Assembler Service and DDX Reference

Adobe LiveCycle ES4

May 2016

Legal Notices

Contents

1 About This Help

The Assembler service 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 Assembler service 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 Assembler service 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.

What's new

Here are the new features that the Assembler service introduces. These features are reflected in new DDX elements:

- **Resolve Assets:** The Assembler service lets embed all referenced images in the source XDP files. You can specify how the image references are resolved—resolve all relative or absolute references, all references, or none.
- **Support for PDF/A-2b standard:** LiveCycle adds the support for PDF/A-2b standard for archiving. You can specify the PDF/A compliance as either PDF/A-1b, PDF/A-2b and PDF/A-3b.

Deprecated

Document Services deprecates the support for PDF/A-1a standard.

Additional information

See the following documentation to learn more about Document Services.

For information about	See
Document Services modules	*LiveCycle ES4 Overview*
The features available with each service. This document also introduces Assembler Installation and Verification Sample (Assembler IVS), which lets you test DDX documents.	*LiveCycle ES4 Service Reference*

For information about	See
Document Builder, which is a Workbench perspective where you can create and test assembly descriptors (DDX documents) without working directly in XML.	*Using Document Builder*
Programmatically invoking Document Services services. For example, to learn how to programmatically invoke the Assembler service.	*Programming with LiveCycle*
Last-minute changes to the product	*LiveCycle ES4 Release Notes*

Part I: DDX User Document

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

2 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 Assembler service determine the best way to produce the desired result.

DDX documents are templates for the documents that the Assembler service 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 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 Assembler service.

[dep root]`\sdk\schemas\PDFAssembler`

The Assembler service runs a complete validation on DDX documents that you submit and reports violations that it finds. For information about programmatically validating a DDX document, see *Validating DDX Documents* in *Programming with LiveCycle.*

DDX building blocks

The primary function of the Assembler service is to assemble multiple PDF or XDP documents into a single document. The Assembler service 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 Assembler service 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 139.

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*

```
<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 Assembler service 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 17 for details.
- A *source* element typically has a `source` attribute and represents initial content. See "Source elements" on page 19 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 20 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. See the "Programmatically Assembling PDF Documents" section in *Programming with LiveCycle*.

Note: Beginning with LiveCycle ES 8.2, 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 24.)

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 23.)

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 25.)

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 62.)

The example from above is repeated here. This example has a single `PDF` result element, which means that the Assembler service creates a stream named doc1.pdf. This stream contains a PDF document. The Assembler service returns the stream to the client.

```
<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*

```
<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 19)
- Filter elements (see "Filter elements" on page 20)
- 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 23). 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 81).

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 32.)

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

- Content that has been provided to the Assembler service (see "Input and output" on page 23).
- 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.

```
<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*

```
<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 165.)

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*

```
<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*

```
<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 65 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*

```
<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*

```
<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*

```
<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 94.)
- `PasswordEncryptionProfile` (See "Specifying passwords" on page 131.)
- `PasswordAccessProfile` (See "Accessing a password-protected document" on page 133.)
- `StyleProfile` (See "Using style profiles" on page 116.)
- `PDFAProfile`

Grouping PDF sources

As described in "Scope of elements that affect PDF or XDP properties" on page 25, 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*

```
<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*

```
<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*

```
<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>
      <PDF source=doc4"/>
```

```
    </PDFGroup>
    <PDF source="doc5"/>
</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 99.)

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 Assembler service 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 32.

For details about specifying the source documents programmatically, see the Assembler service quick starts in *Programming with LiveCycle*.

After the Assembler service 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 62.)

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*

```
<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 Assembler service 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 30.)

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 31.) The following example would concatenate two copies of doc2 and return them as doc1.

Example: *Specifying a source document twice*

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

Using External Data URLs for source and result values

As of LiveCycle ES 8.2, the Assembler service 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 Assembler service 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 Assembler service 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.

See also

"Lists obtained from input maps or External Data URLs" on page 28

"External Data URL" on page 152.

Using External Data URLs for string values

As of LiveCycle ES 8.2, the Assembler service 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 152.)

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*

```
<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*

```
<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*

```
<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 249.)

For more information on how DDX elements affect the contents or properties of PDF pages, see "Setting Page Properties" on page 96, "Adding and Manipulating Page Content" on page 103, and "Specifying Page Labels" on page 123.

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.

```
<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"
```

```
            insertionPoint="myOtherInsertionPoint"
            source="myOtherFragmentSource">
        </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 Assembler service 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.

```
<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.

See also

"Specifying multiple input streams" on page 32

Lists obtained from input maps or External Data URLs

Lists are often used when the Assembler service 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 Assembler service is invoked or using the Assembler service 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 Assembler service invokes another LiveCycle service to convert the file to PDF. In particular, the Assembler service 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 37.)
- 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 303.)

3 Assembling PDF Documents

The Assembler service can assemble PDF and XDP documents from other documents.

Note: If a PDF file contains invalid XMP metadata, the Assembler service 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 Assembler service can assemble documents from one or more source documents. It also supports several options for specifying source documents.

Note: Beginning with LiveCycle ES 8.2, the non-PDF documents can used as PDF source documents. Assembler service automatically converts such documents to PDF documents. The Assembler service performs the conversion by using the Generate PDF service. The `PDFGenerationSettings` element specifies conversion parameters. (See "PDFGenerationSettings" on page 303.)

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 15) 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 Assembler service 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*

```
<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*

```
<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 pagES4-4 from doc1.pdf, deleting page 7 from doc2.pdf, and then concatenating the documents.

Example: *Assembling pages from documents*

```
<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 32.)

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 66).
- `includeInTOC` controls whether the source document appears in the table of contents, if present (see "Adding a table of contents" on page 86).
- `access` specifies a password for opening the source document if it is encrypted (see "Accessing a password-protected document" on page 133).
- 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 32). 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. (See the "Programmatically Assembling PDF Documents" section in the Programming with LiveCycle.)

A source element can have a `sourceMatch` attribute instead of a `source` attribute. (If neither is present, the DDX is invalid.) The Assembler service 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 Assembler service 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 Assembler services 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 Assembler service 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 31). 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 Assembler service (by default) imports the contents of the folder.

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

```
        <StyledText><p>_SourceTitle</p></StyledText>
      </Center></Header>
   </PDF>
</PDF>
```

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 expressions 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*

```
<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:

Logical name	Data stream
intro	*stream_1*
chap1	*stream_2*
chap2	*stream_3*
chap3	*stream_4*
chap4	*stream_5*
appendix1	*stream_6*
appendix2	*stream_7*
index	*stream_8*

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.

Logical name	Data streams
intro	*stream_1*
Chapters	*stream_2, stream_3, stream_4, stream_5*
Appendices	*stream_6, stream_7*
index	*stream_8*

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 34 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 115); 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 Assembler service 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 30). 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 Assembler service.

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 Assembler service 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.

4 Modifying Acrobat and XML Forms

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 Adobe LiveCycle 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 Assembler service 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 38 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 44 and "XFAConversionSettings" on page 305.)

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

The Assembler service 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*

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

```
      <NoForms/>
   </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*

```
<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*

```
<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 40

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 Assembler service retains all form fields by default. In general, there are no restrictions on the operations that the Assembler service can perform on documents containing Acrobat forms. However, if a document contains signature fields, certain operations invalidate the signatures. (See "Digital signatures" on page 133.)

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 43.)

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)

5 Creating and Modifying Acrobat and XML (XFA) Forms

You can use the Assembler service 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:

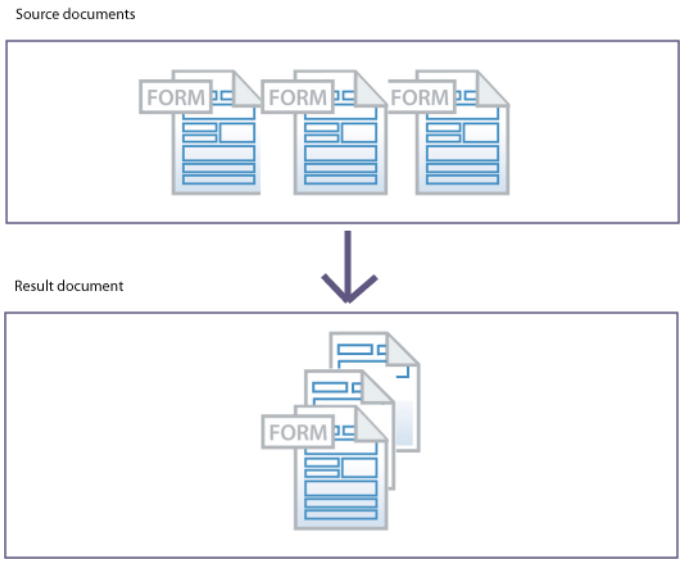
- "Assemble a simple XDP document" on page 40
- "Dynamically insert forms or form fragments into an XFA form" on page 41
- "Package an XDP document as PDF" on page 43
- "Package a PDF document as XDP" on page 46

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.

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

```
      <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

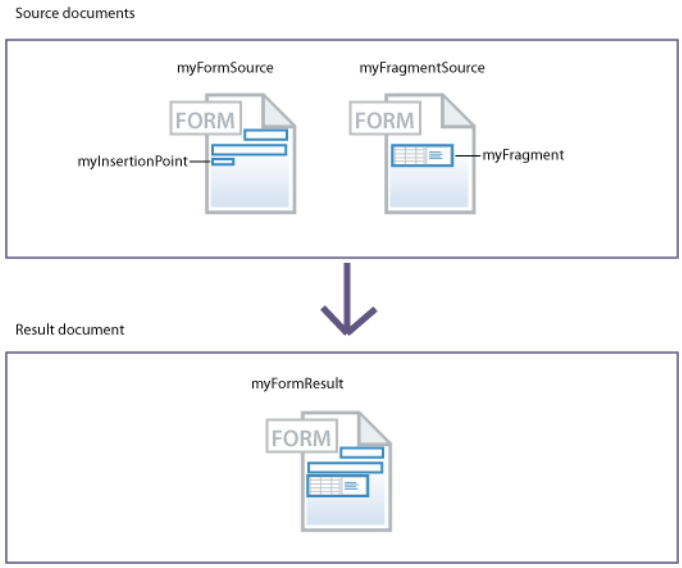
The Assembler service lets you create an XFA form by inserting forms or form fragments into another XFA form.

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 Assembler service 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/">
```

```
   <XDP result="myFormResult">
      <XDP source="myFormSource">
         <XDPContent fragment="myFragment"
         insertionPoint="myInsertionPoint" source="myFragmentSource"/>
      </XDP>
   </XDP>
</DDX>
```

Resolve references

XDP documents can contain images referenced either through absolute or relative references. Assembler service, by default, retains the references to the images in the resultant XDP document.

You can specify how the Assembler service 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:

Value	Description
`none`	Does not resolve any references. All references are retained.
`all`	Embeds all referenced images in the source XDP documents.
`relative`	Embeds all the images referenced through relative references in the source XDP document.
`absolute`	Embeds all the images referenced through absolute references in the source XDP document.

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:

```
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
   <XDP 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.

```
<DDX xmlns="http://ns.adobe.com/DDX/1.0/">
   <XDP result="result.xdp">
      <XDP source="input1.xdp" resolveAssets="all"/>
      <XDP source="input2.xdp" resolveAssets="all"/>
      <XDP source="input3.xdp" resolveAssets="all"/>
```

```
   </XDP>
</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.

```
<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:

```
<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 Assembler service 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*

```
<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*

```
<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)*

```
<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)*

```
<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 Assembler service cannot assemble an XDP (generic) result with other documents. If the following conditions occur, then Assembler 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 Assembler service uses the Output service to flatten dynamic XFA forms. The Assembler service flattens static XFA-based forms and Acrobat forms by itself. (See "Assemble XFA-based forms with other documents" on page 44.)

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>
</PDF>
<PDF result="final_result.pdf">
```

```
   <PDF source="intermediate_result.pdf" baseDocument="true"/>
   <PDF source="nonXFAForm.pdf"/>
   <PDF source="other_nonXFAForm.pdf"/>
</PDF>
```

PDF documents from Acrobat forms

The Assembler service can successfully assemble PDF documents from multiple PDF documents containing Acrobat forms. By default, Assembler 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*

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

Package a PDF document as XDP

The Assembler service 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*

```
<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.

6 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

LiveCycle Doc Team: Using Assembler 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:

```
<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.

```
<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 51.)

Note: Document Services navigates down into the subfolders to include files in subfolders. Earlier versions of LiveCycle did not 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.

```
<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 Assembler service 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 52.)
- 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.

```
<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.

```
<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.

```
<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:

```
<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 Assembler service 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*

```
<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*

```
<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.-->
```

```
            <FieldData name="X">72</FieldData>
            <FieldData name="Y">72</FieldData>
         </PackageFiles>
         <PackageFiles source="process:///process_data/@doc2" required="false">
            <FieldData name="X">72</FieldData>
            <FieldData name="Y">144</FieldData>
         </PackageFiles>
      </Folder>
      <Folder name="Folder of Files">
         <Description>Assembler Dev Test Files</Description>
            <FieldData name="X">72</FieldData>
            <FieldData name="Y">216</FieldData>
            <PackageFiles source=
               "contentspace:///Company Home/User Homes/Assembler Dev Tests/"
               includeSubFolders="true"/>
      </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" prefix="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" prefix="Re:">Lunch on Friday!</FieldData>
      </PackageFiles >
   </PDF>
   <StyleProfile name="letterCollection">
      <Package>
         <Schema>
```

```
            <Field name="from" type="Text" />
            <Field name="to" type="Text" />
            <Field name="date" type="Date" />
            <Field name="subject" type="Text" />
            <Field name="size" type="Size" />
          </Schema >
          <DisplayOrder>
            <Field name="date"/>
            <Field name="subject"/>
            <Field name="from"/>
            <Field name="to"/>
            <Field name="size"/>
          </DisplayOrder>
          <SortOrder>
            <Field name="date" ascending="true"/>
          </SortOrder>
        </Package>
    </StyleProfile>
    <InitialViewProfile
      name="letterCollection"
      packageInitialDocument="FirstSortedDocument"
      packageUIPane="Left"
    />
</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 Assembler service 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">
   <PDF source="single1"/>
   <PackageFiles>
```

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

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*

```
<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*

```
<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*

```
<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 Assembler service 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*

```
<PDF result="resultPkg">
   <TargetLocale locale="de_DE"/>
   <PDF source="_AdobeCoverSheet"/>
   <PackageFiles>
      <PDF source="pkg"/>
```

```
   </PackageFiles>
</PDF>
```

Note: If the locale is not specified in the DDX, then the locale specified via User Management is used. If those locale specifications are absent, `"en_US"` is used.

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*

```
<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*

```
<PDF result="newPackage">
   <Package>
      <PDF source="pkg1.pdf"/>
   </Package>
   <PDF source="pkg2.pdf"/>
   <PDF source="pkg3.pdf"/>
```

```
</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*

```
<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*

```
<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 Assembler service 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)***

```
<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***

```
<PDF result="newPortfolio">
   <PDF source="portfolio.pdf"/>
   <PackageFiles>
      <PDF source="pkg2.pdf"/>
      <PDF source="single4"/>
   </PackageFiles>
```

```
   <PackageFiles nameKeys="doc1.pdf, doc2.pdf">
      <PDF source="pkg3.pdf"/>
      <PDF source="single5"/>
   </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*

```
<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 222.)

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*

```
<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 350.)

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 Assembler service 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 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 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 assemblies 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*

```
<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*

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

7 Disassembling PDF Documents

You can use the Assembler service 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 Assembler service 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*

```
<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 Assembler service 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 pagES4-12
- 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 30). 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 133.)

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 131.)
- `save` can be set to `Full` (the default) or `FastWebView` (see "Saving PDF documents" on page 35). Incremental saving is not applicable to disassembled documents.

8 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 86.

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*

```
<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*

```
<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*

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

```
<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*

```
<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 Assembler service 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 book marks from a PDF document

To export bookmarks, use a `Bookmarks` result element. (See "Bookmarks" on page 165.)

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

Example: *Exporting bookmarks*

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

```
   <PDF source="doc3"/>
</Bookmarks>
```

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*

```
<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*

```
<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*

```
<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*

```
<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">
```

```
   <PDF source="PartI" bookmarkTitle="Part I"/>
   <PDF source="PartII" bookmarkTitle="Part II"/>
   <PDF source="PartIII" bookmarkTitle="Part III"/>
</PDF>
```

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 accordinating to the destination page number.

Example: *Inserting one document into another with sorting*

```
<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*

```
<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

Bookmark	Target page
Ba1	1
-- Ba1-1	2
-- Ba1-2	4
Ba2	5
-- Ba2-1	7
-- Ba2-2	10

Bookmark structure of Doc2

Bookmark	Target page
Bb1	1
-- Bb1-1	2
-- Bb1-2	3
Bb2	4

Bookmark structure of the resultant document with sorting

When the DDX at "Inserting one document into another with sorting" on page 68 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.

Bookmark	Target page
Ba1	1
-- Ba1-1	2
Bb1	3
-- Bb1-1	4
-- Bb1-2	5
Bb2	6
Ba1	No target page
-- Ba1-2	8

Bookmark	Target page
Ba2	9
-- Ba2-1	11
-- Ba2-2	14

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.

Bookmark	Target page
Ba1	1
-- Ba1-1	2
-- Ba1-2	8
Ba2	9
-- Ba2-1	11
-- Ba2-2	14
Bb1	3
-- Bb1-1	4
-- Bb1-2	5
Bb2	6

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 Assembler service 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*

```
<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*

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

9 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 Assembler service 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 81.

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*

```
<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*

```
<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*

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

```
   <PDF source="doc3">
      <NoComments/>
   </PDF>
   <PDF source="doc4"/>
</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*

```
<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*

```
<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.

Note: In addition to comments, FDF can also be used to store information about links and form fields. In the Assembler service, however, you can use FDF only for comments. Links must be imported and

exported using XFDF, and the Assembler service cannot export form fields. For more information, see "Working with links" on page 78).

Some comment types are of special interest:

- The Assembler service 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 81.
- 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 75).

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

Example: *Exporting comments as XFDF*

```
<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*

```
<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*

```
<PDF result="doc2.pdf">
   <PDF source="doc1.pdf"/>
   <Comments source="comments.xfdf"/>
</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>
<PDF result="doc2.pdf">
   <PDF source="doc1_rev2.pdf"/>
   <Comments source="doc1comments.xfdf"/>
</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*

```
<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*

```
<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="FDF" 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*

```
<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 79 for details.

Example: *Exporting links from multiple sources*

```
<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*

```
<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*

```
<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 20.

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*

```
<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*

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

10 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 208.)

Example: *Excluding attachments from a source document*

```
<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*

```
<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 e 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.

Example: *Attaching a file to the document*

```
<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*

```
<PDF result="doc2.pdf">
   <PDF source="doc1.pdf">
   <FileAttachments source="data.pdf" ... additional attributes />
</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 183.)

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*

```
<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 pagES4-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 Assembler service 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 183.)
- 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 187.)

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*

```
<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. In LiveCycle ES 8.0 and later, 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 187.)

11 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 64.)

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*

```
<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*

```
<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*

```
<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 116.) 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*

```
<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*

```
<TableOfContents maxBookmarkLevel="3">
   <Header styleReference="alpha"/>
   <Footer styleReference="beta"/>
   <TableOfContentsPagePattern pages="1">
      <Header styleReference="gamma"/>
   <TableOfContentsPagePattern/>
   <TableOfContentsPagePattern pages="2-last">
```

```
    <Footer styleReference="delta"/>
  <TableOfContentsPagePattern/>
</TableOfContents>
```

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 Assembler service (see "Specifying styled text" on page 112).

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

Example: *Formatting table of contents entries*

```
<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*

```
<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*

```
<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*

```
<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 25.

12 Setting Other Document Properties

Some properties of PDF documents apply to the document as a whole rather than to individual pages. The Assembler service 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 30).

Document properties also include security settings, which are discussed in "Working with Secured Documents" on page 131.

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*

```
<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*

```
<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*

```
<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*

```
<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*

```
<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*

```
<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*

```
<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 196.)

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*

```
<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 210.)

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*

```
<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*

```
<PDF result="resultDoc">
   <PDF source="pdf1"/>
   <PDFGroup>
      <NoJavaScripts/>
      <PDF source="pdf2"/>
      <PDF source="pdf3"/>
      <PDF source="pdf4"/>
   </PDFGroup>
</PDF>
```

13 Setting Page Properties

The Assembler service 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 25.)

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 26 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 103.)

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 135.)

There is an interaction between the `PageSize` and `PageRotation` elements. See "Rotation and orientation" on page 98 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 27 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*

```
<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.

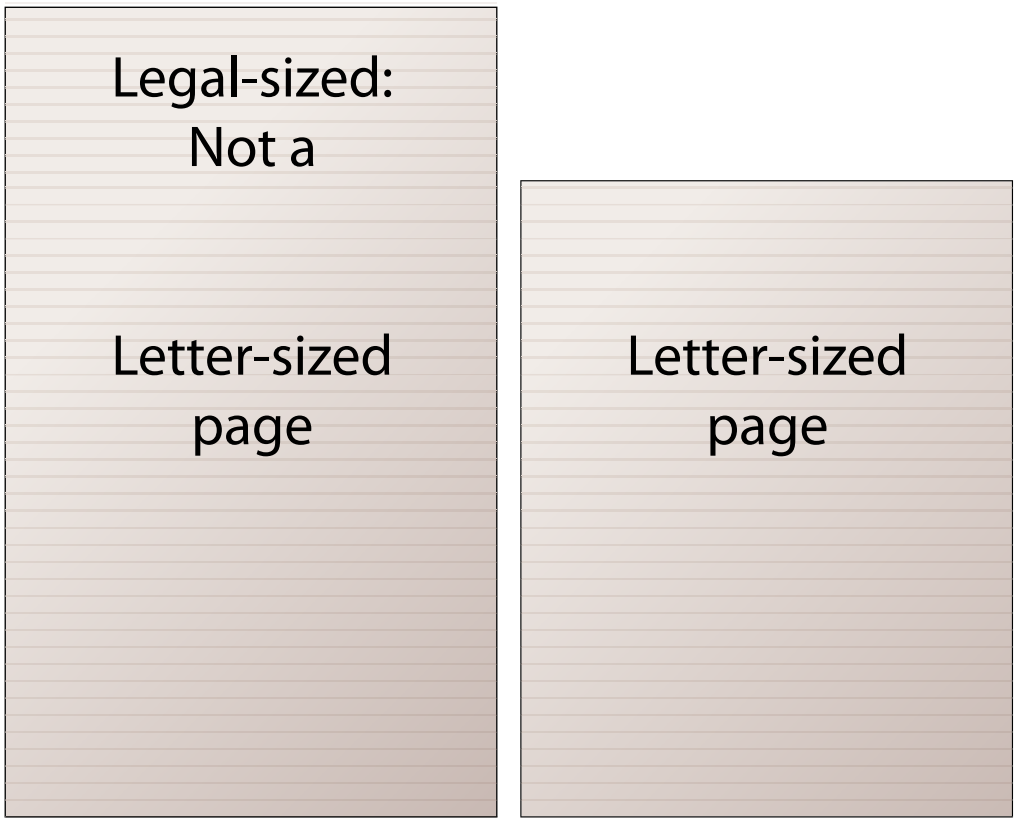
For example, if you specify a page size of 8.5 x 14 inches, any source pages that were 8.5 x 11 inches grows vertically. The default value for the `verticalAnchor` attribute is `Middle`. With this setting, the old page size is centered vertically compared to the new size and white space appears equally 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*

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

```
</PDF>
```



Rotation and orientation

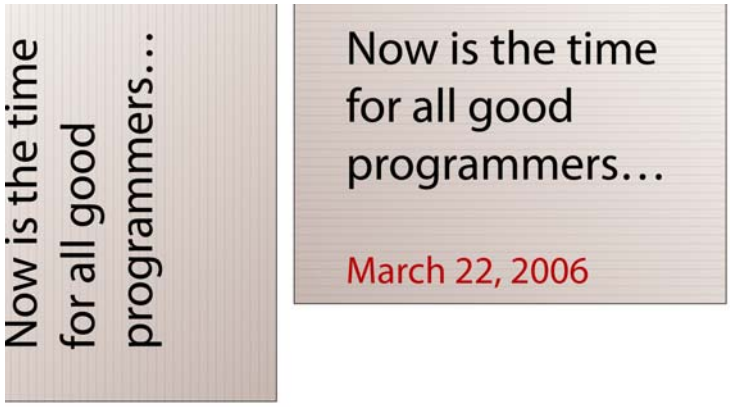
You can use the `PageRotation` element to change the rotation angle of a page. The `rotate90` attribute specifies the rotation angle in increments of 90°, where a positive value is clockwise rotation and a negative value is counterclockwise. This rotation is absolute, not relative. That is, if the `rotate90` 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 Assembler service always assumes that the pages are set the way you want to view them; whether a rotation value is specified or not. The Assembler service 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*

```
<PDF result="newdoc">
   <PDF source="doc1">
      <PageRotation rotate90="90"/>
      <Footer styleReference="myStyle"/>
   </PDF>
</PDF>
```

These figures illustrate how the source page looks before and after rotation and application of the footer.



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 98.)

- `PageSize`: Specifies the new page size. "Changing page size" on page 97 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 121). 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 101.)
- 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 103.)

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 96.)

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 27 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 26.)

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*

```
<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 103.)

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 103.)

Example: *Setting page margins*

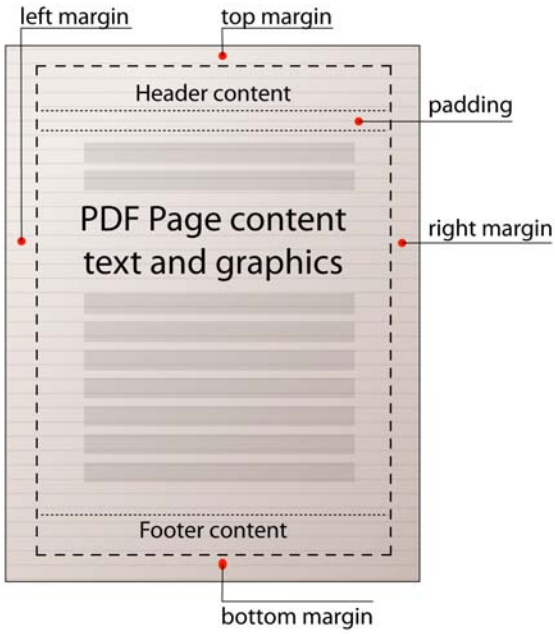
```
<PDF result="doc1">
   <PDF source="doc2">
      <PageMargins left="1in" top="0.75in" right="0.75in" bottom="1in"/>
      <Header padding="0.5in">
         <Center>
            <StyledText><p>Header content</p></StyledText>
         </Center>
      </Header>
      <Footer>
         <Center>
            <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 26.)

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 Assembler service, 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:

```
<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 101.) 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 116.)

In LiveCycle 7.x, when you specified a header or footer for a page, any header or footer that was previously added to the page was removed. This behavior applied even if it contained no text or graphics. In LiveCycle ES 8.0 and later, you can retain pre-existing headers and footers by setting the `replaceExisting` attribute to `false`.

Note: The Assembler service 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 Assembler does not remove page content that contains Bates numbers.

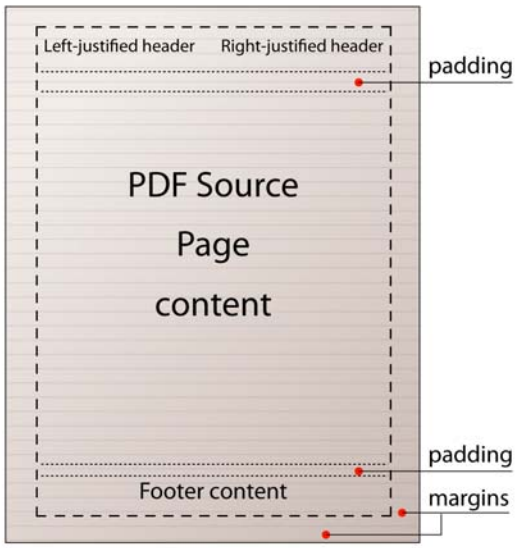
You can specify a different `Header` or `Footer` for even and odd pages within a given scope (see "Odd and even pages" on page 26).

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 112)
- 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*

```
<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*

```
<PDF result="doc2.pdf">
```

```
   <PDF source="doc1.pdf">
      <Header styleReference="myProfile"/>
      <Footer styleReference="myProfile"/>
   </PDF>
</PDF>
<StyleProfile name="myProfile">
   <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>
</StyleProfile>
```

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 116 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*

```
<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 26.

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 Assembler service, are removed.

Example: *Removing headers*

```
<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*

```
<PDF result="assembledDoc.pdf">
   <Header styleReference="general"/>
   <PDF source="doc1.pdf">
   <PDF source="doc2.pdf">
      <NoHeaders/>
   </PDF>
   <PDF source="doc3.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:

- In LiveCycle 7.x, you could add at most one watermark and one background to a destination page. When you added a watermark or background, any previous one was removed. In LiveCycle ES4 8.0 and later, 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 26).

- 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 26.)

Note: The Assembler service 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 25.) 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*

```
<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*

```
<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">
      <NoWatermarks/>
   </PDF>
</PDF>
```

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 225.)
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*

```
<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 source="doc1"/>
</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 Assembler service 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*

```
<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 Assembler service 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 Assembler service 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 Assembler service 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 Assembler service 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 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 stylized 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*

```
<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 115) 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 ` `
- `Space` element for the `DatePattern` element and rich text elements.

The ` ` 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 Assembler service 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 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">
      <PDF source="input2.pdf"/>
      <Watermark verticalOffset="4.5in">
         <StyledText>
            <p><BatesNumber prefix="Ref ID = " start="1"/>.</p>
         </StyledText>
      </Watermark>
   </PDF>
</DDX>
```

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 as 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 Assembler service 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.

  ```
  <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.

```
<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*

```
<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 118 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*

```
<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 26).

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 source="doc1.pdf">
```

```
      <Header styleReference="body"/>
      <Footer styleReference="body"/>
   </PDF>
</PDF>

<StyleProfile name="cover">
   <DatePattern><DayNumber/> / <ShortMonthName/> / <Year/></DatePattern>
   <Header>
      <Left>
         <StyledText>
         <p><_DateTime styleReference="cover"/></p>
         </StyledText>
      </Left>
      <Right><StyledText><p>Draft</p></StyledText></Right>
   </Header>
   <NoFooters/>
</StyleProfile>

<StyleProfile name="body">
   <Header>
      <Center> <StyledText><p>Confidential</p></StyledText></Center>
      <Left>
         <StyledText><p>
            <_DateTime styleReference="cover"/>
         </p></StyledText>
      </Left>
      <Right><StyledText><p>Draft</p></StyledText></Right>
   </Header>
   <Footer alternation="EvenPages">
      <Left>
         <StyledText>
         <p>Page <_PageNumber/> of <_LastPageNumber/></p>
         </StyledText>
      </Left>
   </Footer>
   <Footer alternation="OddPages">
      <Right>
         <StyledText>
         <p>Page <_PageNumber/> of <_LastPageNumber/></p>
         </StyledText>
      </Right>
   </Footer>
</StyleProfile>
```

For an example of using `StyleProfiles` for Package definitions, see "Creating a PDF package" on page 53.

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 ` `
- `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*

```
<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*

```
<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/> <ShortMonthName/><Space/><Year/> at
<Hour24/>:<Minute00/>
   </DatePattern>
</StyleProfile>
```

This DDX would result in a page header with this appearance:

```
01 Jan 2001 at 14:03          Confidential          page 1
```

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 27).

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:

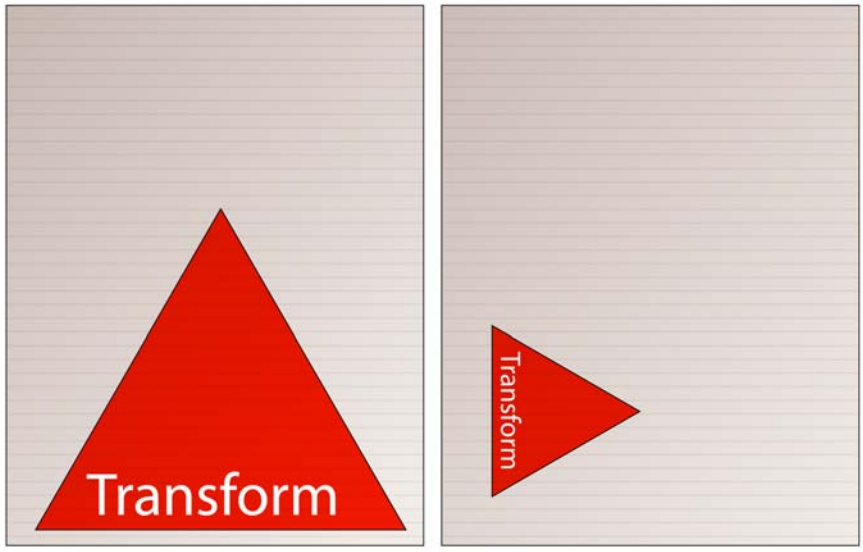
- Shrinks the page content of doc1.pdf to one-quarter size
- Moves the origin from the page to 50 points horizontal and 80 points vertical
- Rotates the content by 90 degrees

The content of doc2.pdf is unchanged.

Example: *Transforming page content*

```
<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.



15 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 Assembler service 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 112 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 129).

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 25.)

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*

```
<PDF result="doc4">
   <PageLabel format="Decimal"/>
   <PDF source="Preface">
```

```
      <PageLabel format="LowerRoman"/>
   </PDF>
   <PDF source="Chapter1"/>
   <PDF source="Chapter2"/>
</PDF>
```

Page labeling is optional. However, when assembling documents using the Assembler service, 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 Assembler service 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 115).

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*

```
<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.

Ordinal page number	Page label
1	Chapter 1 - 1
2	Chapter 1 - 2
3	Chapter 2 - 1
4	Chapter 2 - 2
5	Appendix A - 1
6	Appendix A - 2

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*

```
<PDF result="doc4">
   <PageLabel prefix="page "/>
```

```
   <!-- Each source provides a 2-page document. -->
   <PDF source="doc1"/>
   <PDF source="doc2"/>
   <PDF source="doc3"/>
</PDF>
```

The resulting page numbers and labels are shown in the following table.

Ordinal page number	Page label
1	page 1
2	page 2
3	page 3
4	page 4
5	page 5
6	page 6

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*

```
<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.

Document	Number of pages	PageLabel attributes (if any)
Title	1	`format="None" prefix="Title page"`
FirstHalf	2	None
AdSection	2	`format="Decimal" prefix="Ad - "`
SecondHalf	2	None

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*

```
<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>
```

Ordinal page number	Page label
1	Title page
2	1
3	2
4	Ad - 1
5	Ad - 2
6	3
7	4

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*

```
<PDF result="doc1">
   <PDF source="Title">
      <PageLabel mode="Preserve"/>
   </PDF>
   <PDF source="FirstHalf"/>
   <PDF source="AdSection">
      <PageLabel mode="Preserve"/>
   </PDF>
   <PDF source="SecondHalf"/>
</PDF>
```

Ordinal page number	Page label
1	Title page
2	2
3	3
4	Ad - 1
5	Ad - 2
6	6
7	7

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*

```
<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.

Ordinal page number	Page label
1	Title Page
2	1
3	2
4	Ad - 1
5	Ad - 2
6	5
7	6

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*

```
<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>
```

16 Working with Secured Documents

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 Assembler service, 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 62.

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*

```
<PDF result="doc2" encryption="userProtect">
   <PDF source="doc"/>
</PDF>

<PasswordEncryptionProfile name="userProtect">
```

```
   <OpenPassword>opensesame</OpenPassword>
</PasswordEncryptionProfile>
```

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 35).

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 253.)

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*

```
<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*

```
<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 Assembler service 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*

```
<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 Assembler service can perform any operation on the document. If the password is the open password, the Assembler service 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 Assembler service 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 Assembler service. 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 35.)
- Signatures become invalid when disassembling documents using the `PDFsFromBookmarks` element. (See "Disassembling PDF Documents" on page 62.)
- Signatures become invalid when flattening form fields using the `NoForms` element. (See "Flattening forms" on page 37.)

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 37). 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.

17 Querying Documents

The Assembler service 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 179.)

In this example, information about doc1 is returned as an XML stream.

Example: *Getting document information*

```
<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*

```
<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*

```
<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:

```
<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
      </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*

```
<DDX>
   <About result="AboutDDX.xml"/>
</DDX>
```

The result document looks like this:

```
<About xmlns="http://ns.adobe.com/DDX/AboutDDX/1.0/">
```

```
    <Processor>Adobe® LiveCycle™ 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.

18 DDX Concepts

Each job submitted to the Assembler service 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 Assembler service 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).

```
<Comments
   filter="Exclude" or "Include"
   afterDate=unspecified or "YYYYMMDD"
   …
```

```
</Comments>
```

The syntax in the DDX Reference uses the following conventions.

Expression	Meaning
"**Exclude**"	Indicates the default value
"Include"	An alternative value for the attribute
"*YYYYMMDD*"	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: ? *xs:integer* ? *xs:string* ? *color* ? *File description* ? *page range*
unspecified	Indicates that the attribute is omitted.
or	Indicates a descriptive comment.

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).

```
<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.

Expression	Meaning
<PDF source>	The source designation is a comment that indicates the expected child element contains a source attribute, as described in "Element relationships and roles" on page 139.
[1..n]	The minimum and maximum allowable occurrences of the element within the parent element. The following list describes occurrence designations: [1] The child element must be specified only once. [1..n] The child element must be specified one or more times. [0..2] In this case 2 is allowed for alternating pages. The child element can be omitted, can appear once if used for all pages, or can appear twice if each appearance applies to different alternating pages. That is, one child element applies to the odd pages and the other applies to the even pages.
or	Indicates that either one of the child elements are allowed (not both). The occurrence designator indicates the number of times either child element appears within the parent element.
and/or	Indicates that one or both child elements are allowed, where the occurrence designator indicates the number of times either child element appears within the parent element.

Text content

As shown in the following declaration, elements can include text values.

```
<OpenPassword>
   password
</OpenPassword>
```

As with attributes that take text values, the syntax used for text content uses the following convention.

Expression	Meaning
password	A text value is provided.

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 142
- "Document components" on page 143
- "Document disassembly" on page 145
- "Document properties" on page 145
- "Page labels" on page 146
- "Page properties" on page 146

- "Page content" on page 147
- "Profile" on page 147
- "Query" on page 148

Document assembly

Document assembly elements are used for "PDF assembly" on page 142, "PDF package or portfolio assembly" on page 142, and "XDP assembly" on page 143.

PDF assembly

The following elements are in this category.

Element name	Description
BlankPage	Specifies the addition of an initially blank page to the document.
PDF result	PDF document that is returned to the client.
PDF source	One or more PDF documents that are used to construct the parent result PDF document.
PDFGroup	Specifies a grouping of source documents to which page properties, page content, and document components can be applied.
TableOfContents	Specifies a table of contents in the result document.
TableOfContentsEntryPattern	Specifies the style to apply to table of contents entries.
TableOfContentsPagePattern	Defines the style for table of contents pages.

PDF package or portfolio assembly

The following elements are in this category.

Element name	Description
ColorScheme	Color settings used in the PDF Portfolio.
DisplayOrder	Determines the order in which the viewer displays the fields of a PDF package.
Header (portfolio navigation pane)	Resource that supplies a header used in a PDF Portfolio.
Field	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.
FieldData	References metadata for a package file in a PDF package or portfolio.

Element name	Description
Package	Identifies the parent element, PDF result or PDF source, as being a PDF package. A <"PackageFiles" on page 217> element can either add new package files or select package files for modification.
NoPackage	Assembles a PDF package or portfolio into a single PDF that omits the PDF package navigational features.
Navigator	Specifies a navigator to use for a PDF Portfolio.
PackageFiles	Adds new package files to the PDF package or portfolio being assembled.
NoPackageFiles	The package files in the package are not included, but the package specification remains and the parent element remains a PDF package.
Portfolio	Extends PDF packages with additional navigational features introduced with Adobe Acrobat 8.2.
NoPortfolio	Assembles a PDF Portfolio into a single PDF that omits the PDF Portfolio navigational features.
Schema	Defines the characteristics of the metadata (Field) attributes associated with each package file.
SortOrder	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.
Schema	Defines the metadata (Field) attributes that can be associated with each package file
String	Adds entries to the String name tree in a navigator dictionary.
WelcomePage	Specifies the Welcome Page used in a PDF Portfolio.

XDP assembly

The following elements are in this category.

Element name	Description
XDP	Specifies an XDP document assembled from other XDP documents. Also packages an XDP document as a PDF document.
XDPContent	Specifies XDP content to insert into the XDP source or result being specified.

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.

Element name	Description
`Bookmarks` result	Exports bookmarks from the child elements to an XML document.
`Bookmarks` source	Specifies that bookmarks contained in an XML document are imported into the parent element.
`Bookmarks` filter	Specifies bookmarks from child source elements that can be imported into the parent element but are not returned as XML.
`LinkAlias`	Provides an alternative name for the parent document, for use as a link destination.
`Links` result	Specifies that links be exported from the child elements as an XML Forms Data Format (XFDF) document.
`Links` source	Imports links into the parent element. The links are provided in the source, which must be an XFDF document.
`Links` filter	Specifies that links from child source elements be imported into the parent element.
`NoBookmarks`	Specifies removal of bookmarks from the parent element.
`NoLinks`	Specifies removal of links from the parent element.
`NoThumbnails`	Specifies removal of embedded thumbnails from the parent element.

Comments

The elements in this subcategory represent comments, which can be imported or exported from PDF documents. The following elements fall into this subcategory.

Element name	Description
`Comments` result	Exports the comments from the child elements as an XFDF or Forms Data Format (FDF) document.
`Comments` source	Imports comments into the parent element. The comments are contained in an XFDF or FDF document.
`Comments` filter	Specifies comments from child source elements that can be imported into the parent element but are not returned as FDF or XFDF.
`NoComments`	Specifies removal of annotations from the parent element.

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.

Element name	Description
AttachmentAppearance	Defines the appearance and placement of a page-level file attachment
Description	Provides a description of a file attachment
File	Specifies a file attached to a PDF document as a document-level or page-level file attachment
FileAttachments result	Requests information about file attachments and optionally returns attachments as separate data streams
FileAttachments source	Specifies a file to be attached to the parent document
NoFileAttachments	Specifies removal of all file attachments in the parent element

Document disassembly

The document disassembly element splits a single document into multiple PDF documents.

Element name	Description
PDFsFromBookmarks	Splits a single PDF document into multiple PDF documents, based on top-level bookmarks

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.

Element name	Description
Author	Provides a value for the author metadata in the result document.
"JavaScript" on page 199	A source element that specifies a document-level JavaScript to be added to the result PDF document.
"NoJavaScripts" on page 210	Specifies that the parent PDF result or source element contains no document-level JavaScript.
Keyword	Provides a single keyword for use as metadata.
Keywords	Provides metadata keywords for the result document.
MasterPassword	Specifies a password that is used to change permissions for the result document.
Metadata result	Exports the metadata from the child document as XMP.

Element name	Description
Metadata source	Imports the metadata contained in an XMP stream into the parent document.
NoForms	Flattens all form fields in the parent document.
NoXFA	Flattens XFA-based forms in the parent element.
OpenPassword	Specifies a password that is used to open the result document.
Password	Provides an access password for the Assembler service to use to open encrypted source documents.
Subject	Provides the value for the subject metadata in the result document.
Title	Specifies the value for the title metadata in the result document.

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 Assembler service, this string can also be added to page content such as footers. The following elements fall into this category.

Element name	Description
NoPageLabels	Specifies removal of page labels from the parent element
PageLabel	Specifies the format and content of page labels

Page properties

Page property elements specify how pages are viewed or printed. The following elements fall into this category.

Element name	Description
ArtBox	Defines the extent of the page's meaningful content as intended by the creator of the page
BleedBox	Defines the bounds to which the contents of the page are clipped when output in a production environment
PageMargins	Specifies margins for page content elements
PageRotation	Specifies the rotation angle of pages within the scope of the parent element
PageSize	Defines the page dimensions for purposes of display or print
TrimBox	Defines the intended dimensions of the finished page after printing and trimming

Page content

Page content elements specify new content that is added to pages in PDF documents. The following elements fall into this category.

Element name	Description
Background	Provides styled text or graphics for placement behind existing page content
Center	Specifies a centered header or footer
DatePattern	Provides style information used to construct a string representing the date and time
Footer	Provides styled text or graphics for footer content
graphic	Inserts a graphic as if it were a character in a line of text.
Header	Provides styled text or graphics for header content
Left	Specifies a left-justified header or footer
NoBackgrounds	Specifies removal of backgrounds from the parent element
NoFooters	Specifies removal of footers from the parent element
NoHeaders	Specifies removal of headers from the parent element
NoWatermarks	Specifies removal of watermarks from the parent element
"PageContent" on page 225	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.
PageOverlay	Provides content for placement over the current page content
PageUnderlay	Provides content for placement under the current page content
Right	Specifies a right-justified header or footer
StyledText	Provides a rich text expression
Transform	Specifies transformations applied to the page contents, including scaling, translation, and rotation
Watermark	Provides styled text or graphics for placement over the existing page content

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.

Element name	Description
InitialViewProfile	Specifies information about how a document is viewed when it is opened in a viewer application.
PDFAProfile	Specifies settings for conversion to PDF/A.
StyleProfile	Specifies a named profile that contains style information about page contents or table of contents elements.
PasswordEncryptionProfile	Specifies a named profile containing password security settings.
PasswordAccessProfile	Specifies a named profile containing an access password.

Query

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.

Element name	Description
About	Requests information about the Assembler service.
DocumentInformation	Requests information about a PDF document, such as the document's metadata settings and information about the pages in the document
DocumentText	Requests information about the words that appear in the specified source documents

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 Assembler service replaces these keys when it processes the DDX document.

The following table describes the Assembler service built-in keys.

Built-in key	Description
_PageLabel	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.
_LastPageLabel	Page label of the last page in the result document.
_PageNumber	Ordinal (1-based) page number of the current page in the result document.
_LastPageNumber	Total number of pages in the result document.

Built-in key	Description
`_Title`	Title from the metadata of the result document. If no title is defined, this key is set to an empty string.
`_SourceTitle`	Title from the metadata of the source document. If no title is defined, this key is set to an empty string.
`_Author`	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.
`_SourceAuthor`	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.
`_Subject`	Subject from the metadata of the result document. If no subject is defined, this key is set to an empty string.
`_SourceSubject`	Subject from the metadata of the source document. If no subject is defined, this key is set to an empty string.
`_Created`	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.
`_Modified`	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.
`_DateTime`	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.
`_BookmarkTitle`	Title of the current bookmark. (Used only with the `TableOfContentsEntryPattern` element.)
`_BookmarkPageCitation`	Page reference of the current bookmark. (Used only with the `TableOfContentsEntryPattern` element.)
`_AdobeCoverSheet`	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 150 for more details and information about localization.
`_Filename`	Filename stored with a package file. Its use is restricted to the `FieldData` element, as the value of the `name` attribute.

Built-in key	Description
`_Description`	Description stored with a package file. Its use is restricted to the `FieldData` element.
`_ModificationDate`	Modification date stored with a package file. Its use is restricted to the `FieldData` element.
`_CreationDate`	Filename stored with a package file. Its use is restricted to the `FieldData` element.

_AdobeCoverSheet

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`:

Language	TargetLocale
Bulgarian	`bg_BG`
Chinese (China)	`zh_CN`
Chinese (Taiwan)	`zh_TW`
Croatian	`hr_HR`
Czech	`cs_CZ`
Danish	`da_DK`
Dutch	`nl_NL`
English	`en_US`
Finnish	`fi_FI`
French	`fr_FR`
German	`de_DE`
Hungarian	`hu_HU`
Italian	`it_IT`

Language	TargetLocale
Japanese	`ja_JP`
Korean	`ko_KR`
Latvian	`lv_LV`
Lithuania	`lt_LT`
Norwegian	`no_NO`
Polish	`pl_PL`
Portuguese (Brazil)	`pt_BR`
Romanian	`ro_RO`
Russian	`ru_RU`
Slovak	`sk_SK`
Slovenian	`sl_SL`
Spanish	`es_ES`
Swedish	`sv_SE`
Turkish	`tr_TR`
Ukrainian	`uk_UA`

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.

String or string pattern	Description
`#rrggbb`	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.
`SVG color keyword name`	One of the colors named in the *Scalable Vector Graphics (SVG) Specification, version 1.1*(http://www.w3.org/TR/SVG/). The following examples are SVG color keyword names: ? `black` represents `#000000` ? `aqua` represents `#00FFFF` ? `blue` represents `#0000FF` ? `white` represents `#FFFFFF`

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 24.) LiveCycle ES4 supports the following URL schemes:

- "Application URL" on page 153
- "Contentspace URL" on page 154
- "Process URL" on page 154
- "Repository URL (deprecated)" on page 154
- "File URL" on page 155
- "HTTP/HTTPS URL" on page 155
- "FTP URL" on page 155
- "Inputmap URL" on page 156

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

Element	Attributes
<Author>	value
<BatesNumber>	prefix, suffix
<File>	filename
<Folder>	name
<Keyword>	value
<Resource> (as child of <Navigator> and as child of <Portfolio>)	name
<PackageFiles>	sourceMatch
<PageLabel>	prefix
<PDF>	bookmarkTitle, sourceMatch
<PDFGenerationSettings>	conversionSettings, fileTypeSettings
<ReaderRights>	credentialAlias
<String> (String element as a child of the StyledText element)	url
<String> (String element as child of the Navigator element)	name
<TargetLocale>	locale
<Subject>	value
<TableOfContents>	bookmarkTitle
<Title>	value
<XDP>	fragment, sourceMatch, insertionPoint, retainInsertionPoints, removeInsertionPoints
<XDPContent>	n/a

Application URL

(Since 9.0) Application URLs provide access to data available in an application created in LiveCycle Workbench ES4. DDX elements can use Process URLs to access data independent of the Assembler 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

(Since 8.2) When LiveCycle Contentspace ES4 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 ES4, for example:

Company Home > User Homes > Assembler Dev Tests > doc1.pdf

The DDX can reference the above document using a Contentspace URL:

```
<PDF
   source=
   "contentspace:///Company Home/User Homes/Assembler Dev Tests/doc1.pdf"/>
```

Process URL

(Since 8.2) Process URLs provide access to data available in a process created in LiveCycle Workbench ES4. DDX elements can use Process URLs to access process data independent of the Assembler operation's input map.

The following example shows the syntax of a Process URL:

`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://localhost:8080?get-map-values(/process_data/assemInputs)#docZ.pdf
process://?get-map-values(/process_data/assemInputs)#docZ.pdf
```

Repository URL (deprecated)

Beginning with LiveCycle 9.0, the repository URL is deprecated.

A Repository URL lets DDX elements reference files within the LiveCycle ES4 repository. Such direct access is useful for standard files that are used in multiple documents, such as company logos, cover sheets, and

disclaimers. A Repository URL also enables access to specific versions of files within the LiveCycle ES4 repository.

Here is the syntax for Repository URLs:

```
repository:///dir-path/filename[?rv=myResourceVersion]
```

Where brackets indicate optional parts:

- *dir-path* is a directory path in the LiveCycle ES4 repository service.
- *filename* is the (optional) name of the file being referenced.
- *myResourceVersion* is the version of the file that is accessed. This part is valid only when the filename part is specified.

No authority or host specifications are permitted in a Repository URL.

If a Repository URL omits the *filename* part, the Assembler service includes every file in the *dir-path* folder that satisfies the filtering criteria. The sourceMatch attribute (if present) specifies the filtering criteria.

Caution: The Assembler service extends Repository URLs to support access of all files in a folder (when the filename is omitted). Other LiveCycle ES4 services do not provide such support.

Here is an example of a Repository URL:

```
repository:///FinanceAppFolder/myForm.xdp?rv=1.2
```

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:

```
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 Assembler service 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

(Since LiveCycle ES 8.2) 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

(Since LiveCycle ES4 9.0) 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. Beginning with LiveCycle ES4 (version 9), 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 Assembler service.

Enables the document name to come from an XPath string variable. For example:
`<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:

```
<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,last`
- `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.

Length value	Description
`"1.0in"`	1.0 in
`"-3.77cm"`	-3.77 cm
`"35pt"`	35 points (a point is 1/72 in)
`"0pt"`	0 points

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.

19 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 Assembler service.

```
<About
   result="xs:string"
/>
```

Can be contained in the DDX element, which is the DDX root.

This element requests that the Assembler service 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

Name	Description
result	Required. Name of the stream containing the result of this query or an External Data URL. (See "External Data URL" on page 152.)

ArtBox

Defines the extent of the page's meaningful content as intended by the creator of the page.

```
<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

Name	Description
`left`	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
`top`	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
`right`	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
`bottom`	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
`alternation`	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.

AttachmentAppearance

Defines the appearance and placement of a page-level file attachment (also known as a file attachment annotation).

```
<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

Name	Description
`icon`	Optional. The name of an icon that represents the attachment on the page. Default: `PushPin`
`color`	Optional. The color of the icon as displayed on the page. Default: blue
`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.

```
<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

Name	Description
`value`	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 "External Data URL" on page 152.)

Background

Provides styled text or PDF content for placement behind the existing page content.

```
<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. In LiveCycle ES 8.0 and later, 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.

Category

"Page content" on page 148

Attributes

Name	Description
`alternation`	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.
`fitToPage`	Optional. Specifies whether to scale the background content to fit the page. This attribute can have the following values: `false` (default) - Background content is scaled according to the `scale` attribute. `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.
`horizontalAnchor`	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: `Left` - Left page margin `Center` (default) - Midpoint between the left and right margins `Right` - Right margin
`horizontalOffset`	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.
`opacity`	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.
`replaceExisting`	Optional. If set to `false`, the specified background does not replace any pre-existing background.
`rotation`	Optional. Rotation of the background upon the page. The valid range is -360 to 360°.
`scale`	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.

Name	Description
showOnScreen	Optional. Controls whether the background is displayed when pages are viewed with an application such as Acrobat. This attribute can have the following values: `true` (default) - The background is displayed. `false` - The background is not displayed.
showWhenPrinting	Optional. Controls whether the background is included when pages are printed with an application such as Acrobat. This attribute can have the following values: `true` (default) - The background is included. `false` - The background is not included.
verticalAnchor	Optional. Specifies a vertical anchor point, relative to the PageMargins element of the page on which the background is placed. The verticalOffset attribute lets you modify the placement relative to this anchor. This attribute can have the following values: `Top` - Top page margin `Middle` (default) - Midpoint between the top and bottom margins. `Bottom` - Bottom margin
verticalOffset	Optional. Offset from the vertical anchor point specified in the verticalAnchor attribute. A positive value moves the background up, while a negative value moves it down.

BlankPage

Specifies the addition of an initially blank page to the document.

```
<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 Assembler service 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 occurs:

- 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

Name	Description
`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

Name	Description
`left`	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.
`top`	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.
`right`	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.
`bottom`	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.
`alternation`	Optional. Specifies whether the margin settings apply to all pages or alternating pages. This attribute can have the following values: `None` (default) - The margin settings apply to all pages. `OddPages` - The margin settings apply to odd pages only. `EvenPages` - The margin settings apply to even pages only.

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.

```
<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 Assembler service 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

Name	Description
`result`	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 152.)
`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.

```
<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

Name	Description
source	Required. Identifies the name of the Bookmarks XML document. This name can be specified with an External Data URL. (See "External Data URL" on page 152 and "Bookmarks Language" on page 321.)

Bookmarks filter

Specifies that bookmarks from child source elements be imported into the pages within the scope of the parent element.

```
<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"
   cardBackground="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

Name	Description
`scheme`	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.
`primaryText`	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.
`secondaryText`	Optional. The color of the text other than primary text in a navigator. Overrides the corresponding value in the `scheme` attribute, if specified.
`background`	Optional. The color of the navigator background. Overrides the corresponding value in the `scheme` attribute, if specified.
`cardBackground`	Optional. The color of the navigator card background. Overrides the corresponding value in the `scheme` attribute, if specified.
`cardBorder`	Optional. The color of the navigator card border or the "selected" card. Overrides the corresponding value in the `scheme` attribute, if specified.

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 Assembler service 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.

```
<Comments
```

```
   result="xs:string"
   format="FDF" or "XFDF"
   return="true" or "false"
   afterDate="YYYYMMDD"
   beforeDate="YYYYMMDD"
   byAuthor="Author name"
   byCategory="Notes" or "DrawingMarkups" or "TextEditingMarkups" or "Stamps"
      or "Attachments" or "All"
   byType="name1,name2,name3"
   filter="Exclude" or "Include"

>
   <PDF source> and/or <PDFGroup> [1..n]
   <Comments filter> [0..n]
   <TargetLocale> [0..1]
</Comments>
```

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 171.) 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

Name	Description
result	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 152.)
format	Optional. Specifies the format to use for the exported comments. This attribute can have the following values: FDF XFDF (default)
return	Optional. Specifies whether the exported comments are returned to the client. This attribute can have the following values: true (default) - Exported comments are returned to the client. false - Exported comments are not returned to the client but can be referenced as source from within a subsequent result element.

Selection Attributes

Name	Description
`afterDate`	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.
`beforeDate`	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.
`byAuthor`	Optional. Select comments that match the name specified by this attribute.
`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 183.) `All` (default) - All comments are selected. If `byCategory` is set to this value and `filter` is set to `Exclude`, no comments are exported. **Note:** When `All` is specified and `filter` is set to `Include`, the Assembler service 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.
`byType`	Optional. Select comments by annotation type. Multiple types can be selected by entering the names separated by a comma. Annotation types, such as Text and Line, are listed in the *PDF Reference*.
`filter`	Optional. Specifies whether to include or exclude the comments selected by the other attributes. This attribute can have the following values: `Include` (default) - Selected comments are included. `Exclude` - Selected comments are excluded.

Comments source

Comments contained in an XFDF or FDF document are imported into the pages within the scope of the parent element.

```
<Comments
   source="xs:string"
   format="FDF" 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

Name	Description
source	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 152.)
format	Optional. Specifies the format of the imported comments. This attribute can have the values XFDF (default) or FDF.

Comments filter

Specifies comments from child source elements that can be imported into the pages within the scope of the parent element.

```
<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"
>
   <PDF source> and/or <PDFGroup> [1..n]
   <Comments filter> [0..n]
   <TargetLocale> [0..1]
</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

Name	Description
`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]
   <Minute/> [0..1]
   <Minute00/> [0..1]
   <Hour/> [0..1]
   <Hour01/> [0..1]
   <Hour24/> [0..1]
   <AmPm/> [0..1]
   <DayNumber/> [0..1]
   <NayNumber01/> [0..1]
   <DayName/> [0..1]
   <ShortDayName/> [0..1]
   <MonthNumber/> [0..1]
   <MonthNumber01/> [0..1]
   <MonthName/> [0..1]
   <ShortMonthName/> [0..1]
   <Year/> [0..1]
   <ShortYear/> [0..1]
   <UTCOffset/> [0..1]
   <ShortTimeZone/> [0..1]
   <TimeZone/> [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.

DatePattern children	Formatting applied to given built-in key
xs:string	Optional. Typically, such text provides separators used between values or provides other information.
`Space`	Optional. Specifies a white space between two styled text elements.
`Second`	Optional. 1- or 2-digit (0-59) second of the minute.
`Second00`	Optional. 2-digit (00-59) second of the minute.
`Minute`	Optional. 1- or 2-digit (0-59) minute of the hour.
`Minute00`	Optional. 2-digit (00-59) minute of the hour.
`Hour`	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 `AmPm` element.
`Hour01`	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 `AmPm` element.
`Hour24`	Optional. Zero-padded 2-digit (00-23) hour of the day, expressed as a 24-hour clock.
`AmPm`	Optional. Meridian name (AM or PM) of the prevailing locale.
`DayNumber`	Optional. 1- or 2-digit (1-31) day of the month.
`NayNumber01`	Optional. Zero-padded 2-digit (01-31) day of the month.
`DayName`	Optional. Full weekday name of the prevailing locale.
`ShortDayName`	Optional. Abbreviated weekday name of the prevailing locale.
`MonthNumber`	Optional. 1- or 2-digit (1-12) month of the year.
`MonthNumber01`	Optional. Zero-padded 2-digit (01-12) month of the year.
`MonthName`	Optional. Full month name of the prevailing locale.
`ShortMonthName`	Optional. Abbreviated month name of the prevailing locale.

DatePattern children	Formatting applied to given built-in key
`Year`	Optional. 4-digit year.
`ShortYear`	Optional. 2-digit year, where 00 = 2000, 29 = 2029, 30 = 1930, and 99 = 1999.
`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:

```
<Year/>-<MonthNumber01/>-<DayNumber01/>T<Hour01/>:<Minute00/>:<Second00/>
  <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.

```
<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..1]
   <PDFsFromBookmarks> [0..n]
   <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.

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

<PDF result="resultDoc.pdf">
```

```
   <PDF source="doc2.pdf"/>
   <Metadata source="doc1.xmp"/>
</PDF>
```

Category

"Document assembly" on page 143

DDXProcessorSetting

(Since 8.0.1) Provides generic tuning hints and configuration settings to the DDX processor.

```
<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.

```
<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

Name	Description
result	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 152.)
source	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 152.) 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.

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.

```
<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

Name	Description
`mode`	Optional. Specifies whether the Document Text (DocText) XML document includes word positions. This attribute can have the following values: `TextPerPage` (default) - The DocText document provides the text that appears on each page, without providing the positions for individual words. `WithQuads` - For each word on each page, the DocText document includes a bounding box that describes the word's position on the page. `ParagraphsPerPage` - (since 9.0) Groups the words into sentences and paragraphs per page. There is no overflow to the next page.
`result`	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 152.)

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`.

Attributes

Name	Description
`editable`	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.
`name`	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 (` `).
`type`	Required. Identifies the type of data that is stored in this field. For the types `"Text"`, `"Date"` and `"Number"`, 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 `"Size"` 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*.) LiveCycle 9.0 adds support for the `CompressedSize` attribute.
`visible`	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.

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.

```
<Field
   name="xs:string"
/>
```

Attributes

Name	Description
`name`	Required. The name of a field as described in the `Schema`.

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.

```
<Field
   name="xs:string"
```

```
   ascending="true" or "false"
>
</Field>
```

Attributes

Name	Description
`ascending`	Optional. When sorting on this field, this attribute specifies whether to sort in ascending (`"true"`) or descending (`"false"`) order.
`name`	Required. The name of a field as described in the `Schema`.

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

Name	Description
`name`	Required. The name of a field as described in `Schema`.
`prefix`	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 (` `).

File

Specifies a document-level or page-level file attachment.

```
<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

Name	Description
`filename`	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.
`mimetype`	Optional. The MIME type for the data being attached. MIME types are registered with IANA (http://iana.org/assignments/media-types/).
`creationDate`	Optional. The creation date for the data being attached. If not specified, the current date, as known to the server, is used.
`modificationDate`	Optional. The modification date for the data being attached. If not specified, the current date, as known to the server, is used.

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.

```
<FileAttachments
   result="xml output name"
   extract="true" or "false"
   nameKeys=unspecified or "*" or "nameKey,nameKey,…"
>
   < PDF source> [1]
   <FilenameEncoding> [0..n]
</FileAttachments>
```

Can be contained in the DDX element, which is the DDX root.

The `FileAttachments` result 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.)

In LiveCycle Assembler 7.2 and earlier, the `FilenameEncoding` element was required to provide the character encoding of the original filenames. The original filename is the filename at the time the files are attached. In LiveCycle ES 8.0 and later, 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 Assembler service attempts to decode the filenames using each character encoding until it finds the one that works best.

Category

"Document components" on page 144

Attributes

Name	Description
`result`	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 152.)

Name	Description
extract	Optional. Specifies whether the attachments specified in the nameKeys attribute are returned to the client. This attribute can have the following values: true (default) - The file attachments are returned as separate data streams to the client. false - Only the XML document describing the attachments is returned.
nameKeys	Optional. Identifies file attachments selected from the source document. Here are the supported values: ? Unspecified (default). All page-level file attachments are extracted. ? An asterisk (*). All page-level and document-level file attachments are extracted. ? A string specifying a single name key or a comma-separated list of name keys representing document-level file attachments to extract. The appearance of a pages attribute in the PDF source element specifies pages from the source document. If such a PDF source is a child of the FileAttachments result element, only the specified pages are included in the file attachment.

FileAttachments document-level source

Specifies a file to be attached to the parent document.

```
<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 PDFGroup.

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 186.)

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

Name	Description
`source`	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 152.)
`nameKey`	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.

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.

```
<FileAttachments
   source="xs:string"
>
   <File> [1]
   <FilenameEncoding> [0..1]
   <Description> [0..1]
   <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

Name	Description
`source`	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 152.)

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.

Starting in LiveCycle ES 8.0, 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 FilenameEncoding elements is used in the order specified to decode filenames when exporting or filtering package files or exporting file attachments. Only one 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 useOnImport="true" in the most local set.

Category

"Document components" on page 144

Attributes

Name	Description
encoding	Required. The Assembler service supports the character encodings described in the following table. Depending on the character encodings available through your installation's Java Virtual Machine, the Assembler service 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.
useOnImport	Optional. Flags a FilenameEncoding element as the preferred one to use when importing package files or file attachments. If multiple FilenameEncoding elements are flagged in a set, the following occurs: ? Warning is logged ? First flagged FilenameEncoding element is used to encode the filename for compatibility with older viewers.

Character encodings

Character encoding names	Description
`ASCII`	*ISO/IEC 8859-1:1998 Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1*, published by ISO (not available online)
`BASE64`	*The Base16, Base32, and Base64 Data Encodings*, RFC 3548 (http://ietf.org/rfc/rfc3548.txt)
`UTF-8`	*The Unicode Standard, Version 4.0*, (http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404)
`UTF-16`	*The Unicode Standard, Version 4.0*, (http://.unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404)
`PDFDocEncoding`	*PDF Reference, version 1.6*, (http://partners.adobe.com/public/developer/pdf/index_reference.html)

FileSize

(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.

```
<FileSize
compressNonObjectStreams="true" or "false"
removeDuplicateResources="true" or "false"
useObjectStreams="true" or "false"
/>
```

Can be contained in the elements `PDF` result and `DDX`.

Attributes

Name	Description
`compressNonObjectStreams`	Optional. If `true`, force compression of all non-object streams, such as content streams, using a flate filter. The default is `true`.
`removeDuplicateResources`	Optional. If `true`, remove duplicate resources. The default is `true`.
`useObjectStreams`	Optional. If `true`, use object streams for objects and the crossreference table, and compress them. This requires the PDF document version to be a minimum of 1.5 and for the save mode to be `FastWebView`. The default is `true`.

Example

The following DDX applies all the default `FileSize` settings to FileSizeAll.pdf, but removes only duplicate resources from RemoveDuplicates.pdf.

```
<?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>
   <PDF result="RemoveDuplicates.pdf" save="FastWebView">
   <PDF source="largeFileSize.pdf"/>
   <FileSize useObjectStreams="false" compressNonObjectStreams="false"/>
</PDF>
</DDX>
```

Folder

(Since 9.0) Identifies a folder for the package files in a PDF Portfolio.

```
<Folder
   name="xs:string"
   thumbnail="xs:string"
>
   <Description>[0..1]
   <FieldData> [0..1]
   <Folder> [0..n]
   <PackageFiles> [0..n]
</Folder>
```

Can be contained in the elements `PDF` result, `PDF` source, and `Folder`.

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

Name	Description
`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.

```
<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

Name	Description
`shrinkContentToFit`	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.
`padding`	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 `shrinkContentToFit`.
`replaceExisting`	Optional. If it is set to `"false"`, pre-existing footers in the source PDF document remain. That is, the `Footer` element does not replace the existing footer.

Name	Description
`alternation`	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.
`whiteout`	Optional. Specifies whether the footer area (including padding) obscures the underlying page content. This attribute can have the following values: `false` (default) - Footer area does not obscure underlying page content. `true` - Footer area uses a white background to obscure any underlying page body content.
`styleReference`	Optional. The name of a `StyleProfile` element that contains a `Footer` element describing the footer. If this attribute is present, the other attributes are ignored.
`background-color`	Optional. The color of the background area for the header or footer body region. 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 152. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin`	Optional. A shorthand CSS margin property of the form. This attribute was added in LiveCycle ES 8.2. **Note:** Use XML escape characters (for example, use "<" for "<").
`margin-top`	Optional. Sets the top margin of the content/layout area. Must not be less than 0. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin-right`	Optional. Sets the right margin of the content/layout area. Must not be less than 0. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin-bottom`	Optional. Sets the bottom margin of the content/layout area. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin-left`	Optional. Sets the left margin of the content/layout area. Must not be less than 0. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.

Header

Characteristics of header content placed on a page.

```
<Header>
   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]
</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

Name	Description
`shrinkContentToFit`	Optional. Specifies whether the original page content is scaled down if necessary so that it fits below the header 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.
`padding`	Optional. A nonnegative length value specifying how much white space is added below the header, between it and the page body content. This padding is included in the header area when applying `shrinkContentToFit`.
`replaceExisting`	Optional. If it is set to `"false"`, pre-existing headers in the source PDF document remain. That is, the `Header` element does not replace the existing header.
`alternation`	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.
`whiteout`	Optional. Specifies whether the header area (including padding) obscures the underlying page content. This attribute can have the following values: `false` (default) - Header area does not obscure underlying page content. `true` - Header area uses a white background to obscure any underlying page body content.
`styleReference`	Optional. The name of a `StyleProfile` element that contains a `Header` element describing the header. If this attribute is present, the other attributes must not be present.

Name	Description
`background-color`	Optional. The color of the background area for the header or footer body region. 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 152. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin`	Optional. A shorthand CSS margin property of the form. This attribute was added in LiveCycle ES 8.2. **Note:** Use XML escape characters (for example, use "<" for "<").
`margin-top`	Optional. Sets the top margin of the content/layout area. Must not be less than 0. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin-right`	Optional. Sets the right margin of the content/layout area. Must not be less than 0. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin-bottom`	Optional. Sets the bottom margin of the content/layout area. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.
`margin-left`	Optional. Sets the left margin of the content/layout area. Must not be less than 0. This attribute is not inheritable. This attribute was added in LiveCycle ES 8.2.

Header (portfolio navigation pane)

(Since 9.0) Resource that supplies a navigation header used in a PDF Portfolio.

```
<Header source="xs:string">
   <Resource> [0..n]
</Header>Header_portfolio
```

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 WelcomePage ("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

Name	Description
source	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 <PDF> result.

InitialViewProfile

Specifies how a document is displayed when it is initially opened in a viewer application.

```
<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

Name	Description
name	Required. Name of the initial view profile. The value of this attribute must be unique among other `InitialViewProfile` elements in the same DDX document.
`display`	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.
`magnification`	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.
`openToPage`	Optional. Specifies the page displayed when the document is opened. The value must be an integer between 1 and the last ordinal page number.
`packageInitialDocument`	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.

Name	Description
`packageUIPane`	Optional. Instructions to the viewer. ? `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. ? `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. ? `Minimized` - The viewer minimizes the information. ? `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. ? `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.
`splitterBarPosition`	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. LiveCycle 9.0 adds support for this attribute.

`pageLayout`	Optional. Controls paging through the document. This attribute can have the following values:
	`Default` (default)
	`SinglePage`
	`Continuous`
	`Facing` (same as `FacingRight` with odd-numbered pages on the right)
	`ContinuousFacing` (same as `ContinuousFacingRight` with odd-numbered pages on the right)
	`ContinuousFacingRight`
	`ContinuousFacingLeft`
	`FacingRight`
	`FacingLeft`
	If the `windowOptions` attribute is set to `FullScreenMode`, this attribute is given the value `SinglePage`.

Name	Description

`show`	Optional. Specifies the panel, if any, displayed along with the page. This attribute can have the following values:
	`PageOnly` - Displays the page and no panels.
	`BookmarksPanel` (default)
	`PagesPanel`
	`AttachmentPanel`
	`LayersPanel`

`userInterfaceOptions`	Optional. Controls the interface options displayed with the page. This attribute can have the following values: *`empty-string`* (default) - No user interface options are selected. `HideMenuBar`, `HideToolBars`, `HideWindowControls` - One or more options can be specified in a comma-separated string.
`windowOptions`	Optional. Controls the appearance of the window containing the viewer. This attribute can have the following values: *`empty-string`* (default) - No window options are selected. `ResizeToInitialPage`, `CenterOnScreen`, `FullScreenMode` - One or more options can be specified in a comma-separated string.

JavaScript

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.

```
<JavaScript
   source="unspecified" or "xs:string"
   name="xs:string"
/>
```

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

Name	Description
`name`	Required. A unique name within the PDF document associated with the script.
`source`	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 19.)

Keyword

Provides a single keyword for use as metadata.

```
<Keyword
   value="xs:string"
/>
```

Can be contained in the `Keywords` element.

Category

"Document properties" on page 146

Attributes

Name	Description
`value`	The value of this attribute can be specified as a string or with an External Data URL that returns a string.

Keywords

Provides metadata keywords for the result document.

```
<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

Name	Description
`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.

```
<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.

```
<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.

```
<Links>
   result="xs:string"
   return="true" or "false"
>
   < PDF source> and/or <PDFGroup> [1..n]
   <Links source> [0..n]
   <Links filter> [0..n]
   <TargetLocale> [0..1]
</Links>
```

Can be contained in the DDX 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

Name	Description
`result`	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.
`return`	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.

Links source

Links contained in an XFDF document are imported into the pages within the scope of the parent element.

```
<Links
   source="xs:string"
/>
```

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

Name	Description
`source`	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 152.)

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.

```
<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.

Category

"Document properties" on page 146

Attributes

Name	Description
result	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 152.)
return	Optional. Specifies whether the metadata is returned to the client. This attribute can have the following values: true (default) - The XMP data is returned to the client as a named stream. false - The XMP data stream is not returned to the client but can be referenced as the source from within a subsequent result document.

Metadata source

Metadata contained in an XMP stream that is imported into the parent document.

```
<Metadata
   source="xmp input name"
>
</Metadata>
```

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

Name	Description
source	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 152.)

MetadataSchemaExtension

(Since 9.0) Metadata schema for use with PDF/A conformance.

```
<MetadataSchemaExtension source = "xs:string"/>
```

Can be contained in the <PDFAProfile> 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

Name	Description
source	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 "External Data URL" on page 153.)

Navigator

(Since 9.0) Specifies a navigator to use for a PDF Portfolio.

```
<Navigator source="xs:string">
   <Resource> [0..n]
   <String> [0..n]
</Navigator>
```

Can be contained in the Portfolio element.

Attributes

Name	Description
source	(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.

Note: NAV files and other resources can come from the repository, content services (deprecated), and a LiveCycle ES4 resource-only application.

NoBackgrounds

Specifies removal of backgrounds from the pages within the scope of the parent element.

```
<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

Name	Description
alternation	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.

NoBookmarks

Specifies removal of bookmarks from the pages within the scope of the parent element.

```
<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.

```
<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.

```
<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.

```
<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

Name	Description
`alternation`	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.

NoForms

All Acrobat (Acroform) and XFA-based form fields in the parent document are flattened.

```
<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 Assembler service 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.

```
<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

Name	Description
`alternation`	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.

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

Name	Description
`bookmarkPackageFiles`	Optional. Default is true. Determines whether to add a top-level bookmark for the pages from the Package file in the resulting "flattened" 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.

Category

"Document assembly" on page 143

NoPackageFiles

Inclusion of a `NoPackageFiles` element specifies that the PDF parent does not contain any package files.

```
<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.

```
<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.

```
<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

Name	Description
bookmarkPackageFiles	Optional. Default is true. Determines whether to add a top-level bookmark for the pages from the Package file in the resulting "flattened" 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.

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.

```
<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.

```
<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.

```
<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 Assembler service 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

Name	Description
`flatten`	Optional. Specifies whether full flattening is required. If `true`, the form is flattened. However, an error is raised if the Assembler service 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. LiveCycle 9.0 adds support for this attribute.

Category

"Document properties" on page 146

OpenPassword

User password that is required to open the resultant document.

```
<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. This element was added in LiveCycle ES 8.2.

```
<OutputIntent
   colorSpace="sRGB or CoatedFOGRA27 or JapanColorCoated or SWOP"
/>
</OutputIntent>
```

Can be contained in `PDFAProfile`.

Attributes

Name	Description
`colorSpace`	Optional. Name of color space to use during conversion. Default is sRGB.

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.

```
<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.

```
<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.

```
<Package styleReference="xs:string"/>
<Portfolio styleReference="xs:string"/>
```

Can be contained in `PDF` result and `PDF` source.

Attributes

Name	Description
`styleReference`	Required. The name of a `StyleProfile` element that contains a specification for a PDF package.

PackageFiles

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.

```
<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 182.

Attributes

Name	Description
`access`	Optional. If the source is a password encrypted PDF document, this attribute names the `PasswordAccessProfile` to apply if the document is opened or modified. This attribute was added in LiveCycle ES 8.2.
`includeSubFolders`	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. LiveCycle ES4 (version 9) adds support for this attribute.

Name	Description
`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. This attribute was added in LiveCycle 8.2.
`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. This attribute was added in LiveCycle ES 8.2.

Name	Description
`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 152.) 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.

```
<PackageFiles
   nameKeys="unspecified" or "xs:string"
>
   <PDF> source [1..n]
   <FieldData> [0..n]
   <TargetLocale> [0..1]
   ...  PackageFiles modifying elements  ...
</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

Name	Description
`nameKeys`	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.

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.

```
<PackageFiles
   nameKeys="unspecified" or "xs:string"
>
   <FieldData> [0..n]
   ... PackageFiles modifying elements ...
</PackageFiles>
```

For more information on the FieldData element, see "FieldData" on page 182.

Can be contained in PDF result and PDF source.

Attributes

Name	Description
nameKeys	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.

PackageFiles result elements

Returns an XML file containing information about the package files. Can also return the package files.

```
<PackageFiles
   result="xs:string"
   extract="true" or "false"
   nameKeys="unspecified" or "xs:string"
>
   <PDF> source [1]
   <FilenameEncoding> [0..n]
   <TargetLocale> [0..1]
</PackageFiles>
```

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

Name	Description
`extract`	Optional. If its value is `"true"`, all package files specified are returned as separate data streams to the client. If its value is `"false"`, then only the result XML data, containing information about the package files specified, is returned.
`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.

```
<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 217

"Modifying the package files in a PDF package or portfolio" on page 59

"PackageFiles Language" on page 350

Attributes

Name	Description
`import`	Required. A name to be associated with the data stream containing the XML generated from a `PackageFiles` result specification. The XML provides the mapping of the input data stream names to information stored with the package file. That information includes the filename, creation date, MIME type, and any custom metadata (see "PackageFiles Language" on page 350). The import source can be specified with an External Data URL. (See "External Data URL" on page 153.)

PageContent

(Since 8.2) Adds content to a page similar to `Watermark` but with alternate text and style profiles.

```
<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

Name	Description
alternateText	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.
alternation	Optional. If None, applies to all pages. If OddPages, applies to odd pages only. If EvenPages, applies to even pages only.

Name	Description
appears	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.
fitToPage	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.
horizontalAnchor	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.
horizontalOffset	Optional. The offset from the horizontalAnchor point. A positive value moves right, while a negative value moves left.
opacity	Optional. Controls the transparency of the PageContent text. 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%.
rotation	rotation Optional. The valid range is -360 to 360 degrees.
scale	Optional. The valid range is 8 to 3200 percent.
showOnScreen	Optional. A Boolean value that controls whether the PageContent is displayed when pages are viewed within an application such as Acrobat.
showWhenPrinting	Optional. A Boolean value that controls whether the PageContent appears on the page when printed.
verticalAnchor	Optional. The verticalOffset is relative to the verticalAnchor. Top is the top PageMargin. Middle is the middle of the page and Bottom is the bottom PageMargin
verticalOffset	Optional. The offset from the verticalAnchor point. A positive value moves up, while a negative value moves down.

The following example shows how to reference a PageContent element contained within a named StyleProfile element:

```
<PageContent styleReference="xs:string"/>
```

In the following example, doc1 is a tagged PDF document.

```
<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.

```
<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.

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

Name	Description
`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.

```
<PageMargins
   left="36pt" or "nonnegative length"
   top="36pt" or "nonnegative length"
   right="36pt" or "nonnegative length"
   bottom="36pt" or "nonnegative 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.

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

Name	Description
left	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).
top	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).
right	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).

Name	Description
`bottom`	Optional. Width of bottom margin, which is the distance between the bottom of the page and the bottom of the page content. A negative value defaults to 0. Default is 36 points (0.5 inches).
`alternation`	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.

PageOverlay

Provides content for placement over the current page content.

```
<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

Name	Description
`embedFormsAndAnnots`	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.
`newX`	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.
`newY`	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.
`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.

```
<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

Name	Description
rotate90	Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number is clockwise, and a negative number is counterclockwise.

PageSize

Defines the page dimensions for purposes of display or print.

```
<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 Assembler service 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

Name	Description
width	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.
height	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.
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.

Name	Description
`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`

PageUnderlay

Provides content for placement under the current page content.

```
<PageUnderlay
   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]
</PageUnderlay>
```

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

Name	Description
`embedFormsAndAnnots`	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.
`newX`	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.
`newY`	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.
`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 Assembler service 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.

```
<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.

```
<PasswordAccessProfile
   name="xs:string"
>
   <Password>
```

```
</PasswordAccessProfile>
```

Can be contained in the DDX element, which is the DDX root.

The password specified in this element allows the Assembler service 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 Assembler service 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

Name	Description
name	Required. The name of the PasswordAccessProfile element. The PDF source element contains an access attribute that can reference the password contained in this element.

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.

Category

"Profile" on page 148

Attributes

Name	Description
`name`	Required. Name of the `PasswordEncryptionProfile` element. The `PDF` result element's `encryption` attribute uses this name to reference the profile. The value of this attribute must differ from the names assigned to other `PasswordEncryptionProfile` elements within the same DDX. Also, the name cannot be `None` because that value is reserved to indicate that the result document not be encrypted.
`compatibilityLevel`	Optional. A code specifying the algorithm used to encrypt and decrypt the document. These algorithms are described in the *PDF Reference*. Possible values for this attribute are `Acrobat3`, `Acrobat5` (default), `Acrobat6`, `Acrobat7`, `Acrobat9`. 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. For `Acrobat9`, encryption is 256-bit AES, supports FIPs, and requires "Unlimited Strength Jurisdiction Policy Files" be available in the Java Runtime Environment (JRE).
`encryptionLevel`	Optional. Specifies the parts of the document that are encrypted. This attribute can have the following values: `All` (default) - The entire document is encrypted, including metadata and file attachments. `NotMetadata` - Excludes document-level metadata from encryption. `OnlyFileAttachments` - Only file attachments are encrypted. This setting applies only when the `compatibilityLevel` attribute has a value of `Acrobat7`. This attribute is ignored if the `compatibilityLevel` attribute is set to `Acrobat3` or `Acrobat5`.

PDF

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.

```
<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"
   >

   <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]
   <TableOfContents> [0..1]
   <TargetLocale> [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

Name	Description
result	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.)
format	Optional. This attribute can have the following values: PDF (default) - Result document is returned as a PDF document. 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.

Name	Description
`return`	Optional. Specifies whether the assembled PDF document is returned. This attribute can have the following values: `true` (default) - The assembled PDF document is returned to the client as a stream. `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.
`initialView`	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.
`save`	Optional. Specifies how Acrobat save the result document. If specified, this attribute can have the following values: `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. `Full` - A full save is performed, overwriting any existing incremental saves. This option does not result in optimization for fast web viewing. `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. 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 Assembler service performs a full save. **Note:** The removal of existing incremental saves invalidates any signatures, certification, or reader-enabled rights set on the base document.
`encryption`	Optional. If specified, this attribute can have the following values: *`PasswordEncryptionProfile name`* - This value is the name of the `PasswordEncryptionProfile` element with which to encrypt the result. `None` - Specifies that the output must have no encryption regardless of whether the source base document was encrypted. If this attribute is not specified and the `baseDocument` is encrypted, then the resultant document has the same encryption.

Name	Description
`readerUsageRights`	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 <`ReaderRights`> element is present and a Reader Extension service is available, then the specified Reader Rights overrides the value of `None`.
`pdfa`	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. This attribute was added in LiveCycle ES 8.2.
`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.
`sortBookmarks`	If `true`, bookmarks are sorted relative to their target page numbers in the resultant document. Only bookmarks that have page destinations within the resultant document are sorted. Such bookmarks have destinations such as XYZ, Fit, FitB, FitH, FitV, FitBH, FitBV, and FitR. (See "Sorting bookmarks" on page 69.) Bookmarks that do not have page destinations within the resultant document are added as is. Their position remain nearest to their parent, as they would fall in the order depending upon the ordering of the page-numbered Bookmarks. This attribute was added in LiveCycle ES4 Service Pack 1 (version 9.0.0.1).

PDF source

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]
   <Folder> [0..1]
   <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]
   <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]
    <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]
    <Watermark>
      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

Name	Description
`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 Adobe LiveCycle Assembler service, 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.

Name	Description
`baseDocument`	Optional. Identifies the base document and provides the initial structure that the Assembler service 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 Assembler service, 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 Assembler service 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.
`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.

Name	Description
`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. LiveCycle 9.0 adds support for this attribute.
`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.
`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.
`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.

PDFGroup

Specifies a grouping of source documents to which page properties, page content, and document components can be applied.

```
<PDFGroup>
   <ArtBox> [0..2, where 2 is allowed for alternating pages]
   <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]
   < FileAttachments source> [0..n] or <NoFileAttachments> [0..1]
   <FilenameEncoding> [0..n]
   <Footer>
     or <NoFooters> [0..2, where 2 is allowed for alternating pages]
   <Header>
     or <NoHeaders> [0..2, where 2 is allowed for alternating pages]
   <Links source> and/or <Links filter> [0..n] or <NoLinks> [0..1]
   <NoForms> [0..1]
   <NoJavaScripts> [0..1]
   <NoThumbnails> [0..1]
   <NoXFA> [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]
   <TableOfContents> [0..1]
   <TargetLocale> [0..1]
   <TrimBox> [0..2, where 2 is allowed for alternating pages]
   <Transform> [0..1]
     <Watermark>
     or <NoWatermarks> [0..2, where 2 is allowed for alternating pages]
   <XFAConversionSettings> [0..1]
</PDFGroup>
```

Can be contained in any of the following elements: PDF result, PDFGroup, Bookmarks result, Bookmarks filter, Links result, Links filter, Comments result, Comments filter, and DocumentText.

The PDFGroup element provides a convenient way to specify elements that apply to multiple PDF source elements. All the other child elements of PDFGroup apply to all PDF source elements in the scope of the PDFGroup element.

PDFGroup elements can be nested. The innermost PDFGroup element must contain at least one PDF source.

Category

"Document assembly" on page 143

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 Assembler service 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 Assembler service replaces these characters with an underscore (_). The following characters are invalid file path characters:

- Slash "/"
- Backslash "\"
- 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

Name	Description
`encryption`	Optional. Name of the `PasswordEncryptionProfile` element to use for encrypting the result document. This attribute can have the following values: `None` (default) - No encryption. string - Name of the `PasswordEncryptionProfile` element.
`prefix`	Required. Prefix to use in the names of the result documents. The DDX processor modifies any invalid characters in the name.
`save`	Optional. Specifies how the result documents are saved. This attribute can have the following values: `Full` (default) `FastWebView` - Full save that restructures the result documents for page-at-a-time downloading from web servers. Incremental save is not supported.

PDFAProfile

(Since 8.2) Provides settings for conversion to a PDF/A document. The `PDF` result element's `pdfa` attribute references this element.

```
<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"
>
   <OutputIntent> [0..1]
   <MetadataSchemaExtension> [0..1]
</PDFAProfile>
```

Can be contained in the `DDX` element, which is the DDX root. This element was added in LiveCycle ES 8.2.

Attributes

Name	Description
`compliance`	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`.
`name`	Required. The name of the PDF/A profile.

Name	Description
`resultLevel`	Optional. Specifies the reporting level from the conversion effort. A value of `PassFail` reports whether the conversion was successful for the compliance and options that were specified. A value of `Summary` is similar to Acrobat's PDF/A Preflight results in how it reports a count of each error encountered. `Detailed` attemps 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`.
`optionalContent`	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 `Visible` converts all visible optional content to content and remove the remaining optional content. A value of `All` converts all optional content to visible content regardless of the original optional content's visibility. The default is `Visible`.
`signatures`	Optional. Specifies whether to archive signatures. The default is `ArchiveAsNeeded`. 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.
`verify`	Optional. If `verify` 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`.

PDFAValidation

Used to validate PDF/A conformance.

```
<PDFAValidation
   allowCertificationSignatures="true or false"
   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

Name	Description
`allowCertificationSignatures`	Determines whether the PDF document is PDF/A compliant. Valid values are `true`, `false`, and `notvalidated`. Default value: `true`
`compliance`	Specifies the PDF/A compliance level. Valid values are `PDF/A-1b`, `PDF/A-2b, or PDF/A-3b`. Default value: `PDF/A-1b`
`ignoreUnusedResources`	Specifies whether to ignore resources that are not used. Valid values are `true` or `false`. Default value: `true`
`resultLevel`	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`.

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

Name	Description
`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.

Name	Description
`copy`	Optional. Specifies whether users can copy text, images, and other content. This setting is available only if the parent `PasswordEncryptionProfile` element's `compatibilityLevel` attribute has a value of `Acrobat5` or later. This attribute can have the following values: `Yes` (default) - Users can copy text, images, links, forms, and other content. `No` - Users cannot copy text, images, and other content.
`screenReading`	Optional. Specifies whether users can use a screen reader to read content from the document. If users are allowed to do so, they can use the screen reader to perform these tasks: ? Select text or graphics. ? Copy the selected items into the system buffer. ? Paste the content into another application. This setting is available only if the parent `PasswordEncryptionProfile` element's `compatibilityLevel` attribute has a value of `Acrobat5` or later. This attribute can have the following values: `Yes` (default) - Users can use a screen reader to copy content from the document. `No` - Users cannot read the document with a PDF screen reader. If set to `No`, then copy is set to `No`, even if not specified.

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

Name	Description
styleReference	Required. The name of a StyleProfile element that contains a specification for a PDF Portfolio.

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.

Attributes

Name	Description
`name`	(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.
`source`	(Required) An input map key or External Data URL which specifies the document to add as a resource.

RichMedia

(Since 9.0) Attaches a compiled ActionScript program (SWF file) to a page.

```
<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

Name	Description
`source`	Required. Identifies the SWF content. The source can be specified with an External Data URL.
`page`	Optional. Specifies the page on which the SWF content is attached.
`left`	Optional. Left alignment of the SWF content on the page. The units are points.
`top`	Optional. Top alignment of the SWF content on the page. The units are points.
`width`	Optional. Width of the display for the SWF content. The width and height are determined from the SWF content itself. The units are points.
`height`	Optional. Height of the display for the SWF content. The width and height are determined from the SWF content itself. The units are points.

Right

Specifies the right edge of the page as the anchor point for a header or footer. The `PageMargins` element defines the right edge.

```
<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.

```
<Schema/>
```

which is the equivalent of specifying the following, given the inherited `TargetLocale` is `"en"`:

```
<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.

```
<Schema>
   <Field> [0..n]
</Schema>
```

For more information, see "Field contained in Schema element" on page 180.

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.

```
<SortOrder>
   <Field> [0..n]
</SortOrder>
```

For more information, see "Field contained in SortOrder element" on page 181.

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.

```
<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

Name	Description
`name`	(Required) Name of the resource string. The name can be provided with an External Data URL.
`url`	(Required) An input map key or External Data URL that provides the string to add.

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.

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>color</td><td>Optional and inheritable. Color of the enclosed text, leaders, and text decorations.</td></tr>
<tr><td>font</td><td>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><tr><th>Style</th><th>Weight</th><th>Size</th><th>Line height</th><th>Family</th></tr><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></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.</td></tr>
</table>

Name	Description
`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 Assembler service 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 Assembler service environment variables specify a default font family. Font names that contain a space must be enclosed in single quotes, as shown below: 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 268.)
`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`

Name	Description

`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`

Name	Description
`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 inherits 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 Assembler service determines default values as described in the *Cascading Style Sheets Specification, version 2* (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.

Name	Description
`margin-left`	Optional. Sets the left 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.

`text-align`	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 `span`, `b`, `i`, and `leader`.
	This attribute is not supported in the DDX `Header` and `Footer` elements, which take alignment instructions from their child elements `Left`, `Center` and `Right`.
	This attribute is inheritable if specified in the `p` or `StyledText` element.

`text-decoration`	Optional and inheritable. Specifies special characteristics applied to the enclosed text. The following values are supported:
	`none` (default)
	`underline`
	`overline`
	`line-through`

`text-indent`	Optional and inheritable. Specifies additional indention of the first line of a paragraph. A negative value cannot be greater than the paragraph's `margin-left` value. This attribute is ignored in the elements `span`, `b`, `i`, and `leader`. This attribute is inheritable if specified in the `p` or `StyledText` element.

Name	Description
`vertical-align`	Optional. Specifies vertical positioning of the content described by the element. The following values are supported: `baseline` (default) - Align the baseline of the element (or the bottom, if the element does not have a baseline) with the baseline of the parent `base` - Same meaning as the `baseline` value. `sub` - Subscript the element. `subscript` - Same meaning as the `sub` value. `sup` - Superscript the element. `super` - Same meaning as the `sup` value. `superscript` - Same meaning as the `sup` value. *`length`* - Distance from the current baseline to the new baseline. This attribute is not inheritable.
`xml:space`	Optional and inheritable. Specifies handling for the whitespace characters carriage return, line feed, and space. The following values are supported: `default` (default) - Convert line feeds and carriage returns to spaces and collapse multiple adjacent whitespace to a single space. `preserve` - All spaces, line feeds, and carriage returns are retained as entered. The line feed is interpreted as a line-break.

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.

Desired font	Selected font	Workaround
`Plain`	Decorative variation, extension, or subset, as described in "Font variations, extensions, and subsets" on page 269	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 268
`Decorative variation, extension, or subset`	Plain	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 269

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*

```
<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
- SubCapt
- 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

- 31AB
- 31BC
- 32AB
- 32BC
- 33BC
- Symbol

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 261.)

b

Specifies that the bold typeface of a font is applied to an isolated string of text (`font-weight="700"`).

```
<b
   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"
   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
   <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</b>
```

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

Name	Description
numberOfDigits	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.
prefix	Optional. Any text that appears just before the bates number.
start	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.
suffix	Optional. Any text that is to appear just after the Bates number.

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

Name	Description
styleReference	Optional. The name of a StyleProfile element that itself contains a DatePattern element as appropriate.

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

Name	Description
block-align	Optional. Specifies the horizontal placement of the graphic relative to the specified height.
height	Optional. The optimal height of the graphic. The default of "auto" uses the "intrinsic-height" of the graphic source. It must be greater than 0.
inline-align	Optional. Specifies the vertical placement of the graphic relative to the specified width.
rotate90	Optional. The reference orientation of the graphic with respect to the page.

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.

i

Specifies that italics be applied to an isolated string of text (`font-style="italic"`).

```
<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*.

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"
   <Space/>
   />
```

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
   <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</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   can also be used.

```
<Space/>
```

Can be contained in the p, b, i, span, and DatePattern elements.

span

Specifies formatting for spans of inline text.

```
<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
   <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</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.

```
<String url="xs:string">
   xs:string
</String>
```

Can be contained in the elements p, i, b, and span.

Example

- `<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".
- `<String url="badURL">Dr. Zhivago</String>`, the text is "Dr. Zhivago".
- `<String url="badURL"/>`, the text is "badURL".

StyledText

The root of a rich text expression.

```
<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"
   font-family=unspecified or "font-family-specifier"
   font-size="12pt" or "positive-length"
   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"
   line-height="normal" or "positive-length"
   text-decoration="none" or "underline" or "overline" or "line-through"
>
   <span> [0..n]
   <i> [0..n]
   <b> [0..n]
   <graphic> [0..n]
   <sub> [0..n]
   <sup> [0..n]
   <leader> [0..n]
   <Built-in key> [0..n]
   <TargetLocale>  [0..1]
   xs:string
</sub>
```

Can be contained in the elements p, i, b, span, sub, 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"
   font-family=unspecified or "font-family-specifier"
   font-size="12pt" or "positive-length"
   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"
   line-height="normal" or "positive-length"
   text-decoration="none" or "underline" or "overline" or "line-through"
>
   <span> [0..n]
   <i> [0..n]
   <b> [0..n]
   <graphic> [0..n]
   <sub> [0..n]
   <sup> [0..n]
   <leader> [0..n]
   <Built-in key> [0..n]
```

```
   <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.

```
<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

Name	Description
name	The name of the StyleProfile element.

Subject

Provides the value for the subject metadata in the result document.

```
<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

Name	Description
value	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.

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 Assembler service 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

Name	Description
`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.

Name	Description
`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. This attribute was added in LiveCycle ES 8.2.
`toc-margin`	Optional and inheritable. A shorthand CSS margin property of the form: <margin-top><margin-right><margin-bottom><margin-left>. This attribute was added in LiveCycle ES 8.2.
`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. This attribute was added in LiveCycle ES 8.2.
`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. This attribute was added in LiveCycle ES 8.2.
`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. This attribute was added in LiveCycle ES 8.2.
`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. This attribute was added in LiveCycle ES 8.2.

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

Attributes

Name	Description
applicableLevel	Optional. The bookmark level for which this entry pattern is defined. This attribute can have the following values: all (default) - Entry pattern applies to all bookmark levels. positive non-zero integer - A single bookmark level to which the entry pattern applies.

TableOfContentsPagePattern

Defines the style for table of contents pages.

```
<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

Name	Description
`pages`	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.
`toc-backgroundcolor`	Optional. The toc-background and related toc-margin attributes are used to set up the background for the area occupied by the TOC listings. This attribute was added in LiveCycle ES 8.2.
`toc-margin`	Optional. A shorthand CSS margin property of the form: <margin-top><margin-right><margin-bottom><margin-left>. This attribute was added in LiveCycle ES 8.2.
`toc-margin-top`	Optional: Sets the top margin of the table of contents background area. Must not be less than 0. This attribute was added in LiveCycle ES 8.2.
`toc-margin-right`	Optional. Sets the right margin of the table of contents background area. Must not be less than 0. This attribute was added in LiveCycle ES 8.2.
`toc-margin-bottom`	Optional. Sets the bottom margin of the table of contents background area. Must not be less than 0. This attribute was added in LiveCycle ES 8.2.
`toc-margin-left`	Optional. Sets the left margin of the table of contents background area. Must not be less than 0. This attribute was added in LiveCycle ES 8.2.

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 LiveCycle Assembler service 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

Name	Description
`locale`	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 `"en_US"`, `"de"`, and `"zh_Hans"`.

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

Name	Description
value	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.

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

Name	Description
scale	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.
newX	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.

Name	Description
newY	Optional. Units by which the Y-axis origin is shifted. A positive value shifts the origin upward, and a negative value shifts it downward.
rotate90	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.

TrimBox

Defines the intended dimensions of the finished page after printing and trimming.

```
<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

Name	Description
left	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.
top	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.
right	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.

Name	Description
`bottom`	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.
`alternation`	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.

Watermark

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.

```
<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

Name	Description
`alternation`	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.
`fitToPage`	Optional. Specifies whether to scale the watermark content to fit the page. This attribute can have the following values: `false` (default) - Watermark content is scaled according to the `scale` attribute. `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.
`horizontalAnchor`	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. `Left` - Left page margin `Center` (default) - Midpoint between the left and right margins `Right` - Right margin
`horizontalOffset`	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.
`opacity`	Optional. Transparency of the watermark or background text. 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%.
`replaceExisting`	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.
`rotation`	Optional. Rotation of the watermark upon the page. The valid range is -360 to 360 degrees.
`scale`	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.

Name	Description
`showOnScreen`	Optional. Controls whether the watermark or background is displayed when pages are viewed with an application such as Acrobat. This attribute can have the following values: `true` (default) - The watermark is displayed. `false` - The watermark is not displayed.
`showWhenPrinting`	Optional. Controls whether the watermark or background is included when pages are printed with an application such as Acrobat. This attribute can have the following values: `true` (default) - The watermark is included. `false` - The watermark is not included.
`verticalAnchor`	Optional. Specifies a vertical anchor point, relative to the `PageMargins` element of the page on which the watermark is placed. The `verticalOffset` attribute modifies the placement relative to this anchor. `Top` - Top page margin `Middle` (default) - Midpoint between the top and bottom margins. `Bottom` - Bottom margin
`verticalOffset`	Optional. Offset from the vertical anchor point specified in the `verticalAnchor` attribute. A positive value moves the watermark up, while a negative value moves it down.

WelcomePage

Specifies the Welcome Page used in a PDF Portfolio.

```
<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

Name	Description
source	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 <PDF> result.

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.

```
<XDP
   aggregateXDPContent="None" or "All"
>
   <XDPContent> | <XDP source> | <XFAData> [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

Name	Description
`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.

```
<XDP
   result="xs:string"
   aggregateXDPContent="None" or "All"
   removeInsertionPoints= "xs:string"
   resolveAssets="none" or "all" or "absolute" or "relative"
   retainInsertionPoints= "xs:string"
   return="true" or "false"
>

   <XDPContent> [0..n]
   <XDP source> [1..n]
</XDP>
```

Can be contained in the DDX element.

Attributes

Name	Description
`result`	Required. Name of the assembled XDP document. The name can be provided with an External Data URL. See "External Data URL" on page 153.
`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.

Name	Description
`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.
`return`	Optional. Specifies whether the assembled XDP document is returned. This attribute can have the following values: `true` (default) - The assembled XDP document is returned to the client as a stream. `false` - The assembled XDP document is available as transient data, which can be referenced as source from within a subsequent `XDP` result element. In this case, the assembled XDP document is not returned to the client.

XDP source

Specifies XDP source content to assemble into the XDP result.

```
<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

Name	Description
`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 Assembler service 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 Assembler service, 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.

Name	Description
`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.)

Name	Description
`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

XDPContent

(Since 9.0) Specifies XDP content to insert into the XDP source or result being specified.

```
<XDPContent source="xs:string"
   fragment="xs:string"
   includeSubFolders="true" or "false"
```

```
   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> [0..n]
</XDPContent>
```

Can be contained in the elements XDP result, XDP source, and XDP generic.

Category

"Document assembly" on page 143

Attributes

Name	Description
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 only used if it matches a name entry in the input map.
fragment	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 LiveCycle Designer ES4. The forms are saved as XDP documents.
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.
insertionPoint	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 LiveCycle Designer ES4. The forms are saved as XDP documents.

Name	Description
`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.
`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.
`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. 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.
`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.

Name	Description
`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

Part III: Supporting XML Grammars Reference

This section describes the syntax and semantics of the XML grammars used with the DDX grammar.

20 Extended Services

Some DDX elements provide configuration information for DDX processors that can call other services. For example, the Assembler service can call these LiveCycle ES4 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.

```
<PDFGenerationSettings
   conversionSettings="xs:string"
   fileTypeSettings="xs:string">
/>
```

Can be contained in the DDX, PDF, and PDFGroup elements. This element was added in LiveCycle ES 8.2.

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 LiveCycle ES4 Administration Help for information on default and custom settings.

Attributes

Name	Description

`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 LiveCycle 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 LiveCycle 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.

```
<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. This element was added in LiveCycle ES 8.2.

The attributes to include in the `ReaderRights` element depend on the rights that the specified credential includes.

Attributes

Name	Description
`credentialAlias`	(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.
`formFillIn`	(Optional) Corresponds to the `formFillIn` usage right.
`formDataImportExport`	(Optional) Corresponds to the `formDataImportExport` usage right.
`submitStandalone`	(Optional) Corresponds to the `submitStandalone` usage right.
`onlineForms`	(Optional) Corresponds to the `onlineForms` usage right.
`dynamicFormFields`	(Optional) Corresponds to the `dynamicFormFields` usage right.
`dynamicFormPages`	(Optional) Corresponds to the `dynamicFormPages` usage right.
`barcodeDecoding`	(Optional) Corresponds to the `barcodeDecoding` usage right.
`digitalSignatures`	(Optional) Corresponds to the `digitalSignatures` usage right.
`comments`	(Optional) Corresponds to the `comments` usage right.
`onlineComments`	(Optional) Corresponds to the `onlineComments` usage right.
`embeddedFiles`	(Optional) Corresponds to the `embeddedFiles` usage right.
`mode`	(Optional) Corresponds to the `mode` usage right.

XFAConversionSettings

(Since 8.2) Controls the options for converting XFA documents to PDF documents.

```
<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 Assembler service, 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

Name	Description
renderAtClient	Optional. Enables the delivery of PDF content by using the client-side rendering capability of Acrobat 7 or later. The value auto uses the XFA dynamicRender configuration value. Default is auto, unless NoXFA is specified, and then the default value is false. A value of true results in the creation of a Dynamic XFA PDF.
tagged	Optional. Setting to true results in creating a tagged PDF document. The default is false.

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 LiveCycle is installed on JBoss, the full path is [Install location]\svcdata\XMLFormService.

```
<XFAConversionSettings>
   <XCI source="source key">
</XFAConversionSettings>
```

Can be contained in the XFAConversionSettings element.

Attributes

Name	Description
source	Required. A logical name, associated with an input data stream or an ordered list of data streams, containing XCI content. If the source 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 "Source elements" on page 19.)

XFAData

(Since 8.2) The `XFAData` 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 `XFAData` element.

Note: The Assembler service uses the Forms service to merge data into XFA forms. If that service is not available, the `XFAData` element is ignored.

If the parent `PDF` source attribute or the `XFAData` source attribute points to a list of documents, only the first document in each list is used.

```
<XFAData
   source="xs:string"
>
   <any element in ##other namespace> or <any element in ##local namespace>
[0..1]
</XFAData>
```

Can be contained in the elements PDF source and XDP (generic).

Attributes

Name	Description
source	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.

The following is an example XDF Data.

```
<PDF result="filledinForm.pdf">
   <PDF source="form.pdf">
      <XFAData>
         <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>
      </XFAData>
   </PDF>
```

```
</PDF>
```

21 About Language

The About language is an XML syntax that provides information about the Assembler service.

The Assembler service 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.

```
<About>
   <Processor>
   <Version>
   <Build>
   <Copyright>
</About>
```

Build

Provides the build number for the Assembler service.

```
<Build>
   xs:string
</Build>
```

Copyright

Provides the copyright for the Assembler service.

```
<Copyright>
   xs:string
</Copyright>
```

Processor

Provides the product name of the Assembler service.

```
<Processor>
   xs:string
</Processor>
```

Version

Provides the version number of the Assembler service.

```
<Version>
   xs:string
</Version>
```

22 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 Assembler service returns a `DocInfo` document in response to the `DocumentInformation` element appearing in the DDX document.

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.

```
<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.

```
<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 `<year>-<month>-<day>T<hour>:<minute>:<sec>-<offsetGMTHour><offsetGMTMinute>`

```
<CreatedDate>
   "xs:string"
</CreatedDate>
```

Creator

Name of the application used to produce the content of the document.

```
<Creator>
```

```
   "xs:string"
</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.

```
<DisplayOrder>
   <Field name="xs:string" /> [0..n]
</DisplayOrder>
```

Attributes

Name	Description
`name`	Required. The normalized name of a Field as defined in the Schema. The non-normalized name appears as the content in the `Schema` element.

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.

```
<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.

```
<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"
"XFAForeground"
```

Keyword

Keyword associated with the specified PDF document.

```
<Keyword>
   xs:string
</Keyword>
```

Can be contained in the Keywords element.

Keywords

Keywords associated with the specified PDF document.

```
<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 <year>-<month>-<day>T<hour>:<minute>:<sec>-<offsetGMTHour><offsetGMTMinute>

```
<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

Name	Description
pages	The page range to which the page label format applies. (See "Page and document ranges" on page 156.)
start	First label number for this page range, not necessarily the ordinal page number.
format	Manner in which the number is displayed. The following values are supported: None - Indicates there is no number following the prefix in the page label. Decimal - 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, …
prefix	Text displayed before the number.

Example: *Specifying page labels*

```
<!-- 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

The page labels for the specified PDF document.

```
<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.

```
<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.

```
<PageSize
   pages="pages"
   width="page width" [or rename to shortedge/longedge]
   height="page height"
/>
```

Can be contained in the PageSizes element.

Attributes

Name	Description
pages	Range of pages that share this same width and height. ("Page and document ranges" on page 156)
width	Width of the page (media).
height	Height of the page (media).

Example: *Page ranges qualify page sizes*

```
<?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.

```
<PageSizes>
   <PageSize> [1..n]
</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.

```
<PageRotation
   pages="page range"
   rotate90="degrees in increments of 90"
/>
```

Can be contained in the `PageRotations` element.

Attributes

Name	Description
`pages`	A page range that describes the pages having a the orientation specified by this element. (See "Page and document ranges" on page 156.)
`rotate90`	Optional. Specifies a rotation setting for the page in increments of 90 degrees. A positive number is clockwise, and a negative number is counterclockwise.

PDFAConformance

Specifies the PDF/A conformance level of the PDF.

```
<PDFAConformance
   compliance="PDF/A-1b, PDF/A-2b, or PDF/A-3b"
```

```
   isCompliant="false or true or notValidated"
   resultLevel="PassFail or Summary or Detailed"
   ignoreUnusedResources="xs:boolean"
   allowCertificationSignatures="xs:boolean"
>
   <ViolationDetail> [0..n]
   <Violation> [0..n]
</PDFAConformance>
```

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 `PFDAConformance` is reported as `notvalidated` if the `PDFAValidation` element is not present.

Attributes

Name	Description
`isCompliant`	Determines whether the PDF document is PDF/A compliant. Valid values are `true`, `false`, and `notvalidated`.
`compliance`	Specifies whether this document is PDF/A compliant. Valid values are `PDF/A-1b`, `PDF/A-2b`, or `PDF/A-3b`.
`resultLevel`	Specifies how much information is returned. Valid values are `PassFail`, `Summary`, or `Detailed`.
`ignoreUnusedResources`	Specifies whether to ignore resources that are not used. Valid values are `true` or `false`.
`allowCertificationSignatures`	Specifies whether signatures are allowed. Valid values are `true` or `false`.

Producer

Name of the application used to convert the content into PDF.

```
<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

Name	Description
`editable`	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.
`name`	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.
`type`	Required. The type identifies the type of data that is stored in this field.
`visible`	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.

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

Name	Description
`ascending`	Optional. When sorting on this field, this attribute specifies whether to sort in ascending (`"true"`) or descending (`"false"`) order.
`name`	Required. The name of a field as described in the schema.

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.

Value	PDF Reference
`1.7`	*PDF Reference, Sixth Edition, version 1.7*, available at http://www.adobe.com/go/pdf_developer.
`1.6`	*PDF Reference, Fifth Edition, version 1.6*, available at http://www.adobe.com/go/pdf_developer
`1.5`	*PDF Reference, Fourth Edition, Version 1.5*, available at http://www.adobe.com/go/pdf_developer
`1.4`	*PDF Reference, Third Edition, Version 1.4*, available at http://www.adobe.com/go/pdf_developer
`1.3`	*PDF Reference, Second Edition, Version 1.3*, published by Addison-Wesley, ISBN 0-201-61588-6 and available at http://www.adobe.com/go/pdf_developer

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"
```

```
   fieldAnnot="xs:string"
   colorSpace="xs:string"
   font="xs:string"
   pattern="xs:string"
   xObject ="xs:string"
/>
```

Violation

Contained within `PDFAConformance` element to report a count of each type of detected violation.

```
<Violation
   count="xs:positiveInteger"
   key="xs:string"
   description="xs:string"
/>
```

23 Bookmarks Language

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 Assembler service is 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.

Bookmark title	Level	Effect when clicked
First Slide	1	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="11"` is displayed as page number 12.
Second Slide	1	Changes the view to page 13, fitting that page into the viewer window.
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 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>
         <Title>A backup slide</Title>
            <Dest>
               <XYZ PageNum="13" Left="45.0" Top="530.0" Zoom="2.0"/>
         </Dest>
      </Bookmark>
   </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.

Bookmark title	Level	Effect when clicked
Document Title	1	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.
Document Subtitle	2	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.
Preface	3	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.
Chapter 1	3	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.

Example: *A basic bookmark expression that specifies go-to actions*

```
<?xml version="1.0" encoding="UTF-8"?>
<Bookmarks xmlns="http://ns.adobe.com/pdf/bookmarks" version="1.0">
   <Bookmark>
      <Bookmark>
         <Bookmark>
            <Action>
               <Launch NewWindow="true">
                  <File Name="MyFile.pdf"/>
               </Launch>
            </Action>
            <Title>Contents</Title>
         </Bookmark>
         <Bookmark>
            <Action>
               <GoTo>
                  <Dest>
                     <FitR PageNum="23" Left="28" Bottom="2"
                        Right="23" Top="404"/>
                  </Dest>
               </GoTo>
            </Action>
            <Title>Preface</Title>
         </Bookmark>
         <Dest>
            <FitR PageNum="23" Left="285" Bottom="207" Right="523" Top="404"/>
         </Dest>
         <Title>Document Subtitle</Title>
```

```
      </Bookmark>
      <Dest>
         <FitR PageNum="5" Left="285" Bottom="207" Right="523" Top="404"/>
      </Dest>
      <Title>Document Title</Title>
   </Bookmark>
</Bookmarks>
```

Bookmarks that launch views or applications

The following example shows a Bookmark document that represents the bookmark actions described in the following table.

Bookmark title	Level	Effect when clicked
Open a new document in an external file	1	The viewer application opens a new window in which it displays the PDF document located at `C:/adobe/livecycle/samples/assembler7/articles.pdf`
Launch WordPad	1	The viewer application launches the WordPad application, which is located at `C:/Program Files/Windows NT/Accessories/wordpad.exe` **Note:** Either the forward slash or backward slash separators can be used in paths.

Example: ***A basic bookmark expression that specifies launch actions***

```
<?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\livecycle\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.

```
<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.

```
<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

Name	Description
`Color`	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.

`Styles`	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.

Bookmarks

Root element of the Bookmarks XML language.

```
<Bookmarks
   <Bookmark> [0..n]
</Bookmarks>
```

Desc

A description of a file destination.

```
<Desc>
   xs:string
</Desc>
```

Can be contained in the File element.

Dest

The destination displayed when this item is activated.

```
<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.

```
<File
   FSType="xs:string"
   FSType_enc="xs:token"
   Name="xs:string"
   Name_enc="xs:token"
>
   <Desc> [0..1]
```

Can be contained in the GoToE, GoToR and Launch elements.

The PDF counterpart to this element is the file specification dictionary.

Attributes

Name	Description
`FSType`	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.
`FSType_enc`	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 338.)

`Name`	Required. Filename, using UNIX or DOS notation. The following values are equivalent:
	`C:/adobe/livecycle/samples/assembler7/MyFile.pdf`
	or
	`C:\adobe\livecycle\samples\assembler7\MyFile.pdf`

`Name_enc`	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 338.)

Fit

Specifies a page destination and viewer characteristics that fit the entire page into the viewer window.

```
<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

Name	Description
`PageNum`	Required. The page number of the destination, where 0 is the first page in the document.

FitB

Specifies a page destination and viewer characteristics that fit the page's bounding box into the viewer window.

```
<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

Name	Description
PageNum	Required. The page number of the destination, where 0 is the first page in the document.

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.

```
<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

Name	Description
PageNum	Required. The page number of the destination, where 0 is the first page in the document.
Top	Vertical coordinate in points that specifies the top of the viewed page.

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.

```
<FitBV
   PageNum="xs:nonNegativeInteger"
   Left="xsd:float"
/>
```

Can be contained in the Dest element.

When a user clicks a bookmark that uses the FitBV 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

Name	Description
PageNum	Required. The page number of the destination, where 0 is the first page in the document.
Left	Horizontal coordinate in points that specifies the left side of the viewed page.

FitH

Specifies a page destination and viewer characteristics that set the top of the page and horizontally fits the page into the viewer window.

```
<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

Name	Description
PageNum	Required. The page number of the destination, where 0 is the first page in the document.
Top	Vertical coordinate in points that specifies the top of the displayed portion of the page.

FitR

Specifies a page destination and viewer characteristics that fit a rectangle on the page into the viewer window.

```
<FitR
   PageNum="xs:nonNegativeInteger"
   Left="xsd:float"
   Bottom="xsd:float"
   Right="xsd:float"
```

```
   Top="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

Name	Description
PageNum	Required. The page number of the destination, where 0 is the first page in the document.
Left	The horizontal coordinate for the left side of the rectangle, in points.
Bottom	The vertical coordinate for the bottom of the rectangle, in points.
Right	The horizontal coordinate for the right side of the rectangle, in points.
Top	The vertical coordinate for the top of the rectangle, in points.

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.

```
<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

Name	Description
PageNum	Required. The page number of the destination, where 0 is the first page in the document.
Left	Horizontal coordinate in points that specifies the left side of the displayed portion of the page.

GoTo

Specifies a page destination in the document.

```
<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.

```
<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*

```
<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 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 Assembler service replaces the original package file with the modified package file (the one that you added the bookmark to).

Attributes

Name	Description
`NewWindow`	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.

GoToR

Specifies a destination in another document (go to remote).

```
<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

Name	Description
`NewWindow`	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.

Launch

Launches an application, opens a file, or prints a file.

```
<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 324.

Attributes

Name	Description
`NewWindow`	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.

Named

Execute an action predefined by the viewer application.

```
<Named
   Name="xs:string"
```

```
   Name_enc="xs:token"
/>
```

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

Name	Description
`Name`	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.
`Name_enc`	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 338.)

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 `GoToE` 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

Name	Description
`Relationship`	Required. Specifies whether the target document is a parent or child of the current document.
`Filename`	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 `PackageFiles` result or `FileAttachments` result element to obtain the `nameKey` for the destination file. When DDX processors embed a file, they typically use the filename as the starting point for determining the `nameKey`. They modify that name to ensure each `nameKey` is unique and to represent PDF Portfolio folders. For example, if the original filename is MyFile.pdf and there are other identically named files in the package, the Assembler service may assign `MyFile.001.pdf` as the `nameKey`. Beginning with LiveCycle 9, the value for this attribute has the format `<folder_index>nameKey`. It is not the whole path. If the file is not in a folder, omit `<folder_index>`. If the file is in a folder, the value of *`folder_index`* corresponds to the order in which the folder is defined in the PackageFile. Use escape encoding for the angle brackets. More specifically, use `&lt;` in place of the left angle bracket (<), and use `&gt;` in place of the right angle bracket (>). Consider a target file in the second folder defined in the PackageFile's `Folders` element where the nameKey is MyFile.001.pdf. In this case, use `Filename="&lt;2&gt;MyFile.001.pdf"`. **See also** "Bookmark xml that links to an embedded file in a folder" on page 331 "File Attachments Language" on page 345 "PackageFiles Language" on page 350
`Filename_enc`	Optional. The encoding used to represent the `Filename` value in the PDF document. If this attribute is omitted, the PDFDocEncoding character encoding is used. (See "Supported character encodings" on page 338.)
`PageNum`	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.
`AnnotName`	Optional. If the destination is a file-attachment annotation (page-level attachment), this attribute specifies the name of the destination annotation. Either the `AnnotName` attribute or the `AnnotNum` attribute must be provided to specify file-attachment annotation targets.
`AnnotNum`	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 `AnnotName` attribute or the `AnnotNum` attribute must be provided to specify file-attachment annotation targets.

Title

The text displayed on the screen for this bookmark entry.

`<Title>`

```
   Bookmark title
</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

Name	Description
`URI`	Required. The uniform resource identifier to resolve, encoded in 7-bit ASCII.
`IsMap`	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. **Note:** This attribute is not relevant to bookmarks.

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

Name	Description
`Name`	Required. The filename of the application to launch or the document to open or print. The name format uses standard Windows path format.
`Name_enc`	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 338.)
`Dir`	Optional. A string specifying the default directory in standard DOS syntax.
`Dir_enc`	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 338.)
`Action`	Optional. A string specifying the operation to perform: ? `open` - Open a document. ? `print` - Print a document. If the `Name` attribute designates an application instead of a document, this entry is ignored and the application is launched. Default value: `open`.
`Action_enc`	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 338.)
`Params`	Optional. A parameter string passed to the application designated by the `Name` attribute. This entry is ignored if `Name` designates a document. If `Params_enc` is omitted, encoding is assumed to be PDFDocEncoding or UTF-16, based on the characters encountered in the `Params` value.
`Params_enc`	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 338.)

XYZ

A destination that specifies positioning of a specific page in the viewer window.

```
<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

Name	Description
`PageNum`	Required. The page number for the destination, where 0 is the first page in the document.
`Left`	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.
`Top`	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.
`Zoom`	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 `Fit`, `FitB`, or `FitBH`.

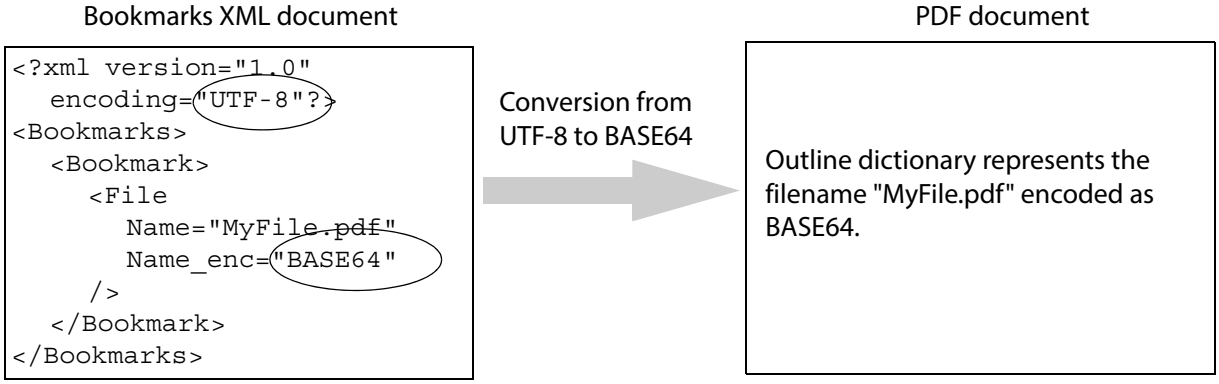
Supported character encodings

The Assembler service 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 Assembler service 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.

Encoding names	Description
`ASCII`	*ISO/IEC 8859-1:1998 Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1*, published by ISO (not available online)
`BASE64`	*The Base16, Base32, and Base64 Data Encodings*, RFC 3548 (http://ietf.org/rfc/rfc3548.txt)
`UTF-8`	*The Unicode Standard, Version 4.0*, (http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404)
`UTF-16`	*The Unicode Standard, Version 4.0*, (http://unicode.org/versions/Unicode4.0.0/ch03.pdf#G7404)
`PDFDocEncoding`	*PDF Reference, version 1.6*, (http://www.adobe.com/go/pdf_developer)

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.



24 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.

The Assembler service returns a DocText document in response to the `DocumentText` element appearing in the DDX document.

About the DocText XML language

The namespace of the DDX language is http://ns.adobe.com/DDX/DocText/1.0/, and the root element is `DocText`. The schema is installed in the product Documentation folder.

Example: *DocText document that provides words and word locations (long form)*

```
<?xml version="1.0" encoding="UTF-8"?>
<DocText xmlns="http://ns.adobe.com/DDX/DocText/1.0/">
   <WithQuads>
      <Page pageNumber="n">
         <Word>word1
            <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>
         </Word>
      </Page>
   </WithQuads>
</DocText>
```

Example: *DocText document that provides only words (short form)*

```
<?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*

```
<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>
```

```
          <Sentence>I sure would. </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.

```
<DocText>
   <WithQuads> or <ParagraphsPerPage> or <TextPerPage> [1]
</DocText>
```

Page

Lists the words that appear on a particular page.

```
<Page pageNumber="n">
   <Word/> [0..n, present only if a child of the WithQuads element]
   word1 word2 word3 [present only if a child of the TextPerPage element]
</Page>
```

Can be contained in the elements `WithQuads` or `TextPerPage`.

If the `Page` element is a child of the `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

Name	Description
`pageNumber`	Ordinal page number of a page in the document.

Paragraph

Sentences contained in a paragraph on the page.

```
<Paragraph>
   <Sentence> [0..n]
      xs:string
   </Sentence>
</Paragraph>
```

A sentence contains a string of words terminated by punctuation.

Can be contained in the `Page` element.

ParagraphsPerPage

Outer element containing paragraphs and sentence strings.

```
<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.

```
<P1
   x="X-coordinate"
   y="Y-coordinate"
</P1>
```

Can be contained in the Quad element.

Attributes

Name	Description
x	X-coordinate of one corner of a bounding box.
y	Y-coordinate of one corner of a bounding box.

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.

```
<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.

```
<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.

```
<WithQuads>
   <Page/> [0..n]
</WithQuads>
```

Can be contained in the DocText element.

Word

Describes the location of an individual word on the page.

```
<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.

25 File Attachments Language

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 Assembler service 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 Assembler service 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.

```
<Attachment
   attachmentKey="xs:string"
   name="xs:string"
>
   <File> [1]
   <Description> [0..1]
   <Page> [0..1]
</Attachment>
```

Can be contained in the `Attachments` element.

Attributes

Name	Description
`attachmentKey`	The contrived name associated with the output stream.
`name`	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.

Attachments

Describes some or all of the file attachments in a PDF document. This element is empty if no file attachments are extracted.

```
<Attachments>
   <Attachment> [1..n]
</Attachments>
```

This element is the root element for the Attachments XML.

Description

Provides the description for the file attachment.

```
<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

```
<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

Name	Description
mimeType	Optional. MIME type of the file. If this information is unknown, the attribute is absent.
size	Optional. Size of the file, in bytes. If this information is unknown, the attribute is absent.
creationDate	Optional. Creation date of the file. If this value is unknown, the attribute is absent.
modificationDate	Optional. Date the file was last modified. If this value is unknown, the attribute is absent.

FileName

Filename and the success of decoding that name from the source PDF document.

```
<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

Name	Description
`unmappableCharacters`	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 with a value of `true`. In addition, the following changes occur: ? Unmappable characters are replaced with the Unicode substitution character (\uFFFD). ? `success` attribute is set to `false`. ? Modified filename text is displayed. If unmappable characters are not found, this attribute is absent.
`fromEncoding`	Encoding applied to produce the string content of the `FileName` element.
`success`	Flag indicating the success of the attempt to decode the filename. This attribute can have the following values: `false` - Decoding was unsuccessful. If the `unmappableCharacters` attribute is present, some characters in the filename could not be decoded. If that attribute is absent, the decoding attempt failed. `true` - Decoding was successful.

Location

Specifies a position on a page as coordinates.

```
<Location
   x="length"
   y="length"
/>
```

Can be contained in the `Page` element.

Attributes

Name	Description
`x`	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.
`y`	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.

Page

Specifies the page on which a page-level file attachment occurs and the position on that page where the annotation is placed.

```
<Page
   pageNumber="xs:integer"
>
   <Location> [0..1]
</Page>
```

Can be contained in the Attachment element.

Attributes

Name	Description
pageNumber	The number of the page in the PDF document to which the file was attached.

26 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 Assembler service returns a PackageFiles document in response to the appearance of the `PackageFiles` result element in a DDX document.

Example: *Resultant PackageFiles*

```
<?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"
```

```
      modificationDate="2009-07-16T10:44:20-08:00"
      size="6508">
      <Filename>chap1.pdf</Filename>
    </File>
  </PackageFile>
  <PackageFile attachmentKey="SimpleFolders.pdf_attach.0000.0002"
    nameKey="/Chapters/Chapter 2/chap2.pdf">
    <File creationDate="2009-08-03T14:05:18-07:00"
      mimeType="application/pdf"
      modificationDate="2009-07-16T10:44:20-08:00" size="12942">
      <Filename>chap2.pdf</Filename>
    </File>
    <Description>File chap2.pdf</Description>
  </PackageFile>
  <PackageFile attachmentKey="SimpleFolders.pdf_attach.0000.0003"
    nameKey="/Biodynamics/biodynamic.pdf">
    <File creationDate="2009-08-03T14:06:00-07:00"
      mimeType="application/pdf"
      modificationDate="2009-07-16T10:44:26-08:00" size="9339">
      <Filename>biodynamic.pdf</Filename>
    </File>
  </PackageFile>
</PackageFiles>
```

PackageFiles reference

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

Name	Description
`name`	Required. The normalized name of a `FieldData` as defined in the `Schema`. The non-normalized name appears as the content in the `Schema` element.

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.

```
<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`, o r`PackageFile` elements.

Attributes

Name	Description
`name`	Required. The normalized name of a `Field` as defined in the `Schema`. The non-normalized name appears as the content in the `Schema` element.
`type`	Required. The type of metadata stored in the `Field` according to the `Schema`.

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.

```
<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

Name	Description
`creationDate`	Optional. The creation date of the file, if known. If not known, this attribute is absent. The default for import is the current date.
`mimeType`	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.
`modificationDate`	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.
`size`	Optional. The size of the file in bytes. The size is calculated as the number of actual bytes in the data stream.

Filename Attributes

Name	Description
`fromEncoding`	Required. The encoding applied to produce the string content of the `Filename` element.
`success`	Optional. A flag indicating whether the encoding was successfully applied to the filename. A `success` value of `"false"`, 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.
`unmappableCharacters`	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 `"true"`. Additionally, the unmappable characters are replaced with the Unicode substitution character (`\uFFFD`), the success attribute is set to `"false"`, and the modified filename text is displayed. If unmappable characters are not found, this attribute omitted from the XML.

Folders

(Since 9.0) Root in the folder structure for the PDF package.

```
<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

Name	Description
name	Required. Folder name. The name must conform to the encoding specified in the DDX FilenameEncoding element.

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.

```
<Package>
   <Schema>[0..1]
   <DisplayOrder>[0..1]
   <SortOrder>[0..1]
</Package>
```

PackageFile

Describes a package file selected from the PDF document.

```
<PackageFile
   attachmentKey= "xs:string"
   nameKey= "xs:string"
   required= "false" or "true"
>
   <File>[1]
   <Description>[0..1]
   <FieldData>[0..n]
</PackageFile>
```

Attributes

Name	Description

`attachmentKey`	Required. The contrived name associated with the output stream for the extracted attachment.
	For document-level attachments, the syntax is
	`documentName` `+ "_attach.0000." +` *`nnnn`*
	The *`documentName`* is the value of the PDF source attribute from which the package files are being extracted. The *`nnnn`* is a sequence number for the package file.
	For example, if PDF source `"docA"` has two package files, the `attachmentKey` values will be:
	`"docA_attach.0000.0001"`
	`"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 Assembler 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.

```
<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.

```
<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

Name	Description
editable	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.
name	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.
type	Required. The type identifies the type of data that is stored in this FieldData.
visible	Optional. The initial visibility of the field in the PDF viewer. At least one FieldData must be specified as visible.

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

Name	Description
ascending	Optional. When sorting with this field, this attribute specifies whether to sort in ascending ("true") or descending ("false") order.
name	Required. The normalized name of a FieldData as defined in the Schema. The non-normalized name appears as the content in the Schema element.

Part IV: Special Topics

More advanced settings let you coordinate between DDX expressions, system settings, and LiveCycle ES4 configuration settings.

27 Handling Out of Memory Problems

Assembling and applying content to large PDF documents can use so much memory that the Assembler service 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 LiveCycle ES4 system to fail. Do not adjust this setting when your system is in a production mode.

Operation checkpoints (DDXProcessorSetting)

Beginning with LiveCycle 8.0.1, 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 Assembler service 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:

```
<DDXProcessorSetting name="checkpoint" value="positive-integer"/>
```

It can appear as a child of the `DDX` or `PDF` result elements.

In the Assembler service, the only supported value for the `name` attribute is `checkpoint`. The checkpoint setting is a unit-less hint to the Assembler service. 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 LiveCycle ES4 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.

```
<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"><p>Adopted!</p></StyledText>
         </Watermark>
      </PDF>
      <PDF source="summary" bookmarkTitle="Summary" required="false" />

   </PDF>

</DDX>
```

Include this expression in the DDX document for your assembly workflow.