

Adobe XML Architecture

Data Handling Specification

Version 2.0

Adobe Systems Incorporated

October 2003

© 2003 Adobe Systems Incorporated. All rights reserved.

This publication and the information herein are furnished AS IS, are subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third-party rights

This page is left intentionally blank.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated.

Any references to company names in the specifications are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

POSIX is a registered certification mark of the Institute of Electrical and Electronic Engineers. Unicode is a registered trademark of Unicode, Inc.

All other trademarks are the property of their respective owners.

This limited right of use does not include the right to copy other copyrighted material from Adobe, or the software in any of Adobe's products that use the Portable Document Format, in whole or in part, nor does it include the right to use any Adobe patents, except as may be permitted by an official Adobe Patent Clarification Notice (see the Bibliography).

Intellectual Property

Adobe will enforce its intellectual property rights. Adobe's intention is to maintain the integrity of the Adobe XML Architecture standard. This enables the public to distinguish between the Adobe XML Architecture and other interchange formats for electronic documents, transactions and information. However, Adobe desires to promote the use of the Adobe XML Architecture for information interchange among diverse products and applications. Accordingly, Adobe gives anyone permission to use Adobe's intellectual property, subject to the conditions stated below, to:

- Prepare files whose content conforms to the Adobe XML Architecture
- Write drivers and applications that produce output represented in the Adobe XML Architecture
- Write software that accepts input in the form of the Adobe XML Architecture specifications and displays, prints, or otherwise interprets the contents
- Copy Adobe's intellectual property, as well as the example code to the extent necessary to use the Adobe XML Architecture for the purposes above

The condition of such intellectual property usage is:

- Anyone who uses the Adobe intellectual property, as stated above, must include the appropriate intellectual property and patent notices.

This limited right to use the example code in this document does not include the right to use other intellectual property from Adobe, or the software in any of Adobe's products that use the Adobe XML Architecture, in whole or in part, nor does it include the right to use any Adobe patents, except as may be permitted by an official Adobe Patent Clarification Notice (see the Bibliography).

Adobe, the Adobe logo, and Acrobat are either trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries. Nothing in this document is intended to grant you any right to use these trademarks for any purpose.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

This page is left intentionally blank.

Table of Contents

- 1 Introduction 1
 - 1.1 Notation..... 1
 - 1.2 Terminology 1
 - 1.3 Definitions 2
 - 1.4 Background and Goals 3
- 2 Basic Concepts..... 4
 - 2.1 XFA Data Document Object Model..... 4
 - 2.2 Data-Values 5
 - 2.3 Data-Groups..... 7
 - 2.4 Relationship Between the XFA-data-DOM and the XML-data-DOM..... 8
 - 2.5 Tree Notation 9
- 3 Default-Mapping-Rules 11
 - 3.1 Document Range 11
 - 3.1.1 XML Logical Structures 11
 - 3.1.2 Start Element 12
 - 3.1.3 Namespaces 13
 - 3.1.4 Record Elements..... 16
 - 3.2 Rich-Text 18
 - 3.3 Data-Value Elements 20
 - 3.3.1 Character-Data Data-Values..... 20
 - 3.3.2 Mixed-Content Data-Values..... 20
 - 3.3.3 Empty-Element Data-Values..... 21
 - 3.3.4 Element-Content Data-Values..... 22
 - 3.4 Data-Group Elements..... 23
 - 3.5 Attributes..... 24
 - 3.6 White Space Handling..... 24
 - 3.6.1 White Space in Data-Groups 24
 - 3.6.2 White Space in Data-Values..... 25
- 4 Extended Mapping Rules 26
 - 4.1 Document Range 27
 - 4.2 Transforms 27
 - 4.3 XSLT Preprocessing 29
 - 4.4 Exclude Namespaces..... 29
 - 4.5 Nominate Start Element..... 33
 - 4.6 Nominate Record Elements 35
 - 4.7 Map Attributes to Data-Values..... 43
 - 4.7.1 Data-Group Elements with Attributes..... 43
 - 4.7.2 Data-Value Elements with Attributes 44
 - 4.8 Exclude Records By Position 46
 - 4.9 Simplify Structure Based on Element Name 47
 - 4.10 Trim White Space..... 50
 - 4.11 Simplify Structure Based on Empty Content 51
 - 4.12 Force Data-Value or Data-Group Mapping..... 55
 - 4.13 Replace Element Names with Attribute Values 57
 - 4.14 Rename Nodes Based On Element Name..... 59

4.15 XSLT Postprocessing..... 60
5 Update Processing 61
6 Unload Processing 62
 6.1 Unloading Node Type Information 62
7 Bibliography..... 64

1 Introduction

This specification specifies how generic well-formed XML documents are processed by XFA processing applications, and presents an object and processing model for interpreting and manipulating the structured data contained within the XML document. This specification refers to such XML documents as XML-data-documents.

Within this specification “XFA” refers to the XML Forms Architecture of which this specification is a part. XFA is an application of [\[XML\]](#) for modeling electronic forms and related processes. XFA provides for the specific needs of electronic forms and the processing applications that use them.

1.1 Notation

Character codes are given using the notation described in the preface to The Unicode Standard, Version 3.0 [\[Unicode Preface\]](#), p. xxvii. Character names are as given in the Unicode character tables.

1.2 Terminology

The following normative terms may be present in this specification. These terms extend the definitions in [\[RFC2119\]](#) in ways based upon similar definitions in ISO/IEC 9945-1:1990 [\[POSIX.1\]](#):

by-default

The term “by-default” is to be interpreted as a value or behavior automatically used in the absence of a choice made by the user.

implementation-defined

The term “implementation-defined” is to be interpreted as delegating to the implementation the definition and documentation of the corresponding requirements for correct processing.

may

The term “may” is to be interpreted as an optional feature or permissible behavior. The feature or behavior is not required by this specification but can legitimately be provided by an implementation. For example, one implementation might choose to include a feature because a particular marketplace requires it or because it enhances the product; another implementation might omit the same feature.

must

The term “must” is to be interpreted as a mandatory requirement on the implementation.

must not

The term “must not” is to be interpreted as a mandatory prohibition on the implementation.

should

recommended

The terms “should” or “recommended” are to be interpreted as an implementation suggestion, but not a requirement.

There might exist valid reasons for a particular implementation not to implement this behavior, but the full implications ought to be understood and carefully considered before deciding not to implement it.

should not

not recommended

The terms “should not” or “not recommended” are to be interpreted as an implementation suggestion, but not a requirement.

There might exist valid reasons for a particular implementation to implement this behavior, but the full implications ought to be understood and carefully considered before implementing it.

1.3 Definitions

character-data

All text within an XML document that is not markup constitutes character-data. See the description of character-data within section “2.4 Character Data and Markup” of the [\[XML\]](#) specification.

data-group

A data-group is an object in the XFA-data-DOM that corresponds to an element holding other elements (as opposed to character-data) in an XML-data-document. Within the XFA-data-DOM interior nodes are usually data-groups. A data-group may have other data-groups and/or data-values descended from it.

data-loader

A program or subsystem responsible for loading data from an XML-data-document into the XFA-data-DOM.

data-unloader

A program or subsystem responsible for unloading data from the XFA-data-DOM into a new XML-data-document.

data-value

A data-value is an object in the XFA-data-DOM that corresponds to an element holding character-data (and possibly other elements) in an XML-data-document. Within the XFA-data-DOM leaf nodes are usually data-values. A data-value may have other data-values descended from it but it must not have any data-group descended from it.

default-mapping-rule

A rule that governs, by-default, how an XML-data-document is mapped to an XFA-data-DOM.

document-range

The section(s) of the XML-data-document that is/are loaded into the XFA-data-DOM.

element-content

XML elements that contain only XML child elements, optionally separated with white space, constitute “element content”. See the description within section “3.2.1 Element Content” of the [\[XML\]](#) specification.

element-type

The first token within an XML start or end tag identifies the “element type”. The element-type is a string containing a qualified name. A qualified name consists of an optional namespace prefix and colon, followed by a mandatory local name. See the description within section “3. Qualified Names” of the [\[XMLNAMES\]](#) specification.

empty-element

An XML element that does not enclose any content.

extended-mapping-rule

A rule that is not in effect by-default but is available as an override or extension to the default-mapping rules.

form-template

A form-template is a collection of related subforms and optional business logic, constraints, and processing rules.

metadata

In this specification, “metadata” refers to data expressed via XML attributes.

mixed-content

XML elements that contain character-data interspersed with XML child elements constitute “mixed-content”. See the description of mixed-content within section “3.2.2 Mixed Content” of the [\[XML\]](#) specification.

plain-text

Text that does not contain any markup signifying formatting, hence, text that is not rich-text.

rich-text

Text containing markup signifying formatting such as bold and underline.

XFA

Within this specification “XFA” refers to the XML Forms Architecture of which this specification is a part.

XFA-configuration-DOM

The “XFA-configuration-DOM” provides a set of software interfaces to the data obtained from an XFA configuration document. The “XFA-configuration-DOM” includes sections for all of the different components of XFA, including a section for the data-loader.

XFA-data-DOM

The “XFA-data-DOM” provides a set of software interfaces to the data loaded from an XML-data-document. The data in the “XFA-data-DOM” is in general a subset of the data in the XML-data-document, but it may also contain other data not present in the XML-data-document as well as data that originated in the XML-data-document but has been modified.

XML-data-document

An “XML-data-document” is an XML document intended to be processed as data in the context of a form or workflow processing application, such as displaying or printing the data with a form, or manipulating the data via a workflow process.

XML-data-DOM

An “XML-data-DOM” provides a set of software interfaces to the data in an XML-data-document.

1.4 Background and Goals

This specification originated from the requirement to define a common set of behavior for XFA processing applications that consume or produce data.

For consistency and predictability of XFA processing applications, it is important that all XFA processing applications handle data according to well-defined rules and a consistent model of the data.

The reader of this specification will learn how XML-data-documents are handled by XFA processing applications, how the data is mapped to an object model known as the XFA-data-DOM, and how the data is mapped back out again during the creation of a new XML-data-document. This information is valuable to authors of form-templates who will be processing existing XML data, and to people or systems producing data that will be serviced by XFA processing applications.

This specification primarily focuses on the processing of XML-data-documents; however, it was a design goal that the same object model could also be used to represent data from non-XML sources such as a database or other data formats such as comma-separated values.

It should be noted that the concepts and guidelines presented by this specification are also valuable to other types of XML processing applications outside the realm of XFA, such as applications concerned with utilizing XML for data exchange and mapping the data into structured documents types other than forms. A reduction in implementation costs and a gain in flexibility can be obtained by taking a simplified view of how XML expresses data. The object model described by this specification represents such a simplified view.

An important philosophy underlying this specification is to place as few requirements as possible on the data. One of the strengths of XML is that it provides a comprehensible, and often self-evident representation of data. Data expressed in XML can be manipulated via off-the-shelf, and often freely available, processing tools; in the worst case the data can be manipulated using a common text editor. Hence, it would be contrary to this inherent property of XML to require that data conform to a specific grammar or schema before it can be used within an XFA processing application.

2 Basic Concepts

2.1 XFA Data Document Object Model

The interpretation of an XML-data-document is a mapping operation of the data into an object model known as the “XFA Data Document Object Model”, commonly referred to as the XFA-data-DOM. The behavior of the mapping is governed by rules described by this specification. The software component responsible for performing the mapping of the XML-data-document into the XFA-data-DOM is referred to in this specification as the data-loader. There is in addition a reverse function that maps from the XFA-data-DOM to a new XML-data-document. The software component responsible for this reverse mapping is referred to in this specification as the data-unloader.

The XFA-data-DOM provides a set of software interfaces to the data mapped from an XML-data-document. In principle, the same model could also be used to represent data from non-XML sources such as a database or other data formats such as comma-separated values. The XFA-data-DOM provides interfaces that are simpler than a generic XML Document Object Model [[XMLDOM2](#)]. There are fewer interfaces in the XFA-data-DOM as compared to the XML DOM, and many of the physical structures of XML are abstracted away. Notwithstanding the wider applicability of the XFA-data-DOM, this specification assumes that the XML-data-document is first loaded into an XML-data-DOM and from there into the XFA-data-DOM.

The XFA-data-DOM encloses a tree structure in which each node is an object of type “data-value” or “data-group”. Each object corresponds to a single XML element and is peered with a single node in the XML-data-DOM. Parent-child relationships correspond to element nesting relationships, that is, the element corresponding to the parent of any given node contains the element corresponding to the given node. In addition the children of any node are ordered by age, that is the first child acquired by the node is the eldest, the next child acquired is the second-eldest and so on. In XFA specifications when a tree is drawn pictorially sibling nodes are shown in order by age, with the oldest at the left and the youngest at the right. In the case of the XFA-data-DOM the result of this ordering is that a tree walk that goes in depth-first order, and left-to-right at any level, traverses the data in the same order as it was present in the original XML-data-document.

The objects in the XFA-data-DOM and their properties are exposed via the XFA-data-DOM interfaces. Every object in the XFA-data-DOM has the following exposed properties:

“name” property

A string of any length (including zero-length, that is empty) which is a non-unique identifier for the object.

This corresponds to the local part of either an element type or an attribute name in the XML-data-document.

“namePrefix” property

The namespace prefix that goes with “name”, if any. This is also a string of any length, including zero.

“xmlns” property

The URI to which “name-prefix” resolves. This is also a string of any length. This may be non-empty even when “name-prefix” is empty, because the default namespace is inherited in accordance with the rules defined in XML Namespaces [[XMLNAMES](#)].

In addition, each object that is created by the data-loader has an internal pointer to the node in the XML-data-DOM with which it is peered. Furthermore, some objects have additional properties appropriate to their types, as described below.

For example, consider the following fragment of XML:

```
<abc:tree xmlns:abc="http://www.example.org/orchard/">apple</abc:tree>
```

When loaded into the XFA-data-DOM using default-mapping-rules, the object representing this data has among other properties a “name” of “tree”, a “name-prefix” of “abc”, and an “xmlns” of “http://www.example.org/orchard”.

After the XFA-data-DOM has been loaded the XFA application may update it. Updates may include adding, deleting, moving, and changing the properties of nodes. These changes are passed through to the XML-data-DOM by the XFA-data-DOM so that the two data DOMs stay synchronized. When the data-unloader runs it creates the new XML-data-document based upon the contents of the XML-data-DOM, as updated by the application.

Note that the exposed properties may be set by an XFA application to any Unicode string, including the empty string “”. This allows XFA applications to construct arbitrary data structures. However the XML 1.0 Specification [\[XML\]](#) imposes additional restrictions upon element types, namespace prefixes, and URIs. Hence when the XFA-data-DOM is unloaded to an XML-data-document the result may be malformed XML. It is up to the application to ensure, if desired, that the restrictions of XML with regard to element types, namespace prefixes and URIs are respected.

The XFA-data-DOM is part of a larger tree that holds all exposed XFA objects. The single large tree makes it possible to refer to XFA objects using a unified format known as a Scripting Object Model (SOM) expression. The grammar of SOM expressions is described in “XFA Scripting Object Model 2.0 Specification” [\[SOM\]](#). Briefly, an expression consists of a sequence of node names separated by periods (“.” characters). Starting from some point in the XFA tree, each name identifies which child of the current node to descend to. The Data DOM descends from a node named “data” which is a child of “datasets”, which is a child of “xfa”, which is the root. The XML-data-document does not supply the “xfa”, “datasets”, or “data” nodes; instead the data-loader must create them automatically and make the node mapped to the outermost element of the data the child of “data”. Consider the following XML-data-document:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
</book>
```

When loaded into the XFA-data-DOM, the node representing the “book” element would be referenced by the SOM expression “xfa.datasets.data.book”. The “ISBN” element would be referenced by “xfa.datasets.data.book.ISBN” and the “title” element by “xfa.datasets.data.book.title”.

Also, SOM expressions recognize the short-form “!” as equivalent to “xfa.datasets.” and “\$data” as equivalent to “xfa.datasets.data”. Thus for example the “title” element above could also be referenced as either “!data.book.title” or “\$data.book.title”.

2.2 Data-Values

A data-value is an object in the XFA-data-DOM that corresponds to an element holding character-data (and possibly other elements) in an XML-data-document. Within the XFA-data-DOM leaf nodes are usually data-values. A data-value may have other data-values descended from it but it must not have any data-group descended from it.

The following properties are unique to data-value objects:

“value” property

A string of Unicode characters holding the data associated with the object. The string may be of any length (including zero-length, that is empty). The string must not include the character code NUL (U0000). Hence, NUL may be used as the string terminator.

“contentType” property

A string identifying the type of the data. By-default this must be set to “text/plain”. Note that this specification is more relaxed than [\[RFC2046\]](#), which sets out the requirements for the MIMEtype “text/plain”, in that the data-loader may signify a line break by a newline character (U000A) without an accompanying carriage-return character (U000D).

“contains” property

A string identifying the source of the data. The string is set to “metadata” if the “value” property originated from an XML attribute, but to “data” if it did not.

In the following fragment of XML, the elements “ISBN”, “title”, “desc” and “keyword” represent data-values.

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <desc>Basic primer on <keyword>XML</keyword> technology.</desc>
</book>
```

In the above example, the element ISBN represents a data-value with a “name” property of ISBN, a “value” property of “15536455”, and a “contains” property of “data”.

When loading the data value from the XML-data-document, the data-loader must remove XML escaping, so that for example “<” is represented literally in the value string rather than by the XML escape sequence “<”. The XML-data-document must not contain character-data containing either a NUL character (U0000) or any escape sequence that evaluates to zero, for example “�”.

Data-values are permitted to be descended from other data-values in order to represent XML mixed-content. The element “keyword” is represented in the XFA-data-DOM by data-value descended from the data-value representing the element “desc”. A common requirement is the ability to ask for the value of a data-value and receive the result of combining any descended data-values. For instance, in the case of the “desc” data-value above, the value is “Basic primer on XML technology.”; the “XML” portion of the value is contributed by the “keyword” data-value to its parent “desc” data-value. The resulting data-values have the properties:

Name	Value	Contains
“ISBN”	“15536455”	“data”
“title”	“Introduction to XML”	“data”
“desc”	“Basic primer on XML technology.”	“data”
“keyword”	“XML”	“data”

XML attributes are also candidates for consideration as data-values as described in section “Map Attributes to Data-Values”. Data-values that resulted from mapping of XML attributes are recognized as a different flavor of data-value; such data-values resulting from attributes for the purpose of this specification are said to represent “metadata”. One benefit of this approach is the potential to place these data-values back into attributes when unloading the XFA-data-DOM into an XML-data-document.

Data-values that are considered to contain metadata are excluded from the value of any ancestor data-value. Consider the following XML-data-document that extends the previous example with the addition of a “language” attribute on the “desc” element:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <desc language="en-US">Basic primer on <keyword>XML</keyword> technology.</desc>
</book>
```

In the above example the value of “desc” is “Basic primer on XML technology.” regardless of the presence of the “language” attribute and its corresponding data-value. The value of the data-value element “keyword” contributes to the value of “desc”, but the data-value resulting from the attribute “language” does not contribute. Hence the data-values resulting from this example (assuming attributes are being loaded) have the properties listed in the following table. The only difference from the preceding example is the addition of a new data-value representing the attribute.

Name	Value	Contains
“ISBN”	“15536455”	“data”
“title”	“Introduction to XML”	“data”
“desc”	“Basic primer on XML technology.”	“data”
“keyword”	“XML”	“data”
“language”	“en-US”	“metadata”

2.3 Data-Groups

A data-group is an object in the XFA-data-DOM that corresponds to element holding other elements (as opposed to character-data) in an XML-data-document. Within the XFA-data-DOM interior nodes are usually data-groups. A data-group may have other data-groups and/or data-values descended from it.

Some XML-data-documents enclose repeating groups of data, each with different content. This specification refers to these repeating groups of data as “records”. Typically a record holds the data from a single form instance. The outermost element of a record must map to a data-group.

The following property is unique to data-group objects:

isRecord

A Boolean variable which must be true if and only if the data-group represents a record (that is if the nodes descending from it represent all the content of a single record).

In the following fragment of XML the element “book” represents a data-group containing data-values “ISBN”, “title”, “desc”, and “keyword”.

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <desc>Basic primer on <keyword>XML</keyword> technology.</desc>
</book>
```

In the above example the element “book” represents a data-group with a “name” property of “book”, a “namePrefix” of an empty string, and a “namespace” set to the default namespace it inherited. “isRecord” may be true or false depending on the context within the XML-data-document and depending on the configuration option described in “Nominate Record Elements”. If the above text was the entire XML-data-document and default-mapping-rules were used “isRecord” would be true.

2.4 Relationship Between the XFA-data-DOM and the XML-data-DOM

After the XFA-data-DOM has been loaded the XFA application may update it. Updates may include adding, deleting, moving, and changing the properties of nodes. These changes are passed through to the XML-data-DOM by the XFA-data-DOM so that the two data DOMs stay synchronized. When the data-unloader runs it creates the new XML-data-document based upon the contents of the XML-data-DOM, as updated by the application.

Note that the exposed properties may be set by an XFA application to any Unicode string, including the empty string “”. This allows XFA applications to construct arbitrary data structures. However the XML 1.0 Specification [[XML](#)] imposes additional restrictions upon element types, namespace prefixes, and URIs. Hence when the XFA-data-DOM is unloaded to an XML-data-document the result may be malformed XML. It is up to the application to ensure, if desired, that the restrictions of XML with regard to element types, namespace prefixes and URIs are respected.

Some data-loader options cause the data-loader to load only a subset of data from the XML-data-DOM into the XFA-data-DOM. The result is that the ignored data is still in the XML-data-DOM but is not accessible through the XFA-data-DOM. Consequently the XFA application is unable to alter the ignored data. When the data-unloader writes out a new XML-data-document, since the ignored data has been kept untouched in the XML-data-DOM, it is written out in the new document without significant changes. This applies by-default to attributes. It also applies to data which is always excluded from the document-range as described in “[Default-Mapping-Rules, Document Range](#)”. It applies to data which is excluded using the extended mapping rules described in “[Exclude Namespaces](#)”, “[Nominate Start Element](#)”, and “[Exclude Records by Position](#)”. Finally, it applies to all extended mapping rules invoked by the “ignore” keyword, as described in “[Map Attributes to Data-Values](#)”, “[Simplify Structure Based on Element Name](#)”, and “[Simplify Structure Based on Empty Content](#)”.

The “name” property of a node in the XFA-data-DOM may be altered by the application or by the data-loader, but doing so does not affect the “name” property of the peered node in the XML-data-DOM. Consequently the node in the XFA-data-DOM is accessible to scripts under its new name, but when the peered node is written out from the XML-data-DOM to a new XML-data-document it retains its original name. This applies to nodes renamed via the data-loading options described in “[Replace Element Names with Attribute Values](#)” and “[Rename Nodes Based on Element Name](#)”. On the other hand when the application creates a new node in the XFA-data-DOM, there is no existing peer in the XML-data-DOM, so one is created and given the same “name” property.

The remaining data-loader options cause the data-loader to alter the content of the XML-data-DOM along with the XFA-data-DOM. When the data is unloaded the alteration is reflected in the new XML-data-document. This applies to extended mapping rules described in “[XSLT Preprocessing](#)”, “[Trim White Space](#)” and “[XSLT Postprocessing](#)”. It also applies to the rules invoked via the “remove”, “dissolve” or “dissolveStructure” keywords as described in “[Simplify Structure Based on Element Name](#)” and “[Simplify Structure Based on Empty Content](#)”.

Other updates by the XFA application carry through to the XML-data-DOM. When updating any property other than “name”, and when deleting, inserting, or moving a node in response to a request from the XFA application, the XFA-data-DOM must propagate the update to the XML-data-DOM..

Moving a node can cause a name conflict if, for example, a data-value containing an attribute (“contains” set to “metadata”) is moved to a location where there is already a peer representing an attribute with the same name. A similar situation can arise from a request to create a new node. Carrying out such a request would result in a structure that violates the linkage rules of the XML-data-DOM. The XML-data-DOM may refuse to carry out such a request and the XFA-data-DOM must in response return an error code to the XFA application and leave the XFA-data-DOM in a state consistent with the XML-data-DOM..

2.5 Tree Notation

To facilitate discussion of the XFA-data-DOM, this specification uses a particular notation when illustrating the contents of an XFA-data-DOM in examples, as described below.

Data-groups are expressed in the following form:

```
[DataGroup (name)]
```

where “name” represents the “name” property of the data-group.

Data-values are expressed in the following form:

```
[DataValue (name) = "value"]
```

where “name” represents the “name” property of the data-value, and “value” represents the “value” property.

The “contains” property of a data-value must be assumed to have a value of “data” when not explicitly shown in the notation within this specification; it will only be expressed in this notation when it has a value of “metadata”, as described by the following two examples:

```
[DataValue (ISBN) = "15536455"]
```

In the above example the data-value ISBN has a “value” property of “15536455” and, although it isn't explicitly stated by the notation, it also has a “contains” property of “data”.

```
[DataValue (status) = "stocked" contains="metadata"]
```

In the above example, the data-value “status” has a “value” property of “stocked” and a “contains” property of “metadata”.

Similarly, within this specification the “contentType” property of a data-value must be assumed to have a value of “text/plain” when not explicitly shown. Likewise the “isRecord” property of a data-group must be assumed to have a value of “false” when not explicitly shown.

In some implementations the “value” property of a data-value may take a null value, as distinct from an empty string. This is shown by an unquoted value consisting of the word “null”, as shown below:

```
[DataValue (ISBN) = null]
```

Indenting is used to show parent-child relationships between objects. In the following example a data-group named “book” is the parent of a data-valuenamed ISBN:

```
[DataGroup (book)]  
  [DataValue (ISBN) = "15536455"]
```

Within a group of sibling nodes, the age relationship is shown by the vertical ordering. The eldest child is at the top, the youngest at the bottom. In the following example “ISBN” is the eldest child of “book”, “title” is the middle child, and “author” is the youngest. Between the children of “author”, “firstname” is the older and “lastname” the younger.

```
[DataGroup (book)]  
  [DataValue (ISBN) = "15536455"]  
  [DataValue (title) = "Introduction to XML"]  
  [DataGroup (author)]  
    [DataValue (firstname) = "Charles"]  
    [DataValue (lastname) = "Porter"]
```

In some complex XML-data-documents the elements that correspond to data-groups or data-values may be annotated with an XML namespace [[XMLNAMES](#)]; in these cases, the namespace is expressed in the notation as follows:

```
[DataGroup (prefix:book) xmlns="uri"]  
  [DataValue (prefix:ISBN) = "15536455" xmlns="uri"]
```

In order to not clutter each example with namespace information, only examples that depend upon namespace information will include this form of the notation. The *prefix* refers to a namespace prefix as described by [[XMLNAMES](#)]. The *uri* refers to the corresponding Uniform Resource Identifier as described by [[URI](#)].

It is worth repeating that the above notation is provided only as means within this specification to depict the structure and content within an XFA-data-DOM.

3 Default-Mapping-Rules

There is a set of rules that govern, by-default, how an XML-data-document is mapped to an XFA-data-DOM and vice-versa. These rules have the effect of directing the data-loader to usefully interpret the XML document content by mapping the physical structures of an XML-data-document into data-values and data-groups. In addition they direct the data-unloader to map the data-values and data-groups into a new XML-data-document.

3.1 Document Range

The term document-range refers to the portion of the XML-data-document that must be processed by the data-loader, such as the whole XML-data-document or a fragment. Assuming the XML-data-document starts as a file containing serialized XML, the first step in processing is to load the entire content of the XML-data-document into an XML-data-DOM. The portion of the XML-data-document corresponding to the document-range is then loaded into and accessible from the XFA-data-DOM, and any portions of the data that are outside of the document-range are not accessible via the XFA-data-DOM. When the data-unloader creates a new XML-data-document it stitches together the data within the XFA-data-DOM and the data excluded from the XFA-data-DOM (but still in the XML-data-DOM) to create a new serialized XML file.

The document-range can be influenced via the use of a number of extended-mapping-rules, however there is a by-default document-range which is described in more detail in the following subsections.

The document-range is the portion of content corresponding to the intersecting result of applying the rules described by the following sections, in order. In other words, the mechanisms described by this section (and in the corresponding sections within the “[Extended-Mapping-Rules](#)”) each excludes more content from the document-range. The order that the data-loader applies the rules must be as follows:

1. [XML Logical Structures](#): Constrain the document-range to a limited set of XML logical structures.
2. [Start Element](#): Additionally constrain the document-range to the document content within a specified element.
3. [Namespaces](#): Additionally constrain the document-range to a set of content belonging to a set of XML namespaces.
4. [Record Elements](#): Partition the document-range into “records” enclosed within specified elements.

3.1.1 XML Logical Structures

An XML-data-document is comprised of a number of physical and logical structures (see the [XML](#) specification for more information on logical and physical structures).

The document-range is constrained to a range of document content not greater than the content that corresponds to the following logical structures:

- elements with character-data
- elements with element-content
- elements with mixed-content
- empty-elements
- attributes

This means that XML processing instructions, for example, are not included in the document-range.

3.1.2 Start Element

The notion of a “start element” represents the element where the XFA data handling processing begins, and consequently the end of the element determines where processing ends; the start element is a way to specify that a particular element within the XML-data-document actually represents the “root” of the document.

By-default the start element corresponds to the root element (also known as the document element) of the XML-data-document (see section “[2.1 Well-Formed XML Documents](#)” of the [\[XML\]](#) specification for more information on the root element). Therefore, the data-loader must by-default map the content of the document beginning with the root element inclusively.

The document-range is constrained to a range of document content not greater than that specified by the start element.

Consider the following XML-data-document:

```
<order>
  <number>1</number>
  <shipto>
    <reference><customer>c001</customer></reference>
  </shipto>
  <contact>Tim Bell</contact>
  <date><day>14</day><month>11</month>
    <year>1998</year></date>
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title>Introduction to XML</title>
      <author>
        <firstname>Charles</firstname>
        <lastname>Porter</lastname>
      </author>
      <quantity>1</quantity>
      <unitprice>25.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <item>
    <book>
      <ISBN>15536456</ISBN>
      <title>XML Power</title>
      <author>
        <firstname>John</firstname>
        <lastname>Smith</lastname>
      </author>
      <quantity>2</quantity>
      <unitprice>30.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <notes>You owe $85.00, please pay up!</notes>
</order>
```

By-default, the start element corresponds to the root element, which in the above example is the “order” element. The data-loader by-default must map the entire document, which results in the following mapping:

```
[DataGroup (order) isRecord="true"]
  [DataValue (number) = "1"]
  [DataGroup (shipTo)]
    [DataGroup (reference)]
      [DataValue (customer) = "c001"]
    [DataValue (contact) = "Tim Bell"]
  [DataGroup (date)]
    [DataValue (day) = "14"]
    [DataValue (month) = "11"]
    [DataValue (year) = "1998"]
  [DataGroup (item)]
    [DataGroup (book)]
      [DataValue (ISBN) = "15536455"]
      [DataValue (title) = "Introduction to XML"]
      [DataGroup (author)]
        [DataValue (firstname) = "Charles"]
        [DataValue (lastname) = "Porter"]
      [DataValue (quantity) = "1"]
      [DataValue (unitprice) = "25.00"]
      [DataValue (discount) = ".40"]
    [DataGroup (item)]
      [DataGroup (book)]
        [DataValue (ISBN) = "15536456"]
        [DataValue (title) = "XML Power"]
        [DataGroup (author)]
          [DataValue (firstname) = "John"]
          [DataValue (lastname) = "Smith"]
        [DataValue (quantity) = "2"]
        [DataValue (unitprice) = "30.00"]
        [DataValue (discount) = ".40"]
      [DataValue (notes) = "You owe $85.00, please pay up!"]
```

In the above example the document-range is the range of document content enclosed by the “order” data-group element.

3.1.3 Namespaces

It is common for XML-data-documentsto be comprised of content belonging to more than one namespace. The formal specification of XML namespaces is provided by the “Namespaces for XML” [\[XMLNAMES\]](#) specification.

Namespace inheritance is described fully in the [\[XMLNAMES\]](#) specification. Briefly, each element or attribute that does not explicitly declare a namespace inherits the namespace declared by its enclosing element. A namespace is declared using a namespace prefix. The namespace prefix must be associated with a URI either in the element using the namespace prefix or in an enclosing element. The same rules apply to the XFA-data-DOM except for namespaces that are reserved for XFA directives, as described below.

The following example illustrates an XML document containing information about a purchase from a bookstore. The information is partitioned into two namespaces. The default namespace represents the order information needed by the bookstore for inventory and shipping purposes. The other namespace represents accounting information pertaining to the e-commerce transaction.

```

<?xml version="1.0" encoding="UTF-8"?>
<invoice xmlns:trn="http://www.example.com/transaction/">
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title>Introduction to XML</title>
      <quantity>1</quantity>
      <unitprice currency="us">25.00</unitprice>
      <discount>.40</discount>
      <trn:identifer>27342712</trn:identifer>
    </book>
  </item>
</invoice>

```

The use of namespaces within an XML-data-document assists in the determination of data-values and data-groups, and can also exclude document content from data-loader processing. The by-default handling of namespaces is described in this section, and additional control over namespace processing is described in the section [“Extended-Mapping-Rules, Exclude Namespaces”](#).

The document-range must also exclude any document content as follows:

- content belonging to the namespace “http://www.xfa.com/schema/xf-a-package/” or “http://www.xfa.org/schema/xf-a-package/”
- attributes with a namespace prefix of “xml”, such as “xml:space” and “xml:lang”
- attributes belonging to the namespace of “http://www.xfa.org/schema/xf-a-data/1.0/”
- namespace declaration attributes, including the default namespace attribute named “xmlns” and specific namespace attributes with the “xmlns” namespace prefix

The data-loader may have additional namespaces that it considers to be excluded from the document-range: this behavior is implementation-defined.

Note that although namespace declaration attributes are excluded from processing as ordinary metadata, they are nonetheless processed separately. Within the XFA-data-DOM each object has “xmlns” and “namespacePrefix” properties holding the namespace information. This can be thought of as out-of-band processing of namespace information. Similarly attributes belonging to the “http://www.xfa.org/schema/xf-a-data/1.0/” namespace may affect processing by the data-loader at load time but, in accordance with the rule above, must not be included as metadata in the XFA-data-DOM.

Consider again the previous example:

```
<?xml version="1.0" encoding="UTF-8"?>
<invoice xmlns:trn="http://www.example.com/transaction/">
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title xml:lang="en">Introduction to XML</title>
      <quantity>1</quantity>
      <unitprice currency="us">25.00</unitprice>
      <discount>.40</discount>
      <trn:identifier>27342712</trn:identifier>
    </book>
  </item>
</invoice>
```

In the above example, by-default there is an “invoice” element that belongs to the default namespace and also includes an “identifier” element that belongs to the namespace “http://www.example.com/transaction/”. The “invoice” element has an attribute which makes the “trn:” prefix a synonym for the “http://www.example.com/transaction/” namespace. This declaration uses an attribute named “xmlns:trn”, which according to the rule above must be excluded from the document-range. Although the declaration has its intended effect (associating subsequent data with the appropriate namespace) it is itself excluded from the document-range. Similarly the “xml:lang” attribute of the “title” element is excluded by another of the rules above. To illustrate this, the following is the same example with the document-range represented in bold type:

```
<?xml version="1.0" encoding="UTF-8"?>
<invoice xmlns:trn="http://www.example.com/transaction/">
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title xml:lang="en">Introduction to XML</title>
      <quantity>1</quantity>
      <unitprice currency="us">25.00</unitprice>
      <discount>.40</discount>
      <trn:identifer>27342712</trn:identifer>
    </book>
  </item>
</invoice>
```

The first element is also excluded from the document range, not on the basis of namespace but it is an XML processing instruction, as described above in [“XML Logical Structures”](#).

The result of mapping this XML-data-document, assuming the XFA-configuration-DOM specifies that attributes are to be preserved, is as follows:

```
[DataGroup (invoice) isRecord="true"]
  [DataGroup (item)]
    [DataGroup (book)]
      [DataValue (ISBN) = "15536455"]
      [DataValue (title) = "Introduction to XML"]
      [DataValue (quantity) = "1"]
      [DataValue (unitprice) = "25.00"]
      [DataValue (currency) = "us" content="metadata"]
      [DataValue (discount) = ".40"]
      [DataValue (trn:identifier) = "27342712"
        xmlns="http://www.example.com/transaction/"]
```

The above example demonstrates the automatic exclusion of attributes having the "xmlns:" prefix and of attributes having the "xml:" namespace prefix.

3.1.4 Record Elements

Some XML-data-documents enclose repeating groups of data, each with different content. This specification refers to these repeating groups of data as “records”. Typically a record holds the data from a single form instance.

By-default, the data-loader considers the document-range to enclose one record of data represented by the first (outermost) data-group within the document-range.

Consider the following XML-data-document:

```
<order>
  <number>1</number>
  <shipto>
    <reference><customer>c001</customer></reference>
  </shipto>
  <contact>Tim Bell</contact>
  <date><day>14</day><month>11</month><year>1998</year></date>
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title>Introduction to XML</title>
      <author>
        <firstname>Charles</firstname>
        <lastname>Porter</lastname>
      </author>
      <quantity>1</quantity>
      <unitprice>25.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <item>
    <book>
      <ISBN>15536456</ISBN>
      <title>XML Power</title>
      <author>
        <firstname>John</firstname>
        <lastname>Smith</lastname>
      </author>
      <quantity>2</quantity>
      <unitprice>30.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <notes>You owe $85.00, please pay up!</notes>
</order>
```

The result of mapping this XML-data-document with default-mapping-rules is as follows:

```
[DataGroup (order) isRecord="true"]
  [DataValue (number) = "1"]
  [DataGroup (shipTo)]
    [DataGroup (reference)]
      [DataValue (customer) = "c001"]
    [DataValue (contact) = "Tim Bell"]
  [DataGroup (date)]
    [DataValue (day) = "14"]
    [DataValue (month) = "11"]
    [DataValue (year) = "1998"]
  [DataGroup (item)]
    [DataGroup (book)]
      [DataValue (ISBN) = "15536455"]
      [DataValue (title) = "Introduction to XML"]
      [DataGroup (author)]
        [DataValue (firstname) = "Charles"]
        [DataValue (lastname) = "Porter"]
      [DataValue (quantity) = "1"]
      [DataValue (unitprice) = "25.00"]
      [DataValue (discount) = ".40"]
    [DataGroup (item)]
      [DataGroup (book)]
        [DataValue (ISBN) = "15536456"]
        [DataValue (title) = "XML Power"]
        [DataGroup (author)]
          [DataValue (firstname) = "John"]
          [DataValue (lastname) = "Smith"]
        [DataValue (quantity) = "2"]
        [DataValue (unitprice) = "30.00"]
        [DataValue (discount) = ".40"]
  [DataValue (notes) = "You owe $85.00, please pay up!"]
```

In the above example the document-range is the range of document content enclosed by the “order” data-group element, and by-default, is partitioned into a single record represented in bold type.

3.2 Rich-Text

Some content may represent "rich-text", which is text with markup representing formatting information such as underlining. A subset of XHTML markup is supported, as described in XFA Data Text Handling Version 2.0 Specification [[TextHandling](#)]. The markup within rich-text data must not be represented by nodes in the XFA-data-DOM. Instead the “value” property of the corresponding data-value must be set to plain-text derived from the rich-text in the manner described in Data Text Handling Version 2.0 Specification [[TextHandling](#)]. The procedure defined there can be summarized as follows: Starting with a copy of the rich-text including all markup, delete the start and end tags for all elements (whether they represent supported markup or not). Convert all XML character escapes to their literal form (for example, “<” to “<”). Then normalize the white space by replacing all contiguous sequences of white space and/or newline characters with single spaces. Finally, trim any leading and trailing spaces.

The effect this rule is that the derived plain-text is included in the document-range but the original rich-text is excluded. When an XFA application requires plain-text it can get it from the data-value's “value” property in the XFA-data-DOM. However when it requires the original rich-text, it must get it from the corresponding node in the XML-

data-DOM and from the descendants of that node. The data-unloader also must obtain the rich-text content from the XML-data-DOM in order to write it out without significant loss.

For content to be recognized as rich-text, as specified in [\[TextHandling\]](#), at least one of the following criteria must be met:

- the enclosing element bears an XFA “contentType” attribute with the value of “text/xhtml”
- the element content belongs to the [\[XHTML 1.0\]](#) namespace

When the data-loader recognizes rich-text, it must set the “contentType” property of the data-value corresponding to the enclosing element to “text/xhtml”. This property tells rich-text capable applications that they should look below the corresponding node of the XML-data-DOM for the original rich-text. (As mentioned in “[XFA Data Document Object Model](#)”, each node in the XFA-data-DOM contains a pointer to the corresponding node in the XML-data-DOM. The pointer is not shown in examples in this specification.)

For example, the following XML fragment contains rich-text. The rich-text content is highlighted.

```
<message>
  <p xmlns='http://www.w3.org/1999/xhtml'>
    You owe <b>$25.00</b>. Please pay up!
  </p>
</message>
```

After loading, the above fragment is represented in the XFA-data-DOM as follows.

```
[dataValue message = "You owe $25.00. Please pay up!"
  contentType="text/xhtml" ]
```

In addition to the above constraints, when specified via a “contentType” attribute on the enclosing element, the rich-text content must have a single outer element. Only white space is allowed within the region of rich-text content and outside the outer element.

The data-loader may emit a warning message when it encounters a construct that violates the above rule. How the application subsequently processes the affected content is implementation defined.

The content in the following example is illegal because the rich-text is not enclosed within a single outer element.

```
<message xmlns:xfa="http://www.xfa.org/schema/xfadata/1.0/"
  xfa:contentType="text/html">
  You owe <b>$25.00</b>. Please pay up!
</message>
```

In the example, the “message” element is not part of the rich-text because the “contentType” attribute applies to the content of the declaring element but not to the element itself. Hence the rich-text does not have include an enclosing element. However, it would not be a good idea to declare that the “message” element was part of the rich-text, because XHTML markup does not include a “message” element. Instead, the above example of illegal content could be made legal by wrapping the text in a “span” element as follows:

```
<message xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
  xfa:contentType="text/html">
  <span>You owe <b>$25.00</b>. Please pay up!</span>
</message>
```

3.3 Data-Value Elements

Consider the following XML-data-document:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
</book>
```

In the above example the elements ISBN and “title” enclose character-data. All such elements within the document-range must map to data-values with a “name” property corresponding to the local part of the element-type (the name given in the element's start and end tags), and a “value” property corresponding to the element content. The result of the mapping is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN) = "15536455"]
  [DataValue (title) = "Introduction to XML"]
```

The rules for mapping content of the XML-data-document into data-values are defined in the following sections:

3.3.1 Character-Data Data-Values

For elements within the document-range enclosing purely character-data, the data-loader must map each element into a data-value as specified:

- the “name” property of the data-value must be set to the local part of the element-type (tag name) of the element
- the “namePrefix” property of the data-value must be set to the namespace prefix of the element-type (tag name) of the element
- the “xmlns” property of the data-value must be set to the resolved namespace corresponding to the namespace prefix
- the “value” property of the data-value must be set to the character-data of the element
- the “contains” property of the data-value must be set to “data”

This behavior is illustrated by the previous example.

3.3.2 Mixed-Content Data-Values

For elements within the document-range enclosing mixed-content, the data-loader must map each element into data-values as specified:

- the “name” property of the data-value must be set to the local part of the element-type (tag name) of the element
- the “namePrefix” property of the data-value must be set to the namespace prefix of the element-type (tag name) of the element

- the “xmlns” property of the data-value must be set to the resolved namespace corresponding to the namespace prefix
- the “value” property of the data-value must be set to the ordered concatenation of all of its child data-value's “value” properties, excluding children that contain metadata (see the section “[Default-Mapping-Rules, Attributes](#)” for more information about the “contains” property)

In addition, each enclosed unit of character-data within the element must map to a data-value with a “name” property of an empty string, a “value” property corresponding to the unit of character-data, and a “contains” property of “data”. Data-values created according to this rule must be the children of the data-value mapped from the enclosing element. The child data-values must be ordered in document order.

Consider the following example where the element “desc” has mixed-content:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
<desc>Basic primer on <keyword>XML</keyword> technology.</desc>
</book>
```

The result of mapping this XML-data-document is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN)      = "15536455"]
  [DataValue (title)    = "Introduction to XML"]
  [DataValue (desc)     = "Basic primer on XML technology."]
    [DataValue ()       = "Basic primer on "]
    [DataValue (keyword) = "XML"]
    [DataValue ()       = " technology."]
```

In the above example the element “desc” maps to a data-value where

- “Basic primer on” is an enclosed unit of character-data so it maps to a data-value named “”
- “XML” is an enclosed element so it maps to a data-value named “XML”
- “ technology.” is another enclosed unit of character-data so it maps to another data-value named “”
- each of these three data-values is a child of the “desc” data-value
- the children of “desc” are in the same order that they occur in the XML-data-document
- the value of “desc” is formed by concatenating “Basic primer on”, “XML”, and “ technology” in that order

3.3.3 Empty-Element Data-Values

For each empty element within the document-range, the data-loader must by-default, map the element into a data-value as specified:

- the “name” property of the data-value must be set to the local part of the element-type (tag name) of the element
- the “namePrefix” property of the data-value must be set to the namespace prefix of the element-type (tag name) of the element
- the “xmlns” property of the data-value must be set to the resolved namespace corresponding to the namespace prefix
- the “value” property of the data-value must be set to an empty string
- the “contains” property of the data-value must be set to “data”

This specification provides an extended-mapping-rule described in section “[Simplify Structure Based on Empty Content](#)” for overriding this by-default behavior by explicitly forcing an empty element to be mapped to a data-group.

Consider the following example where the element “desc” is empty:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <desc></desc>
</book>
```

Of course, as defined by the [XML](#) specification, this example is equivalent to the following:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <desc/>
</book>
```

The result of the mapping is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN)      = "15536455"]
  [DataValue (title)    = "Introduction to XML"]
  [DataValue (desc)     = ""]
```

3.3.4 Element-Content Data-Values

A data-value cannot be the ancestor of a data-group, and therefore once an element is mapped to a data-value, the descendent elements must also be mapped to data-values. For each element enclosing element-content within the document-range, where the enclosing element is mapped to a data-value, the data-loader must by-default map the element into a data-value as specified:

Consider the following example with the element “stamp”:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <stamp>Ordered on <date><yr>2000</yr><mo>06</mo><day>23</day></date></stamp>
</book>
```

In the above example the element “date” encloses element-content and would be mapped to a data-group based upon the rules described in the next section; however, we know that the enclosing element “stamp” maps to a data-value and therefore the element “date” must be mapped to a data-value because data-values cannot be ancestors to data-groups.

The result of mapping this XML-data-document is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN)      = "15536455"]
  [DataValue (title)     = "Introduction to XML"]
  [DataValue (stamp)     = "Ordered on 20000623"]
  [DataValue ()          = "Ordered on "]
  [DataValue (date)      = "20000623"]
  [DataValue (yr)        = "2000"]
  [DataValue (MO)        = "06"]
  [DataValue (day)       = "23"]
```

3.4 Data-Group Elements

For each element enclosing element-content within the document-range, where the element does not have an enclosing element mapped to a data-value, the data-loader must by-default map the element into a data-group as specified:

- the “name” property of the data-group must be set to the local part of the element-type (tag name) of the element
- the “namePrefix” property of the data-value must be set to the namespace prefix of the element-type (tag name) of the element
- the “xmlns” property of the data-value must be set to the resolved namespace corresponding to the namespace prefix

Consider the following XML-data-document:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <author>
    <firstname>Charles</firstname>
    <lastname>Porter</lastname>
  </author>
</book>
```

The result of the mapping is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN)      = "15536455"]
  [DataValue (title)     = "Introduction to XML"]
  [DataGroup (author)]
    [DataValue (firstname) = "Charles"]
    [DataValue (lastname)  = "Porter"]
```

As specified, the element that is a candidate for mapping to a data-group must not be enclosed within an element mapped to a data-value; this is to uphold the defined relationships between data-groups and data-values where data-values must not be ancestors to data-groups. An example illustrating this case is presented in section “[Element-Content Data-Values](#)”.

3.5 Attributes

The data-loader must, by-default, refrain from loading attributes into the XFA-data-DOM. This applies to attributes of elements associated with both data-values and data-groups. On output the data-unloader must, by-default, reinsert the original attributes from the XML-data-DOM into the corresponding elements of the new XML-data-document. For any given element the order in which the data-unloader reinserts the attributes is implementation-defined. This is appropriate because the order of attributes is not significant according to the [\[XML\]](#) specification.

Consider the following XML-data-document:

```
<book status="stocked">
  <ISBN>15536455</ISBN>
  <title alternate="XML in Six Lessons">Introduction to XML</title>
</book>
```

In the above example the XML element “book” maps to a data-group and has a “status” attribute with a value of “stocked”. The XML element “title” maps to a data-value and has an “alternate” attribute with a value of “XML in Six Lessons”. Based upon the by-default rule to ignore attributes the result of the mapping is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN)   = "15536455"]
  [DataValue (title) = "Introduction to XML"]
```

3.6 White Space Handling

XML-data-documents often include additional white space within elements strictly as a legibility aid; this white space is considered insignificant. Establishing comprehensive white space handling rules contributes to predictable processing of XML-data-documents.

A processing application must produce the same result from processing two documents that differ only by insignificant white space.

The following Unicode characters are considered white space, as defined by [\[XML\]](#):

- space U+0020
- tab U+0009
- carriage return U+000D
- line feed U+000A

Note that the [\[XML\]](#) specification allows for white space to be contained within the definition of an element known to enclose only element-content, and that such white space is considered insignificant.

Distinguishing between significant and insignificant white space within an XML-data-document depends upon rules described in the following sections, and is dependent upon whether the white space is contained within a data-group or a data-value.

3.6.1 White Space in Data-Groups

If an element within an XML-data-document has been determined to represent a data-group, then all white space within the data-group must be ignored by the data-loader and must not be present in the XFA-data-DOM.

This rule respects the common cases where authors of XML-data-documents include white space as a legibility aid within elements known to enclose only other child elements.

Consider the following example to illustrate the insignificance of white space within data-groups. In the following fragment of XML the “book” element is known to represent a data-group of data-values ISBN and “title”:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
</book>
```

Note the additional newlines before and after the data-values and the spaces inserted to indent the data-values from the data-group to improve legibility. The additional newlines and other white space within “book” data-group element are considered insignificant.

The data-loader must produce the same result based on the above example, as equivalent to the following example:

```
<book><ISBN>15536455</ISBN><title>Introduction to XML</title></book>
```

3.6.2 White Space in Data-Values

If an element within an XML-data-document has been determined to represent a data-value, then by-default all white space within the data-value must be considered significant. Significant white space must be processed as content by the data-loader and must be present in the XFA-data-DOM.

Consider the following example to illustrate the significance of white space within data-values. In the following fragment of XML, the “book” element is known to represent a data-group of data-values ISBN and “title”:

```
<book>
  <ISBN> 15536455 </ISBN>
  <title>
    Introduction to XML
  </title>
</book>
```

Note the additional newlines before and after the data-values and the spaces inserted to indent the data-values from the data-group to improve legibility. As described in “[White Space in Data-Groups](#)”, the additional newlines and other white space within “book” data-group element is considered insignificant.

However, the data-value element ISBN contains additional leading and trailing space around the text “15536455”; this white space must be considered significant. In addition, the data-value element “title” contains leading and trailing newlines and white space; this white space must also be considered significant.

The data-loader must produce the same result based on the data contained within the above example as equivalent to the following example:

```
<book><ISBN> 15536455 </ISBN><title>
  Introduction to XML
</title></book>
```

4 Extended Mapping Rules

This specification provides a number of rules that are not in effect by-default, but must be made available by the implementation as overrides or extensions to the default-mapping-rules. These extended-mapping-rules may be invoked via configuration settings in the XFA-configuration-DOM or via namespaced attributes in the XML-data-document. The extended-mapping-rules are described in detail in the following sections.

The following table summarizes the relationship between the extended mapping rules. “Sequence” indicates the order in which the rules are applied to the data as it is loaded from an XML-data-document into the XFA-data-DOM, processed, and subsequently unloaded from the XFA-data-DOM into a new XML-data-document. “Category” is a rough grouping of rules that are similar in character. “Description” indicates the specific rule. “Governed By” indicates what part of the XFA-configuration-DOM controls the invocation of the rule.

Sequence	Category ¹	Description	Governed By
1	External	XSLT preprocessing	“xsl” element in configuration
2	document range	Exclude namespaces	“excludeNS” element in configuration
3	document range	Nominate start element	“startNode” element in configuration
4	document range	Nominate record elements	“record” element in configuration
5	document range	Map attributes to data-values	“attributes” element in configuration
6	document range	Exclude records by position	“range” element in configuration
7	Transform	Simplify structure based on element name	“presence” element in configuration
8	Transform	Trim white space	“whitespace” element in configuration
9	Transform	Simplify structure based on empty content	“ifEmpty” element in configuration
10	Transform	Force data-value or data-group mapping	“xfa:dataNode” attribute in XML-data-document
11	Transform	Replace element names with attribute values	“nameAttr” element in configuration
12	Transform	Rename nodes based on element name	“rename” element in configuration
13	External	XSLT postprocessing	“outputXSL” element in configuration

¹The phrases under “Category” have the following meanings:

external

Delegated to code external to XFA

document range

Alters what portion of the original XML-data-document is mapped into the XFA-data-DOM during load and back out again during unload

transform

Modifies data in the XFA-data-DOM and possibly the XML-data-DOM

For simplicity, the descriptions of some of the extended mapping rules below assume that the XFA-data-DOM is first loaded using the default behavior and then the extended mapping rules are applied in subsequent passes by modifying the XFA-data-DOM. Conforming implementations may reduce the number of physical passes to optimize performance provided the same end result is obtained.

The extended mapping rules are invoked by supplying the appropriate content to elements in the XFA-configuration-DOM. For many of those elements the content may be one of the following standardized keywords:

preserve

“preserve” in every case means the data-loader must copy the content as-is from the XML-data-DOM into the XFA-data-DOM. (Note that this description assumes that XML escape sequences such as “"” have already been converted to literal Unicode characters in the XML-data-DOM. The details of an implementation may differ provided the effect is the same.) For most mapping rules “preserve” mimics the default behavior, but there are exceptions which are described along with the individual mapping rules below.

ignore

“ignore” in every case means the data-loader must omit content from the XFA-data-DOM but do not make any changes to it in the XML-data-DOM. If a subsequent unload operation is done the data-unloader must insert the ignored content into the output document at the appropriate place, copying it from the XML-data-DOM.

remove

“remove” in every case means the data-loader must omit content from the XFA-data-DOM and also remove it from the XML-data-DOM. If a subsequent unload operation is done the data-unloader must produce a document without the removed content.

Other keywords and their meanings are defined under the extended mapping rule in which they are used.

4.1 Document Range

The term document range refers to the extent of the XML-data-document that must be processed by the data-loader, such as the whole XML-data-document or a fragment, as previously described in the section [“Default Mapping Rules, Document Range”](#).

Some extended mapping rules affect the position and extent of the document range. The set of elements associated with these rules consists of “startNode”, “record”, “incrementalLoad”, and “window”.

4.2 Transforms

Some of the extended mapping rules are known collectively as “transforms”. The elements that invoke these rules are only valid inside a “transform” element, as specified by the XFA configuration schema [\[Configuration\]](#). The set of elements associated with transforms consists of “presence”, “whitespace”, “ifEmpty”, “nameAttr”, and “rename”.

The XFA-configuration-DOM may include one or more “transform” elements. Each “transform” element must have a “ref” attribute. The value of the “ref” attribute determines where the mapping is applied. The data-loader must apply the mapping to elements whose names match (case-sensitive) the value of the “ref” attribute or, if the value of the “ref” attribute is the empty string (“”), to the entire document. Some transforms apply to all content of the matched element, including sub-elements, whereas other transforms apply only to the element itself and its character-data. The scoping in each case is natural to the type of transform. For example, white space trimming transforms, when applied to data-values, affect all of the descendants of the data-value containing data (as opposed to metadata), whereas renaming

affects only the node corresponding to the nominated element or attribute itself. However, an element mapped by one “transform” may include an element mapped by the same or a different “transform”. The data-loader must respond to such nesting by applying the specified transforms sequentially in a depth-first manner. For example, if the transform for an outer element says it is to be discarded but the transform for an inner element says it is to be preserved, the inner element is at first preserved (by its own transform) but then discarded along with the other descendents of the outer element.

Note that the value of “ref” must be a simple node name. Readers who are familiar with XFA SOM expressions might assume that SOM expression syntax is valid here, but it is not. As a consequence of this restriction, the transforms within the “transform” element apply to every node in the document-range with a matching name and appropriate type, regardless of the node's position in the tree.

Each “transform” element must include at least one operation (child element). There must not be more than one child element of the same name within a single “transform” element.

The data-loader must not apply these transformations to any part of the XML-data-document above the start element, as described in [“Nominate Start Element”](#). Likewise it must not apply these transformations to any part of the XFA-data-DOM representing parts of the XML-data-document above the start element. Also, it must not apply these transformations to any part of the XML-data-document excluded from loading by the namespace rules *except* when carrying out a “remove” directive. When carrying out a “remove”, all descendents of the node to be removed from the XML DOM must also be removed, regardless of namespace.

The data-loader must apply these transformations in the order shown in the table in Extended Mapping Rules. The “ignore” keyword has an effect both when data is loaded into the XFA-data-DOM and when it is unloaded into an XML-data-document. The other transforms only have effects when loading.

It is not recommended for a single XFA-configuration-DOM to contain multiple “transform” elements with the same value for “ref”. However if this does happen the data-loader must process each applicable transformation of the type currently being processed for a particular data element in the order that they occur in the XFA-configuration-DOM. For example, if the XFA configuration document contains:

```
<transform ref="book">
  <presence>dissolve</presence>
  <whitespace>trim</whitespace>
</transform>
<transform ref="book">
  <ifEmpty>remove</ifEmpty>
  <whitespace>normalize</whitespace>
  <presence>preserve</presence>
</transform>
```

the order of processing for a “book” element in the XML-data-document must be:

1. the first “presence” transformation “dissolve”
2. the second “presence” transformation “preserve”
3. the first “whitespace” transformation “trim”
4. the second “whitespace” transformation “normalize”
5. the sole “ifEmpty” transformation “remove”

The XFA-configuration-DOM may contain a “transform” with a “ref” value of “” (thus applying to all elements) and other “transform”s with non-empty values for “ref”. The data-loader must process the applicable “transform”s in the

order that they occur in the XFA-configuration-DOM, that is similarly to the case of “transform”s having identical “ref” values. For example, the following fragment of anXFA configuration document:

```
<transform ref="">
  <whitespace>trim</whitespace>
</transform>
<transform ref="address">
  <isEmpty>dataGroup</isEmpty>
</transform>
```

causes leading and trailing white space to be trimmed from all data-value “value” properties and then for “address” elements, if the “value” property is empty *after trimming*, the data-value is converted to a data-group.

4.3 XSLT Preprocessing

This section describes an extended-mapping-rule that can be used to modify the incoming data under control of an XSLT script [\[XSLT\]](#). This transformation makes changes to the XML-data-DOM before the XFA-data-DOM is loaded by the data-loader. Thus if a new XML-data-document is created it reflects the result of the XSLT preprocessing.

Note that it is possible to incorporate a processing instruction into the XML-data-document that causes an XSLT transformation to be applied before (or as) the data is loaded into the XML-data-DOM. This is quite separate from the facility described here. The facility described here does not require the addition of a processing instruction, or any other modification, to the XML-data-document.

The XFA-configuration-DOM may include an “xsl” element. If present, the “xsl” element must contain a “uri” element that nominates the XSLT script. See [\[Configuration\]](#) for the full schema. When the “xsl” element is supplied the data-loader must execute the script and use its output in place of the original XML-data-document.

The XFA-configuration-DOM may include a “debug” element that nominates a place in which to save a copy of the data after the XSLT transformation but before any other transformations. If the “debug” element is supplied and is non-empty, and the XSLT transformation is performed, the data-loader must copy the output of the XSLT transformation into the nominated place.

For example, the following fragment from an XFA configuration document causes the script “message.xslt” to be used for preprocessing the XML-data-document. The preprocessed document is copied to “message.xml” before being loaded by the data-loader.

```
<xsl>
  <debug>
    <uri>message.xml</uri>
  </debug>
  <uri>message.xslt</uri>
</xsl>
```

4.4 Exclude Namespaces

This specification provides an extended-mapping-rule to exclude document content from data-loaderprocessing by providing one or more XML namespaces that refer to the content intended for exclusion. By-default, as described in the section “[Default-Mapping-Rules, Namespaces](#)”, the data-loader excludes content belonging to a number of predetermined namespaces.

The XFA-configuration-DOM may include an “excludeNS” element which overrides the default behaviour. See [\[Configuration\]](#) for the full schema. The behavior specified by the XFA-configuration-DOM must override the default behavior for the entire XML-data-document.

The content of the “excludeNS” element is a white space separated list of Uniform Resource Identifiers as described by [\[URI\]](#). The data-loader must exclude elements belonging to any namespace associated with any of these URIs. The data-loader must also exclude any attributes belonging to any such namespace. Finally, it must exclude all attributes belonging to any element which is itself excluded.

Note that a namespace specified with a namespace prefix is not inherited, whereas the default namespace is inherited. Thus if an element declares a default namespace that is excluded, elements contained within it are excluded by-default. However any of the contained elements may declare a different namespace and so escape exclusion. When a containing element is excluded but its contained element is included, the node corresponding to the contained element must be appended to the XFA-data-DOM at the same point where the node corresponding to the containing element would have been appended had it not been excluded.

In the example below showing a fragment of an XML-data-document, the default name prefix is declared to be “http://www.example.org/orchard/”. This is inherited by most elements and their attributes, however some override the default with the namespace “http://www.example.org/field/”, which is signified by the namespace prefix “field”.

```
<property xml:lang="en">
  <farm xmlns="http://www.example.org/orchard/"
        xmlns:field="http://www.example.org/field/">
    <tree class="pome">apple</tree>
    <tree class="citrus">lemon</tree>
    <field:plant class="tuber">potato</field:plant>
    <tree field:role="border">poplar</tree>
  </farm>
</property>
```

Assume that the XFA Configuration DOM has been set to load attributes into the XFA-data-DOM, as described in [“Map Attributes to Data-Values”](#). Without the “ExcludeNS” option, the resulting subtree of the XFA-data-DOM contains:

```

[DataGroup (property)
  [DataGroup (farm) xmlns="http://www.example.org/orchard/"]
    [DataValue (tree) = "apple"
      xmlns="http://www.example.org/orchard/"]
      [DataValue (class) = "pome" contains="metadata"
        xmlns="http://www.example.org/orchard/"]
    [DataValue (tree) = "lemon"
      xmlns="http://www.example.org/orchard/"]
      [DataValue (class) = "citrus" contains="metadata"
        xmlns="http://www.example.org/orchard/"]
    [DataValue (plant) = "potato" namePrefix="field"
      xmlns="http://www.example.org/field/"]
      [DataValue (class) = "tuber" contains="metadata"
        xmlns="http://www.example.org/orchard/"]
    [DataValue (tree) = "poplar"
      xmlns="http://www.example.org/orchard/"]
    [DataValue (role) = "border" namePrefix="field"
      contains="metadata" xmlns="http://www.example.org/field/"]

```

Note that the “xml:lang” attribute of the “property” element is excluded by the default namespace exclusion rule described in [“Document Range, Namespaces”](#). The default namespace exclusions are mandatory and not controlled by the XFA Configuration DOM.

If the “excludeNS” option was used to exclude “http://www.example.org/field/”, the XFA-data-DOM would include only those portions shown below in bold:

```
<property xml:lang="en">
  <farm xmlns="http://www.example.org/orchard/"
        xmlns:field="http://www.example.org/field/">
    <tree class="pome">apple</tree>
    <tree class="citrus">lemon</tree>
    <field:plant class="tuber">potato</field:plant>
    <tree field:role="border">poplar</tree>
  </farm>
</property>
```

The namespace declarations on the “farm” element are processed by the data-loader but not placed directly into the data-value corresponding to the element where they are found, as described in “[XFA Data Document Object Model](#)”. However since the “farm” element-type does not include a namespace prefix it adopts the default namespace which is declared by one of its attributes. The resulting subtree of the XFA-data-DOM would contain:

```
[DataGroup (property)]
  [DataGroup (farm) xmlns="http://www.example.org/orchard/"]
    [DataValue (tree) = "apple"
      xlmns="http://www.example.org/orchard/"]
      [DataValue (class) = "pome" contains="metadata"
        xlmns="http://www.example.org/orchard/"]
    [DataValue (tree) = "lemon"
      xlmns="http://www.example.org/orchard/"]
      [DataValue (class) = "citrus" contains="metadata"
        xlmns="http://www.example.org/orchard/"]
    [DataValue (tree) = "poplar"
      xlmns="http://www.example.org/orchard/"]
```

If instead the namespace “http://www.example.org/orchard/” was excluded, the XFA-data-DOM would include the portions shown below in bold:

```
<property xml:lang="en">
  <farm xmlns="http://www.example.org/orchard/"
        xmlns:field="http://www.example.org/field/">
    <tree class="pome">apple</tree>
    <tree class="citrus">lemon</tree>
    <field:plant class="tuber">potato</field:plant>
    <tree field:role="border">poplar</tree>
  </farm>
</property>
```

The resulting subtree of the XFA-data-DOM would contain:

```
[DataGroup (property)]
  [DataValue (plant) = "plant" namePrefix="field"
    xmlns="http://www.example.org/field"]
```

Finally, if both “<http://www.example.org/orchard/>” and “<http://www.example.org/field/>” were excluded, only the “property” element would remain. As an empty element it would by-default be mapped to a data-value.

```
[DataValue (property) = ""]
```

4.5 Nominate Start Element

This section describes an extended-mapping-rule to nominate an element within the XML-data-document that represents the element where the XFA data handling processing must begin; that is, a way to specify that a particular element within the document actually represents the “root” of the document, thus constraining the range of the data-loader’s processing to a subset fragment of the document.

By-default, as described in the section “[Default-Mapping-Rules, Start Element](#)”, the data-loader attempts to map the whole of the XML-data-document. This extended-mapping-rule provides a mechanism for nominating an element where the data-loader must start processing. As a result the document-range is constrained to a range of document content not greater than the fragment specified by the start element.

The XFA-configuration-DOM may include a “startNode” element which overrides the default behaviour. See [\[Configuration\]](#) for the full schema. The behavior specified by the XFA-configuration-DOM must override the default behavior for the entire XML-data-document. The content of the “startNode” element is a string of the form “`xfasom(somExpr)`” where *somExpr* is a restricted SOM expression. The general syntax of SOM expressions is defined in the XFA Scripting Object Model 2.0 Specification [\[SOM\]](#). The expression in the “startNode” element must be a fully-qualified path of element types (tag names) starting with the root of the XML-data-document and referring to a single element, as illustrated in the following example:

```

<order>
  <number>1</number>
  <shipto>
    <reference><customer>c001</customer></reference>
  </shipto>
  <contact>Tim Bell</contact>
  <date><day>14</day><month>11</month>
    <year>1998</year></date>
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title>Introduction to XML</title>
      <author>
        <firstname>Charles</firstname>
        <lastname>Porter</lastname>
      </author>
      <quantity>1</quantity>
      <unitprice>25.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <item>
    <book>
      <ISBN>15536456</ISBN>
      <title>XML Power</title>
      <author>
        <firstname>John</firstname>
        <lastname>Smith</lastname>
      </author>
      <quantity>2</quantity>
      <unitprice>30.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <notes>You owe $85.00, please pay up!</notes>
</order>

```

Assume that the start node has been set with "xfasom(order.item[1].book)". Recall that XFA SOM expressions use zero-based indexing, so "item[1]" refers to the *second* instance of "item". The result of the mapping is as follows:

```

[DataGroup (book)]
  [DataValue (ISBN) = "15536456"]
  [DataValue (title) = "XML Power"]
  [DataGroup (author)]
    [DataValue (firstname) = "John"]
    [DataValue (lastname) = "Smith"]
  [DataValue (quantity) = "2"]
  [DataValue (unitprice) = "30.00"]
  [DataValue (discount) = ".40"]

```

With a start element expressed as "xfasom(order.item)", the result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (firstname) = "Charles"]
      [DataValue (lastname) = "Porter"]
    [DataValue (quantity) = "1"]
    [DataValue (unitprice) = "25.00"]
    [DataValue (discount) = ".40"]
```

It should be noted that an identical mapping must be produced with a start element expressed as "xfasom(order.item[0])".

With a start element expressed as "xfasom(order.item[1])", the result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536456"]
    [DataValue (title) = "XML Power"]
    [DataGroup (author)]
      [DataValue (firstname) = "John"]
      [DataValue (lastname) = "Smith"]
    [DataValue (quantity) = "2"]
    [DataValue (unitprice) = "30.00"]
    [DataValue (discount) = ".40"]
```

4.6 Nominate Record Elements

This section describes an extended-mapping-rule to nominate the data-group elements within the document-range that represent records.

By-default, as described in section "[Default-Mapping-Rules, Document Range, Record Elements](#)", the data-loader considers the document-range to include one record of data represented by the first data-group. This data handling option provides a mechanism for controlling the partitioning of the document into one or more records by nominating data-group elements where partitioning occurs. The extent of the document-range is not affected by record elements, rather the content within the document-range is partitioned by record elements.

Notionally a record is a complete logical unit which the application processes separately. For example a tax department collects tax returns from many individuals. All of the tax returns could be expressed as a single large XML-data-document with each individual return comprising one record. The division of a document into records may have side-effects in the way they are processed by the XFA application. It is expected that the application will process records sequentially and separately. For example, it is common to sequentially merge each record with a template, then lay out and print the merged record before moving on to the next record. In this operation the output for each record starts on a new page, even if there is space remaining on the last page of the previous record. In addition, portions of the XFA-data-DOM that lie outside any record may passively supply data but in this operation do not force any output to occur.

The XFA-configuration-DOM may include a “record” element which overrides the default behaviour. See [\[Configuration\]](#) for the full schema. The behavior specified by the XFA-configuration-DOM must override the default behavior for the entire XML-data-document.

The “record” element nominates data-groups which must be marked as records by setting their “isRecord” properties to true. The content of the “record” element must be either a node level or an element type, where:

- A node level is an integer that specifies the level in the XML-data-DOM hierarchy for which the data-loader must mark each corresponding data-group node in the XFA-data-DOM as a record. The node level is relative to the start node, as follows: the start node is located at level “0”, its children are at “1”, its grandchildren at “2” and so on.
- An element-type (tag name) of an element present within the XML-data-document where the data-loader must mark each corresponding data-group element at the same level in the hierarchy with the specified element-type (tag name) as a record.

The second case requires some clarification. If an element-type is supplied, the data-loader must traverse the nodes in the XML-data-DOM beneath the start node in depth-first left-to-right (document) order, until it encounters a node that corresponds to a data-group in the XFA-data-DOM and for which the local part of the element-type matches the supplied string (case-sensitive). The corresponding data-group in the XFA-data-DOM must be marked as a record. Subsequent nodes at the same depth in the XML-data-DOM (but not necessarily siblings) that also correspond to data-groups and have the same “name” property must also be marked as records.

The following XML-data-document shows a situation in which there are two records but they are not siblings. Assume that in the configuration document the “record” element contains “book”. The result is that the subsections shown in bold print are processed as records.

```
<order>
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title>Introduction to XML</title>
    </book>
  </item>
  <item>
    <book>
      <ISBN>15536456</ISBN>
      <title>XML Power</title>
    </book>
  </item>
</order>
```

Note that there are two cases in which a node in the XML-data-DOM might have the correct name and level but not cause a record to be marked. First, the node in the XML-data-DOM might be mapped into a data-value in the XFA-data-DOM, but data-values cannot be marked as records because they have no “isRecord” property. Second, the node in the XML-data-DOM might be excluded from the XFA-data-DOM because of its namespace, as described in [“Exclude Namespaces”](#), in which case there is nothing to mark as a record.

It is also worth noting that namespace exclusion may raise the level of a node in the XFA-data-DOM relative to the corresponding node in the XML-data-DOM, when a container element is excluded but not its contained element, as described in [“Exclude Namespaces”](#). The level that matters in determining which data-groups are records is the level in the XML-data-DOM, *not* the level in the XFA-data-DOM. This is appropriate because the record structure is normally expressed in the structure of the XML-data-document as it is supplied before any exclusions.

Consider the following XML-data-document:

```
<order>
  <number>1</number>
  <shipto>
    <reference><customer>c001</customer></reference>
  </shipto>
  <contact>Tim Bell</contact>
  <date><day>14</day><month>11</month>
    <year>1998</year></date>
  <item>
    <book>
      <ISBN>15536455</ISBN>
      <title>Introduction to XML</title>
      <author>
        <firstname>Charles</firstname>
        <lastname>Porter</lastname>
      </author>
      <quantity>1</quantity>
      <unitprice>25.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <item>
    <book>
      <ISBN>15536456</ISBN>
      <title>XML Power</title>
      <author>
        <firstname>John</firstname>
        <lastname>Smith</lastname>
      </author>
      <quantity>2</quantity>
      <unitprice>30.00</unitprice>
      <discount>.40</discount>
    </book>
  </item>
  <notes>You owe $85.00, please pay up!</notes>
</order>
```

With no “record” element in the XFA-configuration-DOM, the “order” element is considered to be the one and only record. The data-group representing the single record is represented in bold type:

```
[DataGroup (order)]
  [DataValue (number) = "1"]
  [DataGroup (shipTo)]
    [DataGroup (reference)]
      [DataValue (customer) = "c001"]
  [DataValue (contact) = "Tim Bell"]
  [DataGroup (date)]
    [DataValue (day) = "14"]
    [DataValue (month) = "11"]
    [DataValue (year) = "1998"]
  [DataGroup (item)]
    [DataGroup (book)]
      [DataValue (ISBN) = "15536455"]
      [DataValue (title) = "Introduction to XML"]
      [DataGroup (author)]
        [DataValue (firstname) = "Charles"]
        [DataValue (lastname) = "Porter"]
      [DataValue (quantity) = "1"]
      [DataValue (unitprice) = "25.00"]
      [DataValue (discount) = ".40"]
    [DataGroup (item)]
      [DataGroup (book)]
        [DataValue (ISBN) = "15536456"]
        [DataValue (title) = "XML Power"]
        [DataGroup (author)]
          [DataValue (firstname) = "John"]
          [DataValue (lastname) = "Smith"]
        [DataValue (quantity) = "2"]
        [DataValue (unitprice) = "30.00"]
        [DataValue (discount) = ".40"]
  [DataValue (notes) = "You owe $85.00, please pay up!"]
```

The following example assumes that the “record” element contains “item”. The result of mapping the same XML-data-document is as follows, with the data-groups representing records appearing in bold type:

```
[DataGroup (order)]
  [DataValue (number) = "1"]
  [DataGroup (shipTo)]
    [DataGroup (reference)]
      [DataValue (customer) = "c001"]
    [DataValue (contact) = "Tim Bell"]
  [DataGroup (date)]
    [DataValue (day) = "14"]
    [DataValue (month) = "11"]
    [DataValue (year) = "1998"]
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (firstname) = "Charles"]
      [DataValue (lastname) = "Porter"]
    [DataValue (quantity) = "1"]
    [DataValue (unitprice) = "25.00"]
    [DataValue (discount) = ".40"]
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536456"]
    [DataValue (title) = "XML Power"]
    [DataGroup (author)]
      [DataValue (firstname) = "John"]
      [DataValue (lastname) = "Smith"]
    [DataValue (quantity) = "2"]
    [DataValue (unitprice) = "30.00"]
    [DataValue (discount) = ".40"]
  [DataValue (notes) = "You owe $85.00, please pay up!"]
```

The expression of data records via a node level is likely not the typical usage of this extended-mapping-rule, but does have particular utility in specific cases similar to the one illustrated below. Using the previous XML-data-document as an example, with a “record” element containing “1”, the result of the mapping is as follows:

```
[DataGroup (order)]
  [DataValue (number) = "1"]
  [DataGroup (shipTo)]
    [DataGroup (reference)]
      [DataValue (customer) = "c001"]
    [DataValue (contact) = "Tim Bell"]
  [DataGroup (date)]
    [DataValue (day) = "14"]
    [DataValue (month) = "11"]
    [DataValue (year) = "1998"]
  [DataGroup (item)]
    [DataGroup (book)]
      [DataValue (ISBN) = "15536455"]
      [DataValue (title) = "Introduction to XML"]
      [DataGroup (author)]
        [DataValue (firstname) = "Charles"]
        [DataValue (lastname) = "Porter"]
      [DataValue (quantity) = "1"]
      [DataValue (unitprice) = "25.00"]
      [DataValue (discount) = ".40"]
  [DataGroup (item)]
    [DataGroup (book)]
      [DataValue (ISBN) = "15536456"]
      [DataValue (title) = "XML Power"]
      [DataGroup (author)]
        [DataValue (firstname) = "John"]
        [DataValue (lastname) = "Smith"]
      [DataValue (quantity) = "2"]
      [DataValue (unitprice) = "30.00"]
      [DataValue (discount) = ".40"]
  [DataValue (notes) = "You owe $85.00, please pay up!"]
```

As a result of the above mapping, the XML-data-document is partitioned into four records: “shipTo”, “date”, “item”, “item”. The two “item” records represent the same grouping of data as order items. The other two records “shipTo” and “date”, don't represent the same grouping of data as order items, and they don't even relate directly to each other. Given this XML-data-document, such a mapping is only useful if the processing application is able to discriminate among the data-group records that are of interest. This example illustrates how the expression of data records via a node level can easily produce a mapping of heterogeneous data records.

In cases where the XML-data-document makes use of different element-types for roughly the same grouping of data, the ability to express data records via a node level is very useful, as illustrated by the following example with a “record” element containing “1”.

```

<order>
  <bookitem>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <author>
      <firstname>Charles</firstname>
      <lastname>Porter</lastname>
    </author>
    <quantity>1</quantity>
    <unitprice>25.00</unitprice>
    <discount>.40</discount>
  </bookitem>
  <musicitem>
    <cdid>4344-31020-2</cdid>
    <title>Big Calm</title>
    <artist>Morcheeba</artist>
    <quantity>1</quantity>
    <unitprice>19.00</unitprice>
  </musicitem>
</order>

```

The result of the mapping, with the data-groups representing records appearing in bold type, is as follows:

```

[DataGroup (order)]
  [DataGroup (bookitem)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (firstname) = "Charles"]
      [DataValue (lastname) = "Porter"]
    [DataValue (quantity) = "1"]
    [DataValue (unitprice) = "25.00"]
    [DataValue (discount) = ".40"]
  [DataGroup (musicitem)]
    [DataValue (cdid) = "4344-31020-2"]
    [DataValue (title) = "Big Calm"]
    [DataValue (artist) = "Morcheeba"]
    [DataValue (quantity) = "1"]
    [DataValue (unitprice) = "19.00"]

```

The XML-data-document may have content that is outside any record. Although such content must not be marked as part of a record, it must be loaded into the XFA-data-DOM. Although it is not inside any record it can still be used in special circumstances, for example if a calculation explicitly makes reference to it. For example, the previous example could be modified with some extra-record data as follows:

```

<order>
  <customername>Delta Books</customername>
  <customerorder>300179</customerorder>
  <bookitem>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <author>
      <firstname>Charles</firstname>
      <lastname>Porter</lastname>
    </author>
    <quantity>1</quantity>
    <unitprice>25.00</unitprice>
    <discount>.40</discount>
  </bookitem>
  <musicitem>
    <cdid>4344-31020-2</cdid>
    <title>Big Calm</title>
    <artist>Morcheeba</artist>
    <quantity>1</quantity>
    <unitprice>19.00</unitprice>
  </musicitem>
</order>

```

When this is loaded into the XFA-data-DOM the additional data is not part of any record but is available for use.

```

[DataGroup (order)]
  [DataValue (customername) = "Delta Books"]
  [DataValue (customerorder) = "300179"]
  [DataGroup (bookitem)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (firstname) = "Charles"]
      [DataValue (lastname) = "Porter"]
    [DataValue (quantity) = "1"]
    [DataValue (unitprice) = "25.00"]
    [DataValue (discount) = ".40"]
  [DataGroup (musicitem)]
    [DataValue (cdid) = "4344-31020-2"]
    [DataValue (title) = "Big Calm"]
    [DataValue (artist) = "Morcheeba"]
    [DataValue (quantity) = "1"]
    [DataValue (unitprice) = "19.00"]

```

4.7 Map Attributes to Data-Values

This section defines an extended-mapping-rule that can be used to map XML attributes to data-values.

The XFA-configuration-DOM may include an “attributes” element which overrides the default behaviour. See [\[Configuration\]](#) for the full schema. The content of the “attributes” element must be one of the following keywords:

- “delegate” allows an implementation-defined behavior which may have the effect of either “ignore” or “preserve”, as described below
- “ignore” means that the data-loader must not map attributes to data-values
- “preserve” means that the data-loader must map attributes to data-values, unless the attribute is excluded by namespace, as described in the following sections

Invoking the extended-mapping-rule with “ignore” produces the same results as the default mapping behavior for attributes as described in section [“Default-Mapping-Rules, Attributes”](#).

Attributes may also be excluded from loading based on namespace (as described in [“Exclude Namespaces”](#)). The data-loader must not load attributes with excluded namespaces regardless of the value of the “attributes” element in the XFA-configuration-DOM.

Apart from the above exclusions, the behavior specified by the XFA-configuration-DOM must override the default behavior for the entire XML-data-document.

Assuming that this extended-mapping-rule has been invoked with “preserve”, the following sections describes how the mapping occurs for attributes which are not excluded by the above rules.

4.7.1 Data-Group Elements with Attributes

Consider the following XML-data-document:

```
<item xmlns:abc="http://www.example.org/abc/">
  <book abc:status="stocked">
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
  </book>
</item>
```

In the above example, the “book” element, maps to a data-group and has an “abc:status” attribute with a value of “stocked”.

The data-loader must map each attribute of a data-group element to data-values that are children of the data-group as follows:

- the “name” property of the data-value must be set to the attribute name
- the “namePrefix” property of the data-value must be set to the namespace prefix of the attribute name
- the “xmlns” property of the data-value must be set to the resolved namespace of the attribute name
- the “value” property of the data-value must be set to the attribute value
- the “contains” property of the data-value must be set to “metadata”

The result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (status) = "stocked" namePrefix="abc"
      xmlns="http://www.example.org/abc/" contains="metadata" ]
    [DataValue (ISBN) = "15536455" ]
    [DataValue (title) = "Introduction to XML" ]
```

The data-loader is expected to process attributes prior to processing any content of the data-group element. The effect must be that any data-values resulting from attributes appear as children of the data-group before (left of) any children resulting from processing content of the data-group. This is in keeping with the general structure of the XFA-data-DOM as described in “[XFA Document Object Model](#)”, whereby a top-down left-to-right traversal reproduces document order.

The set of data-values resulting from processing attributes are ordered in an implementation-defined order. The [XML](#) specification states that by definition the order of attributes is not meaningful. Hence there is no justification for imposing any attribute ordering restriction upon the data-loader. On the other hand applications using the XFA-data-DOM must not rely on any particular ordering of attributes, apart from the previously stated requirement that attribute nodes precede content nodes.

4.7.2 Data-Value Elements with Attributes

Consider the following XML-data-document:

```
<item>
  <book>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <unitprice currency="USD">25.00</unitprice>
  </book>
</item>
```

In the above example the “unitprice” element maps to a data-value and has a “currency” attribute.

Assume the XFA configuration document has an <attribute> element containing “preserve”.

When an element mapped to a data-value has one or more attributes, the data-loader must map each attribute to a data-values that is a child of the data-value mapped to the enclosing element as specified:

- the “name” property of the data-value must be set to the local part of the attribute name
- the “namePrefix” property of the data-value must be set to the namespace prefix of the attribute name
- the “xmlns” property of the data-value must be set to the resolved namespace of the attribute name
- the “value” property of the data-value must be set to the attribute value
- the “contains” property of the data-value must be set to “metadata”

As a result of processing attributes of a data-value, the data-loader produces a child data-value for each attribute.

The result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN)           = "15536455"]
    [DataValue (title)          = "Introduction to XML"
    [DataValue (unitprice)      = "25.00"]
    [DataValue (currency)       = "USD" contains="metadata"]
```

Note that in the above mapping the value of “unitprice” does not include the value of the data-value “currency” because mapped attributes have the characteristic that they are considered to contain metadata; only data-values that contain data are included in the value of a parent data-value as described in section “[Default-Mapping-Rules, Mixed-Content Data-Values](#)”.

The data-loader is expected to process attributes prior to processing any content of the data-value element. The effect must be that any data-values resulting from attributes appear as children of the data-value before (left of) any children resulting from processing content of the data-value. This is in keeping with the general structure of the XFA-data-DOM as described in “[XFA Data Document Object Model](#)”, whereby a top-down left-to-right traversal reproduces significant document order.

The set of data-values resulting from processing attributes are ordered in an implementation-defined order. The [XML](#) specification states that by definition the order of attributes is not meaningful. Hence there is no justification for imposing any attribute ordering restriction upon the data-loader. On the other hand applications using the XFA-data-DOM must not rely on any particular ordering of attributes, apart from the previously stated requirement that attribute nodes precede content nodes.

The same rules apply to attributes of elements containing mixed-content. Consider the following XML-data-document:

```
<book>
<desc class="textbook">Basic primer on <keyword
  class="technical">XML</keyword> technology.</desc>
</book>
```

The result of the mapping is as follows:

```
[DataGroup (book)]
  [DataValue (desc) = "Basic primer on XML technology."
  [DataValue (class) = "textbook" contains="metadata"]
  [DataValue () = "Basic primer on"]
  [DataValue (keyword) = "XML"]
  [DataValue (class) = "technical" contains="metadata"]
  [DataValue () = " technology."]
```

Attributes of empty data-value elements are processed via the same rules as other elements. Consider the following XML-data-document:

```
<item>
  <book>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <unitprice currency="USD">25.00</unitprice>
    <desc language="en-US" />
  </book>
</item>
```

In the above example the empty “desc” element maps to a data-value and has a “language” attribute.

Assume the XFA-configuration-DOM has an “attribute” element containing “preserve”. Given that empty elements map to data-values, as described in section “[Empty Element Data-Values](#)”, the result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataValue (unitprice) = "25.00"]
    [DataValue (currency) = "USD" contains="metadata"]
    [DataValue (desc) = ""]
    [DataValue (language) = "en-US" contains="metadata"]
```

4.8 Exclude Records By Position

This section describes an extended-mapping-rule to specify a subset of records for processing.

By-default the data-loader must process all records in the XML-data-document. However the XFA-configuration-DOM may contain a “range” element which specifies a set of record indices. See [\[Configuration\]](#) for the full schema. When this element is provided and is non-empty, records with indices not included in the list must not be processed. Hence the mere existence of the list reverses the default behavior. The behavior specified by the XFA-configuration-DOM must override the default behavior for the entire XML-data-document.

The content of the “range” element must be a comma-separated list of record numbers and/or record number ranges, where:

- A record number is a positive base ten integer specifying a record index that must be added to the set. The index of the first record in the XML-data-document is “0”, of the second is “1”, and so on.

- A record number range is a positive base ten integer followed by a hyphen character (“-”) followed by another positive base ten integer, specifying a range of record indices that must be added to the set. The range includes the two numbers given. The two integers may be in either order, smallest-first or largest-first.

All records with indices in the set must be processed. The set of indices may extend beyond the index of the last record in the XML-data-document (indeed it is expected that this will frequently be the case).

Use of this mapping rule must not affect the order in which records are processed, only which ones are included. Therefore the order of processing will be the same as described elsewhere in this specification.

4.9 Simplify Structure Based on Element Name

This section describes an extended-mapping-rule that can be used to alter the handling of data groups in the XML-data-document.

By-default, as described in [Empty-Element Data-Values](#), elements without content (containing only other elements) are represented in the Data DOM by data group nodes. Elements with content (including mixed content) are represented by data value nodes. This extended-mapping-rule provides for alternate behaviors in which elements without content are omitted or represented by data values.

The XFA-configuration-DOM may include one or more “transform” elements, each of which may contain a “presence” element that overrides the default behaviour. See [[Configuration](#)] for the full schema. The behavior specified by the “presence” element must override the default behavior within the scope described for each keyword below.

The content of the “presence” element must be one of “preserve”, “ignore”, “dissolve”, or “dissolveStructure”, where:

- “preserve” means that the data-loader must preserve the original hierarchy.
- “ignore” means that the data-loader must remove the node matching the “ref” property and also its descendents from the XFA-data-DOM but not from the XML-data-DOM, unless the node is the root node in which case the original hierarchy must be preserved.
- “dissolve” means that the data-loader must remove the data-group node matching the “ref” property from the XFA and XML-data-DOMs, promoting its immediate children to children of its parent, unless the node is the root node in which case it must be preserved. Descendents are not affected by this operation.
- “dissolveStructure” means that the data-loader must remove all data-groups that are descendants of the node matching the “ref” property from the XFA and XML-data-DOMs. The matching node itself must not be removed. All data-values that are descendants of the node must be promoted to children of the node.

Invoking the extended-mapping-rule with “preserve” produces the same results as the default mapping behavior for element names as described in section “[Basic Concepts, XFA Data Document Object Model](#)”.

Note the different scope for each keyword. In summary, “ignore” affects the specified node and its descendents, whereas “dissolve” affects just the specified node and “dissolveStructure” affects just the descendents.

Consider the following XML-data-document:

```
<item>
  <book>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <author>
      <firstname>Charles</firstname>
      <lastname>Porter</lastname>
    </author>
  </book>
</item>
```

With no suppression of data-groupmapping, expressed in the XFA configuration document as:

```
<transform ref="book">
  <presence>preserve</presence>
</transform>
```

the result of the mapping is identical to the default mapping as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (firstname) = "Charles"]
      [DataValue (lastname) = "Porter"]
```

The following example suppresses the “author” data-group and its substructure, expressed in the XFA configuration document as:

```
<transform ref="author">
  <presence>ignore</presence>
</transform>
```

The result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
```

The following example suppresses the data-group mapping of the “book” element, expressed in the XFA configuration document as:

```
<transform ref="book">
  <presence>dissolve</presence>
</transform>
```

The result of the mapping is as follows:

```
[DataGroup (item)]
  [DataValue (ISBN) = "15536455"]
  [DataValue (title) = "Introduction to XML"]
  [DataGroup (author)]
    [DataValue (firstname) = "Charles"]
    [DataValue (lastname) = "Porter"]
```

Because “dissolve” also modifies the XML-data-DOM, when the data-unloader creates a new XML-data-document the new document reflects the “dissolve” operation. In this case the resulting XML-data-document contains:

```
<item>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <author>
    <firstname>Charles</firstname>
    <lastname>Porter</lastname>
  </author>
</item>
```

The following example suppresses all data-group mapping within the “book” element, expressed in the XFA configuration document as:

```
<transform ref="book">
  <presence>dissolveStructure</presence>
</transform>
```

The result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = "15536455"]
    [DataValue (title) = "Introduction to XML"]
    [DataValue (firstname) = "Charles"]
    [DataValue (lastname) = "Porter"]
```

Because “dissolveStructure” also modifies the XML-data-DOM, when the data-unloader creates a new XML-data-document the new document reflects the “dissolveStructure” operation. In this case the resulting XML-data-document contains:

```
<item>
  <book>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <firstname>Charles</firstname>
    <lastname>Porter</lastname>
  </book>
</item>
```

The data-loader should not issue a warning when “ignore” or “dissolve” is suppressed at the root node, because the same element-type may legitimately appear other places in the XML-data-document.

4.10 Trim White Space

This section describes an extended-mapping-rule that can be used to alter the handling of white space in the XML-data-document. This handling affects the XFA-data-DOM and also the XML-data-DOM.

By-default, as described in White Space Handling, white space is preserved in the “value” property of data-valuenodes. This extended-mapping-rule provides for alternate behaviors in which white space is trimmed from both ends, trimmed only on the left, trimmed only on the right, or normalized. Note that both element content and attribute values are represented by data-value nodes, so this operation applies equally to both.

The XFA-configuration-DOM may include one or more “transform” elements, each of which may contain a “whitespace” element that overrides the default behaviour. See [[Configuration](#)] for the full schema. When applied to a data-group this transform has no effect, either upon the data-group or upon its children. When applied to a data-value this transform must override the default behavior for the data-value and its descendents. However its descendents may have different white space transforms applied to them, and such transforms must be applied depth-first as described in [[Transforms](#)].

The content of the “whitespace” element must be one of “trim”, “rtrim”, “ltrim”, “normalize”, or “preserve”, where:

- “trim” means that the data-loader must trim white space from both ends but preserve embedded white space, both in the XFA-data-DOM and in the XML-data-DOM.
- “rtrim” means that the data-loader must trim trailing white space but preserve leading and embedded white space, both in the XFA-data-DOM and in the XML-data-DOM.
- “ltrim” means that the data-loader must trim leading white space but preserve embedded and trailing white space, both in the XFA-data-DOM and in the XML-data-DOM.
- “normalize” means that the data-loader must trim leading and trailing white space and replace each instance of embedded white space with a single SPACE character (U+0020), both in the XFA-data-DOM and in the XML-data-DOM.
- “preserve” means that the data-loader must preserve all white space.

When the document contains mixed content, the operation must be performed on the complete text of the “value” property of the outermost data-value, then the inner data-values must be modified as necessary to remain consistent. When the operation is “normalize”, where within mixed content there is embedded white space at the end of one data-value, whether or not the the next data-value starts with white space, the replacement SPACE character must be assigned to the first data-value; but where a data-value ends in non-white space and the next data-value starts with white space, the replacement SPACE character must be assigned to the second data-value.

For example, consider the following XML data:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
<desc>Primer on <keyword> XML </keyword> technology.</desc>
</book>
```

The resulting XFA-data-DOM, after default white space handling (“preserve”), is as follows:

```
[DataGroup (book)]
  [DataValue (ISBN)      = "15536455"]
  [DataValue (title)     = "Introduction to XML"]
  [DataValue (desc)      = "Primer on XML technology."]
    [DataValue ()        = "Primer on "]
    [DataValue (keyword) = " XML "]
    [DataValue ()        = " technology."]
```

After the normalize operation no more than one blank separates each word:

```
[DataGroup (book)]
  [DataValue (ISBN)      = "15536455"]
  [DataValue (title)     = "Introduction to XML"]
  [DataValue (desc)      = "Primer on XML technology."]
    [DataValue ()        = "Primer on "]
    [DataValue (keyword) = "XML"]
    [DataValue ()        = " technology."]
```

In the following example two different transforms are defined, “rtrim” for one element and “ltrim” for the other.

```
<transform ref="desc">
  <whitespace>rtrim</whitespace>
</transform>
<transform ref="keyword">
  <whitespace>ltrim</whitespace>
</transform>
```

In the XML-data-document an element that is nominated for “rtrim” encloses an element nominated for “ltrim”.

```
<desc>Primer on<keyword> XML </keyword> </desc>
```

The result is that the inner element is first subject to “ltrim” on its own, and then to “rtrim” as content of the outer element. Hence the value of the inner element starts off as “ XML ”, is trimmed on the left to become “XML ”, and then trimmed on the right as part of the string “Primer on XML ”, resulting finally in:

```
[DataValue (desc) = "Primer onXML"]
  [DataValue (keyword) = "XML"]
```

4.11 Simplify Structure Based on Empty Content

This section describes an extended-mapping-rule that can be used to modify the handling of elements that are empty in the XML-data-document. This handling affects the XFA-data-DOM and may optionally alter the XML-data-DOM.

By-default, as described in [Empty-Element Data-Values](#), the data-loader represents empty elements in the XML-data-document with data-values in the XFA-data-DOM. This extended-mapping-rule provides for alternate behaviors in which the data-value node is removed from the Data DOM(s) or is converted to a data-group.

For purposes of this specification:

- A data-group is considered empty if and only if it has no children.

- A data-value is considered empty if and only if it has no children and its “value” property is equal to the empty string (“”).

As described in [Mixed-Content Data-Values](#), a data-value representing mixed content has a “value” property equal to the ordered concatenation of the “value” properties of its content-containing children. Hence it can have a “value” property equal to the empty string if all of its children also have “value” properties equal to the empty string. However even in this case it is not considered empty because it has children.

Similarly a data-value representing an element with no content has a “value” property equal to the empty string, but if it has a child representing an attribute it is not considered empty. Attributes are not loaded by-default but they may be loaded under control of a configuration option, as described in [Map Attributes to Data-Values](#).

Consider the following XML-data-document:

```
<item>
  <book>
    <ISBN registered="no"></ISBN>
    <title>Introduction to XML</title>
    <author>
      <firstname></firstname>
      <lastname></lastname>
    </author>
  </book>
</item>
```

The elements “firstname” and “lastname” are considered empty. The “author”, “book” and “item” elements are not empty because each of them has children. Assuming attributes were not loaded the element “ISBN” is empty, however if attributes were loaded it is not empty.

The XFA-configuration-DOM may include one or more “transform” elements, each of which may include an “ifEmpty” element that overrides the default behavior. See [\[Configuration\]](#) for the full schema. If an “ifEmpty” element is present and the corresponding node is empty, the behavior specified by the “ifEmpty” element must override the default behavior within the scope of the “transform” element.

The “ifEmpty” element must contain one of “ignore”, “remove”, “dataGroup”, or “dataValue”, where:

- “ignore” means that the data-loader must remove the node representing the element from the XFA-data-DOM, unless it is the root node. This does not affect the XML-data-DOM.
- “remove” means that the data-loader must remove the node representing the element from the XFA-data-DOM, unless it is the root node. In addition, if that node was removed from the XFA-data-DOM, the data-loader must remove the node corresponding to the element from the XML-data-DOM, so that if a new XML-data-document is subsequently generated it does not contain the element.
- “dataGroup” means that the data-loader must replace the data-value node in the XFA-data-DOM with a data-group node, unless the node's parent is a data-value. This does not affect the XML-data-DOM.
- “dataValue” means that the data-loader must retain the data-value node representing the element.

Invoking the extended-mapping-rule with “dataValue” produces the same results as the default behavior for empty elements as described in section [“Default Mapping Rules, Data-Value Elements, Empty-Element Data-Values”](#).

As described above, an “ifEmpty” operation will be suppressed if it is trying to remove the root node or if it is trying to make a data-group the child of a data-value. When an operation is suppressed in this way, the data-loader should not

issue an error message because the same operation may operate legitimately on elements with the same name elsewhere in the XML-data-document, hence this is expected to be a common occurrence.

Consider the following XML-data-document:

```
<item>
  <book>
    <ISBN></ISBN>
    <title>Introduction to XML</title>
    <author>
      <firstname></firstname>
      <lastname></lastname>
    </author>
  </book>
</item>
```

With “dataValue” empty element handling, expressed in theXFA configuration document as:

```
<transform ref="ISBN">
  <ifEmpty>dataValue</ifEmpty>
</transform>
<transform ref="firstname">
  <ifEmpty>dataValue</ifEmpty>
</transform>
<transform ref="author">
  <ifEmpty>dataValue</ifEmpty>
</transform>
```

the “author” data-group is not changed to a data-value because it has children, hence is not empty. The result of the mapping is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (ISBN) = ""]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (firstname) = ""]
      [DataValue (lastname) = ""]
```

The above mapping is identical to that produced by default empty element handling, however it has a higher priority. Default empty element handling can be overridden by forced data-group mapping as described below in the section [“Force Data-Value or Data-Group Mapping”](#). By contrast “dataValue” empty element handling takes precedence over forced data-group mapping. This is a consequence of coming earlier in the processing sequence.

With “dataGroup” empty element handling, expressed in theXFA configuration document as:

```
<transform ref="ISBN">
  <ifEmpty>dataGroup</ifEmpty>
</transform>
<transform ref="firstname">
  <ifEmpty>dataGroup</ifEmpty>
</transform>
<transform ref="author">
  <ifEmpty>dataGroup</ifEmpty>
</transform>
```

the result of the mapping in the XFA-data-DOM is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataGroup (ISBN)]
      [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataGroup (firstname)]
      [DataValue (lastname) = ""]
```

With “ignore” empty element handling, expressed in the XFA configuration document as:

```
<transform ref="ISBN">
  <ifEmpty>ignore</ifEmpty>
</transform>
<transform ref="firstname">
  <ifEmpty>ignore</ifEmpty>
</transform>
<transform ref="author">
  <ifEmpty>ignore</ifEmpty>
</transform>
```

the “author” data-group is not ignored because it has a child and is therefore not empty. Hence the result of the mapping in the XFA-data-DOM is as follows:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (title) = "Introduction to XML"]
    [DataGroup (author)]
      [DataValue (lastname) = ""]
```

If on the other hand the XFA configuration document contains:

```
<transform ref="ISBN">
  <ifEmpty>ignore</ifEmpty>
</transform>
<transform ref="firstname">
  <ifEmpty>ignore</ifEmpty>
</transform>
<transform ref="lastname">
  <ifEmpty>ignore</ifEmpty>
</transform>
<transform ref="author">
  <ifEmpty>ignore</ifEmpty>
</transform>
```

the “firstname” and “lastname” data-values are both deleted from the XFA-data-DOM, leaving the “author” data-group empty, so it in turn is deleted. This results in the following mapping in the XFA-data-DOM:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (title) = "Introduction to XML"]
```

The data-loader must ensure that the order of declaration of “ifEmpty” rules in the configuration document does not affect the outcome of empty node processing. One way to do this is to perform an “ifEmpty” pass over the XFA-data-

DOM in bottom-up order, that is to perform the “ifEmpty” processing on the return back from a leaf node towards the root.

Alternatively “ignore” can be applied to the whole document as follows:

```
<transform ref="">
  <ifEmpty>ignore</ifEmpty>
</transform>
```

and in this case the recursive deletion of all empty nodes results in the following mapping in the XFA-data-DOM:

```
[DataGroup (item)]
  [DataGroup (book)]
    [DataValue (title) = "Introduction to XML"]
```

With “remove” in place of “ignore”, the result of the mapping in the XFA-data-DOM is the same in every case as for “ignore” (above), but for every node deleted from the XFA-data-DOM the corresponding node in the XML-data-DOM is also deleted.

4.12 Force Data-Value or Data-Group Mapping

This specification provides an extended mapping rule that can be used to force the creation of data-values and data-groups from a particular element within the document range. For instance, this is useful in circumstances where the structure of the original XML-data-document is not considered to be useful information; only the data-value content is desired.

Unlike all the other extended-mapping-rules, this rule is invoked from within the XML-data-document. The rule may be invoked for any element within the document range by placing a particular attribute in the element. The name of the attribute must be “dataNode” and it must belong to the namespace “<http://www.xfa.org/schema/xfadata/1.0/>”. The attribute is defined as:

```
dataNode="dataValue" | "dataGroup"
```

where:

- “dataValue” means the data-loader must map the associated element to a data-value according the rules defined by this specification
- “dataGroup” means the data-loader must map the associated element to a data-group according the rules defined by this specification

The data-loader must not permit this extended mapping rule to be used to violate the relationships between data-groups and data-values, as described by this specification. For instance, as a result of a forced mapping of an element to a data-value, the element's descendant elements must not be considered as candidate data-groups, because data-values can enclose only other data-values. Similarly, a forced mapping attempt to a data-group must not succeed where the data-group would be enclosed by a data-value, again because data-values can enclose only other data-values. Any attempt to force the mapping of an element that would violate the relationships between data-groups and data-values must be detected by the data-loader and the request refused.

The following examples illustrate the usage of this extended mapping rule.

Consider the following example:

```
<book xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <ISBN xfa:dataNode="dataGroup">15536455</ISBN>
  <title>Introduction to XML</title>
  <author xfa:dataNode="dataValue"
><firstname>Charles</firstname
><lastname>Porter</lastname
></author>
  <desc xfa:dataNode="dataGroup">Basic primer on <keyword>XML</keyword
> technology.</desc>
</book>
```

The result of mapping this XML-data-document is as follows:

```
[DataGroup (book)]
  [DataGroup (ISBN)]
    [DataValue () = "15536455"]
  [DataValue (title) = "Introduction to XML"]
  [DataValue (author) = "CharlesPorter"]
    [DataValue (firstname) = "Charles"]
    [DataValue (lastname) = "Porter"]
  [DataGroup (desc)]
    [DataValue () = "Basic primer on "]
    [DataValue (keyword) = "XML"]
    [DataValue () = " technology."]
```

In the above example, the XML for the “author” element has been modified from previous similar examples which had the “firstname” and “lastname” on separate lines and indented to aid legibility. This white space was removed for this example in order to produce the mapping above. See the section “[Whitespace Handling](#)” for more information on white space handling inside data-values.

When this extended-mapping-rule causes an element containing character-data, which would otherwise be mapped to a data-value, to be mapped instead to a data-group, the data-loader must insert an unnamed data-value as child of the data-group to hold the character-data. For example, given the following XML-data-document:

```
<book xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <ISBN>15536455</ISBN>
  <title xfa:dataNode="dataGroup">Introduction to XML</title>
</book>
```

After loading with default-mapping-rules the XFA-data-DOM must contain:

```
[DataGroup (book)]
  [DataValue (ISBN) = "15536455"]
  [DataGroup (title)]
    [DataValue () = "Introduction to XML"]
```

A common use for this extended-mapping-rule is to ensure that an empty element that represents a data-group is not mapped to a data-value as would occur based on the default mapping rules described in the section “[Empty-Element Data-Values](#)”. Consider the following example:

```
<book>
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <author/>
</book>
```

We know from previous examples that the element “author” is a data-group element that usually encloses data-value elements “firstname” and “lastname”; however, in this specific example, the “author” element is empty and therefore would, by-default, map to a data-value.

To ensure that the “author” element maps to a data-group the following example uses this extended mapping rule:

```
<book xmlns:xfa="http://www.xfa.org/schema/xfadata/1.0/">
  <ISBN>15536455</ISBN>
  <title>Introduction to XML</title>
  <author xfa:dataNode="dataGroup"/>
</book>
```

Note however that this extended-mapping-rule only overrides the default-mapping-rule for empty elements. It must not override the extended-mapping-rule for empty elements which is described in the section “[Simplify Structure Based on Empty Content](#)” if the extended-mapping-rule for empty elements is declared in the XFA configuration document.

4.13 Replace Element Names with Attribute Values

Some XML-data-documents use the same element-type for different families of elements, with an attribute indicating the element family. This style of XML is referred to in the following paragraphs as “inverted XML”. (“Inverted” is not meant pejoratively.) For example, a document which conventionally would contain:

```
<?xml version="1.0"?>
<directory>
  <address>
    <street>10 King</street>
  </address>
</directory>
```

could be expressed in inverted XML as:

```
<?xml version="1.0"?>
<directory>
  <item nodename="address">
    <item nodename="street">10 King</item>
  </item>
</directory>
```

This section describes an extended-mapping-rule that can be used to alter the handling of data expressed as inverted XML. This affects the XFA-data-DOM but has no effect on the XML-data-DOM.

By-default, as described in [Default-Mapping-Rules](#), the “name” property of each node in the XFA-data-DOM is copied from the local part of the element-type of the corresponding element in the XML-data-document. This extended-mapping-rule provides for behaviors in which the “name” property of a nominated element is taken from the value of its nominated attribute.

The XFA-configuration-DOM may include one or more “transform” elements, each of which may enclose a “nameAttr” element that overrides the default behavior. See [[Configuration](#)] for the full schema. The behavior specified by the “nameAttr” element must override the default behavior for elements matching the “ref” property of the “transform” element. For each such element which has an attribute with the given name that has a non-empty value, the data-loader must copy the value of the nominated attribute into the “name” property of the associated node in the XFA-data-DOM. For such elements the data-loader must not load the nominated attribute as a data-value even when attribute loading is enabled, as described in “[Map Attributes to Data-Values](#)”; this applies even if the attribute has an empty value.

For the example data above, the following fragment in the XFA configuration document would cause the inverted XML to be represented in the XFA-data-DOM the same way as the conventional XML:

```
<transform ref="item">
  <nameAttr>nodename</nameAttr>
</transform>
```

The XFA configuration document must not declare multiple different “nameAttr” mappings for the same value of “ref”. For example, the following fragment from an XFA configuration document illegally declares two different “nameAttr” mappings for “foo”:

```
<transform ref="foo">
  <nameAttr>x</nameAttr>
</transform>
<transform ref="foo">
  <nameAttr>y</nameAttr>
</transform>
```

This is forbidden because it can lead to paradox. For example, consider the above illegal configuration fragment and the following XML-data-document:

```
<foo x="abc" y="def">some content</foo>
```

If the configuration was legal the data-loader would be required to map the data-value simultaneously to both names “abc” and “def”, which is impossible.

If the value supplied by the nominated attribute is not a valid XML node name the behavior of the data-loader is implementation-defined.

Note that this mapping is data-dependent in that the data determines what node names result. Consequently it is not reversible; inverted XML can be loaded into the XFA-data-DOM but cannot be copied back into the XML-data-DOM. (Ways to support this are under consideration for a future version of this specification.)

4.14 Rename Nodes Based On Element Name

This section describes an extended-mapping-rule that can be used to cause the substitution of new names for names of elements from the XML-data-document. This substitution affects the XFA-data-DOM but does not affect the XML-data-DOM.

By-default, as described in [Default-Mapping-Rules](#), the data-loader copies the local parts of element-types from elements into the corresponding “name” properties of nodes in the Data DOM. This extended-mapping-rule provides for an alternate behavior in which the local (non-namespace) part of each element-type is matched against a set of (name, substitute) pairs and, if it is in the set, the substitute string is copied into the “name” property of the node in its place. Matching is case-sensitive in keeping with XML norms. If the local part of the element-type does not match any member of the set it is used verbatim, as in the default case.

The XFA-configuration-DOM may include one or more “transform” elements, each of which may include one “rename” elements that override the default behaviour. See [\[Configuration\]](#) for the full schema. The behavior specified by the “rename” must override the default behavior for every element in the XML-data-document with the local part of the element-type matching the “ref” property of the “transform”. It must also override the default behavior for every attribute in the XML-data-document with the local part of the name matching the “ref” property of the “transform”, when attributes are being mapped to data-values as described in [“Map Attributes to Data-Values”](#).

If the “rename” element is non-empty the data-loader must set the “name” property of the corresponding node in the XFA-data-DOM to the content of the “rename” element. The content of the “rename” element must be a valid local name as specified by [\[XMLNAMES\]](#).

Note that name mapping only applies to the local part of a name. It must not be affected by and must not affect namespace designators.

Consider the following XML-data-document:

```

<item>
  <book>
    <ISBN>15536455</ISBN>
    <title>Introduction to XML</title>
    <author>
      <firstname>Charles</firstname>
      <lastname>Porter</lastname>
    </author>
  </book>
</item>

```

With name mapping for “author” and “title” elements, expressed in the XFA configuration document as:

```

<transform ref="author">
  <rename>writer</rename>
</transform>
<transform ref="title">
  <rename>bookName</rename>
</transform>

```

the result of the mapping in the XFA-data-DOM is as follows:

```

[DataGroup (item)]
  [DataGroup (book)]
    [DataGroup (ISBN)]
    [DataValue (bookName) = "Introduction to XML"]
    [DataGroup (writer)]
      [DataValue (firstname) = "Charles"]
      [DataValue (lastname) = "Porter"]

```

4.15 XSLT Postprocessing

This section describes an extended-mapping-rule that can be used to modify data being written to a new XML-data-document under control of an XSLT script [[XSLT](#)].

The XFA-configuration-DOM may include an “outputXSL” element. If present, the “outputXSL” element must contain a “uri” element that nominates the XSLT script. See [[Configuration](#)] for the full schema. When the “outputXSL” element is supplied the data-loader must execute the script after applying all other transformations to the data.

Note that “debug” may not be used inside “outputXSL”.

For example, an XFA-configuration-DOM includes the following fragment:

```

<outputXSL>
  <uri>out.xslt</uri>
</outputXSL>

```

This causes the output XML document from the data-loader to be passed to an XSLT interpreter, along with the local file “out.xslt” which contains an XSLT style sheet. The style sheet supplies all required additional configuration such as the destination of the transformed document.

5 Update Processing

The XFA application may make edits to the XFA-data-DOM during processing. These edits may include insertions and deletions of nodes, moving nodes, and updating the content of nodes. The XFA-data-DOM must ensure that all such edits are propagated to the XML-data-DOM so that the XFA-data-DOM and the XML-data-DOM stay synchronized. The XFA-data-DOM should detect and refuse to carry out edits that would lead to the production of invalid XML during data-unloading. For example, if the XFA application attempts to create a dataValue with the same name as a sibling and both siblings have their “contains” properties set to “metadata”, the XFA-data-DOM should refuse to create the new dataValue on the grounds that attributes of the same element must have unique names.

When the XFA application updates the value property of a dataValue node in the XFA-data-DOM representing rich-text, the XFA-data-DOM must break the association between the dataValue node and the rich-text in the XML-data-DOM by setting the “namespace” and “namePrefix” properties to null or to the empty string. The result must be that the dataValue node becomes an ordinary plain-text node.

The XFA-data-DOM may provide a facility for loading data from an XML-data-document into an XFA-data-DOM that already contains nodes. This specification refers to such a load as an append-load. When carrying out an append-load the data-loader must follow the same rules described elsewhere in this specification, except that:

- The start element for the load must be determined at invocation time, without regard to “startNode” option of the XFA Configuration DOM.
- The new data must be appended as a subtree, the root of which is the child of an existing node. The node to which the new subtree is appended must be determined at invocation time.

Note that one use for an append-load is to insert rich-text into the XFA-data-DOM during processing. No other mechanism for doing so is defined by this specification.

6 Unload Processing

The data-unloader must provide a facility for creating or updating an XML-data-document that represents the XML-data-DOM.

When invoked, the data-unloader must produce an XML-data-document which reflects the contents of the XML-data-DOM, as of the moment at which the data-unloader was invoked.

When unloading the XML-data-DOM, the XML-data-document produced by the data-unloader must be such that the data can make a round-trip back to the XFA-data-DOM. This means that when the new document is subsequently loaded into an empty XFA-data-DOM using all the same data-loader configuration options, the resulting XFA-data-DOM must be indistinguishable from the original XFA-data-DOM at the moment the data-unloader was invoked.

The data-unloader must encode characters using XML character references when necessary to conform to [\[XML\]](#). In lieu of character references it may use the special strings defined by [\[XML\]](#) which includes “<” for “<” (less-than sign), “>” for “>” (greater-than sign), and “&” for “&” (ampersand). Inside attribute values it may also use “"” for “” (quotation mark) and “'” for “'” (apostrophe) as defined by [\[XML\]](#).

The data-unloader may insert XML-style comment(s) into the output document. It should insert a comment near the beginning identifying itself and its version number.

6.1 Unloading Node Type Information

In order to support round-tripping the data-unloader must when necessary insert attributes in particular elements to cause them to be loaded as the correct type of node, as described in “[Force Data-Value or Data-Group Mapping](#)”.

The need for this attribute can arise because of deletions during processing. Consider the following excerpt from an XML-data-document:

```
<author>
  <firstname>Charles</firstname>
  <lastname>Porter</lastname>
</author>
```

When loaded into the XFA-data-DOM using default-mapping-rules the result is:

```
[dataGroup (author)]
  [dataValue (firstname) = "Charles"]
  [dataValue (lastname) = "Porter"]
```

Suppose that during processing the “firstname” and “lastname” dataValue nodes are deleted. The result is that the XFA-data-DOM contains:

```
[dataGroup (author)]
```

By-default empty elements are loaded as data-values. To prevent this, the data-unloader writes out the “author” element with an attribute that marks it as a dataGroup.

```
<author xmlns:xfa="http://www.xfa.org/schema/xfadata/1.0/"
  xfa:dataNode="dataGroup" />
```

Similarly if the configuration options are such that an empty element would be loaded as a dataGroup, but the element is being written to represent the content of a dataValue, the data-unloader writes out the element with an “xfa:datasets_dataNode” attribute having a value of “dataValue”. For example, suppose default-mapping-rules are in force and the XML-data-DOM (after some processing) contains:

```
[dataValue (foo) = "xyz"]
  [dataValue (bar) = "xyz"]
```

The node “foo” corresponds to an element containing nothing but another element, but such elements are normally loaded as data-groups. Yet “foo” is a data-value. When the new XML-data-document is created the data-unloader adds an attribute to mark “foo” as a data-value.

```
<foo xmlns:xfa="http://www.xfa.org/schema/xfadata/1.0/"
  xfa:dataNode="dataValue"><bar>xyz</bar></foo>
```

The rules stated above ensure that every implementation produces logically equivalent output given the same inputs. Logical equivalence includes exact character-for-character reproduction of the content of elements that map (or would map, if they were within the document-range) to data-values, including attribute values. However it does not include white space within the character-data of elements that map (or would map) to data-groups. Hence, the data-unloader may insert white space characters and newlines within elements representing data-groups. This is useful for improving readability. When the output XML-data-document is loaded into a new XML-data-DOM the new XML-data-DOM does not necessarily have the same content as the original XML-data-DOM, however the XFA-data-DOM that derives from it has the same content as the original XFA-data-DOM.

7 Bibliography

THIS BIBLIOGRAPHY PROVIDES details on books and documents, from both Adobe Systems and other sources, that are referred to in this specification.

Resources from Adobe Systems Incorporated

All of these resources from Adobe Systems are available on the Adobe Solutions Network (ASN) Developer Program site on the World Wide Web, located at

<http://partners.adobe.com/asn/developer/>

Document version numbers and dates given in this Bibliography are the latest at the time of publication; more recent versions may be found on the Web site.

The ASN can also be contacted as follows:

Adobe Solutions Network
Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704

(800) 685-3510 (from North America)
(206) 675-6145 (from other areas)

acrodevsup@adobe.com

[Adobe Patent Clarification Notice]

“*Adobe Patent Clarification Notice*”. Available on the Legal Notices page of the ASN Developer Program Web site.

[Configuration]

“*Configuration 2.0 Information Specification*”, Adobe Systems Incorporated, October 2003.
Available from the <http://partner.adobe.com> site.

[Data Binding]

“*Data Binding 2.0 Specification*”, Adobe Systems Incorporated, October 2003.
Available from the <http://partner.adobe.com> site.

[SOM]

“*Scripting Object Model Expression Specification*”, Adobe Systems Incorporated, October 2003.
Available from the <http://partner.adobe.com> site.

[Template]

“*Template Specification*”, Adobe Systems Incorporated, October 2003
Available from the <http://partner.adobe.com> site.

[TextHandling]

“*Data Text Handling Version 2.0 Specification*”, Adobe Systems Incorporated, October 2003.
Available from the <http://partner.adobe.com> site.

Resources from other sources

[POSIX.1]

“*ISO/IEC 9945-1:1990 Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]*”, Institute of Electrical and Electronics Engineers, Inc, 1990.

- [RFC2119]
“RFC2119: Key words for use in RFCs to Indicate Requirement Levels”, S. Bradner, March 1997.
Available at <http://www.ietf.org/rfc/rfc2119.txt>
- [RFC2046]
“RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types”, N. Freed, N. Borenstein, November 1996.
Available at <http://www.ietf.org/rfc/rfc2046.txt>
- [Unicode Preface]
Preface, “The Unicode Standard, Version 3.0”, The Unicode Consortium, 2000.
Available at <http://www.unicode.org/uni2book/Preface.pdf>
- [URI]
“RFC2396: Uniform Resource Identifiers (URI): Generic Syntax”, T. Berners-Lee, R. Fielding, L. Masinter, August 1998.
This document updates RFC1738 and RFC1808.
Available at <http://www.ietf.org/rfc/rfc2396.txt>
- [XHTML 1.0]
“XHTML™ 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0”, S. Pemberton, et al., January 2000
Available at <http://www.w3.org/TR/xhtml1>
- [XML]
“Extensible Markup Language (XML) 1.0 Specification”, T. Bray, J. Paoli, C. M. Sperberg-McQueen, February 1998.
Available at: <http://www.w3.org/TR/REC-xml>
- [XMLDOM2]
“Document Object Model (DOM) Level 2 Specification: Version 1.0”, World Wide Web Consortium, 1999.
Available at <http://www.w3.org/TR/DOM-Level-2/>
- [XMLNAMES]
“Namespaces in XML”, T. Bray, D. Hollander, A. Layman, 14 January 1999.
XML namespaces provide a simple method for qualifying names used in XML documents by associating them with namespaces identified by URI.
Available at: <http://www.w3.org/TR/REC-xml-names>
- [XML-Schema-Part1]
“XML Schema, Part 1: Structures”, H. S. Thompson, et al., 2001.
Available at <http://www.w3.org/TR/xmlschema-1/>
- [XSLT]
“XSL Transformations (XSLT) Version 1.0”, James Clark, ed., November 1999.
Available at <http://www.w3.org/TR/xslt>