "super" or "superscript" or "length"
/>
```

Can be contained in the elements p, i, b, span, sub, and sup.

**Category**

“Page content” on page 148

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block-align</td>
<td>Optional. Specifies the horizontal placement of the graphic relative to the specified height.</td>
</tr>
<tr>
<td>height</td>
<td>Optional. The optimal height of the graphic. The default of &quot;auto&quot; uses the &quot;intrinsic-height&quot; of the graphic source. It must be greater than 0.</td>
</tr>
<tr>
<td>inline-align</td>
<td>Optional. Specifies the vertical placement of the graphic relative to the specified width.</td>
</tr>
<tr>
<td>rotate90</td>
<td>Optional. The reference orientation of the graphic with respect to the page.</td>
</tr>
</tbody>
</table>
Specifies that italics be applied to an isolated string of text (font-style="italic").

```xml
<i
color="black" or "color-specifier"
font-family=unspecified or "font-family-specifier"
font-size="12pt" or "positive length"
font-stretch="normal" or "extra-condensed" or "condensed"
or "semi-condensed" or "semi-expanded" or "expanded" or "extra-expanded"
font-weight="normal" or "bold " or "100" or "200" or "300" or "400"
or "500" or "600" or "700" or "800" or "900"
line-height=normal or "positive length"
text-decoration="none" or "underline" or "overline" or "line-through"
text-indent=0pt or "length"
vertical-align=baseline or "base" or "sub" or "subscript" or "sup"
or "super" or "superscript" or "length"
xml:space=preserve or "default"

> xs:string
<Built-in key> [0..n]
<span> [0..n]
<i> [0..n]
<b> [0..n]
<graphic> [0..n]
<sub> [0..n]
<sup> [0..n]
<leader> [0..n]
<BatesNumber> [0..n]
<TargetLocale> [0..n]
<String> [0..n]
xs:string</i>
```

Can be contained in the elements `p`, `i`, `b`, and `span` and in the element contents described by `xs:string`.  

**Name** | **Description**
---|---
scale-to-fit | Optional. Specifies a strategy for scaling the graphic to fit the line.
source | Required. A logical name in the input map associated with a data stream containing graphical content to include in the output text. Currently, only content of type PDF is accepted.
vertical-align | Optional. A "length" specifier is an amount by which to adjust the baseline of the enclosed text. A positive value indicates a superscript; a negative value indicates a subscript. Otherwise, "base" or "baseline" is equivalent to 0. "sub" or "subscript" means to lower the baseline to the proper position for subscripts. "sup", "super", or "superscript" means to raise the baseline to the proper position for superscripts. This attribute is not inheritable.
width | Optional. The optimal width of the graphic. The default of "auto" uses the "intrinsic-width" of the graphic source. It must be greater than 0.
If the font-style attribute is specified, it is ignored.

**leader**

Specifies the addition of a repetitious pattern of characters (spaces, dashes, dots, or lines) to fill a line. Typically, leaders appear between the bookmark title and bookmark page reference in a table of contents entry.

```
<leader
    color="black" or "color"
    font-size="12pt" or "positive length"
    leader-pattern="space" or "dashed" or "double-dashed" or "triple-dashed"
        or "dotted" or "double-dotted" or "triple-dotted" or "solid"
        or "double" or "triple"
</leader>
```

Can be contained in the elements `<p>`, `<i>`, `<b>`, and `<span>` and in the element contents described by `<xs:string>`.

**p**

Specifies a paragraph of styled text.

```
<p
    color="black" or "color"
    font-family=unspecified or "font-family-specifer"
    font-size="12pt" or "positive length"
    font-stretch="normal" or "extra-condensed" or "condensed"
        or "semi-condensed" or "semi-expanded" or "expanded" or "extra-expanded"
    font-style="normal" or "italic" or "oblique"
    font-weight="normal" or "bold " or "100" or "200" or "300" or "400" or "500"
        or "600" or "700" or "800" or "900"
    line-height="normal" or "positive length"
    margin="margin-shorthand-specifier"
    margin-top="0pt" or "nonnegative length"
    margin-right="0pt" or "nonnegative length"
    margin-bottom="0pt" or "nonnegative length"
    margin-left="0pt" or "nonnegative length"
    text-align="left" or "right" or "center" or "justify" or "justify-all"
    text-decoration="none" or "underline" or "overline" or "line-through"
    text-indent="0pt" or "length"
    xml:space="preserve" or "default"

  <xs:string> [0..n]
  </span> [0..n]
  </i> [0..n]
  </b> [0..n]
  </graphic> [0..n]
  </sub> [0..n]
  </sup> [0..n]
  </leader> [0..n]
  </BatesNumber> [0..n]
  </TargetLocale> [0..n]
```
<String> [0..n]
xs:string</p>

Can be included in the StyledText element.

The **p** element is thought of as the paragraph element. An empty paragraph is legal and can be specified with an empty **p** element or as containing only white space.

**Space**

Specifies a space between two styled text elements. The nonbreaking space entity number &amp;#160; can also be used.

```xml
<Space/>
```

Can be contained in the **p**, **b**, **i**, **span**, and **DatePattern** elements.

**span**

Specifies formatting for spans of inline text.

```xml
<span
  color="black" or "color"
  font-family=unspecified or "font-family-specifier"
  font-size="12pt" or "positive-length"
  font-stretch="normal" or "extra-condensed" or "condensed"
  or "semi-condensed" or "semi-expanded" or "expanded" or "extra-expanded"
  font-style="normal" or "italic" or "oblique"
  font-weight="normal" or "bold " or "100" or "200" or "300" or "400"
  or "500" or "600" or "700" or "800" or "900"
  line-height=normal or "positive length"
  text-decoration="none" or "underline" or "overline" or "line-through"
  vertical-align="baseline" or "base" or "sub" or "subscript" or "sup"
  or "super" or "superscript" or "length"
  xml:space="preserve" or "default"
>
  xs:string
</span>
```

Can be contained in the elements **p**, **i**, **b**, and **span**.
**String**

For DDX processors that can resolve URL references, the `String` element allows text content to come from an URL. When possible DDX processors treat the resolved URL as a string. If the URL does not resolve or is unspecified, DDX processors use the `String` element’s content (if provided). If the URL does not resolve and there is no text content, then the URL itself is used as the string.

```xml
<String url="xs:string">
  xs:string
</String>
```

Can be contained in the elements `p`, `i`, `b`, and `span`.

**Example**

```xml
<String url="process:///process_data/@lcStringVar">Dr. Zhivago</String>, assuming that `lcStringVar` contains "The King and I", then the text is "The King and I".
```

```xml
<String url="badURL">Dr. Zhivago</String>, the text is "Dr. Zhivago".
```

```xml
<String url="badURL"/>
```

**StyledText**

The root of a rich text expression.

```xml
<StyledText
  color="black" or "color-specifier"
  font-family="unspecified" or "font-family-specifier"
  font-size="12pt" or "positive length"
  font-stretch="normal" or "extra-condensed" or "condensed"
    or "semi-condensed" or "semi-expanded" or "expanded" or "extra-expanded"
    font-style="normal" or "italic" or "oblique"
  font-weight="normal" or "bold" or "100" or "200" or "300" or "400" or "500"
    or "600" or "700" or "800" or "900"
  line-height="normal" or "positive length"
  text-align="left" or "right" or "center" or "justify" or "justify-all"
  text-decoration="none" or "underline" or "overline" or "line-through"
  text-indent="0pt" or "length"
  xml:space="preserve" or "default"
>
  <p> [1..n]
  <TargetLocale> [0..1]
</StyledText>
```

Can be contained in the elements `TableOfContentsEntryPattern`, `Left`, `Right`, `Center`, `Watermark`, `Background`, and `PageContent`.

The `StyledText` element is the container for all rich text within DDX. For attribute inheritance, this element is the root.

**sub**

Provides a convenient way to apply `vertical-align="subscript"` to an isolated text string. The approximate size of the subscript is 60% of the font size, and the approximate shift is 31% of the font size. While `sub` and `sup` can be nested, only the innermost element is honored.
<sub>
<color>"black" or "color"</color>
<font-family>"unspecified" or "font-family-specifier"</font-family>
<font-size>"12pt" or "positive-length"</font-size>
<font-stretch>"normal" or "ultra-condensed" or "extra-condensed" or "condensed" or "semi-condensed" or "semi-expanded" or "expanded" or "extra-expanded" or "ultra-expanded"
<font-style>"normal" or "italic" or "oblique"
<font-weight>"normal" or "bold" or "100" or "200" or "300" or "400" or "500" or "600" or "700" or "800" or "900"
<text-decoration="none" or "positive-length"
</sub>

Can be contained in the elements <p>, <i>, <b>, <span>, <sub>, <sup>, and <sup>, and in the element contents described by <xs:string>.

**sup**

Provides a convenient way to apply vertical-align="superscript" to an isolated string of text. The approximate size of the superscript is 60% of the font size and the approximate shift is -15% of the font size. While <sub> and <sup> can be nested, only the innermost element is honored.

<sup>
<color>"black" or "color"</color>
<font-family>"unspecified" or "font-family-specifier"</font-family>
<font-size>"12pt" or "positive-length"</font-size>
<font-stretch>"normal" or "ultra-condensed" or "extra-condensed" or "condensed" or "semi-condensed" or "semi-expanded" or "expanded" or "extra-expanded" or "ultra-expanded"
<font-style>"normal" or "italic" or "oblique"
<font-weight>"normal" or "bold" or "100" or "200" or "300" or "400" or "500" or "600" or "700" or "800" or "900"
<text-decoration="none" or "positive-length"
</sup>

Can be contained in the elements <p>, <i>, <b>, <span>, <sub>, <sup>, and <sup>, and in the element contents described by <xs:string>.
<TargetLocale> [0..1] xs:string</sup>

Can be contained in the elements p, i, b, span, sub, and sup, and in the element contents described by xs:string.

**StyleProfile**

The **StyleProfile** element contains elements that add package specifications, page content, or add a table of contents (TableOfContents) element. Page content elements include Header and Watermark elements. The descriptions of content found in a **StyleProfile** element are external to a document description. The elements that describe an assembly include styleReference attributes that can reference **StyleProfile** element by name.

Named references allow for the creation and maintenance of a repository of named styles for different types of assemblies. Those styles are included within a DDX document when needed.

```xml
<StyleProfile name="xs:string">
  <Background> [0..1] 
  <DatePattern> [0..1] 
  <Footer> [0..1] 
  <Header> [0..1] 
  <Package> [0..1] 
  <Portfolio> [0..1] 
  <PageContent> [0..1] 
  <TableOfContents> [0..1] 
  <Watermark> [0..1] 
</StyleProfile>
```

Can be contained in the DDX element, which is the DDX root.

The **DatePattern** element in a style profile can be referenced from the styleReference attribute of the _DateTime, _Created, and _Modified built-in keys when used within a **StyledText** element.

**Category**

“Profile” on page 148

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the <strong>StyleProfile</strong> element.</td>
</tr>
</tbody>
</table>

**Subject**

Provides the value for the subject metadata in the result document.

```xml
<Subject value="xs:string"/>
```
Can be contained in the PDF result element, and the PackageFiles filter, select, or conversion elements.

If specified as a sibling to a Metadata source element, then the Metadata source is imported first and the subject value overrides the imported Metadata.

**Category**

"Document properties" on page 146

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Required. Subject to use in the metadata. An empty string blanks out the subject metadata. The string value can be specified with an External Data URL.</td>
</tr>
</tbody>
</table>

**TableOfContents**

Specifies a table of contents in the result document.

```
<TableOfContents
  maxBookmarkLevel="1" or "infinite" or "xs:positiveInteger"
  bookmarkTitle="unspecified" or "name of a bookmark title"
  createLiveLinks="true" or "false"
  includeInTOC="true" or "false"
  toc-backgroundcolor="transparent" or color-specifier"
  toc-margin="margin-shorthand-specifier"
  toc-margin-top-left="0 pt" or nonnegative-length-specifier"
  toc-margin-right="0 pt" or nonnegative-length-specifier"
  toc-margin-bottom="0 pt" or nonnegative-length-specifier"
  toc-margin-left="0 pt" or nonnegative-length-specifier"
>
  <ArtBox> [0..1]
  <Background>
    or <NoBackgrounds> [0..2, where 2 is allowed for alternating pages]
  <BleedBox> [0..1]
  <Footer> or <NoFooters> [0..2, where 2 is allowed for alternating pages]
  <Header> or <NoHeaders> [0..2, where 2 is allowed for alternating pages]
  <PageContent> [0..n]
  <PageLabel> [0..1]
  <PageMargins> [0..2, where 2 is allowed for alternating pages]
  <PageOverlay> [0..n]
  <PageRotation> [0..1]
  <PageSize> [0..1]
  <PageUnderlay> [0..n]
  <TableOfContentsEntryPattern> [0..toclevels]
  <TableOfContentsPagePattern> [0..2]
  <TargetLocale> [0..1]
  <TrimBox> [0..1]
  <Watermark>
    or <NoWatermarks> [0..2, where 2 is allowed for alternating pages]
</TableOfContents>
```
Note: PageContent adds only the content, not alternateText, as TableOfContent pages are not tagged.

Can be contained in the elements PDF result, PDFGroup, and StyleProfile.

This element has two forms, depending on the appearance of the styleReference attribute:

Reference to a StyleProfile element. If the styleReference attribute is specified, it references the name attribute of a StyleProfile element that defines the table of contents. Any other attributes or any child elements in the TableOfContents element are ignored.

Definition of style properties. Otherwise, the appearance of the table of contents is specified in the element’s other attributes and child elements.

The placement of the TableOfContents element relative to PDF source elements influences the placement and contents of the table of contents. The Document Manipulation APIs adds a table of contents just before the content obtained from the next PDF source element. By default, the entries in the table of contents include the bookmarks from PDF source elements that come after the TableOfContents element. (The includeInTOC attribute of PDF source element lets you change the entries added to the table of contents.)

Category

“Document assembly” on page 143

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| bookmarkTitle         | Optional. Specifies whether a bookmark is created for the table of contents, and if so, the title of the bookmark. The string value can be specified with an External Data URL. This attribute can have the following values:  
|                       | unspecified (default) - If this attribute is omitted, no bookmark is created for the table of contents.  
|                       | bookmark title - A bookmark is created for the table of contents. The bookmark has the title provided. The string can contain built-in keys. The new bookmark becomes the top-level bookmark for the generated table of contents. |
| maxBookmarkLevel      | Optional. Specifies the maximum nesting level of bookmarks used to construct the table of contents. For example, a value of 1 means that only top-level bookmarks are used to build the table. Further, only bookmarks that reference pages within the result document are included in the table of contents. |
| createLiveLinks       | Optional. Specifies whether entries in the table of contents have PDF links associated with them. This attribute can have the following values:  
|                       | true (default) - Entries in the table of contents include links to their destinations  
|                       | false - Entries in the table of contents do not have links to their destinations. This setting is useful if the result document is intended solely for print. |
TableOfContentsEntryPattern

Specifies the style to apply to table of contents entries.

```
<TableOfContentsEntryPattern
  applicableLevel="all" or "positive non-zero integer"
>
  <StyledText> [1]
  <TargetLocale> [0..1]
</TableOfContentsEntryPattern>
```

Can be contained in the TableOfContents element.

**Category**

“Document assembly” on page 143

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| includeInTOC       | Optional. Controls whether the table of contents includes an entry to itself. The entry is included only when a bookmark title is provided in the **bookmarkTitle** attribute. This attribute can have the following values:  
  true (default) - If the bookmark title is provided, the table of contents includes it in the table of contents.  
  false - The bookmark title (if provided) is omitted from the table of contents. |
| toc-backgroundcolor | Optional and inheritable. The color of the background area for the table of contents body region.                                           |
| toc-margin         | Optional and inheritable. A shorthand CSS margin property of the form: `<margin-top><margin-right><margin-bottom><margin-left>`. |
| toc-margin-top     | Optional and inheritable. Sets the top margin of the table of contents background area. This value must not be less than 0.                |
| toc-margin-right   | Optional and inheritable. Sets the right margin of the table of contents background area. This value must not be less than 0.            |
| toc-margin-bottom  | Optional and inheritable. Sets the bottom margin of the table of contents background area. This value must not be less than 0.           |
| toc-margin-left    | Optional and inheritable. Sets the left margin of the table of contents background area. This value must not be less than 0.           |
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicableLevel</td>
<td>Optional. The bookmark level for which this entry pattern is defined. This</td>
</tr>
<tr>
<td></td>
<td>attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>all (default) - Entry pattern applies to all bookmark levels.</td>
</tr>
<tr>
<td></td>
<td>positive non-zero integer - A single bookmark level to which the entry</td>
</tr>
<tr>
<td></td>
<td>pattern applies.</td>
</tr>
</tbody>
</table>

`<TableOfContentsPagePattern>
  pages="1-last" or "1" or "2-last"
  toc-backgroundcolor="transparent or color-specifier"
  toc-margin="margin-shorthand-specifier"
  toc-margin-top-left="0 pt or nonnegative-length-specifier"
  toc-margin-right="0 pt or nonnegative-length-specifier"
  toc-margin-bottom="0 pt or nonnegative-length-specifier"
  toc-margin-left="0 pt or nonnegative-length-specifier"
  <ArtBox> [0..1]
  <Background> or <NoBackgrounds> [0..2, where 2 is allowed for alternating pages]
  <BleedBox> [0..1]
  <Footer> or <NoFooters> [0..2, where 2 is allowed for alternating pages]
  <Header> or <NoHeaders> [0..2, where 2 is allowed for alternating pages]
  <PageContent> [0..n]
  <PageLabel> [0..1, where 2 is allowed if applied to alternating pages]
  <PageMargins> [0..2, where 2 is allowed for alternating pages]
  <PageOverlay> [0..n]
  <PageRotation> [0..1]
  <PageSize> [0..1]
  <PageUnderlay> [0..n]
  <TargetLocale> [0..1]
  <TrimBox> [0..1]
  <Watermark> or <NoWatermarks> [0..2, where 2 is allowed for alternating pages]
</TableOfContentsPagePattern>

**Note:** PageContent adds only the content, not alternateText, as TableOfContent pages are not tagged.

Can be contained in the `<TableOfContents>` element.

This element specifies the style applied to one or more pages of the table of contents described by the parent `<TableOfContents>` element.

The `toc-background` and related `toc-margin` attributes are used to set up the background for the area occupied by the TOC listings.

The left border of the toc-background area is drawn inside the left page margin.

The top border is drawn at the bottom of the padding on the header area.
The right border is drawn inside the right page margin.
The bottom border is the top of the padding on the footer area.
The content/layout area of the background is inset from each of the four borders by the corresponding side’s toc-margin setting.
If backgrounds are also desired on the header or footer areas, they must be set up individually on the <Header> and <Footer> entries in the <TableOfContentsPagePattern>.

**Category**

“Document assembly” on page 143

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pages</td>
<td>Optional. Associates the style described by the TableOfContentsPagePattern element with pages in the table of content described by the parent TableOfContents element. This attribute can have the following values: 1-last (default) - The style applies to all pages in the table of contents. 1 - The style applies to the first page. 2-last - The style applies to all pages in the table of contents but the first page.</td>
</tr>
<tr>
<td>toc-backgroundcolor</td>
<td>Optional. The toc-background and related toc-margin attributes are used to set up the background for the area occupied by the TOC listings.</td>
</tr>
<tr>
<td>toc-margin</td>
<td>Optional. A shorthand CSS margin property of the form: &lt;margin-top&gt;&lt;margin-right&gt;&lt;margin-bottom&gt;&lt;margin-left&gt;.</td>
</tr>
<tr>
<td>toc-margin-top</td>
<td>Optional: Sets the top margin of the table of contents background area. Must not be less than 0.</td>
</tr>
<tr>
<td>toc-margin-right</td>
<td>Optional. Sets the right margin of the table of contents background area. Must not be less than 0.</td>
</tr>
<tr>
<td>toc-margin-bottom</td>
<td>Optional. Sets the bottom margin of the table of contents background area. Must not be less than 0.</td>
</tr>
<tr>
<td>toc-margin-left</td>
<td>Optional. Sets the left margin of the table of contents background area. Must not be less than 0.</td>
</tr>
</tbody>
</table>

**TargetLocale**

The TargetLocale indicates the locale ID to apply to strings for the following purposes:

Sorting a list of source names specified by the PDF source, XDP source and PackageFiles source elements.

Sorting names specified by Schema for a PDF package or portfolio when exporting package files or assembling package files into a single PDF file.

Selecting a localized version of the default Schema for a package or portfolio specification.
Selecting a localized version of a navigator from a multi-lingual .NAV file.
Selecting a localized version of the built-in "_AdobeCoverSheet" PDF document.
Formatting text specified within the StyledText element.
Converting strings to a dateTime value when working with DatePattern elements.

The locale is an ID consisting of optional language, script, country, and variant fields in that order, separated by underscores, followed by an optional keyword list. The script, if present, is four characters long. The character length distinguishes it from a country code, which is two characters long. A keyword list begins with an at-sign (@) and consists of one or more keyword/value pairs separated by commas. The recommended format is a 2-letter language code followed by an underscore and then a 2-letter country code. For example, ja_JP.

TargetLocale can be specified globally as well as locally. Therefore, TargetLocale can be specified as a child of the DDX root element. The locale ID applies to its parent element and inherits elements that can also contain TargetLocale elements. The most local TargetLocale specification is the one used at any point in the DDX document.

There is a legacy case of a locale specified as an attribute of another element. The PDF source element includes the attribute "sortLocale" used for sorting a list of input names (via the "sourceMatch" attribute). If not specified, "sortLocale" inherits from a more global specification of TargetLocale. A TargetLocale contained within a PDF source element is considered more local than its "sortLocale" attribute.

If no locale is specified, the default locale ID is specific to the DDX processor. The Document Manipulation APIs infers the default locale ID from the user's context. If it cannot be inferred, the locale defaults to en.

<TargetLocale locale="xs:string"/>

Can be contained in DDX, PDF result, PDF source, PDFGroup, TableOfContentsEntryPattern, TableOfContents, TableOfContentsPagePattern, BlankPage, Bookmarks result, Bookmarks filter, Comments result, Comments filter, Links result, Links filter, Package, PackageFiles, Portfolio, StyleProfile, StyledText, p, span, i, b, sub, sup, Header, Footer, Watermark, PageContent, and Background.

Category

"Document properties" on page 146

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>Required. A string representation of the locale as defined in RFC 3066. The string value can be specified with an External Data URL. Examples are &quot;en_US&quot;, &quot;de&quot;, and &quot;zh_Hans&quot;.</td>
</tr>
</tbody>
</table>

Title

Specifies the value for the title metadata in the result document.

<Title
  value="xs:string"
/>
Can be contained in the PDF result element and the PackageFiles filter, select, or conversion elements.

If specified as a sibling to a Metadata source element, then the Metadata source is imported first and the title value overrides the imported Metadata.

**Category**

“Document properties” on page 146

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Required. Title name to use in the metadata. The string value can be specified with an External Data URL. An empty string blanks out the title name.</td>
</tr>
</tbody>
</table>

---

**Transform**

Specifies transformations applied to preexisting page contents.

```
<Transform
    scale="1.0" or "percentage or decimal"
    newX="0pt" or "length"
    newY="0pt" or "length"
    rotate90="0" or "degrees of rotation in intervals of 90"
 />
```

Can be contained in the elements PDF result, PDF source, PackageFiles filter, select, or conversion elements, and PDFGroup.

This element enables the page contents to be scaled, translated, and rotated, without changing the page orientation. In contrast, the PageRotation element affects the page orientation. The Transform element does not apply to new content specified with page content elements. The Header element is a page content element. Also, it does not apply to new pages specified with the TableOfContents or BlankPage elements.

**Category**

“Page content” on page 148

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>Optional. The percentage by which the page contents are scaled. The value can be a decimal in the range of .0 to 1.0 or a percentage in the range of 0% to 100%. In the latter case, the percentage sign (%) is required.</td>
</tr>
<tr>
<td>newX</td>
<td>Optional. Units by which the X-axis origin is shifted. A positive value shifts the origin to the right, and a negative value shifts it to the left.</td>
</tr>
</tbody>
</table>
TrimBox

Defines the intended dimensions of the finished page after printing and trimming.

```xml
<TrimBox
   left="0pt" or "length"
   top="0pt" or "length"
   right="0pt" or "length"
   bottom="0pt" or "length"
   alternation="None" or "OddPages" or "EvenPages"
/>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, TableOfContentsPagePattern, PackageFiles filter, select, or conversion elements, and BlankPage.

This element is intended for use by print professionals only.

**Category**

“Page properties” on page 147

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newY</td>
<td>Optional. Units by which the Y-axis origin is shifted. A positive value shifts the origin upward, and a negative value shifts it downward.</td>
</tr>
<tr>
<td>rotate90</td>
<td>Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number rotates the axes clockwise, and a negative number rotates them counter clockwise.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Optional. Width of the margin as measured from the left side of the page to the left side of the trim box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>top</td>
<td>Optional. Width of the margin as measured from the top of the page to the top of the trim box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>right</td>
<td>Optional. Width of the margin as measured from the right side of the page to the right side of the trim box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bottom</td>
<td>Optional. Width of the margin as measured from the bottom of the page to the bottom of the trim box. In this case, the page is the visible page as viewed in Acrobat.</td>
</tr>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element’s settings apply to all pages or to alternating pages. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>- None (default) - Settings apply to all pages.</td>
</tr>
<tr>
<td></td>
<td>- OddPages - Settings apply to odd pages only.</td>
</tr>
<tr>
<td></td>
<td>- EvenPages - Settings apply to even pages only.</td>
</tr>
<tr>
<td></td>
<td>Pages are considered odd or even depending on their ordinal page number in the result document.</td>
</tr>
</tbody>
</table>
Provides styled text or a PDF file for placement over the existing page content. The Watermark element applies a watermark to all pages in the scope of its parent element. It replaces any preexisting watermarks on those pages if the replaceExisting attribute has not been specified or is "true".

The replaceExisting attribute lets you replace or retain pre-existing watermarks when adding one new watermark. If replaceExisting is "false", a new Watermark is added to the existing Watermark. There can only be one watermark specified (added) per page. Transient results allow for the addition of multiple watermarks.

**Note:** This element removes only watermarks added with Acrobat 8 or earlier. It cannot remove watermarks added with later versions. Acrobat 9 and later does not distinguish between watermarks, backgrounds, headers, and footers. Also, this element does not remove watermarks that contain Bates numbers.

If the watermark content is provided in a PDF source element, the first page from the source document is used for the watermark.

Use the Background element to place content behind the existing page content.

```xml
<Watermark
   alternation="None" or "OddPages" or "EvenPages"
   fitToPage="true" or "false"
   horizontalAnchor="Left" or "Center" or "Right"
   horizontalOffset="0pt" or "length"
   opacity="100%" or "percentage"
   rotation="0" or "xs:integer"
   scale="100%" or "percentage"
   showOnScreen="true" or "false"
   showWhenPrinting="true" or "false"
   verticalAnchor="Top" or "Middle" or "Bottom"
   verticalOffset="0pt" or "length"
   replaceExisting="true" or "false"
>
   <StyledText> or < PDF source> [1]
   <TargetLocale> [0..1]
</Watermark>
```

Can be contained in the elements PDF result, PDF source, PDFGroup, TableOfContents, PackageFiles filter, select, or conversion, TableOfContentsPagePattern, BlankPage, and StyleProfile.

**Category**

"Page content" on page 148
### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternation</td>
<td>Optional. Specifies whether the element's settings apply to all pages or to alternating pages. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>None (default) - Settings apply to all pages.</td>
</tr>
<tr>
<td></td>
<td>OddPages - Settings apply to odd pages only.</td>
</tr>
<tr>
<td></td>
<td>EvenPages - Settings apply to even pages only.</td>
</tr>
<tr>
<td></td>
<td>Pages are considered odd or even depending on their ordinal page number in the result document.</td>
</tr>
<tr>
<td>fitToPage</td>
<td>Optional. Specifies whether to scale the watermark content to fit the page. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>false (default) - Watermark content is scaled according to the scale attribute.</td>
</tr>
<tr>
<td></td>
<td>true - Watermark content is forced to fit the boundaries of the page size, by either expanding or shrinking the text. The scale attribute is ignored.</td>
</tr>
<tr>
<td>horizontalAnchor</td>
<td>Optional. Specifies a horizontal anchor point, relative to the PageMargins element of the page on which the watermark is placed. The horizontalOffset attribute modifies the placement relative to this anchor.</td>
</tr>
<tr>
<td></td>
<td>Left - Left page margin</td>
</tr>
<tr>
<td></td>
<td>Center (default) - Midpoint between the left and right margins</td>
</tr>
<tr>
<td></td>
<td>Right - Right margin</td>
</tr>
<tr>
<td>horizontalOffset</td>
<td>Optional. Offset from the horizontal anchor point specified in the horizontalAnchor attribute. A positive value moves the watermark right, while a negative value moves it left.</td>
</tr>
<tr>
<td>opacity</td>
<td>Optional. Transparency of the watermark or background text. The value of this attribute can have the following forms:</td>
</tr>
<tr>
<td></td>
<td>Decimal in the range of .0 to 1.0</td>
</tr>
<tr>
<td></td>
<td>Percentage in the range of 0% to 100%. In this case, the percentage sign (%) is required.</td>
</tr>
<tr>
<td></td>
<td>The default is 100%.</td>
</tr>
<tr>
<td>replaceExisting</td>
<td>Optional. If set to false, then any pre-existing watermarks in the source PDF document remain. The specified watermark does not replace any existing watermarks.</td>
</tr>
<tr>
<td>rotation</td>
<td>Optional. Rotation of the watermark upon the page. The valid range is -360 to 360 degrees.</td>
</tr>
<tr>
<td>scale</td>
<td>Optional. Scaling of the watermark. The valid range is 8% to 3200%. The value can be a decimal in the range of .0 to 1.0 or a percentage in the range of 0% to 100%. In the latter case, the percentage sign (%) is required.</td>
</tr>
</tbody>
</table>
WelcomePage

Specifies the Welcome Page used in a PDF Portfolio.

```xml
<WelcomePage source="xs:string">
  <Resource> [0..n]
</WelcomePage>
```

Can be contained in the `Portfolio` element.

The welcome page can be a PDF. It can also be an XML document that conforms to the Template Navigator specification.

The welcome page is a resource with a name of "welcome/model.xml". If the welcome page source is specified, then all resources excluding the Header ("header/model.xml") in the PDF source are specified. Only resources referenced by a "header/model.xml" or "welcome/model.xml" are visible in the PDF Portfolio navigation pane (also called the PDF Portfolio Layout pane). If the source specified is an XML source, then any resources it references must also be specified. Such resources include images and localized strings.

**Note:** Acrobat 9 does not fully support the spec. If the portfolio welcome page is designed with the Acrobat user interface, there are no issues of whether the implementation is supported.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Optional. Input map key or URL mapped to either a PDF document which contains a Welcome Page or to an XML document. If the source is a PDF, then more resources than are necessary can be included. If the source attribute is not specified, then it defaults to the identified base document for the &lt;PDF&gt; result.</td>
</tr>
</tbody>
</table>

**XDP**

(Since 9.0) The XDP element allows you to insert part or all of an Adobe XML form (an XFA template) into a root XDP document. This process is called XFA dynamic assembly. The XDP element also allows you to package an XDP document as a PDF document.

**XDP (generic)**

(Since 9.0) Describes an XDP assembly that is dynamically assembled into a PDF document. After the children of the XDP element are assembled, the result is converted to a PDF for assembly with the siblings of the XDP element. The XDP (generic) element is the only form of the XDP element that can be a child of the PDF result element.

```xml
<XDP aggregateXDPContent="None" or "All">
  <XDPContent> | <XDP source> | <XFADat\a> [0..n]
</XDP>
```

Can be contained in the PDF result element.

**Note:** It is an error to assemble multiple XDP (generic) elements. To avoid this error, include the NoXFA or NoForms element as a sibling of the XDP (generic) elements. Alternatively, use the XDP result element to assemble the XDP forms, and then include that result as an XDP source within an XDP (generic) element. (See “Creating and Modifying Acrobat and XML (XFA) Forms” on page 41 and Guidelines for Dynamically Assembling Customized Forms and Documents.)
## Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| aggregateXDPContent | Optional. A string that controls whether inner-most XDPContent overrides or adds to XDPContent elements in an outer (higher) level. Here are the supported values:  
  - **All** Indicates that all of the XDP content from every level are added at the insertion point in inner-to-outer level order.  
  - **None** (default) Indicates that within a XDPContent hierarchy the most inner level XDPContent is for an insertion point. If there are additional XDPContent elements applied to the same insertion point at an outer level, then that XDPContent is not inserted. The insertion is blocked because the insertion point has already been used.  
If aggregateXDPContent is None, then only the local scope is used. So, you insert the fragment into the insertion point defined as the child. If the same insertion point is used as a sibling, then it is ignored. Whereas if aggregateXDPContent=All, then the insertion point is reused as a sibling and the fragments inserted are aggregated together. |

## XDP result

(Since 9.0) Describes the contents to assemble into an XDP document.

```xml
<XDPAPIs>  
  <result>xs:string</result>  
  <aggregateXDPContent>"None" or "All"</aggregateXDPContent>  
  <removeInsertionPoints>"xs:string"</removeInsertionPoints>  
  <resolveAssets>"none" or "all" or "absolute" or "relative"</resolveAssets>  
  <retainInsertionPoints>"xs:string"</retainInsertionPoints>  
  <return>"true" or "false"</return>  
</XDPAPIs>
```

Can be contained in the **DDX** element.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>result</strong></td>
<td>Required. Name of the assembled XDP document. The name can be provided with an External Data URL. See “External Data URL” on page 153.</td>
</tr>
</tbody>
</table>
| **resolveAssets**  | (Since 10.0) Optional. A string that specifies how the referenced images in the source documents must be resolved in the PDF. You can specify the value for this attribute either for the XDP result attribute or for individual source XDP documents. Supported values are:  
  - none  Indicates that none of the image references are resolved. This is the default value.  
  - all   Indicates that all the images linked through absolute or relative references by the source XDP documents are embedded in the result XDP document.  
  - absolute  Indicates that only images linked through absolute references in the source XDP documents are embedded in the result XDP document.  
  - relative  Indicates that only images linked through relative references in the source XDP documents are embedded in the result XDP document. |
| **aggregateXDPContent** | Optional. A string that controls whether inner-most XDPContent overrides or adds to XDPContent elements in an outer (higher) level. Here are the supported values:  
  - All  Indicates that all of the XDP content from every level is added at the insertion point in inner-to-outer level order.  
  - None (default)  Indicates that within a XDPContent hierarchy the most inner level XDPContent is for an insertion point. If there are additional XDPContent elements applied to the same insertion point at an outer level, then that XDPContent is not inserted. The insertion is blocked because the insertion point has already been used.  
  
  If *aggregateXDPContent* is None, then only the local scope is used. So, you insert the fragment into the insertion point defined as the child. If the same insertion point is used as a sibling, then it is ignored. Whereas if *aggregateXDPContent*=All, then the insertion point is reused as a sibling and the fragments inserted are aggregated together. |
| **removeInsertionPoints** | Optional. A string that specifies which insertion points to remove after the form fragments are assembled into the XDP result. Here are the supported values:  
  - All  All of the insertion points are removed  
  - None (default)  None of the insertion points are removed.  
  - A comma-separated list of the insertion points to remove. |
**XDP source**

Specifies XDP source content to assemble into the XDP result.

```xml
<XDP source="xs:string"
    baseDocument = "true" or "false"
    fragment="xs:string"
    includeSubFolders="true" or "false"
    matchMode="Include" or "Exclude"
    removeInsertionPoints="xs:string"
    resolveAssets="none" or "all" or "absolute" or "relative"
    required="true" or "false"
    retainInsertionPoints="xs:string"
    select="1-last" or "range"
    sourceMatch = "xs:string"
    sortOrder="Ascending" or "Descending"
    sortLocale="xs:string"
>
  <XDPContent> [0..n]
</XDP>
```

Can be contained in the elements XDP result and XDP generic.
### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **source**      | Optional, but required if the `sourceMatch` attribute is not specified. A logical name associated with a single input data stream or an ordered list of data streams. The source can be specified with an External Data URL. See "External Data URL" on page 153.  
If both the `source` and `sourceMatch` attributes are specified, this attribute is used only if it matches a name entry in the input map. |
| **baseDocument**| Optional. Identifies the base document and provides the initial structure that the Document Manipulation APIs uses to set certain document-level properties of the result XDP document. These include document properties, form data, document-level JavaScript code, and viewer preferences. The result document can contain only one source identified as a base document. Documents other than the base document only contribute the following items to the resulting document:  
- Pages  
- Document components (such as bookmarks, links, file attachments)  
- Page labels  
- Page content  
- Page properties  
Document-level components, are only included from a document once, even if the document is specified multiple times.  
A file mapped to a `baseDocument` is always required in the input map for the Document Manipulation APIs, even if the `required` attribute is set to "false".  
This attribute can have the following values:  
  - `true` - Identifies the parent XDP source element as the base document.  
  - `false` (default) - Does not identify the parent XDP source element as the base document, though a base document is always required. If none of the source documents in an XDP result are specified as the base document, the DDX processor determines the base document. |
| **fragment**    | Optional. A string that identifies the form fragments to include in the XDP result. The name corresponds to the names of subforms in the XDP forms that this element identifies. The value can be an External Data URL that resolves to a string.  
If fragment is omitted, the entire XDP form specified by this element is included in the XDP result. |
| **includeSubFolders** | Optional. If true, all files in the folder and subfolders are included. The result is a list of documents for the XDP source element. If false, only the files in the specified folder are included. |
| **matchMode**   | Optional. Specifies whether to include the matched results in the resultant document. This attribute can have the following values:  
  - `Include` (default) - Includes the matched data streams.  
  - `Exclude` - Excludes the matched data streams. |
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `removeInsertionPoints` | Optional. A string that specifies which insertion points to remove after the form fragments are assembled into the XDP result. Here are the supported values:  
  - **All**: All of the insertion points are removed  
  - **None** *(default)*: None of the insertion points are removed.  
  - A comma-separated list of the insertion points to remove. |
| `required`         | Optional. The default *(true)* requires the element to add XDP content to the assembly. If no XDP content is added, then the DDX processor declares an error.  
  If set to false and no data streams are identified for this element, then this element does not contribute any XDP content to the assembly. No error is declared. |
| `resolveAssets`    | *(Since 10.0)* Optional. A string that specifies how the referenced images in the source documents must be resolved in the PDF. You can specify the value for this attribute either for the XDP result attribute or for individual source XDP documents. Supported values are:  
  - **none**: Indicates that none of the image references are resolved. This is the default value.  
  - **all**: Indicates that all the images linked through absolute or relative references by the source XDP document are embedded in the result XDP document.  
  - **absolute**: Indicates that only images linked through absolute references in the source XDP documents are embedded in the result XDP document.  
  - **relative**: Indicates that only images linked through relative references in the source XDP documents are embedded in the result XDP document. |
| `retainInsertionPoints` | Optional. A string that specifies the insertion points to retain. Here are the supported values:  
  - **All** *(default)*: All of the insertion points are retained  
  - **None**: None of the insertion points are retained.  
  - A comma-separated list of the insertion points to retain.  
  If `retainInsertionPoints` and `removeInsertionPoints` attributes are specified for the same XDP result, the `removeInsertionPoints` value is used and the `retainInsertionPoints` value is ignored. |
| `select`           | Optional. Specifies which streams from the source document to include in the result document. The default value is `1-last`, which signifies the entire document is included. *(See “Page and document ranges” on page 157.)* |
### XDPContent

(Since 9.0) Specifies XDP content to insert into the XDP source or result being specified.

```xml
<XDPContent source="xs:string"
    fragment="xs:string"
    includeSubFolders="true" or "false"
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| sortOrder       | Optional. If the regular expression specified in the sourceMatch attribute matches multiple documents, this attribute specifies the order in which those documents are sorted. The sort order is used to create an ordered list of documents. This attribute is not used if the source attribute matches an entry in the input map.  
This attribute can have the following values:  
Ascending (default) - Matched documents are sorted in ascending order: A-Z.  
Descending - Matched documents are sorted in descending order: Z-A.                                                                                                     |
| sortLocale      | Optional. Specifies the locale to use for sorting, according to sortOrder, names matched by the sourceMatch attribute. The value of this attribute must contain a valid 2-character ISO language code (see ISO 639). Any locale passes schema validation; however, if the requested locale is not available, a ValidationException is thrown.  
The default value for this attribute is obtained from the TargetLocale element.                                                                                           |
| sourceMatch     | Optional, but required if the source attribute is not specified. The value is a regular expression pattern that selects source names and their associated data streams from the input map or URL.  
Source specifies an input map. If source specifies a non-URL name and sourceMatch is specified, sourceMatch is used only when the source attribute does not match an entry in the input map or URL.  
Source specifies a URL. If the source attribute specifies a URL that references a folder of files, then sourceMatch can select specific files from the folder.  
The regular expression syntax is a standard regular expression syntax as implemented in the java.util.regex class for Java.  
Depending on the matchMode attribute, the matched documents are either included or excluded in the assembled document. If more than one name matches, the names are sorted, as specified in the sortOrder and sortLocale attributes.  
The string value can be specified with an External Data URL.                                                                                                                |

See also

- “External Data URL” on page 153
- “Specifying multiple input streams” on page 32
insertionPoint="xs:string"
matchMode="Include" or "Exclude"
removeInsertionPoints="xs:string"
required="true" or "false"
retainInsertionPoints="xs:string"
select="1-last" or "document range specifier"
sortLocale="xs:string"
sortOrder="xs:string"
sourceMatch="xs:string"

>  
 </XDPContent>

Can be contained in the elements XDP result, XDP source, and XDP generic.

Category

"Document assembly" on page 143

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Optional, but required if the sourceMatch attribute is not specified. A logical name associated with a single input data stream or an ordered list of data streams. The source can be specified with an External Data URL. See “External Data URL” on page 153. If both the source and sourceMatch attributes are specified, this attribute is only used if it matches a name entry in the input map.</td>
</tr>
<tr>
<td>fragment</td>
<td>Optional. A string that identifies the XDP content to insert into the parent element’s XDP document. The name corresponds to the names of subforms in the XDP form that this element identifies. If fragment is omitted, the entire XDP document specified by this attribute is inserted into the parent element's XDP document. The value can be an External Data URL that resolves to a string. Form designers assign names to XML form templates by using Designer. The forms are saved as XDP documents.</td>
</tr>
<tr>
<td>includeSubFolders</td>
<td>Optional. If true, all files in the folder and subfolders are included. The result is a list of documents for the XDP source element. If false, only the files in the specified folder are included.</td>
</tr>
<tr>
<td>insertionPoint</td>
<td>Required. Name of the insertion point where this element’s XDP content is inserted. The insertion point is a property of the parent element’s XDP document. The value can be an External Data URL that resolves to a string. Form designers add insertion points in XML form templates by using Designer. The forms are saved as XDP documents.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>matchMode</td>
<td>Optional. Specifies whether to include the matched results in the resultant document. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>Include (default) - Includes the matched data streams.</td>
</tr>
<tr>
<td></td>
<td>Exclude - Excludes the matched data streams.</td>
</tr>
<tr>
<td>removeInsertionPoints</td>
<td>Optional. A string that specifies which insertion points to remove after the form fragments are assembled into the XDP result. Here are the supported values:</td>
</tr>
<tr>
<td></td>
<td>All - All of the insertion points are removed</td>
</tr>
<tr>
<td></td>
<td>None (default) - None of the insertion points are removed.</td>
</tr>
<tr>
<td></td>
<td>A comma-separated list of the insertion points to remove.</td>
</tr>
<tr>
<td>required</td>
<td>Optional. The default (true) requires the element to add XDP content to the assembly. If no XDP content is added, then the DDX processor declares an error.</td>
</tr>
<tr>
<td></td>
<td>If set to false and no data streams are identified for this element, then this element does not contribute any XDP content to the assembly. No error is declared.</td>
</tr>
<tr>
<td>retainInsertionPoints</td>
<td>Optional. A string that specifies the insertion points to retain. Here are the supported values:</td>
</tr>
<tr>
<td></td>
<td>All (default) - All of the insertion points are retained</td>
</tr>
<tr>
<td></td>
<td>None - None of the insertion points are retained.</td>
</tr>
<tr>
<td></td>
<td>A comma-separated list of the insertion points to retain.</td>
</tr>
<tr>
<td></td>
<td>If retainInsertionPoints and removeInsertionPoints attributes are specified for the same XDP result, the removeInsertionPoints value is used and the retainInsertionPoints value is ignored.</td>
</tr>
<tr>
<td>select</td>
<td>Optional. Determines which documents are selected when an ordered list of input streams is provided. The default value is 1-last, indicating that all streams be selected. For the syntax of specifying ranges, see “Page and document ranges” on page 157.</td>
</tr>
<tr>
<td>sortLocale</td>
<td>Optional. Specifies the locale to use for sorting, according to sortOrder, names matched by the sourceMatch attribute. The value of this attribute must contain a valid 2-character ISO language code (see ISO 639). Any locale passes schema validation; however, if the requested locale is not available, a ValidationException is thrown.</td>
</tr>
<tr>
<td></td>
<td>The default value for this attribute is obtained from the TargetLocale element.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| sortOrder   | Optional. If the regular expression specified in the sourceMatch attribute matches multiple documents, this attribute specifies the order in which those documents are sorted. The sort order is used to create an ordered list of documents. This attribute is not used if the source attribute matches an entry in the input map.  
This attribute can have the following values:  
Ascending (default) - Matched documents are sorted in ascending order: A-Z.  
Descending - Matched documents are sorted in descending order: Z-A. |
| sourceMatch | Optional, but required if the source attribute is not specified. The value is a regular expression pattern that selects source names and their associated data streams from the input map or URL.  
Source specifies an input map. If source specifies a non-URL name and sourceMatch is specified, sourceMatch is used only when the source attribute does not match an entry in the input map or URL.  
Source specifies a URL. If the source attribute specifies a URL that references a folder of files, then sourceMatch can select specific files from the folder.  
The regular expression syntax is a standard regular expression syntax as implemented in the java.util.regex class for Java.  
Depending on the matchMode attribute, the matched documents are either included or excluded in the assembled document. If more than one name matches, the names are sorted, as specified in the sortOrder and sortLocale attributes.  
The string value can be specified with an External Data URL.  
See also  
“External Data URL” on page 153  
“Specifying multiple input streams” on page 32 |
This section describes the syntax and semantics of the XML grammars used with the DDX grammar.
Some DDX elements provide configuration information for DDX processors that can call other services. For example, the Communications Document Manipulation APIs can call these Experience Manager Forms services: Reader Extensions, Generate PDF, Forms, and Output.

Not all DDX processors support the extended service elements. When a DDX processor encounters an unsupported element, it may raise an exception.

**PDFGenerationSettings**

Controls the options that are used when converting native documents, to PDF documents. Examples of native documents include images, Microsoft® Word documents, and Microsoft PowerPoint.

```xml
<PDFGenerationSettings
    conversionSettings="xs:string"
    fileTypeSettings="xs:string">
</PDFGenerationSettings>
```

Can be contained in the [DDX](#), [PDF](#), and [PDFGroup](#) elements.

Conversion is performed automatically when native documents are supplied to PDF source elements. Conversion requires the service to be available for generating PDF, such as the Generate PDF service.

For the Generate PDF service, see [Administration Help](#) for information on default and custom settings.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| conversionSettings | Optional. Name of the PDF settings to apply to the resultant PDF. This name corresponds to settings on the Adobe PDF Settings in the Administration Console. The following names are examples of conversion settings for English installations:  
  - High Quality Print  
  - Oversized Pages  
  - PDFA1b 2005 CMYK  
  The default setting is Standard.  
  The string value can be specified with an External Data URL. |
| fileTypeSettings   | Optional. Name of conversion settings to use for converting the native document to PDF. This name corresponds to named settings on the Administration Console. The default setting is Standard.  
  The string value can be specified with an External Data URL. |
ReaderRights

Specifies the rights that are enabled when the document is viewed in Adobe Reader. This element is meaningful only if the Reader Extensions service is available.

```xml
<ReaderRights
    credentialAlias="xs:string"
    formFillIn="true or false"
    formDataImportExport="true or false"
    submitStandalone="true or false"
    onlineForms="true or false"
    dynamicFormFields="true or false"
    dynamicFormPages="true or false"
    barcodeDecoding="true or false"
    digitalSignatures="true or false"
    comments="true or false"
    onlineComments="true or false"
    embeddedFiles="true or false"
    mode="Final or Draft"
/>
```

Can be contained in the PDF result element.

The attributes to include in the ReaderRights element depend on the rights that the specified credential includes.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialAlias</td>
<td>(Required) Specifies the alias of the credential used to apply usage rights to a PDF document. The string value can be specified with an External Data URL.</td>
</tr>
<tr>
<td>formFillIn</td>
<td>(Optional) Corresponds to the formFillIn usage right.</td>
</tr>
<tr>
<td>formDataImportExport</td>
<td>(Optional) Corresponds to the formDataImportExport usage right.</td>
</tr>
<tr>
<td>submitStandalone</td>
<td>(Optional) Corresponds to the submitStandalone usage right.</td>
</tr>
<tr>
<td>onlineForms</td>
<td>(Optional) Corresponds to the onlineForms usage right.</td>
</tr>
<tr>
<td>dynamicFormFields</td>
<td>(Optional) Corresponds to the dynamicFormFields usage right.</td>
</tr>
<tr>
<td>dynamicFormPages</td>
<td>(Optional) Corresponds to the dynamicFormPages usage right.</td>
</tr>
<tr>
<td>barcodeDecoding</td>
<td>(Optional) Corresponds to the barcodeDecoding usage right.</td>
</tr>
<tr>
<td>digitalSignatures</td>
<td>(Optional) Corresponds to the digitalSignatures usage right.</td>
</tr>
<tr>
<td>comments</td>
<td>(Optional) Corresponds to the comments usage right.</td>
</tr>
<tr>
<td>onlineComments</td>
<td>(Optional) Corresponds to the onlineComments usage right.</td>
</tr>
<tr>
<td>embeddedFiles</td>
<td>(Optional) Corresponds to the embeddedFiles usage right.</td>
</tr>
<tr>
<td>mode</td>
<td>(Optional) Corresponds to the mode usage right.</td>
</tr>
</tbody>
</table>
**XFAConversionSettings**

(Since 8.2) Controls the options for converting XFA documents to PDF documents.

```xml
<XFAConversionSettings
    renderAtClient="auto or true or false"
    tagged="true or false"
    retainSignatureFields="None or All or Signed or Unsigned"
/>
```

Can be contained in the [DDX](#), [PDF](#), and [PDFGroup](#) elements.

For the Document Manipulation APIs, this setting is used when calling the Forms service and the Output service. Other DDX elements can override these settings, such as [NoForms](#) or [NoXFA](#). Here are some examples:

When [NoForms](#) or [NoXFA](#) is specified and XFA stream is required, then the `renderAtClient` attribute is ignored. That is, the default behavior prevails.

When [NoForms](#) is specified as some form fields remain for tagged PDF, the `tagged` attribute is ignored. That is, the default behavior prevails.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>renderAtClient</td>
<td>Optional. Enables the delivery of PDF content by using the client-side rendering capability of Acrobat 7 or later. The value <code>auto</code> uses the XFA <code>dynamicRender</code> configuration value. Default is <code>auto</code>, unless <code>NoXFA</code> is specified, and then the default value is <code>false</code>. A value of <code>true</code> results in the creation of a Dynamic XFA PDF.</td>
</tr>
<tr>
<td>tagged</td>
<td>Optional. Setting to <code>true</code> results in creating a tagged PDF document. The default is <code>false</code>.</td>
</tr>
</tbody>
</table>

**XCI**

Specifies the path of the XCI file. An XCI file is a configuration file to perform tasks, such as embedding a font into a document. The default.xci file is located in the svcdata\XMLFormService folder. For example, assuming that AEM Forms is installed on JBoss, the full path is [Install location]\svcdata\XMLFormService.

```xml
<XFAConversionSettings>
    <XCI source="source key">
</XFAConversionSettings>
```

Can be contained in the [XFAConversionSettings](#) element.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Required. A logical name, associated with an input data stream or an ordered list of data streams, containing XCI content. If the <code>source</code> is not provided, the rendition options specified in the XCI file not applied to resultant PDF document. The source can be specified with an External Data URL. (See &quot;Source elements&quot; on page 19.)</td>
</tr>
</tbody>
</table>
**XFData**

(Since 8.2) The **XFData** element lets one provide form data to be merged into a PDF when it is an XFA-based form. The parent element can be a **PDF** source or an **XDP** (generic) element. An XDP (generic) element contains a dynamically assembled XFA-based form that is being assembled into a PDF result, an XFA form. If the source attribute is provided, the XFA data is taken from the specified input source. If the source attribute is not provided, the XFA data is provided as the content of the **XFData** element.

**Note:** The Document Manipulation APIs use the Forms service to merge data into XFA forms. If that service is not available, the **XFData** element is ignored.

If the parent **PDF** source attribute or the **XFData** source attribute points to a list of documents, only the first document in each list is used.

```xml
<XFData
   source="xs:string"
>
   <any element in ##other namespace> or <any element in ##local namespace>[0..1]
</XFData>
```

Can be contained in the elements **PDF** source and **XDP** (generic).

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Optional. The name of an input document that contains XFA form data in XML format. The string value can be specified with an External Data URL.</td>
</tr>
</tbody>
</table>

The following is an example XDF Data.

```xml
<PDF result="filledinForm.pdf">
   <PDF source="form.pdf">
      <XFData>
         <form1>
            <header>
               <dtmDate>20040606T101010</dtmDate>
               <txtOrderedByCompanyName>
                  Any Company Name
               </txtOrderedByCompanyName>
               <txtOrderedByAddress>555, Any Blvd.</txtOrderedByAddress>
               <txtOrderedByCity>Any City</txtOrderedByCity>
               <txtOrderedByStateProv>Alabama</txtOrderedByStateProv>
               <txtOrderedByZipCode>12345</txtOrderedByZipCode>
               <txtOrderedByCountry>United States</txtOrderedByCountry>
               <txtPONum>8745236985</txtPONum>
            </header>
            <requirements/>
            <scope/>
            <correspondence/>
            <questions/>
            <closing/>
         </form1>
      </XFData>
   </PDF>
</PDF>
```
The About language is an XML syntax that provides information about the Communications - Document Manipulation APIs.

The Document Manipulation APIs returns an About document in response to the `About` element appearing in the DDX document.

The namespace of the About language is `http://ns.adobe.com/DDX/About/1.0/` and the root element is the `About` element. The schema is installed in the product Documentation folder.

### About

The root element for the About language.

```xml
<About>
  <Processor>
    <Version>
      <Build>
        <Copyright>
          ...<About>
```

### Build

Provides the build number for the Document Manipulation APIs.

```xml
<Build>
  xs:string
</Build>
```

### Copyright

Provides the copyright for the Document Manipulation APIs.

```xml
<Copyright>
  xs:string
</Copyright>
```

### Processor

Provides the product name of the Document Manipulation APIs.

```xml
<Processor>
  xs:string
</Processor>
```
Version

Provides the version number of the Document Manipulation APIs.

```xml
<Version>
  xs:string
</Version>
```
Document Information Language

The Document Information (DocInfo) language provides information about a PDF document, such as title, author, and number of pages. Some of the data, such as title and author, in a DocInfo document is metadata taken from the document's document information dictionary. Other information, such as number of pages and page labels, reflects the PDF document's contents.


The namespace of the DDX language is http://ns.adobe.com/DDX/DocInfo/1.0/, and the root element is DocInfo. The schema is installed in the product Documentation folder.

Categories of DocInfo data

The data in a DocInfo document can be categorized as metadata or derived information. The DocInfo language does not group data according to category; however, these categories are discussed to clarify the sources of data included in a DocInfo document.

Metadata includes document-level information can come from the following sources:

- Provided by the application used to produce the PDF document, for example the PDF version and total number of pages in the document
- Applied to the PDF document, usually by human interaction. Examples of such metadata are the document's title, author, subject, and keywords.

Derived information is data obtained by examining the structure and contents of the PDF document, for example:

- Page size as viewed or printed and the page rotation setting for all pages of the document, listing the settings per range of pages. Derived information also includes the page label settings for all pages of the document, again listing the page labels per range of pages.
DocInfo reference

DocInfo

Root element for the DocInfo language.

```xml
<DocInfo>
  <Title> [0..1]
  <Author> [0..1]
  <Subject> [0..1]
  <Keywords> [0..1]
  <CreatedDate> [0..1]
  <ModifiedDate> [0..1]
  <Creator> [0..1]
  <Producer> [0..1]
  <Version> [1]
  <NumPages> [1]
  <Package> [0..1]
  <FormType> [1]
  <PageSizes> [0..1]
  <PageRotations> [0..1]
  <PageLabels> [0..1]
  <PDFAConformance> [0..1]
  <Extensions> [0..1]
</DocInfo>
```

Author

The name of the person who created the specified PDF document.

```xml
<Author>
  "xs:string"
</Author>
```

Can be contained in the DocInfo element.

This information is obtained from the PDF document’s metadata. If that information is missing, the Author element is omitted. If there is more than one author listed in the metadata, only the first one is represented in this element.

CreatedDate

The Created date from the PDF document properties. The format of the date is

```xml
<year>-<month>-<day>T<hour>:<minute>:<sec>-<offsetGMTHour><offsetGMTMinute>
```

```xml
<CreatedDate>
  "xs:string"
</CreatedDate>
```

Creator

Name of the application used to produce the content of the document.

```xml
<Creator>
```
Can be contained in the DocInfo element.

The result PDF document inherits this value from the base document.

**DisplayOrder**

Contains Field elements that identify the specified order of the fields when displayed in a viewer.

```xml
<DisplayOrder>
  <Field name="xs:string" /> [0..n]
</DisplayOrder>
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. The normalized name of a Field as defined in the Schema. The non-normalized name appears as the content in the Schema element.</td>
</tr>
</tbody>
</table>

Can be contained in the Package element.

**Extensions**

The PDF specification is now the ISO/DIS 32000 standard and the PDF version number will not increment beyond version 1.7. Instead, an extensions dictionary identifies the versions of vendor-specific extensions to the PDF specification. Each vendor or entity must register a prefix with the PDF Name Registry (ISO 32000, Annex E). Each entry in this dictionary has a key derived from the registered prefix assigned to the vendor that defines the extension.

The associated value is a dictionary containing BaseVersion and ExtensionLevel entries. BaseVersion is a name that identifies the PDF version upon which the extension is based. ExtensionLevel is an integer that identifies the version of the extension.

```xml
<Extensions>
  <Extension> [0..n]
</Extensions>

<Extension>
  <Vendor>
    xs:string
  </Vendor>
  <BaseVersion>
    xs:string
  </BaseVersion>
  <ExtensionLevel>
    xs:string
  </ExtensionLevel>
</Extension>
```

**Note:** The BaseVersion can differ from the <Version> included in this result.
FormType

Type of form used in the document, if any.

```xml
<FormType>
  "xs:string"
</FormType>
```

Can be contained in the DocInfo element.

The contents of this element can be any of the following values:

- "NotAForm"
- "Acroform"
- "Static-XFA"
- "Dynamic-XFA"
- "XPAGround"

Keyword

Keyword associated with the specified PDF document.

```xml
<Keyword>
  xs:string
</Keyword>
```

Can be contained in the Keywords element.

Keywords

Keywords associated with the specified PDF document.

```xml
<Keywords>
  <Keyword>[1..n]
</Keywords>
```

Can be contained in the DocInfo element.

This information is obtained from the PDF document’s metadata. If that information is missing, the Keywords element is omitted.

ModifiedDate

The Modified date from the PDF document properties. The format of the date is

```xml
<year>-<month>-<day>T<hour>:<minute>:<sec>-<offsetGMTHour><offsetGMTMinute>
```

```xml
<ModifiedDate>
  xs:string
</ModifiedDate>
```

NumPages

Number of pages in the specified PDF document.
<NumPages>
  number of pages
</NumPages>

Can be contained in the DocInfo element.

Package

Only present if the PDF document is a PDF package. It describes the package specification for the package, if there is one. If specified as an empty element (<Package/ >), the package specification is the default consisting of basic metadata known for file attachments in general.

<Package>
  <Schema> [0..1]
  <DisplayOrder> [0..1]
  <SortOrder> [0..1]
</Package>

Can be contained in the DocInfo element.

PageLabel

The page label for a continuous range of pages with the same page label.

<PageLabel
  pages="pages to which the label applies"
  start="Integer page number"
  format="None" or "Decimal" or "LowerRoman" or "UpperRoman" or 
  "LowerAlpha" or "UpperAlpha"
  prefix="xs:string"
/>

Can be contained in the PageLabels element.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pages</td>
<td>The page range to which the page label format applies. (See “Page and document ranges” on page 154.)</td>
</tr>
<tr>
<td>start</td>
<td>First label number for this page range, not necessarily the ordinal page number.</td>
</tr>
<tr>
<td>format</td>
<td>Manner in which the number is displayed. The following values are supported:</td>
</tr>
<tr>
<td></td>
<td>None - Indicates there is no number following the prefix in the page label.</td>
</tr>
<tr>
<td></td>
<td>Decimal - For example, 1, 2, 3, …</td>
</tr>
<tr>
<td></td>
<td>LowerRoman - For example, i, ii, iii, …</td>
</tr>
<tr>
<td></td>
<td>UpperRoman - For example, I, II, III, …</td>
</tr>
<tr>
<td></td>
<td>LowerAlpha - For example, a, b, c, …</td>
</tr>
<tr>
<td></td>
<td>UpperAlpha - For example, A, B, C, …</td>
</tr>
<tr>
<td>prefix</td>
<td>Text displayed before the number.</td>
</tr>
</tbody>
</table>
**Example: Specifying page labels**

```xml
<!-- Pages 1 through 5 bear a page label with the text "Page: " followed by the page number in lower-case Roman numerals, where page numbering begins with 1 ("i"). -->
(PageLabel
    pages="1-5"
    startingNumber="1"
    style="LowerRoman"
    prefix="Page: ">
</PageLabels>
```

**PageLabels**

The page labels for the specified PDF document.

```xml
<PageLabels>
    <PageLabel> [1..n]
</PageLabels>
```

Can be contained in the `DocInfo` element.

If the PDF document omits page labels, this element is omitted.

**PageRotations**

Specifies the page rotations used in the document.

```xml
<PageRotations>
    <PageRotation> [1..n]
</PageRotations>
```

Can be contained in the `DocInfo` element.

**PageSize**

Page size for a continuous range of pages with equal page size.

```xml
<PageSize
    pages="pages"
    width="page width" [or rename to shortedge/longedge]
    height="page height"
/>
```

Can be contained in the `PageSizes` element.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pages</td>
<td>Range of pages that share this same width and height. (&quot;Page and document ranges&quot; on page 154)</td>
</tr>
<tr>
<td>width</td>
<td>Width of the page (media).</td>
</tr>
<tr>
<td>height</td>
<td>Height of the page (media).</td>
</tr>
</tbody>
</table>
Example: Page ranges qualify page sizes

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DocInfo xmlns="http://ns.adobe.com/DDX/DocInfo/1.0/">
...
<!-- All pages use a page size of 8.5x11 inches. -->
<PageSizes>
  <PageSize
    pages="1-last"
    width="8.5in"
    height="11.0in"
  />
</PageSizes>
</DocInfo>
```

Can be contained in the `PageSizes` element.

**PageSizes**

Page sizes used in the specified PDF document.

```xml
<PageSizes>
  <PageSize>
    pages="1..n"
  </PageSize>
</PageSizes>
```

Can be contained in the `DocInfo` element.

The `PageSizes` element contains one `PageSize` element for each distinct page size.

**PageRotation**

Specifies page rotation setting for a continuous range of pages with the same page rotation setting.

```xml
<PageRotation
  pages="page range"
  rotate90="degrees in increments of 90"
/>
```

Can be contained in the `PageRotations` element.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pages</td>
<td>A page range that describes the pages having a the orientation specified by this element. (See &quot;Page and document ranges&quot; on page 154.)</td>
</tr>
<tr>
<td>rotate90</td>
<td>Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number is clockwise, and a negative number is counterclockwise.</td>
</tr>
</tbody>
</table>

**PDFAConformance**

Specifies the PDF/A conformance level of the PDF.

```xml
<PDFAConformance
  compliance="PDF/A-1b, PDF/A-2b, or PDF/A-3b"
/>
If the PDFAValidation element is present on the DocumentInformation query, the PDF is validated for conformance with the result presented in the PDFAConformance element. If the PDFAValidation element is not present, the PDFAConformance returns the PDF/A metadata if it exists in the PDF document.

The PDFAConformance element is added to the DocInfo results if the PDF contains a PDF/A version metadata element. It is always added when a PDFAValidation element is present on the DocumentInformation query. The metadata PDF/A version is reported whenever it exists in the PDF document. The PDFAConformance is reported as notvalidated if the PDFAValidation element is not present.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isCompliant</td>
<td>Determines whether the PDF document is PDF/A compliant. Valid values are true, false, and notvalidated.</td>
</tr>
<tr>
<td>compliance</td>
<td>Specifies whether this document is PDF/A compliant. Valid values are PDF/A-1b, PDF/A-2b, or PDF/A-3b.</td>
</tr>
<tr>
<td>resultLevel</td>
<td>Specifies how much information is returned. Valid values are PassFail, Summary, or Detailed.</td>
</tr>
<tr>
<td>ignoreUnusedResources</td>
<td>Specifies whether to ignore resources that are not used. Valid values are true or false.</td>
</tr>
<tr>
<td>allowCertificationSignatures</td>
<td>Specifies whether signatures are allowed. Valid values are true or false.</td>
</tr>
</tbody>
</table>

### Producer

Name of the application used to convert the content into PDF.

```xml
<Producer>
  xs:string
</Producer>
```

The result PDF document inherits this value from the base document.

Can be contained in the DocInfo element.

### Schema

The description of the PDF package fields (metadata). The element content of the Field element is the textual name displayed to the user in the user interface of the PDF viewing application.
<Schema>
  <Field name="xs:string" type="Text" or "Date" or "Number" or "Filename" or "Description" or "ModificationDate" or "CreationDate" or "Size"
    visible="true" or "false"
    editable="true" or "false"
  >
    xs:string
  </Field> [0..n]
</Schema>

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>editable</td>
<td>Optional. Indicates whether the PDF viewing application allows the field value to be edited. This attribute has no effect on whether the field is editable by a DDX processor.</td>
</tr>
<tr>
<td>name</td>
<td>Required. The normalized name of a field as defined in the Schema. The non-normalized name appears as the element content. The normalized name is used when assembling PDF packages and comparing field names.</td>
</tr>
<tr>
<td>type</td>
<td>Required. The type identifies the type of data that is stored in this field.</td>
</tr>
<tr>
<td>visible</td>
<td>Optional. The initial visibility of the field in the PDF viewer. At least one field must be specified as visible; otherwise, an error is thrown.</td>
</tr>
</tbody>
</table>

Can be contained in the `Package` element.

SortOrder

The `SortOrder` element contains `Field` elements that identify which field values are used for sorting. The first field listed is the main sort. Any subsequent fields are used when the first sort results in duplicate values.

<SortOrder>
  <Field name="xs:string" ascending="true" or "false"/> [0..n]
</SortOrder>

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascending</td>
<td>Optional. When sorting on this field, this attribute specifies whether to sort in ascending (&quot;true&quot;) or descending (&quot;false&quot;) order.</td>
</tr>
<tr>
<td>name</td>
<td>Required. The name of a field as described in the schema.</td>
</tr>
</tbody>
</table>

Can be contained in the `Package` element.

Subject

The subject of the specified PDF document.

<Subject>
Document subject
</Subject>

Can be contained in the DocInfo element.

This information is obtained from the PDF document’s metadata. If that information is missing, the Subject element is omitted.

Title

The title of the specified PDF document, obtained from the document’s metadata.

<Title>
  document title
</Title>

Can be contained in the DocInfo element.

This information is obtained from the PDF document’s metadata. If that information is missing, the Title element is omitted.

Version

The PDF version of the specified PDF document.

<Version>
  version specification
</Version>

The value of the Version element corresponds to the version of the PDF Reference to which the file conforms. Possible values are shown below.

<table>
<thead>
<tr>
<th>Value</th>
<th>PDF Reference</th>
</tr>
</thead>
</table>

ViolationDetail

Contained within PDFAConformance element to report the details of one instance of a detected violation. The elements are not sorted.

<ViolationDetail
  key="xs:string"
  page="xs:positiveInteger"
  field="xs:string"
  annot="xs:string"
Violation

Contained within PDFAConformance element to report a count of each type of detected violation.

```xml
<Violation
    count="xs:positiveInteger"
    key="xs:string"
    description="xs:string"
/>```
The Bookmarks language is an XML representation of bookmarks that can be extracted from PDF documents. It can also be imported into PDF documents, as directed by the appearance of a `Bookmarks` element in a DDX document.

Bookmarks are a tree-structured hierarchy of outline items in the PDF document that enable navigation of the document.

**About the Bookmarks language**

The namespace for the Bookmarks language supported by the Communications - Document Manipulation APIs is [http://ns.adobe.com/pdf/bookmarks](http://ns.adobe.com/pdf/bookmarks) and the namespace version is 1.0.

The root of the Bookmarks XML language is the `Bookmarks` element.

**Intent of bookmarks in a PDF document**

A PDF document viewer can optionally display a document outline on the screen. This display allows the user to navigate interactively from one part of the document to another. The outline consists of a tree-structured hierarchy of bookmarks (also called outline objects). The outline serves as a visual table of contents to display the document’s structure to the user. The user can interactively open and close individual items by clicking them with the mouse. When an item is open, its immediate children in the hierarchy become visible on the screen. Each child can in turn be open or closed, selectively revealing or hiding further parts of the hierarchy. When an item is closed, all of its descendants in the hierarchy are hidden. Clicking the text of any visible item causes the viewer application to jump to the corresponding destination. It can also trigger an action associated with the item.

In addition to supporting navigation within a document, bookmarks can be used to navigate from the source document to another document. They can also be used to launch applications.

**XML representation of bookmarks**

The Bookmarks language describes the structure, appearance, and actions associated with bookmarks that are part of a PDF document.

When a Bookmarks XML document is imported into a PDF document, the bookmarks information becomes part of the PDF document. That is, the bookmarks information is represented within the PDF document, as entries in the PDF outline dictionary. When bookmarks are exported from a PDF document, the entries in the PDF outline dictionary are represented as a Bookmarks XML document.
**Bookmarks that navigate within a PDF document**

The example below shows a Bookmark document that represents the bookmark actions described in the following table.

<table>
<thead>
<tr>
<th>Bookmark title</th>
<th>Level</th>
<th>Effect when clicked</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Slide</td>
<td>1</td>
<td>Changes the view to page 12, fitting that page into the viewer window. The page numbers provided in Bookmarks XML and PDF use zero-based page numbers, but viewer applications show one-based page numbers. As a result, the Bookmarks XML expression PageNum=&quot;11&quot; is displayed as page number 12.</td>
</tr>
<tr>
<td>Second Slide</td>
<td>1</td>
<td>Changes the view to page 13, fitting that page into the viewer window.</td>
</tr>
</tbody>
</table>
| A backup slide       | 2     | Changes the view to page 14. The viewer application also makes the following adjustments to the view:  
  Positions the designated spot of the page at the upper left corner of the viewer window  
  Displays the content at twice its original size                                                                                                           |

**Example: A basic bookmark expression that specifies destinations and views**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Bookmarks xmlns="http://ns.adobe.com/pdf/bookmarks" version="1.0">
  <Bookmark>
    <Dest>
      <Fit PageNum="11"/>
    </Dest>
    <Title>First Slide</Title>
  </Bookmark>
  <Bookmark>
    <Dest>
      <Fit PageNum="12"/>
    </Dest>
    <Title>Second Slide</Title>
  </Bookmark>
  <Bookmark>
    <Dest>
      <XYZ PageNum="13" Left="45.0" Top="530.0" Zoom="2.0"/>
    </Dest>
    <Title>A backup slide</Title>
  </Bookmark>
</Bookmarks>
```
Bookmarks that navigate within a document and change views

The following example shows a Bookmark document that represents the bookmark actions described in the following table.

<table>
<thead>
<tr>
<th>Bookmark title</th>
<th>Level</th>
<th>Effect when clicked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Title</td>
<td>1</td>
<td>The viewer application displays the page contents that fall within a rectangle at the lower right side of page 6. It also magnifies the contents to fit the viewer window.</td>
</tr>
<tr>
<td>Document Subtitle</td>
<td>2</td>
<td>The viewer application displays the page contents that fall within a rectangle at the lower right side of page 24. It also magnifies the contents to fit the viewer window.</td>
</tr>
<tr>
<td>Preface</td>
<td>3</td>
<td>The viewer application displays the page contents that fall within a rectangle at the lower-left side of page 24. It also magnifies the contents to fit the viewer window.</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>3</td>
<td>The viewer application displays the first page of the file called MyOtherFile.pdf. The lack of a path indicates the file is co-located with the PDF file in which the bookmarks reside. Depending on viewer application preferences, the viewer application either displays the file in the current viewer window or opens a new viewer window.</td>
</tr>
</tbody>
</table>

Example: A basic bookmark expression that specifies go-to actions

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Bookmarks xmlns="http://ns.adobe.com/pdf/bookmarks" version="1.0">
  <Bookmark>
    <Action>
      <Launch NewWindow="true"> <File Name="MyFile.pdf"/>
    </Launch>
    <Title>Contents</Title>
  </Bookmark>
  <Bookmark>
    <Action>
      <GoTo>
        <Dest>
          <FitR PageNum="23" Left="28" Bottom="2" Right="23" Top="404"/>
        </Dest>
      </GoTo>
      <Title>Preface</Title>
    </Bookmark>
    <Dest>
      <FitR PageNum="23" Left="285" Bottom="207" Right="523" Top="404"/>
    </Dest>
    <Title>Document Subtitle</Title>
  </Bookmarks>
```
Bookmarks that launch views or applications

The following example shows a Bookmark document that represents the bookmark actions described in the following table.

<table>
<thead>
<tr>
<th>Bookmark title</th>
<th>Level</th>
<th>Effect when clicked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open a new document in an external file</td>
<td>1</td>
<td>The viewer application opens a new window in which it displays the PDF document located at C:/adobe/samples/assembler7/articles.pdf</td>
</tr>
<tr>
<td>Launch WordPad</td>
<td>1</td>
<td>The viewer application launches the WordPad application, which is located at C:/Program Files/Windows NT/Accessories/wordpad.exe</td>
</tr>
</tbody>
</table>

**Note:** Either the forward slash or backward slash separators can be used in paths.

**Example: A basic bookmark expression that specifies launch actions**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Bookmarks xmlns="http://ns.adobe.com/pdf/bookmarks" version="1.0">
  <Bookmark>
    <Action>
      <Launch NewWindow="true">
        <File Name="C:\adobe\samples\assembler7\articles.pdf"/>
      </Launch>
    </Action>
    <Title>Open a new document in an external file</Title>
  </Bookmark>
  <Bookmark>
    <Action>
      <Launch>
        <Win Name="C:/Program Files/Windows NT/Accessories/wordpad.exe"/>
      </Launch>
    </Action>
    <Title>Launch WordPad</Title>
  </Bookmark>
</Bookmarks>
```
Bookmarks XML language reference

Action

The action performed when the bookmark is activated.

```xml
<Action>
    <GoTo> [1] or
    <GoToR> [1] or
    <GoToE> [1] or
    <Launch> [1] or
    <URI> [1] or
    <Named> [1]
</Action>
```

Can be contained in the `Bookmark` element.

The PDF counterpart to this element is the `A` entry of the `outline` dictionary.

The `Action` element expresses actions that are performed when the user clicks the bookmark. The actions can launch an application or change the displayed page’s appearance state.

Bookmark

An individual bookmark item within the hierarchy.

```xml
<Bookmark
    <Color
    <Styles

>[
    <Title> [1]
    <Dest> [0..1]
    <Action> [0..1]
    <Bookmark> [0..n]
</Bookmark>
```

Can be contained in the `Bookmarks` element, which is the Bookmarks XML language root.

The PDF counterpart to this element is the `outline` dictionary.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>Optional. An array of three numbers in the range 0.0 to 1.0. The array represents the components in the DeviceRGB color space of the color used for the outline entry’s text. Default value: 0.0 0.0 0.0, which corresponds to black.</td>
</tr>
<tr>
<td>Styles</td>
<td>Optional. The font style to use for the bookmark entry. The following values are supported: Italic, Bold, Italic Bold - Specifies that both the italics and bold font faces are used. The words must be separated with a space.</td>
</tr>
</tbody>
</table>
**Bookmarks**

Root element of the Bookmarks XML language.

```xml
<Bookmarks
    <Bookmark> [0..n]
</Bookmarks>
```

**Desc**

A description of a file destination.

```xml
.Desc
    xs:string
</Desc>
```

Can be contained in the File element.

**Dest**

The destination displayed when this item is activated.

```xml
.Dest
    <XYZ> [0..1] or
    <Fit> [0..1] or
    <FitH> [0..1] or
    <FitV> [0..1] or
    <FitR> [0..1] or
    <FitB> [0..1] or
    <FitBH> [0..1] or
    <FitBV> [0..1]
</Dest>
```

Can be contained in the elements Bookmark, GoTo, GoToE, and GoToR.

A destination defines a particular view of a document. It consists of the following attributes:

- Page of the document displayed
- Location of the document window on that page
- Magnification (zoom) factor to use when displaying the page

**File**

The full path of a file or the file’s identity within a PDF package or portfolio.

```xml
.File
    FSType="xs:string"
    FSType_enc="xs:token"
    Name="xs:string"
    Name_enc="xs:token"
>
    <Desc> [0..1]
</Desc>
```

Can be contained in the GoToE, GoToR and Launch elements.
The PDF counterpart to this element is the file specification dictionary.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSType</td>
<td>Optional. The name of the file system used to interpret this file specification. If this entry is present, the designated file system interprets the value of the Name attribute. PDF defines only one standard file system name, URL (see the PDF Reference); an application or plug-in extension can register the names of other file systems.</td>
</tr>
<tr>
<td>FSType_enc</td>
<td>Optional. The encoding used to represent the FSType value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
<tr>
<td>Name</td>
<td>Required. Filename, using UNIX or DOS notation. The following values are equivalent: C:/adobe/samples/assembler7/MyFile.pdf or C:\adobe\samples\assembler7\MyFile.pdf</td>
</tr>
<tr>
<td>Name_enc</td>
<td>Optional. The encoding used to represent the Name value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
</tbody>
</table>

**Fit**

Specifies a page destination and viewer characteristics that fit the entire page into the viewer window.

```xml
<Fit
    PageNum="xs:nonNegativeInteger"
/>
```

Can be contained in the `Dest` element.

When a user clicks a bookmark that uses the `Fit` element, the viewer application adjusts the page display to fit the window. Specifically, it magnifies the contents to fit the entire page within the window, both horizontally and vertically. If the required horizontal and vertical magnification factors are different, the viewer application uses the smaller of the two, centering the page within the window in the other dimension.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNum</td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
</tbody>
</table>

**FitB**

Specifies a page destination and viewer characteristics that fit the page's bounding box into the viewer window.

```xml
<FitB
    PageNum="xs:nonNegativeInteger"
/>```
Can be contained in the `Dest` element.

When a user clicks a bookmark that uses the `FitBH` element, the viewer application adjusts the page to fit the bounding box. Specifically, it displays the designated page with its contents magnified enough to fit its bounding box entirely within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, the viewer application uses the smaller of the two, centering the bounding box within the window in the other dimension.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageNum</td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
</tbody>
</table>

**FitBH**

Specifies a page destination and viewer characteristics that set the top boundary of the page. This element also fits the width of the page's bounding box into the viewer window.

```xml
<FitBH
    pageNum="xs:nonNegativeInteger"
    top="xsd:float"
/>
```

Can be contained in the `Dest` element.

When a user clicks a bookmark that uses the `FitBH` element, the viewer application makes the following adjustments to the displayed page:

- Position a vertical coordinate on the page at the top edge of the window. The `top` attribute specifies the vertical coordinate.
- Contents of the page magnified enough to fit the entire width of its bounding box within the window.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageNum</td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
<tr>
<td>top</td>
<td>Vertical coordinate in points that specifies the top of the viewed page.</td>
</tr>
</tbody>
</table>

**FitBV**

Specifies a page destination and viewer characteristics that set the left boundary of the page. This element also sets fits the height of the page's bounding box into the viewer window.

```xml
<FitBV
    pageNum="xs:nonNegativeInteger"
    left="xsd:float"
/>
```
Can be contained in the `Dest` element.

When a user clicks a bookmark that uses the `FitH` element, the viewer application makes the following adjustments to the displayed page:

- Positions the horizontal coordinate specified in the `Left` attribute at the left edge of the window.
- Magnifies the contents of the page to fit the entire height of its bounding box within the window.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNum</td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
<tr>
<td>Left</td>
<td>Horizontal coordinate in points that specifies the left side of the viewed page.</td>
</tr>
</tbody>
</table>

**FitH**

Specifies a page destination and viewer characteristics that set the top of the page and horizontally fits the page into the viewer window.

```xml
<FitH
    PageNum="xs:nonNegativeInteger"
    Top="xsd:float"
/>
```

Can be contained in the `Dest` element.

When a user clicks a bookmark that uses the `FitH` element, the viewer application makes the following adjustments to the displayed page:

- Positions the vertical coordinate specified in the `Top` attribute positioned at the top edge of the window.
- Magnifies the contents of the page to fit the entire width of the page within the window.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNum</td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
<tr>
<td>Top</td>
<td>Vertical coordinate in points that specifies the top of the displayed portion of the page.</td>
</tr>
</tbody>
</table>

**FitR**

Specifies a page destination and viewer characteristics that fit a rectangle on the page into the viewer window.

```xml
<FitR
    PageNum="xs:nonNegativeInteger"
    Left="xsd:float"
    Bottom="xsd:float"
    Right="xsd:float"
/>
```
Can be contained in the `Dest` element.

When a user clicks a bookmark with the `FitR` element, the viewer application displays the portion of the page designated by the rectangle. It also magnifies the contents in the rectangle to fit the viewer window. The viewer application magnifies the page contents horizontally and vertically. If the required horizontal and vertical magnification factors are different, the application uses the smaller of the two, centering the rectangle within the window in the other dimension.

The `Left`, `Bottom`, `Right`, and `Top` attributes define the rectangle. The rectangle coordinates are in points, with the origin at the lower left corner of the page.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PageNum</code></td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
<tr>
<td><code>Left</code></td>
<td>The horizontal coordinate for the left side of the rectangle, in points.</td>
</tr>
<tr>
<td><code>Bottom</code></td>
<td>The vertical coordinate for the bottom of the rectangle, in points.</td>
</tr>
<tr>
<td><code>Right</code></td>
<td>The horizontal coordinate for the right side of the rectangle, in points.</td>
</tr>
<tr>
<td><code>Top</code></td>
<td>The vertical coordinate for the top of the rectangle, in points.</td>
</tr>
</tbody>
</table>

### FitV

Specifies a page destination and viewer characteristics that set the left border of the page and vertically fit the page into the viewer window.

```xml
<FitV
    PageNum="xs:nonNegativeInteger"
    Left="xsd:float"
/>```

Can be contained in the `Dest` element.

When a user clicks a bookmark that uses the `FitV` element, the viewer application makes the following adjustments to the displayed page:

- Positions the horizontal coordinate specified by the `Left` attribute positioned at the left edge of the window.
- Magnifies the page content to fit the entire height of the page within the window.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PageNum</code></td>
<td>Required. The page number of the destination, where 0 is the first page in the document.</td>
</tr>
<tr>
<td><code>Left</code></td>
<td>Horizontal coordinate in points that specifies the left side of the displayed portion of the page.</td>
</tr>
</tbody>
</table>
GoTo

Specifies a page destination in the document.

```xml
<GoTo>
    <Dest> [1]
</GoTo>
```

Can be contained in the `Action` element.

The PDF counterpart to this element is an `Action` dictionary with the `S` key set to `GoTo`.

GoToE

Specifies a destination in an embedded or embedding file. Embedded files are file attachments or package files. Embedding files are the parent files of attachments files or package files.

```xml
<GoToE
    NewWindow="true" or "false"
>
    <File> [0..1] or
    <Dest> [0..1] or
    <Target> [0..1]
</GoToE>
```

Can be contained in the `Action` element.

The PDF counterpart to this element is an `Action` dictionary with the `S` key set to `GoToE`.

An embedded go-to action is similar to a remote go-to action (`GoToR` element). The embedded go-to action differs in that it allows jumping to or from a PDF file that is embedded in another PDF file. Such embedded files are package files or file attachments.

Here is an example that adds a bookmark with a GoToE action that links to a package file within FileB.pdf, a separate file. The destination package file within FileB.pdf has the path `/FolderA/FolderA1/foo.pdf`. The file that contains the bookmark and FileB.pdf are stored in the same folder.

The `File` element's `Name` attribute specifies the separate PDF file that contains the package. The `Target` element's `Filename` attribute precedes the filename with an index that indicates the folder that contains the file. In this example, the folder containing foo.pdf is the second folder in the `Folders` element. The PackageFile XML file exported from FileB.pdf appears after the example. The index angle brackets use escape sequences to avoid conflicting with the bookmarks XML declarations.

**Example: Bookmark xml that links to an embedded file in a folder**

```xml
<Bookmarks xmlns="http://ns.adobe.com/pdf/bookmarks" version="1.0">  
  <Bookmark>  
    <Title>Go to package file</Title>  
    <Action>  
      <GoToE NewWindow="true">  
        <Dest>  
          <FitH PageNum="0" />  
        </Dest>  
        <File Name="FileB.pdf"/>  
        <Target Relationship="Child" Filename_enc="UTF-16"  
            Filename="&lt;2&gt;foo.pdf"/>  
      </GoToE>  
    </Action>  
  </Bookmark>  
</Bookmarks>
```
For clarity, here is the PackageFile XML file exported from FileB.pdf. This file shows the folder structure and
nameKeys.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PackageFiles xmlns="http://ns.adobe.com/DDX/PackageFiles/1.0/">
  <Package/>
  <Folders>
    <Folder name="FolderA">
      <Folder name="FolderA1"/>
    </Folder>
    <Folder name="FolderB"/>
  </Folders>
  ...
  <PackageFile attachmentKey="doc1_attach.0000.0002"
     nameKey="/FolderA/FolderA1/foo.pdf">
    <File creationDate="2009-12-18T08:09:16-08:00"
       mimeType="application/pdf"
       modificationDate="2009-10-06T09:32:10-05:00"
       size="39916">
      <Filename>foo.pdf</Filename>
    </File>
  </PackageFile>
</PackageFiles>
```

You can add a bookmark to a PDF file that is then included as a package file in a PDF package or portfolio. You can also use this example to add a bookmark to a PDF file that is already a package file in a PDF package or portfolio. To add a bookmark to a package file, use the following steps:

1. **Use the PackageFiles result element** to export the package file to modify. This element also exports information about the package files.

2. **Use the Bookmarks source element** to add the bookmark to the exported package file.

3. **From the PackageFiles result**, remove all PackageFile declarations except the one for the modified package file.

4. **Use the PackageFiles import element** to update the original PDF file with modified PackageFile XML file. The Document Manipulation APIs replaces the original package file with the modified package file (the one that you added the bookmark to).

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewWindow</td>
<td>Optional. Specifies whether the destination is displayed in a new window. If true, the destination document is opened in a new window; if false, the destination document replaces the current document in the same window. If this entry is absent, the viewer application honors the current user preference.</td>
</tr>
</tbody>
</table>
GoToR

Specifies a destination in another document (go to remote).

```xml
<GoToR
    NewWindow="true" or "false"
>
    <File> [0..1]
    <Dest> [0..1]
</GoToR>
```

Can be contained in the `Action` element.

The PDF counterpart to this element is an `Action` dictionary with the `S` key set to `GoToR`.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewWindow</td>
<td>Optional. Specifies whether the destination is displayed in a new window. If true, the destination document is opened in a new window; if false, the destination document replaces the current document in the same window. If this entry is absent, the viewer application honors the current user preference.</td>
</tr>
</tbody>
</table>

Launch

Launches an application, opens a file, or prints a file.

```xml
<Launch
    NewWindow="true" or "false"
>
    <File> [0..1] or
    <Win> [0..1]
</Launch>
```

Can be contained in the `Action` element.

The PDF counterpart to this element is an `Action` dictionary with the `S` key set to `Launch`. See examples in "Bookmarks that launch views or applications" on page 321.

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewWindow</td>
<td>Optional. Specifies whether the destination is displayed in a new window. If true, the destination document is opened in a new window; if false, the destination document replaces the current document in the same window. If this entry is absent, the viewer application honors the current user preference.</td>
</tr>
</tbody>
</table>

Named

Execute an action predefined by the viewer application.

```xml
<Named
    Name="xs:string"
```
Can be contained in the Action element.

The PDF counterpart to this element is an Action dictionary with the S key set to Named.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Required. Name of the action to perform. PDF viewer applications are expected to support the following named actions: NextPage, PrevPage, FirstPage, LastPage. Non-standard PDF viewer applications can support additional named actions.</td>
</tr>
<tr>
<td>Name_enc</td>
<td>Optional. The encoding used to represent the Name value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
</tbody>
</table>

Target

Specifies the embedded or embedding document in which the destination resides.

<Target
   Relationship="Parent" or "Child"
   Filename="xs:string"
   Filename_enc="xs:token"
   PageNum="xs:nonNegativeInteger"
   AnnotName="xs:string"
   AnnotNum="xs:nonNegativeInteger"
>
   <Target> [0..1]
</Target>

Can be contained in the GoTo element.

The PDF counterpart to this element is a Target dictionary.

A PDF document can contain document-level attachments or page-level attachments.

The Target element provides path information to the embedded or embedding document. Target elements can be nested, with each nesting level specifying one additional embedding level. Consider a PDF document that contains an embedded (attached) document. The embedded document can also contain an embedded document. Nested Target elements allow a bookmark to navigate from the top-level document to the lowest-level one, or the reverse.
### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship</td>
<td>Required. Specifies whether the target document is a parent or child of the current document.</td>
</tr>
<tr>
<td>Filename</td>
<td>Required if the destination is an embedded file such as a file attachment or package file. To determine the value to use for this attribute, use the <code>PackageFiles</code> result or <code>FileAttachments</code> result element to obtain the <code>nameKey</code> for the destination file. When DDX processors embed a file, they typically use the filename as the starting point for determining the <code>nameKey</code>. They modify that name to ensure each <code>nameKey</code> is unique and to represent PDF Portfolio folders. For example, if the original filename is <code>MyFile.pdf</code> and there are other identically named files in the package, the Document Manipulation APIs may assign <code>MyFile.001.pdf</code> as the <code>nameKey</code>. The value for this attribute has the format <code>&lt;folder_index&gt;nameKey</code>. It is not the whole path. If the file is not in a folder, omit <code>&lt;folder_index&gt;</code>. If the file is in a folder, the value of <code>folder_index</code> corresponds to the order in which the folder is defined in the PackageFile. Use escape encoding for the angle brackets. More specifically, use <code>&amp;lt;</code> in place of the left angle bracket <code>&lt;</code>, and use <code>&amp;gt;</code> in place of the right angle bracket <code>&gt;</code>. Consider a target file in the second folder defined in the PackageFile’s <code>Folders</code> element where the <code>nameKey</code> is <code>MyFile.001.pdf</code>. In this case, use <code>Filename=&quot;&amp;lt;2&amp;gt;MyFile.001.pdf&quot;</code>.</td>
</tr>
<tr>
<td>Filename_en</td>
<td>Optional. The encoding used to represent the <code>Filename</code> value in the PDF document. If this attribute is omitted, the <code>PDFDocEncoding</code> character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
<tr>
<td>PageNum</td>
<td>Required if the destination is a file-attachment annotation (page-level attachment). This attribute specifies the page number that bears the destination annotation, where 0 is the first page in the document.</td>
</tr>
<tr>
<td>AnnotName</td>
<td>Optional. If the destination is a file-attachment annotation (page-level attachment), this attribute specifies the name of the destination annotation. Either the <code>AnnotName</code> attribute or the <code>AnnotNum</code> attribute must be provided to specify file-attachment annotation targets.</td>
</tr>
<tr>
<td>AnnotNum</td>
<td>Optional. If the destination is a file-attachment annotation (page-level attachment), this attribute specifies the number of the destination annotation. A value of 0 specifies the first annotation on the page. Either the <code>AnnotName</code> attribute or the <code>AnnotNum</code> attribute must be provided to specify file-attachment annotation targets.</td>
</tr>
</tbody>
</table>

**Title**

The text displayed on the screen for this bookmark entry.

`<Title>`
**Bookmark title**

Can be contained in the **Bookmark** element.

**URI**

An action that resolves and displays a URL in the viewer window.

```
<URI
   URI="xs:string"
   IsMap="true" or "false"
/>
```

Can be contained in the **Action** element.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>Required. The uniform resource identifier to resolve, encoded in 7-bit ASCII.</td>
</tr>
<tr>
<td>IsMap</td>
<td>Optional. Specifies whether to track the mouse position when the URL is resolved. This entry applies only to actions triggered by the user’s clicking an annotation. It is ignored for actions associated with outline items or with a document’s PDF OpenAction entry. <strong>Note:</strong> This attribute is not relevant to bookmarks.</td>
</tr>
</tbody>
</table>

**Win**

Windows-specific launch parameters.

```
<Win
   Name="xs:string"
   Name_enc="xs:token"
   Dir="xs:string"
   Dir_enc="xs:token"
   Action="open" or "print"
   Action_enc="xs:token"
   Params="xs:string"
   Params_enc="xs:token"
/>
```

Can be contained in the **Launch** element.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Required. The filename of the application to launch or the document to open or print. The name format uses standard Windows path format.</td>
</tr>
<tr>
<td>Name_enc</td>
<td>Optional. The encoding used to represent the Name value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
<tr>
<td>Dir</td>
<td>Optional. A string specifying the default directory in standard DOS syntax.</td>
</tr>
<tr>
<td>Dir_enc</td>
<td>Optional. The encoding used to represent the Dir value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
<tr>
<td>Action</td>
<td>Optional. A string specifying the operation to perform:</td>
</tr>
<tr>
<td></td>
<td>- open - Open a document.</td>
</tr>
<tr>
<td></td>
<td>- print - Print a document.</td>
</tr>
<tr>
<td></td>
<td>If the Name attribute designates an application instead of a document, this entry is ignored and the application is launched. Default value: open.</td>
</tr>
<tr>
<td>Action_enc</td>
<td>Optional. The encoding used to represent the Action value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
<tr>
<td>Params</td>
<td>Optional. A parameter string passed to the application designated by the Name attribute. This entry is ignored if Name designates a document.</td>
</tr>
<tr>
<td></td>
<td>If Params_enc is omitted, encoding is assumed to be PDFDocEncoding or UTF-16, based on the characters encountered in the Params value.</td>
</tr>
<tr>
<td>Params_enc</td>
<td>Optional. The encoding used to represent the Params value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See “Supported character encodings” on page 335.)</td>
</tr>
</tbody>
</table>

A destination that specifies positioning of a specific page in the viewer window.

```xml
<XYZ
    PageNum="xs:nonNegativeInteger"
    Left="xsd:float"
    Top="xsd:float"
    Zoom="xsd:float"
/>
```

Can be contained in the Dest element.

When a user clicks a bookmark that uses the XYZ element, the viewer application makes the following adjustments to the displayed page:

- Positions the upper-left corner of the page at the upper-left corner of the window.
- Magnifies the contents of the page as specified in the Zoom attribute.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageNum</td>
<td>Required. The page number for the destination, where 0 is the first page in the document.</td>
</tr>
<tr>
<td>Left</td>
<td>Optional. Horizontal coordinate for the upper left side of the view, expressed in points. This value describes the distance from the left side of the page to the left side of the view. If omitted, the PDF viewer application uses the current value for this attribute.</td>
</tr>
<tr>
<td>Top</td>
<td>Optional. Vertical coordinate for the upper left side of the view, expressed in points. This value describes the distance from the bottom of the page to the top of the view. If omitted, the PDF viewer application uses the current value for this attribute.</td>
</tr>
<tr>
<td>Zoom</td>
<td>Optional. Zoom factor, expressed as a fraction. For example a value of 0.3 scales down the document to 30%. If this attribute has a value of 0 or is omitted, the PDF viewer application uses the current value for this attribute. To express the scaling factor in terms of Fit Page or Fit Width, use destination child elements such as <code>Fit</code>, <code>FitB</code>, or <code>FitBH</code>.</td>
</tr>
</tbody>
</table>

Supported character encodings

The Document Manipulation APIs supports the character encodings described in the following table. These encodings are used to convert a string or name into encoding in the PDF document. The original string or name is an attribute provided in the Bookmarks XML document.

Depending on the character encodings available through your installation's Java Virtual Machine, the Document Manipulation APIs can also support additional encodings. These additional encodings are described in the Extensible Markup Language (XML) Specification, 1.0. Examples of such additional character encodings are ISO-8859-1, ISO-10646-UCS-2, and ISO-2022.

<table>
<thead>
<tr>
<th>Encoding names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE64</td>
<td>The Base16, Base32, and Base64 Data Encodings, RFC 3548 (<a href="http://ietf.org/rfc/rfc3548.txt">http://ietf.org/rfc/rfc3548.txt</a>)</td>
</tr>
<tr>
<td>UTF-8</td>
<td>The Unicode Standard, Version 4.0, (<a href="http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404">http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404</a>)</td>
</tr>
<tr>
<td>UTF-16</td>
<td>The Unicode Standard, Version 4.0, (<a href="http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404">http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404</a>)</td>
</tr>
<tr>
<td>PDFDocEncoding</td>
<td>PDF Reference, version 1.6, (<a href="http://www.adobe.com/go/pdf_developer">http://www.adobe.com/go/pdf_developer</a>)</td>
</tr>
</tbody>
</table>
The following illustration shows how the filename is converted from the character encoding of the XML file (UTF-8) into the character encoding specified in the `Name_enc` attribute. The resulting encoded name is then incorporated into the PDF document, along with an indication of its character encoding, BASE64.

```
<?xml version="1.0" encoding="UTF-8"?>
<Bookmarks>
  <Bookmark>
    <File
      Name="MyFile.pdf"
      Name_enc="BASE64"
    />
  </Bookmark>
</Bookmarks>
```

Outline dictionary represents the filename "MyFile.pdf" encoded as BASE64.
Document Text Language

The DocText language is an XML representation of the words or sentences used in a PDF document. The language includes words in the body, headers, footers, watermarks, and backgrounds. A DocText document can optionally include the positions on the page of each word. The order in which the words are listed is not guaranteed to be the reading order.


About the DocText XML language


**Example: DocText document that provides words and word locations (long form)**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DocText xmlns="http://ns.adobe.com/DDX/DocText/1.0/">
  <WithQuads>
    <Page pageNumber="n">
      <Word>word1</Word>
      <Quad>
        <P1 X="59.588" Y="590.5"/>
        <P2 X="54.044" Y="590.5"/>
        <P3 X="54.044" Y="575.366"/>
        <P4 X="59.588" Y="575.366"/>
      </Quad>
    </Page>
  </WithQuads>
</DocText>
```

**Example: DocText document that provides only words (short form)**

```xml
<?xml version="1.0" encoding="UTF-16"?>
<DocText xmlns="http://ns.adobe.com/DDX/DocText/1.0/">
  <TextPerPage>
    <Page pageNumber="n">word1 word2 word3...</Page>
  </TextPerPage>
</DocText>
```

**Example: DocText document that provides the sentences for each paragraph**

```xml
<DocText xmlns="http://ns.adobe.com/DDX/DocText/1.0/">
  < ParagraphsPerPage>
    <Page pageNumber="1">
      <Paragraph>
        <Sentence>The Tell Tale Heart Revealed Would you call a man crazy, calm, or patient for killing a man just because he feared the man’s eye? </Sentence>
      </Paragraph>
    </Page>
  </ParagraphsPerPage>
</DocText>
```
Example: DocText document that provides only words in paragraphs and sentences

```
<DocText>
  <ParagraphsPerPage>
    <Page>
      <Paragraph>
        <Sentence>
          xs:string
        </Sentence>
        <Sentence>word1 word2 word3...</Sentence>
      </Paragraph>
      <Paragraph>...</Paragraph>
    </Page>
    <Page>...</Page>
  </ParagraphsPerPage>
</DocText>
```

Text encoding

The text encoding used for the words included in a DocText document is specified in the encoding of the result XML document, typically UTF-8.
DocText reference

This reference provides the syntax and grammar for the DocText language.

DocText

Root element of a DocText document.

\[
\text{<DocText>}
\text{ \<WithQuads\> or \<ParagraphsPerPage\> or \<TextPerPage\> [1]}\]
\[
\text{</DocText>}
\]

Page

Lists the words that appear on a particular page.

\[
\text{<Page \ page\text{-}number="n">}
\text{ \<Word\> [0..n, present only if a child of the WithQuads element]}\]
\text{ \ word1 word2 word3 [present only if a child of the TextPerPage element]}\]
\[
\text{</Page>}
\]

Can be contained in the elements \text{WithQuads} or \text{TextPerPage}.

If the \text{Page} element is a child of the \text{TextPerPage} element, it contains a space-separated list of the words that appear on the page. The words are not sorted alphabetically, but they are roughly in reading order on the page.

The words are encoded using the encoding specified in the first line of the DocText file, usually UTF-8.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageNumber</td>
<td>Ordinal page number of a page in the document.</td>
</tr>
</tbody>
</table>

Paragraph

Sentences contained in a paragraph on the page.

\[
\text{<Paragraph>}
\text{ \<Sentence\> [0..n]}\]
\text{ \ xs:string}\]
\[
\text{</Sentence>}
\text{</Paragraph>}
\]

A sentence contains a string of words terminated by punctuation.

Can be contained in the \text{Page} element.

ParagraphsPerPage

Outer element containing paragraphs and sentence strings.

\[
\text{<ParagraphsPerPage>}
\]
<Page pageNumber="xs:decimal"> [0..n]
  <Paragraph> [0..n]
</ParagraphsPerPage>

Can be contained in the DocText element.

**P1**

Specifies one corner of a word’s bounding box.

```xml
<P1 x="X-coordinate" y="Y-coordinate"/>
```

Can be contained in the Quad element.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-coordinate of one corner of a bounding box.</td>
</tr>
<tr>
<td>y</td>
<td>Y-coordinate of one corner of a bounding box.</td>
</tr>
</tbody>
</table>

**P2**

See the P1 element.

**P3**

See the P1 element.

**P4**

See the P1 element.

**Quad**

Specifies the four corners that describe an area on the page where the word appears.

```xml
<Quad>
  <P1> [1]
  <P2> [1]
  <P3> [1]
  <P4> [1]
</Quad>
```

Can be contained in the Page element.
TextPerPage

Specifies the words used on an individual page, without specifying the location of those words.

```xml
<TextPerPage>
  <Page> [0..n]
</TextPerPage>
```

Can be contained in the `DocText` element.

WithQuads

Provides the page number and placement on the page of each word in the document.

```xml
<WithQuads>
  <Page/> [0..n]
</WithQuads>
```

Can be contained in the `DocText` element.

Word

Describes the location of an individual word on the page.

```xml
<Word>
  word
  <Quad/>
</Word>
```

Can be contained in the `Page` element.

The contents of this element is a single word that appears on the page. Text encoding used for this work is specified in the first line of the DocText file, usually UTF-8.
The File Attachments language is an XML language that describes file attachments in a set of PDF documents. The file attachments can be returned as named data streams. An Attachments document does not contain file attachments.

About the Attachments XML language

The namespace of the Attachments language is http://ns.adobe.com/DDX/Attachments/1.0/, and the root element is Attachments. The schema is installed in the product Documentation folder.

The Communications - Document Manipulation APIs returns an Attachments document in response to the appearance of the FileAttachments result element in a DDX document. The FileAttachments result element specifies the source documents for which file attachment information is desired. It can also specify keys that identify which file attachments to consider and whether those attachments are returned to the client as separate data streams.

An Attachments document contains a description for each file attachment specified in the FileAttachments result element. The description includes the file attachment’s unique identifier, filename, description, and MIME-type. The Document Manipulation APIs assigns a unique identifier to each file attachment, regardless of whether the attachments are returned to the client.

File attachments are identified using unique identifiers rather than filenames because the original filename cannot always be decoded.
Attachments reference

Attachment

Describes a single file attachment.

```xml
<Attachment
    attachmentKey="xs:string"
    name="xs:string"
>
    <File> [1]
    <Description> [0..1]
    <Page> [0..1]
</Attachment>
```

Can be contained in the Attachments element.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachmentKey</td>
<td>The contrived name associated with the output stream.</td>
</tr>
<tr>
<td>name</td>
<td>Name under which the file was attached to the PDF document, if attached at the document level. If the file was attached at the page level, there is no name. This attribute corresponds to the nameKeys attribute in the DDX language FileAttachments element.</td>
</tr>
</tbody>
</table>

Attachments

Describes some or all of the file attachments in a PDF document. This element is empty if no file attachments are extracted.

```xml
<Attachments>
    <Attachment> [1..n]
</Attachments>
```

This element is the root element for the Attachments XML.

Description

Provides the description for the file attachment.

```xml
<Description>
    xs:string
</Description>
```

Can be contained in the Attachment element.

The value provided is taken from the source PDF document Attachment Description property. A user can set this property by selecting the attachment and then selecting the properties menu. If the source document omits a value for the Attachment Description, this element is omitted.
File

```xml
<File
    mimeType="xs:string"
    size="xs:integer"
    creationDate="xs:dateTime"
    modificationDate="xs:dateTime"
>
    <FileName> [1..n]
</File>
```

Can be contained in the `Attachment` element.

## Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mimeType</td>
<td>Optional. MIME type of the file. If this information is unknown, the attribute is absent.</td>
</tr>
<tr>
<td>size</td>
<td>Optional. Size of the file, in bytes. If this information is unknown, the attribute is absent.</td>
</tr>
<tr>
<td>creationDate</td>
<td>Optional. Creation date of the file. If this value is unknown, the attribute is absent.</td>
</tr>
<tr>
<td>modificationDate</td>
<td>Optional. Date the file was last modified. If this value is unknown, the attribute is absent.</td>
</tr>
</tbody>
</table>

### FileName

Filename and the success of decoding that name from the source PDF document.

```xml
<FileName
    unmappableCharacters="true"
    fromEncoding="algorithm name"
    success="true" or "false"
>
    decoded file name
</FileName>
```

Can be contained in the `File` element.

Occasionally, the filename encoding algorithm in the original PDF document is unknown. In such cases, the decoding process is a trial and error process that involves trying multiple algorithms supplied by the DDX `FilenameEncoding` element.

The name is encoded using UTF-8, as specified in the XML encoding attribute.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unmappableCharacters</td>
<td>Flag indicating whether, after applying the encoding specified by the <code>fromEncoding</code> attribute, unmappable characters were found in the filename. If unmappable characters are found, this attribute appears in the XML with a value of <code>true</code>. In addition, the following changes occur: Unmappable characters are replaced with the Unicode substitution character (<code>uFFFD</code>). <code>success</code> attribute is set to <code>false</code>. Modified filename text is displayed. If unmappable characters are not found, this attribute is absent.</td>
</tr>
<tr>
<td>fromEncoding</td>
<td>Encoding applied to produce the string content of the <code>FileName</code> element.</td>
</tr>
<tr>
<td>success</td>
<td>Flag indicating the success of the attempt to decode the filename. This attribute can have the following values:</td>
</tr>
<tr>
<td></td>
<td>false - Decoding was unsuccessful. If the <code>unmappableCharacters</code> attribute is present, some characters in the filename could not be decoded. If that attribute is absent, the decoding attempt failed.</td>
</tr>
<tr>
<td></td>
<td>true - Decoding was successful.</td>
</tr>
</tbody>
</table>

Location

Specifies a position on a page as coordinates.

```xml
<Location
  x="length"
  y="length"
/>
```

Can be contained in the `Page` element.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Specifies the horizontal location on the page where the icon is placed. The value provides the horizontal distance from the lower left corner of the page to the upper right corner of the icon.</td>
</tr>
<tr>
<td>y</td>
<td>Specifies the vertical location on the page where the icon is placed. The value provides the vertical distance from the lower left corner of the page to the upper right corner of the icon.</td>
</tr>
</tbody>
</table>
Page

Specifies the page on which a page-level file attachment occurs and the position on that page where the annotation is placed.

```xml
<Page
    pageNumber="xs:integer"
>
    <Location> [0..1]
</Page>
```

Can be contained in the `<Attachment>` element.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageNumber</td>
<td>The number of the page in the PDF document to which the file was attached.</td>
</tr>
</tbody>
</table>
PackageFiles Language

The PackageFiles language is an XML grammar that provides information about package files in a PDF document.

The PackageFiles language is important for obtaining the nameKey that identifies package files. The nameKey for a package file is a contrived name for these reasons:

- Original filename may be known from the original encoding for the given filename.
- Multiple files with the same filename may be present.
- Package file may be contained in a Folder.

Thus, a PackageFiles document helps with identification by providing a mapping of the contrived name to information about the file. The PackageFiles result element returns a PackageFiles document.

About the PackageFiles language

The namespace of the PackageFiles language is http://ns.adobe.com/DDX/PackageFiles/1.0/, and the root element is PackageFiles. The schema is installed in the product Documentation folder.

The Communications - Document Manipulation APIs returns a PackageFiles document in response to the appearance of the PackageFiles result element in a DDX document.

Example: Resultant PackageFiles

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PackageFiles xmlns="http://ns.adobe.com/DDX/PackageFiles/1.0/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ns.adobe.com/DDX/PackageFiles/1.0/
F:\lc\assembler\kendall\schemas\pdfm\packagefiles.xsd"
<Package/>
<Folders>
<Description>A few test files</Description>
<Folder name="Chapters">
<Description>Folder for Chapters</Description>
<Folder name="Chapter 1">
<Description>Folder Chapter 1</Description>
</Folder>
<Folder name="Chapter 2">
</Folder>
</Folder>
<Folder name="Biodynamics">
<Description>Folder for Biodynamics</Description>
</Folder>
</Folders>
<PackageFile
attachmentKey="SimpleFolders.pdf_attach.0000.0001"
nameKey="/Chapters/Chapter 1/chap1.pdf">
<File creationDate="2009-08-03T14:05:10-07:00"
mimeType="application/pdf"
```
The PackageFiles schema provided below represents the XML contained in a PackageFiles result element. The PackageFiles element is the root element.

**Description**

The description associated with this package file or folder.

<Description> "xs:string" </Description>

**DisplayOrder**

The DisplayOrder element contains FieldData elements that identify the specified order of the fields when displayed in a viewer.

<DisplayOrder>
  <FieldData name="xs:string"/> [0..n]
</DisplayOrder>

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. The normalized name of a FieldData as defined in the Schema. The non-normalized name appears as the content in the Schema element.</td>
</tr>
</tbody>
</table>
FieldData

There is one FieldData element for each custom field defined in the Schema, even if no value is set. The FieldData element content is the value (the metadata) for this package file.

```xml
<FieldData
    name="xs:string"
    type="Text" or "Date" or "Number" or "Filename" or "Description" or "ModificationDate" or "CreationDate" or "Size"
>
    "xs:string"
</FieldData>
```

Can be contained in the Folders, Folder, or PackageFile elements.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. The normalized name of a Field as defined in the Schema. The non-normalized name appears as the content in the Schema element.</td>
</tr>
<tr>
<td>type</td>
<td>Required. The type of metadata stored in the Field according to the Schema.</td>
</tr>
</tbody>
</table>

File

The File element provides the basic metadata for the package file extracted from the PDF document. When the PackageFiles schema is applied to an XML source for the PackageFiles import element, the input is similar to that provided to the PackageFiles and FileAttachments source elements.

```xml
<File
    creationDate="xs:dateTime"
    mimeType="xs:string"
    modificationDate="xs:dateTime"
    size="xs:integer"
>
    <Filename
        fromEncoding="xs:dateTime"
        success="true" or "false"
        unmappableCharacters="false" or "true"
        >[1..n]
        "xs:string"
    </Filename>
</File>
```

Within the File element, there is one Filename element that is output for each FilenameEncoding specified in the PackageFiles result element. Multiple encodings can be successfully used to decode the filename, but some successful decoding operations may not yield the correct result.

Using a specified encoding to decode the filename can fail for one of two reasons:

- The filename bytes cannot be mapped to anything in the encoding.
- Some unmappable characters are found, but the decoding process did not completely fail. In this case a filename is provided. The unmappable characters are replaced with Unicode substitution characters.
With PDF documents that conform to PDF 1.7 and later, the filename encoding is known. With PDF documents that conform to earlier versions of PDF, the encoding is unknown. If the `FilenameEncoding` element is in a PDF result block that includes an earlier PDF version, the DDX processor tries default encodings.

The `PackageFiles` import element uses a PackageFiles document. The DDX processor uses that document to update the package file in the PDF result block. A PackageFiles document can have multiple Filename elements for each file, with one Filename per decoding attempt. For each File in the PackageFiles document, the DDX processor determines the filename by selecting the first Filename element with the `success` attribute set or defaulted to "true".

**File Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>creationDate</td>
<td>Optional. The creation date of the file, if known. If not known, this attribute is absent. The default for import is the current date.</td>
</tr>
<tr>
<td>mimeType</td>
<td>Optional. The MIME type of the file, if known. If it is not provided, then for import, this entry is left blank. Leaving the mimeType blank can cause problems when attempting to open the document within the resultant PDF package or portfolio. The default is the empty string.</td>
</tr>
<tr>
<td>modificationDate</td>
<td>Optional. The date on which the file was last modified, if known. If not known, this attribute is absent. The default for import is the current date.</td>
</tr>
<tr>
<td>size</td>
<td>Optional. The size of the file in bytes. The size is calculated as the number of actual bytes in the data stream.</td>
</tr>
</tbody>
</table>

**Filename Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromEncoding</td>
<td>Required. The encoding applied to produce the string content of the Filename element.</td>
</tr>
<tr>
<td>success</td>
<td>Optional. A flag indicating whether the encoding was successfully applied to the filename. A success value of &quot;false&quot;, when the unmappableCharacters attribute is not present, means that the filename failed to completely decode based on the fromEncoding value. It is not required when used for PackageFiles import.</td>
</tr>
<tr>
<td>unmappableCharacters</td>
<td>Optional. A flag indicating whether, after applying the encoding specified by the fromEncoding attribute, unmappable characters were found in the filename. If unmappable characters are found, this attribute appears in the XML and is set to &quot;true&quot;. Additionally, the unmappable characters are replaced with the Unicode substitution character (\uFFFD), the success attribute is set to &quot;false&quot;, and the modified filename text is displayed.</td>
</tr>
<tr>
<td></td>
<td>If unmappable characters are not found, this attribute omitted from the XML.</td>
</tr>
</tbody>
</table>
Folders

(Since 9.0) Root in the folder structure for the PDF package.

```xml
<Folders
    name = "xs:string"
>
    <Description>[0..1]
    <Folder> [0..n]
    <FieldData>[0..n]
</Folders>
```

Contained in the `PackageFiles` elements.

This element describes the folders and subfolders within the PDF package or portfolio. The nesting of folders shows the folder hierarchy.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. Folder name. The name must conform to the encoding specified in the DDX <code>FilenameEncoding</code> element.</td>
</tr>
</tbody>
</table>

Folder

(Since 9.0) Node in the folder structure for the PDF package.

This element has the same properties as the `Folders` element.

Contained in the `Folders` elements.

Package

The package specification for the PDF package.

```xml
<Package>
    <Schema>[0..1]
    <DisplayOrder>[0..1]
    <SortOrder>[0..1]
</Package>
```

PackageFile

Describes a package file selected from the PDF document.

```xml
<PackageFile
    attachmentKey= "xs:string"
    nameKey= "xs:string"
    required= "false" or "true"
>
    <File>[1]
    <Description>[0..1]
    <FieldData>[0..n]
</PackageFile>
```
**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| attachmentKey | Required. The contrived name associated with the output stream for the extracted attachment. For document-level attachments, the syntax is 
\[\text{documentName} + \text{"_attach.0000." + nnnn}\] 
\[\text{The } \text{documentName} \text{ is the value of the PDF source attribute from which the package files are being extracted. The } \text{nnnn} \text{ is a sequence number for the package file.} \] 
\[\text{For example, if PDF source "docA" has two package files, the } \text{attachmentKey} \text{ values will be:} \] 
\[\text{"docA_attach.0000.0001"} \]
\[\text{"docA_attach.0000.0002"} \] |
| nameKey | Optional. The internal name under which the package file was added to the PDF package, when generated by the PackageFiles result element. If the package file is generated by another process and is used by the PackageFiles import DDX element, it is recommended that the nameKey values be the filename and that all filenames be unique. Duplicate file names are allowed if they exist in different folders. Thus, the nameKey would include the folder path. For example, \myFoldername\myFilename. See the description of nameKey for the PackageFiles source DDX element. |
| required | Optional. If the package file is not found in the input map passed into the Document Manipulation APIs operation and the value of this attribute is "true", an exception is thrown and the operation stops. The exception contains the message that the required file associated with attachmentKey is missing from the input map. If this attribute's value is "false" or is not specified, then a warning similar to the exception message is logged and the operation continues. |

**PackageFiles (root element)**

Contains one Package element which contains the package specification and a PackageFile element for each package file selected from the PDF document. The PackageFiles root element will be empty if the PDF document is a single PDF and not a PDF package, or if no package files are extracted.

```xml
<PackageFiles>
  <Package>[0..1]
  <Folders> [0..1]
  <PackageFile> [0..n]
</PackageFiles>
```

**Schema**

The description of the PDF package fields (metadata). The element content of the FieldData element is the textual name displayed to the user in the user interface of the PDF viewing application.

```xml
<Schema>
  <FieldData
```
name="xs:string"
type="Text" or "Date" or "Number" or "Filename" or "Description" or "ModificationDate" or "CreationDate" or "Size" or "CompressedSize"
visible="true" or "false"
editable="false" or "false"
>
"xs:stringxs:string"
</Field> [0..n]
</Schema>

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>editable</td>
<td>Optional. Indicates whether the PDF viewing application should provide support for editing the field value. This attribute has no effect on whether the field is editable by a DDX processor.</td>
</tr>
<tr>
<td>name</td>
<td>Required. The normalized name of a FieldData as defined in the Schema. The non-normalized name appears as the element content. The normalized name is used when assembling PDF packages and comparing FieldData names.</td>
</tr>
<tr>
<td>type</td>
<td>Required. The type identifies the type of data that is stored in this FieldData.</td>
</tr>
<tr>
<td>visible</td>
<td>Optional. The initial visibility of the field in the PDF viewer. At least one FieldData must be specified as visible.</td>
</tr>
</tbody>
</table>

SortOrder

The SortOrder element contains FieldData elements that identify which field values are used for sorting. The first field listed is the main sorting value. Any subsequent fields are used when the first sorting attempt results in duplicate values.

<SortOrder>
  <FieldData name="xs:string"
    ascending="true" or "false"/> [0..n]
</SortOrder>

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascending</td>
<td>Optional. When sorting with this field, this attribute specifies whether to sort in ascending (&quot;true&quot;) or descending (&quot;false&quot;) order.</td>
</tr>
<tr>
<td>name</td>
<td>Required. The normalized name of a FieldData as defined in the Schema. The non-normalized name appears as the content in the Schema element.</td>
</tr>
</tbody>
</table>
Part IV: Special Topics

More advanced settings let you coordinate between DDX expressions, system settings, and Adobe Experience Manager Forms configuration settings.
Handling Out of Memory Problems

Assembling and applying content to large PDF documents can use so much memory that the Document Manipulation APIs is terminated with an out of memory (OOM) exception. You can use operation checkpoints to avoid triggering such exceptions.

**Caution:** Incorrectly setting values in the `DDXProcessorSetting` element can cause your Adobe Experience Manager Forms system to fail. Do not adjust this setting when your system is in a production mode.

Operation checkpoints (`DDXProcessorSetting`)

The DDX grammar defines a new element (`DDXProcessorSetting`) that lets you specify a checkpoint setting. This element can help avoid out of memory (OOM) conditions. This setting specifies when the Communications - Document Manipulation APIs temporarily saves the PDF document to disk when applying `Header` (portfolio navigation pane), `Footer`, `Watermark`, `Background`, or `PageContent` elements. These elements are called the **checkpointed elements**.

About operation checkpoints

Setting operation checkpoints can reduce the amount of memory required to process the document, which can help avoid out of memory conditions on the server. The penalty of setting operation checkpoints is slower performance due to writing the documents to disk and to reapplying credentials for encrypted documents.

The `DDXProcessorSetting` element has the following form:

```xml
<DDXProcessorSetting name="checkpoint" value="positive-integer"/>
```

It can appear as a child of the `DDX` or `PDF` result elements.

In the Document Manipulation APIs, the only supported value for the `name` attribute is `checkpoint`. The checkpoint setting is a unit-less hint to the Document Manipulation APIs. It specifies how many operations to perform in memory before doing a temporary save (checkpoint) of the file to disk. Its value can be set to zero (0), which turns checkpointing off or to any positive integer value. Smaller values trigger more frequent checkpointing and therefore use less memory while applying the checkpointed elements. Conversely, larger values mean that more operations are processed between checkpoints. By default, the value is zero (0).

Only the operations for applying the checkpointed elements count toward the checkpoint trigger.

Determine a checkpoint value

The checkpoint value is specific to individual Adobe Experience Manager Forms environments and is empirically obtained to avoid out of memory situations for particular workflows.

A guideline is every 1000 operations consume 25 MB of memory. A checkpoint value of 4000 allows 100 MB of memory for these operations. The memory consumption is reset when the service begins interpreting a new DDX or when the checkpoint count rolls over.
The suggested checkpoint values are between 500 to 5000. Start your setting at 4000 (the suggested starting point) and reduce it until the assembly completes successfully. Here is a DDX expression that shows the DDXProcessorSetting element with the suggested starting point.

```xml
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
  <DDXProcessorSetting name="checkpoint" value="4000"/>
  <PDF result="outDoc" >
    <Header>
      <Right><StyledText><p>Oct. 2007</p></StyledText></Right>
      <Left><PDF source="paw_icon"/></Left>
    </Header>
    <Footer alternation="OddPages">
      <Right><StyledText><p>Page <_PageNumber/></p></StyledText></Right>
    </Footer>
    <Footer alternation="EvenPages">
      <Left><StyledText><p>Page <_PageNumber/></p></StyledText></Left>
    </Footer>
    <PDF source="cover" bookmarkTitle="Cover" />
    <PDF source="shasta" bookmarkTitle="Shasta" required="false" />
    <PDF source="sam" bookmarkTitle="Sam" required="false" />
    <PDF source="bobby" bookmarkTitle="Bobby" required="false" />
    <PDF source="billy" bookmarkTitle="Billy" required="false" />
    <PDF source="joaquin" bookmarkTitle="Joaquin" required="false" />
    <PDF source="reese" bookmarkTitle="Reese" required="false">
      <Watermark rotation="45" opacity="60%">
        <StyledText color="red" font-size="72pt">Adopted!</StyledText>
      </Watermark>
    </PDF>
    <PDF source="summary" bookmarkTitle="Summary" required="false" />
  </PDF>
</DDX>

Include this expression in the DDX document for your assembly workflow.