ISO/IEC 29500-2:201x

Office Open XML File Formats — Open Packaging Conventions

Working Draft WD3.5

2018-~~07~~08-~~10~~20

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 29500-2 was prepared by ISO/IEC JTC 1, Information technology, Subcommittee SC 34, Document description and processing languages.

This fourth edition cancels and replaces the third edition (ISO/IEC 29500-2:2012).

The major changes from the previous edition include:

- Removed the allowance for media type to be an empty string, as this conflicts with the definition of media type in RFC 2046 and the existing regular expression defined in Annex C.
- Dropped requirements around streaming consumption.
- xx

Commented [rcj1]:

The major changes in the third edition include:

- Added new terms *byte*, *id*, *relationship type*, *source part*, *target part*, and *unique identifier*, and removed the term *well-known part*.
- Removed subclause §9.2.2, “Fragments”
- Added subclause §C.2, “Data Descriptor Signature”
- Applied changes to resolve numerous Defect Reports

There were no major changes in the second edition.

ISO/IEC 29500 consists of the following parts, under the general title *Information technology — Document description and processing languages — Office Open XML File Formats*:

- *Part 1: Fundamentals and Markup Language Reference*
- *Part 2: Open Packaging Conventions*
- *Part 3: Markup Compatibility and Extensibility*
- *Part 4: Transitional Migration Features*

Annexes A, B, C, and E form a normative part of this Part of ISO/IEC 29500. Annexes D, F, and G are for information only.

This Part of ISO/IEC 29500 includes two annexes (Annex C and Annex D) that refer to data files provided in electronic form.

The document representation formats defined by this Part are different from the formats defined in the corresponding Part of ECMA-376:2006. Some of the differences are reflected in schema changes, as shown in Annex G of this Part.

Introduction

ISO/IEC 29500 specifies a family of XML schemas, collectively called *Office Open XML*, which define the XML vocabularies for word-processing, spreadsheet, and presentation documents, as well as the packaging of documents that conform to these schemas.

The goal is to enable the implementation of the Office Open XML formats by the widest set of tools and platforms, fostering interoperability across office productivity applications and line-of-business systems, as well as to support and strengthen document archival and preservation, all in a way that is fully compatible with the existing corpus of Microsoft Office documents.

Information technology — Document description and processing languages — Office Open XML File Formats

Part 2:
Open Packaging Conventions

1 Scope

This Part of ISO/IEC 29500 defines a set of conventions for packaging one or more interrelated byte stream (part) as a single resource (package). These conventions are applicable not only to Office Open XML specifications as described in Parts 1 and 4 of this Standard, but also to other markup specifications.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

American National Standards Institute, *Coded Character Set — 7-bit American Standard Code for Information Interchange*, ANSI X3.4, 1986.

ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*.

ISO/IEC 9594-8 | ITU-T Rec. X.509, *Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks*.

ISO/IEC 10646, *Information technology — Universal Coded Character Set (UCS)*.

ISO/IEC 29500-3, *Information technology — Document description and processing languages — Office Open XML File Formats, Part 3: Markup Compatibility and Extensibility.*

ISO 15836-1, *Information and documentation — The Dublin Core metadata element set, Part 1: Core elements*

Dublin Core Terms Namespace. http://purl.org/dc/terms/

Commented [MM2]: If an IS for DCMI terms is published, we can reference it.

RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, The Internet Society, N. Freed and N. Borenstein, 1996, http://www.ietf.org/rfc/rfc2046.txt.

RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax*, The Internet Society, Berners-Lee, T., R. Fielding, and L. Masinter, 2005, http://www.ietf.org/rfc/rfc3986.txt.

RFC 3987 *Internationalized Resource Identifiers (IRIs)*, The Internet Society, Duerst, M. and M. Suignard, 2005, http://www.ietf.org/rfc/rfc3987.txt.

RFC 5234 *Augmented BNF for Syntax Specifications: ABNF*, The Internet Society, D. Crocker and P.Overell, (editors), 2008, http://www.ietf.org/rfc/rfc5234.txt.

RFC 7231 *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, The Internet Society, R. Fielding and J. Reschke, 2014, http://www.ietf.org/rfc/rfc7231.txt.

The Unicode Consortium. The Unicode Standard, http://www.unicode.org/standard/standard.html.

XML, Tim Bray, Jean Paoli, Eve Maler, C. M. Sperberg-McQueen, and François Yergeau (editors). *Extensible Markup Language (XML) 1.0, Fourth Edition*. World Wide Web Consortium. 2006. http://www.w3.org/TR/2006/REC-xml-20060816/. [Implementers should be aware that a further correction of the normative reference to XML to refer to the 5th Edition will be necessary when the related Reference Specifications to which this International Standard also makes normative reference, and which also depend upon XML, such as XSLT, XML Namespaces and XML Base, are all aligned with the 5th Edition.]

XML Namespaces, Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin (editors). *Namespaces in XML 1.0 (Third Edition)*, 8 December 2009. World Wide Web Consortium. http://www.w3.org/TR/2009/REC-xml-names-20091208/

XML Base, W3C Recommendation, 28 January 2009. https://www.w3.org/TR/2009/REC-xmlbase-20090128/

XML Schema Part 1: Structures, W3C Recommendation, 28 October 2004. https://www.w3.org/TR/xmlschema-1/

XML Schema Part 2: Datatypes, W3C Recommendation, 28 October 2004. https://www.w3.org/TR/xmlschema-2/

XML-Signature Syntax and Processing, W3C Recommendation, 12 February 2002. http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

ZIP File Format Specification from PKWARE, Inc., version 6.2.0 (2004), as specified in http://www.pkware.com/documents/APPNOTE/APPNOTE_6.2.0.txt. [*Note*: The supported compression algorithm is inferred from Tables C-3 and C-4 in Annex B. *end note*]

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply. Other terms are defined where they appear in italic typeface. Terms explicitly defined in this Part of ISO/IEC 29500 are not to be presumed to refer implicitly to similar terms defined elsewhere.

The terms *base URI*, *relative reference, URI scheme*, *authority*, *fragment*, *path*, *query*, and *segment* are used in accordance with RFC 3986. The term *media type* is used in accordance with RFC 2046.

3.1 Basics

3.1.1
byte
sequence of 8 bits treated as a unit

3.1.2
stream
linearly ordered sequence of bytes

3.1.3
behavior
external appearance or action

3.1.4
behavior, implementation-defined
behavior, application-defined
unspecified behavior where each implementation shall document that behavior, thereby promoting predictability and reproducibility within any given implementation

3.1.5
behavior, unspecified
behavior where this document imposes no requirements

3.2 Abstract package model

3.2.1
part
stream with a name, a MIME media type and associated common properties

3.2.2
package, abstract
logical entity that holds a collection of parts and relationships

3.2.3
relationship
a package relationship or a part relationship

3.2.4
relationship, package
connection from a package to a specific part in the same package, or to an external resource

3.2.5
relationship, part
connection from a part in a package to another part in the same package, or to an external resource

3.2.6
source
part or package from which a connection is established by a relationship

3.2.7
target
part or external resource to which a connection is established by a relationship

3.2.8
relationship type
absolute IRI for specifying the role of a relationship

3.2.9
Relationships part
part containing an XML representation of relationships

3.2.10
package model, abstract
abstract model that define abstract packages

3.2.11
growth hint
suggested number of bytes to reserve for a part to grow in place

3.2.12
pack scheme
URI scheme that allows IRIs to be used as a uniform mechanism for addressing parts within a package

3.2.13
pack IRI
IRI that conforms to the pack scheme

3.2.14
part name
a Unicode string that uniquely identifies a part within a package and satisfies the requirements in §8.2.2.2path component of a pack URI

3.2.15
relationship identifier
string that consists of XML name characters and uniquely identifies a relationship among those from the same source

3.2.16
target mode
mode of resolution of relative references to targets

3.3 Physical package model

3.3.1
physical format
specific file format, or other persistence or transport mechanism

3.3.2
physical package
the result of a mapping an abstract package to a physical format

3.3.3
physical package model
a pair of a physical format and a mapping between the abstract package model and that physical format

3.3.4
piece
portion of a part.

3.3.5
logical item
a non-interleaved part, the non-interleaved Media Types stream, a piece of an interleaved part, or a piece of the interleaved Media Types stream

3.3.6
physical package item
an atomic set of data in a physical package

3.3.7
ZIP item
an atomic set of data in a ZIP file that becomes a file when the archive is uncompressed

3.3.8
ZIP file
ZIP file as defined in the ZIP file format specification

3.3.9
simple ordering
defined ordering for laying out the parts in a package in which all the bits comprising each part are stored contiguously

3.3.10
interleaved ordering
defined ordering for laying out the parts in a package in which parts are broken into pieces and “mixed-in” with pieces from other parts

3.4 Digital signature and thumbnail

3.4.1
signature policy
application-defined policy that specifies what configuration of parts and relationships shall or might be included in a signature and what additional behaviors are required for generating and validating signatures following that signature policy

3.4.2
thumbnail
small image that is a graphical representation of a part or the package as a whole

3.5 Implementations

3.5.1
package implementer
software that implements the physical input-output operations to a package according to the requirements and recommendations of this document

4 Notational Conventions

The following typographical conventions are used in ISO/IEC 29500:

1) The first occurrence of a new term is written in italics. [*Example*: The text in ISO/IEC 29500 is divided into *normative* and *informative* categories. *end example*]
2) In each definition of a term in §3 (Terms and Definitions), the term is written in bold. [*Example*: **behavior** — External appearance or action. *end example*]
3) The tag name of an XML element is written using a distinct style and typeface. [*Example*: The bookmarkStart and bookmarkEnd elements specify … *end example*]
4) The name of an XML attribute is written using a distinct style and typeface. [*Example*: The dropCap attribute specifies … *end example*]
5) The value of an XML attribute is written using a constant-width style. [*Example*: The attribute value of `auto` specifies … *end example*]
6) The qualified or unqualified name of a simple type, complex type, or base datatype is written using a distinct style and typeface. [*Example*: The possible values for this attribute are defined by the ST_HexColor simple type. *end example*]

5 General Description

Commented [rcj3]: Needs to be checked once all edits are completed.

This document is divided into the following subdivisions:

1) Front matter;
2) Overview;
3) Main body;
4) Annexes

Examples are provided to illustrate possible forms of the constructions described. References are used to refer to related clauses. Notes are provided to give advice or guidance to implementers or programmers. Annexes provide additional information and summarize the information contained in this document.

The following form the normative part of this document:

- Introduction
- Clauses 1–5, and xx–12
- Annex A–Annex C
- Annex E

The following form the informative part of this document:

- Clause 6
- Annex D
- Annex F–~~Annex G~~Annex G
- All notes
- All examples

Except for whole clauses or annexes that are identified as being informative, informative text that is contained within normative text is indicated in the following ways:

1) [*Example*: code fragment, possibly with some narrative … *end example*]
2) [*Note*: narrative … *end note*]
3) [*Rationale*: narrative … *end rationale*]

6 Conformance

A document is of conformance class OPC if it obeys all syntactic constraints specified in this Part of ISO/IEC 29500.

OPC conformance is purely syntactic.

7 Overview

This clause is informative.

This document describes an abstract package model (§8) and a physical package model (§9) for the use of XML, Unicode, ZIP, and other available technologies and specifications to organize the content and resources of a document within a package. The package structure is intended to support the organization of constituent resources for various applications and categories of content. An example package is shown in Annex H.

In addition, this document defines common services that can be included in a package, such as Core Properties and Digital Signatures.

The *abstract package model* is a package abstraction that holds a collection of *parts* and relationships. The parts are composed, processed, and persisted according to a set of rules. Parts can have relationships to other parts or external resources, and the package as a whole can have relationships to parts it contains or to external resources. The abstract package model specifies how the parts of a package are named and related. Parts have MIME media types and are uniquely identified using the well-defined naming rules provided in this document.

The *physical package model* defines the mapping of the components of the abstract package model to the features of a specific physical format, namely a ZIP file.

This document also describes certain features that might be supported in a package, including *core properties* for package metadata, a thumbnail (term 3.4.2) for graphical representation of a package, and *digital signatures* of package contents. Because this document might evolve, packages are designed to accommodate extensions and to support compatibility goals in a limited way. The versioning and extensibility mechanisms described in ISO/IEC 29500-3 support compatibility between software systems based on different versions of this document while allowing package creators to make use of new or proprietary features.

This document specifies requirements for documents. Conformance requirements are identified throughout the text of this document. A formal conformance statement is given in §6.

End of informative text.

8 Abstract Package Model

8.1 General

This subclause is informative.

This clause introduces abstract packages (term 3.2.2) in terms of parts (term 3.2.1, §8.2) and relationships (term 3.2.3, §8.5). It also introduces the pack scheme (term 3.2.12, §8.3.2).

The purpose of the abstract package is to aggregate constituent components of a document (or other type of content) into a single object. For example, an abstract package holding a document with a picture might contain two parts: an XML markup part representing the document and another part representing the picture.

An example abstract package is shown in H.2.

End of informative text.

8.2 Parts

8.2.1 General

This subclause is informative.

Parts (term 3.2.1) are analogous to a file in a file system or to a resource on an HTTP server.

End of informative text.

8.2.2 Part Names

8.2.2.1 General

A part shall have a *part name*, which shall uniquely identify a part within an abstract package.

8.2.2.2 Syntax

A part name shall be a Unicode string that matches the following production rules in the ABNF syntax defined in RFC 5234

```
part_name = 1*( "/" isegment-nz )
isegment-nz = <isegment-nz, see RFC3987, Section 2.2>
```

and further satisfies the following constraints.

- No I18N segments shall contain percent-encoded forward slash (“/”), or backward slash (“\”) characters.
- No I18N segments shall contain percent-encoded characters that match the non-terminal iunreserved in RFC 3987.

- No I18N segments shall end with a dot (".") character.

where an I18N segment is a Unicode string that matches the non-terminal isegment-nz and percent-encoding represents a character by the percent character "%" followed by two hexadecimal digits, as specified in RFC 3986.

The part name "/_rels/.rels" shall be reserved (§8.5.2.2). Part names in which the second-to-last I18N segment is equivalent to '_rels' and the final segment is equivalent to any string ending with '.rels' shall be reserved (§8.5.2.3).

[*Example*: The part name "/hello/world/doc.xml" contains three path segments, namely, "hello", "world", and "doc.xml". *end example*]

[*Example*: The part name "/é" contains a path segment "é" where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9). *end example*]

[*Note*: Path segments are not explicitly represented as folders in the abstract package model, and no directory of folders exists in the abstract package model. *end note*]

A package implementer is not required to support non-ASCII part names, although doing so is recommended.

8.2.2.3 Part Name Equivalence and Integrity in an Abstract Package

Equivalence of part names shall be determined by ASCII case-insensitive matching. Such matching compares a sequence of code points as if all ASCII code points in the range 0x41–0x5A (A–Z) were mapped to the corresponding code points in the range 0x61–0x7A (a–z). See Character Model for the World Wide Web: String Matching and Searching [1].

The names of two different parts within an abstract package shall not be equivalent.

[*Example*: If an abstract package contains a part named "/a", the name of another part in that abstract package must not be "/a" or "/A". *end example*]

For each part name N and string S, let the result of concatenating N, the forward slash, and S be denoted by N[S]. A part name N1 is said to be *derivable* from another part name N2 if, for some string S, N1 is equivalent to N2[S].

[*Example*: "/a/b" is derivable from "/a", where N is "/a" and S is "b". *end example*]

The name of a part shall not be derivable from the name of another part.

[*Example*: Suppose that an abstract package contains a part named "/segment1/segment2/…/segment*n*". For it not to be derivable, other parts in that abstract package must not have names such as "/segment1", "/SEGMENT1", "/segment1/segment2", "/segment1/SEGMENT2", or "/segment1/segment2/…/segment*n*-1". *end example*]

This subclause further introduces recommendations, so that Unicode Normalization Form C (NFC) or Unicode Normalization Form D (NFD) of part names does not cause part-name collisions. [*Note*: Some implementations of directory structures always apply NFD normalization. *end note*]

The application of NFC or NFD normalization to the names of two different parts within an abstract package should not yield equivalent strings.

[*Example*: If an abstract package contains a part named "/é", where é is 'LATIN SMALL LETTER E' (U+0065) followed by 'COMBINING ACUTE ACCENT' (U+0301), the name of another part in that abstract package should not be "/é", where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9), or "/É", where É is 'LATIN CAPITAL LETTER E WITH ACUTE '(U+00C9). *end example*]

[*Example*: If an abstract package contains a part named "/Å", where Å is 'ANGSTROM SIGN' (U+212B), the name of another part in that abstract package should not be "/Å" where Å is 'LATIN CAPITAL LETTER A WITH RING ABOVE' (U+00C5) because U+212B and U+00C5 are normalized to the same character sequence. *end example*]

A part name N1 is said to be *weakly derivable* from another part name N2 if, for some string S, the result of applying NFC or NFD to N1 is equivalent to the result of applying NFC or NFD to N2[S].

[*Example*: Consider a part name "/é", where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9). Another part name "/é/a", where é is 'LATIN SMALL LETTER E' (U+0065) followed by 'COMBINING ACUTE ACCENT' (U+0301) is weakly derivable from "/é". Another part name "/É/a", where É is 'LATIN CAPITAL LETTER E' (U+0045) followed by 'COMBINING ACUTE ACCENT' (U+0301) is also weakly derivable. *end example*]

The name of a part should not be weakly derivable from the name of another part.

[*Example*: Suppose that an abstract package contains a part named "/é/ Å /foo", where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9) and Å is 'ANGSTROM SIGN' (U+212B). For it not to be weakly derivable, no other parts in that abstract package should have names such as "/É" and "/É/Å", where É is 'LATIN CAPITAL LETTER E' (U+0045) followed by 'COMBINING ACUTE ACCENT' (U+0301) and Å is 'LATIN CAPITAL LETTER A WITH RING ABOVE' (U+00C5). *end example*]

8.2.3 Media types

Each part shall have a MIME media type, as defined in RFC 2046, to identify the type of content in that part, consisting of a top-level media type and a subtype, optionally qualified by a set of parameters. Media types of OPC-specific parts defined in this document shall not contain parameters.

Media types for parts defined in this standard are listed in Annex E.

8.2.4 Growth Hint

The growth hint (term 3.2.11) is an optional common property of a part.

[*Rationale*: Sometimes a part is modified after it is placed in a physical package. Depending on the nature of the modification, the part might need to grow. For some physical formats, this could be an expensive operation and could damage an otherwise efficient physical package. To allow the part to grow in place, moving as few bytes as possible, the growth hint might be used to reserve space in a mapping to a particular physical format. *end rationale*]

8.2.5 XML Usage

XML content in parts and streams defined in this document (specifically, the Media Types stream, the Core Properties part, Digital Signature XML Signature parts, and Relationships parts) shall conform to the following:

1) XML content shall be encoded using either UTF-8 or UTF-16. If any part includes an encoding declaration, as defined in §4.3.3 of the XML 1.0 specification, that declaration shall not name any encoding other than UTF-8 or UTF-16.
2) The XML 1.0 specification allows for the usage of Document Type Definitions (DTDs), which enable Denial of Service attacks, typically through the use of an internal entity expansion technique. As mitigation for this potential threat, DTD declarations shall not be used in the XML markup defined in this document.
3) Validation of the XML content shall be done only after it is processed by an MCE processor as specified in ISO/IEC 29500-3.
4) XML content shall be valid against the corresponding XSD schema defined in this document. In particular, the XML content shall not contain elements or attributes drawn from namespaces that are not explicitly defined in the corresponding XSD unless the XSD allows elements or attributes drawn from any namespace to be present in particular locations in the XML markup.
5) XML content shall not contain elements or attributes drawn from “xml” or “xsi” namespaces unless they are explicitly defined in the XSD schema or by other means described in this document.

8.3 Part Addressing

8.3.1 General

This subclause is informative.

This document provides the pack scheme as a way to use IRIs (RFC 3987) to reference part resources inside a package.

Schemes are represented in an IRI by the prefix before the colon. A well-known example is "http".

An example of an IRI in the pack scheme is:

pack://http%3c,,www.openxmlformats.org,my.container/a/b/foo.xml

The substring between the double slash and the first single slash represents an IRI in the http scheme for a package, transformed to allow embedding within an IRI in the pack scheme.

References from outside of a package are absolute IRIs of the pack scheme, while those from inside are relative IRIs, which are resolved to absolute IRIs of this scheme.

End of informative subclause.

8.3.2 Pack Scheme

This document defines a specific scheme used to refer to parts in a package: the pack scheme. An IRI that uses the pack scheme is called a *pack IRI*.

[*Note*: The pack scheme is a historical scheme in the IANA-maintained registry of schemes located at https://www.iana.org/assignments/uri-schemes/historic/pack. It was a provisional scheme but was changed to a historical scheme due to a mistake (see the next Note) in the registration proposal. *end note*]

The syntax of pack IRIs is defined by the EBNF (see RFC 5234) as follows:

```
pack_IRI  = "pack://" iauthority [ "/" | ipath ]
iauthority     = *( iunreserved | sub-delims | pct-encoded )
ipath     = 1*( "/" isegment )
isegment  = 1*( ipchar )
ipchar = <ipchar, see [RFC3987], Section 2.2>
iunreserved = <iunreserved, see [RFC3987], Section 2.2>
sub-delims = <sub-delims, see [RFC3986], Section 2.2>
pct-encoded = <pct-encoded, see [RFC3986], Section 2.1>
```

The authority component (`iauthority`) contains an embedded IRI that points to a package. (See §8.3.4 for the procedure for transforming the IRI for the package to permit embedding in the pack IRI as the authority component.) The authority component shall not reference a package embedded in another package.

[*Note*: The definition of the authority component requires that the colon character (:) be escaped as %3c. However, in the proposed registration of the pack scheme, an unescaped colon (:) character was mistakenly used. *end note*]

The optional path component (`ipath`) identifies a particular part within the package. When the path component is missing, the resource identified by the pack IRI is the package as a whole.

A pack IRI might have a query component (as specified in RFC 3986). A query component in a pack IRI is not used when resolving the IRI to a part.

A pack IRI might have a fragment component as specified in RFC 3986. If present, this fragment applies to whatever resource the pack IRI identifies.

[*Example*:

Using the pack IRI to identify a part

The following IRI identifies the “/a/b/foo.xml” part within the “http://www.openxmlformats.org/my.container” package resource:

```
pack://http%3c,,www.openxmlformats.org,my.container/a/b/foo.xml
```

end example]

[*Example*:

Equivalent pack IRIs

The following pack IRIs are equivalent:

```
pack://http%3c,,www.openxmlformats.org,my.container
pack://http%3c,,www.openxmlformats.org,my.container/
```

end example]

[*Example*:

A pack IRI with percent-encoded characters

The following IRI identifies the “/c/d/bar.xml” part within the “http://myalias:pswr@www.my.com/containers.aspx?my.container” package:

```
pack://http%3c,,myalias%3cpswr%40www.my.com,containers.aspx%3fmy.container
/c/d/bar.xml
```

end example]

8.3.3 Resolving a Pack IRI to a Resource

The following is an algorithm for resolving a pack IRI to a resource (either a package or a part):

1) Parse the pack IRI into the potential three components: scheme, authority, path, as well as any fragment identifier.
2) In the authority component, replace all commas (“,”) with forward slashes (“/”).
3) Un-percent-encode ASCII characters in the resulting authority component.
4) The resultant authority component shall be a valid IRI for the package as a whole. If it is not, the pack IRI is invalid.
5) If the path component is empty, the pack IRI resolves to the package as a whole and the resolution process is complete.
6) A non-empty path component shall be a valid part name. If it is not, the pack IRI is invalid.
7) The pack IRI resolves to the part with this part name in the package identified by the authority component.

[*Example:*

Resolving a pack IRI to a resource

Given the pack IRI:

```
pack://http%3c,,www.my.com,packages.aspx%3fmy.package/a/b/foo.xml
```

The components:

```
<authority>= http%3c,,www.my.com,packages.aspx%3fmy.package
<path>= /a/b/foo.xml
```

are converted to the package IRI:

```
http://www.my.com/packages.aspx?my.package
```

and the path:

```
/a/b/foo.xml
```

Therefore, this IRI refers to a part named “/a/b/foo.xml” in the package at the following IRI: http://www.my.com/packages.aspx?my.package.

end example]

8.3.4 Composing a Pack IRI

The following is an algorithm for composing a pack IRI from the IRI of an entire package and a part name.

In order to be suitable for creating a pack IRI, the IRI of a package shall conform to RFC 3986 requirements for absolute IRIs.

To compose a pack IRI from the absolute package IRI and a part name, the following steps shall be performed, in order:

1) Remove the fragment identifier from the absolute package IRI, if present.
2) Percent-encode all percent signs (“%”), question marks (“?”), at signs (“@”), colons (“:”) and commas (“,”) in the package IRI.
3) Replace all forward slashes (“/”) with commas (“,”) in the resulting string.
4) Append the resulting string to the string “pack://”.
5) Append a forward slash (“/”) to the resulting string. The constructed string represents a pack IRI with a blank path component.
6) Using this constructed string as a base IRI and the part name as a relative reference, apply the rules defined in RFC 3986 for resolving relative references against the base IRI.

The result of this operation is the pack IRI that refers to the resource specified by the part name.

[*Example:*

Composing a pack IRI

Given the package IRI:

```
http://www.my.com/packages.aspx?my.package
```

and the part name:

```
/a/foo.xml
```

The pack IRI is:

```
pack://http%3c,,www.my.com,packages.aspx%3fmy.package/a/foo.xml
```

end example]

8.3.5 Equivalence

Two pack IRIs are equivalent if:

1) The scheme components are octet-by-octet identical after they are both converted to lowercase; *and*
2) The IRIs, decoded as described in §8.3.3 from the authority components, are equivalent (the equivalency rules by scheme, as specified in RFC 3986); *and*
3) The path components are equivalent part names, as specified in §8.2.2.

[*Note*: In some scenarios, such as caching or writing parts to a package, it is necessary to determine if two pack IRIs are equivalent without resolving them. *end note*]

8.4 Resolving Relative References

8.4.1 General

Relative references in parts shall be resolved as specified in RFC 3986 (§5 Reference Resolution), as extended in RFC 3987 (§6.5 Relative IRI References).

This document introduces no changes to the resolution procedure, but Annex A introduces a preprocessing for generating relative references. ~~[Editor's note: Should Annex A be dropped?]~~

8.4.2 Base IRIs

This subclause defines a procedure for determining base IRIs for resolving relative references within parts in packages.

[*Note*: RFC 3986 (§5.1 Establishing a Base URI) provides four general methods, in order of precedence, for establishing base IRIs for resolving relative references. The procedure in this subclause provides an OPC-specific method corresponding to the second general method (RFC 3986, §5.1.2 Base URI from the Encapsulating Entity). *end note*]

The base IRI depends on where that reference occurs within the package. This subclause covers the case where a relative reference occurs in a part that is not a Relationships part. §8.5.2 covers the case where a relative reference occurs in a Relationships part.

The base IRI shall be the pack IRI created from the IRI of the package and the name of the part within which the relative reference occurs.

[*Example:*

Consider a part `/a/b/foo.xml` in a package available at

```
http://www.mysite.com/my.package
```

The base IRI is

```
pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml
```

end example]

8.4.3 Examples

This subclause is informative.

8.4.3.1 General

This subclause shows examples of resolving relative references. For each example, this subclause considers three cases.

Case 1: the base IRI is a pack IRI, "pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml", which is constructed from an absolute IRI of the package and a part name.

Case 2: the base IRI is a pack IRI, "pack://http%3c,,www.mysite.com,my.package/", which is created from an absolute IRI of the package.

Case 3: the base IRI is the absolute IRI of the package, http://www.mysite.com/my.package.

8.4.3.2 Leading slash: "/b/bar.xml"

Case 1: The base IRI is "pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml".

Since this relative reference begins with the slash character, the path component of the base IRI ("/a/b/foo.xml") is ignored by the algorithm in §5.2.2 of RFC 3986. The scheme and authority of the resulting IRI are the same as those of the base pack IRI. Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/b/bar.xml"
```

Case 2: The base IRI is "pack://http%3c,,www.mysite.com,my.package/"

Likewise, the path component of the base IRI ("/") is ignored. The rest is the same.

Case 3: The base IRI is "http://www.mysite.com/my.package"

Likewise, the path component of the base IRI ("/my.package") is ignored. Thus, the resulting IRI is:

```
"http://www.mysite.com/my.package/b/bar.xml"
```

8.4.3.3 No leading slash: "bar.xml"

Case 1: The base IRI is "pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml"

Since this relative reference does not begin with the slash character, the path component of the base IRI ("/a/b/foo.xml") and that of the relative reference ("bar.xml") are merged. The merge routine in §5.2.3 of RFC 3986 first removes "foo.xml" from the path component of the base IRI, and emits "/a/b/bar.xml". Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/a/b/bar.xml"
```

Case 2: The base IRI is "pack://http%3c,,www.mysite.com,my.package/"

Likewise, the path component of the base IRI ("/") and that of the relative reference ("bar.xml") are merged. The merge routine emits "/bar.xml". Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/bar.xml"
```

Case 3: The base IRI is "http://www.mysite.com/my.package"

Likewise, the path component of the base IRI ("/my.package") and that of the relative reference ("bar.xml") are merged. The merge routine first removes "my.package" from the path component of the base IRI, and emits "/bar.xml". Thus, the resulting IRI is:

```
"http://www.mysite.com/my.package/bar.xml"
```

8.4.3.4 Dot segment: ./bar.xml

Case 1: The base IRI is "pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml"

As in §8.4.3.3, the merge routine removes "foo.xml" from the path component of the base IRI, and emits "/a/b/./bar.xml". But the remove_dot_segments routine in §5.2.4 of RFC 3986 further handles this result and emits "/a/b/bar.xml". Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/a/b/bar.xml"
```

Case 2: The base IRI is "pack://http%3c,,www.mysite.com,my.package/"

The merge routine emits "/./bar.xml" but the remove_dot_segments routine removes "./" and emits "/bar.xml". Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/bar.xml"
```

Case 3: The base IRI is "http://www.mysite.com/my.package"

Likewise, the path component of the base IRI ("/my.package") and that of the relative reference ("./bar.xml") are merged. The merge routine first removes "my.package" from the path component of the base IRI, and emits "/./bar.xml". But the remove_dot_segments routine removes "./" and emits "/bar.xml". Thus, the resulting IRI is:

```
"http://www.mysite.com/bar.xml"
```

8.4.3.5 Dot segment: "../bar.xml

Case 1: The base IRI is "pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml"

The merge routine emits "/a/b/../bar.xml" but the remove_dot_segments routine removes "b/..". Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/a/bar.xml"
```

Case 2: The base IRI is "pack://http%3c,,www.mysite.com,my.package/"

The merge routine emits "/../bar.xml", but the remove_dot_segments routine replaces "/../" by "/". Thus, the resulting IRI is:

```
"pack://http%3c,,www.mysite.com,my.package/bar.xml"
```

Case 3: The base IRI is "http://www.mysite.com/my.package"

Likewise, the path component of the base IRI ("/my.package") and that of the relative reference ("../bar.xml") are merged. The merge routine first removes "my.package" from the path component of the base IRI, and emits "/../bar.xml". The remove_dot_segments routine replaces "/.." by "/" and emits "/bar.xml". The resulting IRI is:

```
"http://www.mysite.com/bar.xml"
```

End of informative subclause.

8.5 Relationships

8.5.1 General

This subclause is informative.

Parts may contain references to other parts in the package and to resources outside of the package. These references are represented inside the referring part in ways that are specific to the media type of the part; that is, in arbitrary markup or an application-defined encoding. This effectively hides the links between parts from applications that do not understand the media types of the parts containing such references.

This document introduces a higher-level mechanism to describe references from parts to other parts or external resources, namely, relationships (term 3.2.3). Relationships represent connections from a source part or source package (term 3.2.6) to a target part or target resource (term 3.2.7). Relationships from parts are called part relationships (term 3.2.5), while those from packages are called package relationships (term 3.2.4). Relationships make the connection directly discoverable without looking at the part contents, so they are independent of content-specific schemas and are quick to resolve.

There are two modes (term 3.2.16) to resolve relative references to targets. Resolution in the internal target mode provides parts and that in the external target mode provides external resources.

Relationships have relationship identifiers (term 3.2.15) such that they may be distinguished from one another and may be referred to within a source part. If it is necessary to associate a relationship with a specific point in a source part, the identifier of the relationship is embedded at that point.

Commented [MM4]: Caroline's comment: Sentence not clear to me. I initially read it as meaning IDs were not required. But I think it means that the IDs may be referred to within the source part in order to establish a link to a target part or resource at a specific location in the part.

Commented [MM5R4]: A part relationship is established by a Relationship element within a relationships part.

But within an source part, we might want to specify the exact location from which the relationship is established. To do so, we can use some XML construct having the identifier of the relationship. This identifier chooses one of the relationships from this source part.

This is my interpretation, but I do not think that this is clearly described somewhere in OPC.

Commented [MM346R4]: Caroline wrote: Third paragraph. I'm afraid I don't understand Murata–san's response to my comment or what his intent is wrt the specification text. I still find the sentence unclear.

A relationship has a relationship type (term 3.2.8), an absolute IRI for identifying the role of the relationship.

Relationships are represented in XML in a Relationships part. Each part in the package that is the source of one or more relationships has an associated Relationships part. This part holds the list of relationships for the source part. For more information on the Relationships namespace and relationship types, see Annex E.

Relationships have a second important function: providing additional information about parts without modifying their content. [*Note*: Some scenarios require information to be attached to an existing part without modifying that part, for example, because the part is encrypted and cannot be decrypted, or because it is digitally signed and changing it would invalidate the signature. *end note*]

End of informative subclause.

8.5.2 Relationships Part

8.5.2.1 Relationships Part

media Type:	application/vnd.openxmlformats-package.relationships+xml
Root Namespace:	http://schemas.openxmlformats.org/package/2006/relationships

Each set of relationships sharing a common source is represented by a Relationships part. There shall be no relationships from or to a Relationships part.

A Relationships part (term 3.2.9) shall be either a package Relationships part (§8.5.2.2) or a part Relationships part (§8.5.2.3).

8.5.2.2 Package Relationships Part

Every relationship contained in a package Relationships part shall be a package relationship.

The name of a package Relationships part shall be “/_rels/.rels”.

When a relative reference occurs in a package Relationships part, the base IRI depends on the target mode of the relationship. If the target mode is external, the base IRI shall be the absolute IRI of the package. If the target mode is internal, the base IRI shall be the pack IRI created from the absolute IRI of the package.

[*Example:*

Consider the package Relationships part for a package available at http://www.mysite.com/my.package.

If the target mode is external, the base IRI is

```
http://www.mysite.com/my.package
```

If the target mode is internal, the base IRI is

```
pack://http%3c,,www.mysite.com,my.package/
```

end example]

8.5.2.3 Part Relationships Part

Every relationship contained in a part Relationships part shall be a part relationship from the same source part.

The name of a part Relationships part shall be constructed from the name of the source part by adding ".rels" to the end of the last I18N segment and inserting an I18N segment "_rels" immediately before the last I18N segment.

[*Example*: If the source part name is "/foo", the part Relationships part name is "/_rels/foo.rels". Conversely, if the name of a part is "/_rels/foo.rels", it is a part Relationships part for the source part "/foo". If the source part name is "/foo/bar.xml", the part Relationships part name is "/foo/_rels/bar.xml.rels". Conversely, if the name of a part is "/foo/_rels/bar.xml.rels", it is a part Relationships part for the source part "/foo/bar.xml". *end example*]

When a relative reference occurs in a part Relationships part, the base IRI depends on the target mode of the relationship. If the target mode is external, the base IRI shall be the absolute IRI of the package. If the target mode is internal, the base IRI shall be the pack IRI created from the absolute IRI of the package and the source part name.

[*Example*:

Consider a part Relationships part /a/b/_rels/foo.xml.rels in a package available at

```
http://www.mysite.com/my.package
```

If the target mode is external, the base IRI is

```
http://www.mysite.com/my.package
```

If the target mode is internal, the base IRI is

```
pack://http%3c,,www.mysite.com,my.package/a/b/foo.xml
```

end example]

8.5.3 **Relationship Markup**

8.5.3.1 General

The content of a Relationships part shall be an XML document. The requirements (including MCE processing before validation and subsequent processing) specified in §8.2.5 apply.

After the removal of any extensions by an MCE processor as specified in ISO/IEC 29500-3, a Relationships part shall be a schema-valid XML document against opc-relationships.xsd, as described in Annex C.

An xml:base attribute shall not exist in the output document resulting from any MCE processing (as specified in ISO/IEC 29500-3) of the Relationships part.

8.5.3.2 Relationships Element

A Relationships element is the root element of a Relationships part. It is the container for zero or more Relationship elements. It has no attributes. [*Note*: The W3C XML Schema definition of this element's content model (CT_Relationships) is located in §C.4. *end note*]

8.5.3.3 Relationship Element

A Relationship element shall represent a relationship. The source of a relationship shall be either a package or part with which the Relationships part containing this Relationship element is associated. [*Note*: The target of a relationship is specified by the attributes of the Relationship element. *end note*]

Attributes	Description
TargetMode	This attribute specifies the target mode of a relationship. This attribute is optional, and the default value is `Internal`. The possible values for this attribute are `Internal` and `External`, as defined by the ST_TargetMode simple type (§C.5).
Target	This attribute specifies the target of a relationship. This attribute is required. If the value of the TargetMode attribute is `Internal`, the Target attribute shall be a relative reference to a part. If the value of the TargetMode attribute is `External`, the Target attribute shall be a relative reference or an absolute IRI. Base IRIs for resolving relative references are defined in §8.4. The possible values for this attribute are defined by the xsd:anyURI simple type of the W3C Recommendation "XML Schema Part 2: Datatypes".

Commented [WD37]: Add a diagram to the top-level clause that shows the way the source of a relationship is specified. Then improve the wording here to reflect that.

In feedback on the Day 2 minutes, Murata pointed out that a big picture of relationships (naming rule, part/package relationship, and internal/external relationships) is scattered into several attribute descriptions. Aarti suggested the addition of some diagram for providing such a big picture.

After some discussion, WG4 agreed to introduce such a diagram as well as prose descriptions.

(Murata-san will do this in a separate stand-alone document, which Rex will merge into the OPC WD later.)

Commented [MM8R7]: I created a diagram, available at https://onedrive.live.com/view.aspx?resid=4106E423DCEF597E!29880&ithint=file%2cpptx&app=PowerPoint&authkey=!AG0Bpabvvb6XfUU

But I now think that this diagram is not very useful any more. Rewrite of Clause 8 has significantly improved readability, in my opinion.

Attributes	Description
Type	This attribute specifies the relationship type of a relationship. This attribute is required. Relationship types can be compared to determine whether two Relationship elements are of the same type. This comparison is conducted in the same way as when comparing URIs that identify XML namespaces: the two URIs are treated as strings and considered identical if and only if the strings have the same sequence of characters. The comparison is case-sensitive, and no escaping is done or undone. [*Example*: `Type="http://schemas.openxmlformats.org/package/2006/relationships/ digital-signature/signature"` *end example*] The possible values for this attribute are defined by the xsd:anyURI simple type of the W3C Recommendation "XML Schema Part 2: Datatypes".
Id	This attribute specifies the identifier of a relationship. The value of the Id attribute shall be unique within the Relationships part. This attribute is required. [*Example*: `Id="A5FFC797514BC"` *end example*] The possible values for this attribute are defined by the xsd:ID simple type of the W3C Recommendation "XML Schema Part 2: Datatypes".

[*Note*: The W3C XML Schema definition of this element's content model (CT_Relationship) is located in §C.4. *end note*]

8.5.4 Examples

This subclause is informative.

8.5.4.1 Relationships Part Associated with the Entire Package

Consider a package, available at http://www.example.com/ex.opc. Suppose that the package contains a Relationships part "/_rels/.rels". This Relationships part is a package Relationships part, which is associated with the entire package.

Also, suppose that the content of this package Relationships part is the XML document shown below:

```
<Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
```

```
      <Relationship
         Target="./a.xml"
         Id="IDI1"
         Type="http://example.com/relTypeInt1"/>
      <Relationship
         Target="./a.xml"
         TargetMode="External"
         Id="IDE1"
         Type="http://example.com/relTypeExt1"/>
   </Relationships>
```

The two Relationship elements in this package Relationships part specify two relationships. The source of each relationship is the package.

The first relationship:

- The target mode is Internal (default). Thus, the base IRI for resolving "./a.xml" is the pack IRI (pack://http%3c,,www.example.com,ex.opc) created from the IRI of the package (http://www.example.com/ex.opc).
- The result of resolving "./a.xml" is "pack://http%3c,,www.example.com,ex.opc/a.xml". The target of this relationship is thus the part "/a.xml" in this package.
- The relationship type of this relationship is "http://example.com/relTypeInt1".
- The identifier of this relationship is "IDI1".

The second relationship:

- The target mode is External. Thus, the base IRI for resolving "./a.xml" is the IRI ("http://www.example.com/ex.opc") of the package.
- The target of this relationship is thus the resource at "http://www.example.com/a.xml".
- The relationship type of this relationship is "http://example.com/relTypeExt1".
- The identifier of this relationship is "IDE1".

8.5.4.2 Relationships Part Associated with a Part

Consider a package, available at http://www.example.com/ex.opc. Suppose that the package contains a Relationships part "/foo_rels/test.xml.rels". This Relationships part is a part Relationships part, the source of which is a part "/foo/test.xml".

Also, suppose that the content of this part Relationships part is the XML document shown below:

```
   <Relationships
      xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
      <Relationship
         Target="./b.xml"
         Id="IDI2"
```

```
        Type="http://example.com/relTypeInt2"/>
     <Relationship
        Target="./b.xml"
        TargetMode="External"
        Id="IDE2"
        Type="http://example.com/relTypeExt2"/>
  </Relationships>
```

The two Relationship elements in this part Relationships part specify two relationships. The source of each relationship is the part "/foo/test.xml".

The first relationship:

- The mode of the first relationship is Internal (default). Thus, the base IRI ("pack://http%3c,,www.example.com,ex.opc/foo/test.xml") is the pack IRI created from the IRI (http://www.example.com/ex.opc) of the package and the part name "/foo/test.xml".
- The result of resolving "./b.xml" is "pack://http%3c,,www.example.com,ex.opc/foo/b.xml". The target of this relationship is thus the part "/foo/b.xml" in this package.
- The relationship type of this relationship is "http://example.com/relTypeInt2".
- The identifier of this relationship is "IDI2".

The second relationship:

- The mode of the second relationship is External. Thus, the base IRI is the IRI (http://www.example.com/ex.opc) of the package.
- The target of this relationship is thus the resource at "http://www.example.com/b.xml".
- The relationship type of this relationship is "http://example.com/relTypeExt2".
- The identifier of this relationship is "IDE2".

8.5.4.3 Relationships Parts Related to Digital Signature Markup

The figure below shows a Digital Signature Origin part and a Digital Signature XML Signature part (see §12.4). The Digital Signature Origin part is targeted by a package relationship. The connection from the Digital Signature Origin to the Digital Signature XML Signature part is represented by a part relationship.

The relationship targeting the Digital Signature Origin part is stored in a package Relationships part, /_rels/.rels, and the relationship for the Digital Signature XML Signature part is stored in a part Relationships part, /_rels/origin.rels.

The part Relationships part contains a relationship that connects the Digital Signature Origin part to the Digital Signature XML Signature part. This relationship is expressed as follows:

```
  <Relationships
     xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
     <Relationship
        Target="./Signature.xml"
        Id="A5FFC797514BC"
        Type="http://schemas.openxmlformats.org/package/2006/relationships/
           digital-signature/signature"/>
  </Relationships>
```

8.5.4.4 Relationships Targeting External Resources

Relationships can target resources outside the package at an absolute location and resources located relative to the current location of the package. The following Relationships part specifies relationships that connect a package or part to pic1.jpg at an external absolute location, and to my_house.jpg at an external location relative to the location of the package:

```
  <Relationships
     xmlns="http://schemas.openxmlformats.org/package/2006/relationships"
     <Relationship
        TargetMode="External"
        Id="A9EFC627517BC"
        Target="http://www.example.com/images/pic1.jpg"
        Type="http://www.example.com/external-resource"/>
     <Relationship
        TargetMode="External"
        Id="A5EFC797514BC"
        Target="./images/my_house.jpg"
        Type="http://www.example.com/external-resource"/>
  </Relationships>
```

end example]

8.5.4.5 Multiple Relationships that have the Same Target

The following Relationships part contains two relationships, each using a unique Id value. The relationships share the same Target, but have different relationship types.

```
  <Relationships
     xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
     <Relationship
        Target="./Signature.xml"
        Id="A5FFC797514BC"
        Type="http://schemas.openxmlformats.org/package/2006/
           relationships/digital-signature/signature"/>
     <Relationship
        Target="./Signature.xml"
        Id="B5F32797CC4B7"
        Type="http://www.example.com/internal-resource"/>
  </Relationships>
```

end example]

End of informative subclause.

8.5.5 Support for Versioning and Extensibility

Relationships parts might contain the versioning and extensibility mechanisms defined in ISO/IEC 29500-3 to incorporate elements and attributes drawn from other XML namespaces.

9 Physical Package Model

9.1 General

This subclause is informative.

This clause introduces a physical package model (term 3.3.3) in terms of a physical format (such as the ZIP format) and a mapping from the abstract package model to this physical format. See Annex E for additional discussion of the physical package model design considerations.

This clause further specifies general guidelines and common mechanisms for physical package models and defines a ZIP-based physical package model. The interleaving mechanism (see §9.2.4) is such a common mechanism.

An example physical package is shown in H.3.

End of informative subclause.

9.2 Physical Mapping Guidelines

9.2.1 Using Features of Physical Formats

Many physical formats have features that partially match components in the abstract package model. A mapping from the abstract package model to a physical format should take advantage of any similarities in capabilities between the abstract package model and the physical format while using layers of mapping to provide additional capabilities not inherently present in the physical format. [*Example*: Some physical formats store parts as individual files in a file system, in which case, it is advantageous to map many part names directly to identical physical file names. *end example*]

9.2.2 Mapped Components

A physical package model is required to represent packages, parts (including Relationships parts), part names, and part media types, but is not required to represent a growth hint.

9.2.3 Mapping Media Types to Parts

9.2.3.1 General

A physical format may have a native mechanism for associating media types with parts. [*Example*: The Content-Type field in the header of a MIME entity associates a media type with that MIME entity. *end example*] For such a physical format, mappings from the abstract package model should use the native mechanism.

For all other physical formats, the package shall include an XML stream that is referred to in this document as the *Media Types stream.* The Media Types stream shall not represent a part. This stream shall not be URI-addressable. However, it can be interleaved in the physical package using the same mechanisms used for interleaving parts.

9.2.3.2 Media Types Stream Markup

9.2.3.2.1 General

The content of the Media Types stream shall be an XML document. The requirements (including MCE processing before validation and subsequent processing) specified in §8.2.5 apply.

The XML document in the Media Types stream shall have a top-level Types element, and one or more Default and Override child elements. Default elements shall define default mappings from the extensions of part names to media types. Override elements shall specify media types on parts that are not covered by, or are not consistent with, the default mappings. [*Note*: Default elements can be used to reduce the number of Override elements on a part. *end note*]

For all parts of the package other than Relationships parts (§8.5.2), the Media Types stream shall specify either:

- One matching Default element, or
- One matching Override element, or
- Both a matching Default element and a matching Override element, in which case, the Override element takes precedence.

There shall not be more than one Default element for any given extension, and there shall not be more than one Override element for any given part name.

The order of Default and Override elements in the Media Types stream shall not be significant.

The Media Types stream may define Default elements even though no parts use them.

9.2.3.2.2 Types Element

A Types element shall be the root element of the XML document contained in the Media Types stream.

This element shall have no attributes.

[*Note*: The W3C XML Schema definition of this element’s content model (CT_Types) is located in §C.2. *end note*]

9.2.3.2.3 Default Element

A Default element shall specify the default mappings from the extensions of part names to media types.

Attributes	Description
Extension	This attribute shall specify a string as a file extension. This attribute is required. A Default element shall match any part whose name ends with a period (".") followed by the value of this attribute. The possible values for this attribute are defined by the ST_Extension simple type shown in §C.2.
ContentType	This attribute shall specify a media type using the syntax defined in RFC 7231 §3.1.1.1. This attribute is required. The specified media type shall apply to any matching parts (unless overridden by Override elements). The possible values for this attribute are defined by the ST_ContentType simple type shown in §C.2.

[*Note*: The W3C XML Schema definition of this element's content model (CT_Default) is located in §C.2. *end note*]

9.2.3.2.4 Override Element

An Override element shall specify a media type for a part that is not covered by, or is not consistent with, the default mappings.

Attributes	Description
ContentType	This attribute shall specify a media type using the syntax defined in RFC 7231 §3.1.1.1. This attribute is required. The specified media type shall apply to the part named in the attribute PartName. The possible values for this attribute are defined by the ST_ContentType simple type shown in in §C.2.

Attributes	Description
PartName	This attribute shall specify a part name. This attribute is required. An Override element shall match a part whose name is equal to the value of this attribute. The possible values for this attribute are defined by the xsd:anyURI simple type of the W3C Recommendation “XML Schema Part 2: Datatypes.

[*Note*: The W3C XML Schema definition of this element’s content model (CT_Override) is located in §C.2. *end note*]

9.2.3.3 Media Types Stream Markup Example

[*Example*:

Example 9–1. Media Types stream markup

```
<Types
   xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
   <Default Extension="txt" ContentType="text/plain" />
   <Default Extension="jpeg" ContentType="image/jpeg" />
   <Default Extension="picture" ContentType="image/gif" />
   <Override PartName="/a/b/sample4.picture" ContentType="image/jpeg" />
</Types>
```

The Types element is a container for media types to be used within the package.

The following is a sample list of parts and their corresponding media types as defined by the Media Types stream markup above.

Part name	Media type
/a/b/sample1.txt	text/plain
/a/b/sample2.jpg	image/jpeg
/a/b/sample3.picture	image/gif
/a/b/sample4.picture	image/jpeg

end example]

9.2.3.4 Setting a Part Media Type in the Media Types Stream

When adding a new part to a package, the package implementer shall ensure that a media type for that part is specified in the Media Types stream; the package implementer shall perform the following steps to do so:

1) Get the extension from the part name by taking the substring to the right of the rightmost occurrence of the dot character (“.”) from the rightmost segment.
2) If a part name has no extension, a corresponding Override element shall be added to the Media Types stream.
3) Compare the resulting extension with the values specified for the Extension attributes of the Default elements in the Media Types stream. The comparison shall be ASCII case-insensitive matching.
4) If there is a Default element with a matching Extension attribute, then the media type of the new part shall be compared with the value of the ContentType attribute. The comparison might be case-sensitive and include every character regardless of the role it plays in the content-type grammar of RFC 7231, or it might follow the grammar of RFC 7231. [Editor's note: This statement is undesirable, since the result becomes unpredictable. But what do existing implementations do?]

    a) If the media types match, no further action is required.
    b) If the media types do not match, a new Override element shall be added to the Media Types stream.

5) If there is no Default element with a matching Extension attribute, a new Default element or Override element shall be added to the Media Types stream.

Commented [JH9]: Should this be changed?

Commented [MM10R9]: According to RFCs 6838 and 4288, both top-level type and subtype names are case-insensitive. But we cannot change the behaviors of existing implementations. Should we touch this text?

Commented [MM11R9]: We need feedback from Microsoft experts.

Commented [ril12]:

9.2.3.5 Determining a Part Media Type from the Media Types Stream

To get the media type of a part, the package implementer shall perform the following steps:

1) Compare the part name with the values specified for the PartName attribute of the Override elements. The comparison shall be ASCII case-insensitive matching.
2) If there is an Override element with a matching PartName attribute, return the value of its ContentType attribute. No further action is required.
3) If there is no Override element with a matching PartName attribute, then

    a) Get the extension from the part name by taking the substring to the right of the rightmost occurrence of the dot character (“.”) from the rightmost segment.
    b) Check the Default elements of the Media Types stream, comparing the extension with the value of the Extension attribute. The comparison shall be ASCII case-insensitive matching.

4) If there is a Default element with a matching Extension attribute, return the value of its ContentType attribute. No further action is required.

[*Note*: Given a conformant package, either an Override is found by the third step or a Default element is found by the fourth step. *end note*]

9.2.3.6 Support for Versioning and Extensibility

The Media Types stream shall not use the versioning and extensibility mechanisms defined in ISO/IEC 29500-3 .

9.2.4 Interleaving

During the mapping of an abstract package to a physical package, the data stream of a part or the Media Types stream may be interleaved. This document does not require a physical package model to support interleaving.

An interleaved part or the Media Types stream shall be broken into pieces, which can be interleaved with pieces of other parts or with whole parts. Pieces in physical packages shall occur in their natural order for optimal efficiency. Pieces can later be joined together, forming the original stream.

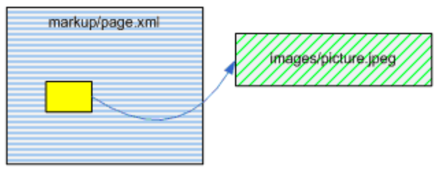
Pieces exist only in the physical package and are not addressable in the abstract package model. A piece might be empty.

A physical package may contain both interleaved parts and non-interleaved parts. However, it shall not contain both a whole part and a piece of the same part

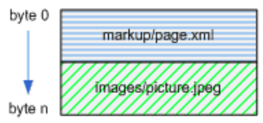
[*Example*:

Example 9–2. Performance benefits with interleaved ordering

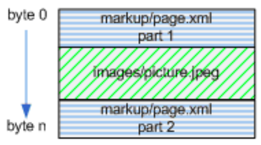
The figure below contains two parts: a page part (markup/page.xml) describing the contents of a page, and an image part (images/picture.jpg) referring to an image that appears on the page.



With simple ordering, *all* of the bytes of the page part are delivered before the bytes of the image part. The figure below illustrates this scenario. The image cannot be displayed until the entire page part *and* the image part have been received. In some circumstances, such as small packages on a high-speed network, this might be acceptable. In others, having to read through all of markup/page.xml to get to the image results in unacceptable performance or places unreasonable memory demands.



With interleaved ordering, performance is improved by splitting the page part into pieces and inserting the image part immediately following the reference to the image. This allows the image to be processed as soon as the reference is encountered.



end example]

9.2.5 Mapping Part Names to Physical Package Item Names

9.2.5.1 General

A mapping from an abstract package to a physical package shall use logical items as intermediate objects. If it is not interleaved, a part or the Media Types stream shall be a logical item. Otherwise, each piece constructed from it shall be a logical item.

9.2.5.2 Logical Item Names

Names of logical items shall be Unicode strings. The support of non-ASCII characters is not required.

If a logical item is a piece, its name shall have suffixes of the following syntax:

```
SuffixName      = "/" "[" PieceNumber "]" [".last"] ".piece"
PieceNumber     = "0" | NonZeroDigit [1*Digit]
Digit           = "0" | NonZeroDigit
NonZeroDigit    = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The prefix of a logical item name is the result of removing a suffix, if any, from the logical item name.

Equivalence of prefixes, and of suffixes shall be determined by ASCII case-insensitive matching. Logical names shall be equivalent if their prefixes and suffixes are equivalent. A physical package shall not contain equivalent logical item names.

Logical item names that use suffix names shall form a complete sequence if and only if:

1) The prefix names of all logical item names in the sequence are equivalent, and
2) The suffix names of the sequence start with “/[0].piece” and end with “/[*n*].last.piece” and include a piece for every piece number between 0 and *n*, without gaps, when the piece numbers are interpreted as decimal integer values.

9.2.5.3 Mapping Part Names to Logical Item Names

Names of non-interleaved parts shall be mapped to logical item names that have an equivalent prefix and no suffix.

Names of interleaved parts shall be mapped to the complete sequence of logical item names with an equivalent prefix.

9.2.5.4 Mapping Logical Item Names and Physical Package Item Names

The mapping of logical item names and physical package item names shall be specific to the particular physical package.

9.2.5.5 Mapping Logical Item Names to Part Names

A logical item name without a suffix shall be mapped to a part name with an equivalent prefix, provided that the prefix name conforms to the part name syntax.

A complete sequence of logical item names shall be mapped to the part name that is equal to the prefix of the logical item name having the suffix “/[0].piece”, provided that the prefix name conforms to the part name syntax.

A physical package may contain logical item names and complete sequences of logical item names that cannot be mapped to a part name because the logical item name does not follow the part naming grammar. Such logical items or complete sequences of logical items shall not be mapped to parts.

[*Example*: A ZIP item name [trash]/0000.dat cannot be mapped to a logical item.   Thus, this ZIP item does not represent a logical item or part. *end example*]

9.3 Mapping to a ZIP file

9.3.1 General

This document defines a mapping for the ZIP file (term 3.3.8) format. Future versions of this document might provide additional mappings.

A ZIP file representing a physical package shall not have any features of the ZIP file format specification related to encryption, decryption, or digital signatures.

The ZIP format includes a number of features that this document does not use. See Annex B for OPC-specific ZIP information.

Physical package items of ZIP files shall be ZIP items (term 3.3.7). [*Note*: When users unzip a ZIP-based package, they see a set of files and folders that reflects the parts in the package and their hierarchical naming structure. *end note*]

Table 9–1, shows the various components of the abstract package model and their corresponding physical representation in a ZIP file.

Table 9–1. Abstract package model components and their physical representations

Abstract package model component	Physical representation
Package	ZIP file

Abstract package model component	Physical representation
Part	ZIP item
Part name	Stored in item header (and ZIP central directory as appropriate). See §9.3.4 for conversion rules.
Part media type	ZIP item containing the Media Types stream described in §9.2.3.2. See §9.3.7 for details about the ZIP item name.
Growth hint	Padding reserved in the ZIP Extra field in the local header that precedes the item. See §9.3.8 for a detailed description of the data structure.

9.3.2 Mapping Part Data

Each non-interleaved part shall be represented as a single ZIP item. Each piece of an interleaved part, as described in §9.2.4, shall be represented as a single ZIP item.

9.3.3 ZIP Item Names

ZIP item names are Unicode strings. ZIP item names shall conform to the ZIP File Format Specification. ZIP item names shall be unique within a given ZIP file.~~.~~ The support of non-ASCII ZIP item names is not required.

[*Example*: A ZIP file might contain the following ZIP item names mapped to part pieces and whole parts:

```
spine.xml/[0].piece
pages/page0.xml
spine.xml/[1].piece
pages/page1.xml
spine.xml/[2].last.piece
pages/page2.xml
```

end example]

9.3.4 Mapping Logical Item ~~Part~~ Names to ZIP Item Names

Mapping of ~~part~~ logical item names to ZIP item names shall perform, in order, the following steps:

~~1) Convert the part name to a logical item name or, in the case of interleaved parts, to a complete sequence of logical item names, as specified in §9.2.5.3.~~

1) Remove the leading forward slash ("/") from the logical item name or, in the case of interleaved parts, from each of the logical item names within the complete sequence.
2) Percent-encode every non-ASCII character.

~~A logical item name or complete sequence of logical item names sharing a common prefix shall not be mapped to a part name if the logical item prefix has no corresponding media type.~~

9.3.5 Mapping ZIP Item Names to ~~Part~~ Logical Item Names

All ZIP item names shall be mapped to ~~part~~ logical item names except for MS-DOS ZIP items, as defined in the ZIP specification, that are not MS-DOS files.

Mapping of ZIP item names to part names shall perform, in order for each ZIP item, the following steps:

1) Un-percent-encode every non-ASCII character.
~~1)~~2) Add~~Map the ZIP item name to a logical item name by adding~~ a forward slash ("/").
~~2) Attempt to map the obtained logical item name to a part name, as specified in §9.2.5.5.~~

~~If Step 2 fails, the ZIP item does not represent a part.~~

[*Note*: The ZIP specification specifies that ZIP items recognized as MS-DOS files are those with a "version made by" field and an "external file attributes" field in the "file header" record in the central directory that have a value of 0. *end note*]

9.3.6 ZIP Package Limitations

This document requires that a file header in the central directory structure within a ZIP file shall not exceed 65,535 bytes (see "F. Central directory structure" in the ZIP Appnote.txt). Each file header contains a zip item name, Extra field (including bytes representing Growth Hint as specified in §8.2.4), File Comment, and 42 more bytes representing miscellaneous fields.

Package implementers should restrict part naming to accommodate file system limitations when naming parts to be stored as ZIP items.

[*Example*: Examples of these limitations are:

- On MS Windows file systems, the asterisk ("*") and colon (":") are not supported, so parts named with this character do not unzip successfully.
- On MS Windows file systems, many programs can handle only file names that are less than 256 characters including the full path; parts with longer names might not behave properly once unzipped.

end example]

ZIP-based packages shall not include encryption as described in the ZIP specification.

ZIP-based packages shall not use compression algorithms except DEFLATE, as described in the .ZIP specification.

9.3.7 Mapping the Media Types Stream

In ZIP files, the Media Types stream shall be stored in an item with the name "[Content_Types].xml" or, in the interleaved case, in the complete sequence of ZIP items "[Content_Types].xml/[0].piece", "[Content_Types].xml/[1].piece", …, and "[Content_Types].xml/[n].last.piece".

[*Note*: Bracket characters "[" and "]" were chosen for the Media Types stream name specifically because these characters violate the part naming grammar, thus reinforcing this requirement. *end note*]

9.3.8 Mapping the Growth Hint

The additional space suggested by growth hint (term 3.2.11) is stored in the Extra field, as defined in the ZIP file format specification. If the growth hint is used for an interleaved part, the padding is stored in the Extra field of the ZIP item representing the first piece of the part.

The format of the ZIP item's Extra field, when used for growth hints, is shown in Table 9–2, Structure of the Extra field for growth hints below.

Table 9–2. Structure of the Extra field for growth hints

Field	Size	Value
Header ID	2 bytes	A220
Length of Extra field	2 bytes	The signature length (2 bytes) + the padding initial value length (2 bytes) + Length of the padding (variable)
Signature (for verification)	2 bytes	A028
Padding Initial Value	2 bytes	Hex number value is set by a package implementer when the item is created
<padding>	[Padding Length]	Should be filled with NULL characters

[*Editor's note*: What does "Padding Initial Value" mean? *end note*]

Commented [rcj13]:

10 Core Properties

10.1 General

This subclause is informative.

Users can associate core properties with packages. Such core properties enable users to get and set well-known and common sets of property metadata to packages. The core properties and the specifications that describe them are shown in Table 10–1. Core properties:

Table 10–1. Core properties

Property	Specification	Description
category	Open Packaging Conventions	A categorization of the content of this package.
contentStatus	Open Packaging Conventions	The status of the content.
created	DCMI Metadata Terms	Date of creation of the resource.
creator	Dublin Core Metadata Element Set	An entity primarily responsible for making the content of the resource.
description	Dublin Core Metadata Element Set	An explanation of the content of the resource.
identifier	Dublin Core Metadata Element Set	An unambiguous reference to the resource within a given context.
keywords	Open Packaging Conventions	A delimited set of keywords to support searching and indexing. This is typically a list of terms that are not available elsewhere in the properties.
language	Dublin Core Metadata Element Set	The language of the intellectual content of the resource. [*Note*: IETF RFC 3066 provides guidance on encoding to represent languages. *end note*]
lastModifiedBy	Open Packaging Conventions	The user who performed the last modification. The identification is environment-specific.
lastPrinted	Open Packaging Conventions	The date and time of the last printing.
modified	DCMI Metadata Terms	Date on which the resource was changed.
revision	Open Packaging Conventions	The revision number.

Property	Specification	Description
subject	Dublin Core Metadata Element Set	The topic of the content of the resource.
title	Dublin Core Metadata Element Set	The name given to the resource.
version	Open Packaging Conventions	The version number.

End of informative subclause.

10.2 Core Properties Part

A Core Properties part shall be a part of the Core Properties part media type, as defined in Annex E. A package may contain at most one Core Properties part.

The content of the Core Properties part shall be an XML document that satisfies the requirements specified in 8.2.5.

A Core Properties part within the package shall be referenced by a core properties relationship from the package, as listed in Annex E. A package shall contain at most one core properties relationship.

The namespaces for the properties in this table in the Open Packaging Conventions domain are defined in Annex E.

10.3 Core Properties Markup

10.3.1 General

Unless specified otherwise, elements representing a Core Properties part shall be of the namespace "http://schemas.openxmlformats.org/package/2006/metadata/core-properties".

Core property elements shall be elements representing core properties. Core property elements are non-repeatable. They can be empty or omitted.

[*Example:*

An example of a core properties part is shown below.

```
<coreProperties
   xmlns="http://schemas.openxmlformats.org/package/2006/metadata/
      core-properties"
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <dc:creator>Alan Shen</dc:creator>
```

```
      <dcterms:created xsi:type="dcterms:W3CDTF">
         2005-06-12
      </dcterms:created>

      <dc:title>OPC Core Properties</dc:title>
      <dc:description>Spec defines the schema for OPC Core Properties and their
   location within the package</dc:description>
      <dc:language>eng</dc:language>
      <version>1.0</version>
      <lastModifiedBy>Alan Shen</lastModifiedBy>
      <dcterms:modified xsi:type="dcterms:W3CDTF">2005-11-23</dcterms:modified>
      <contentStatus>Reviewed</contentStatus>
      <category>Specification</category>
   </coreProperties>
```

In this example dc:creator, dcterms:created dc:title, dc:description, dc:language, version, lastModifiedBy, dcterms:modified, contentStatus, and category are core property elements.

end example]

10.3.2 coreProperties element

A coreProperties element is the root element of a Core Properties part.

This element shall have no attributes.

Children of this element shall be core property elements.

The content of this element is defined by the complex type CT_CoreProperties.

10.3.3 Property elements from Dublin Core Metadata Element Set, Version 1.1

This standard allows creator, description, identifier, language, subject, and title elements as defined by ISO 15836-1 (The Dublin Core metadata element set, Part 1: Core elements) as core property elements.

[*Note*: These elements belong to the namespace "http://purl.org/dc/elements/1.1/". *end note*]

These elements shall not have child elements and shall not have the xsi:type attribute or the xml:lang attribute.

[*Example*:

The example in §10.3.1 contains four elements from ISO 15836-1.

```
   <dc:creator>Alan Shen</dc:creator>
   <dc:title>OPC Core Properties</dc:title>
   <dc:description>Spec defines the schema for OPC Core Properties and their
   location within the package</dc:description>
   <dc:language>eng</dc:language>
```

end example]

10.3.4 Property Elements from DCMI Metadata Terms

This standard allows created and modified elements as defined by DCMI Metadata Terms as core property elements.

[*Note*: These elements belong to the namespace http://purl.org/dc/terms/. *end note*]

In this document, these elements shall not have child elements and shall not have the xml:lang attribute. These elements shall have the xsi:type attribute whose value is "`dcterms:W3CDTF`" (the W3C Note "Date and Time Formats" [2]) and `dcterms` shall be declared as the prefix of the Dublin Core namespace "http://purl.org/dc/terms/".

[*Example*:

The example in §10.3.1 contains two elements from DCMI Metadata Terms.

```
<dcterms:created xsi:type="dcterms:W3CDTF">2005-06-12</dcterms:created>
<dcterms:modified xsi:type="dcterms:W3CDTF">2005-11-23</dcterms:modified>
```

end example]

10.3.5 Property Elements defined in this Document

10.3.5.1 category Element

A category element specifies the category of the content of the package.

[*Example*: Example values for this property might include Resume, Letter, Financial Forecast, Proposal, and Technical Presentation. This value might be used by an application's user interface to facilitate navigation of a large set of documents. *end example*]

This element shall have no attributes.

The content of this element is defined by the xsd:string simple type.

[*Note*: The W3C XML Schema definition of this element is located in §D.2. *end note*]

[*Example*:

A category element is in the example in §10.3.1.

```
<category>Specification</category>
```

end example]

10.3.5.2 contentStatus Element

A contentStatus element specifies the status of the content of the package.

[*Note*: Values might include “Draft”, “Reviewed”, and “Final”. *end note*]

This element shall have no attributes.

The content of this element is defined by the xsd:string simple type.

[*Note*: The W3C XML Schema definition of this element is located in §D.2. *end note*]

[*Example*:

The example in §10.3.1 contains

```
<contentStatus>Reviewed</contentStatus>
```

end example]

10.3.5.3 keywords Element

A keywords element specifies the keywords for the content of the package.

A keywords element shall have an optional attribute xml:lang, as defined by XML 1.0. A keywords element has a mixed content model such that each keyword can be wrapped by a value element having an xml:lang attribute individually.

[*Example*: The following instance of the keywords element has keywords in English (Canada), English (U.S.), and French (France):

```
<keywords xml:lang="en-US">
  color
  <value xml:lang="en-CA">colour</value>
  <value xml:lang="fr-FR">couleur</value>
</keywords>
```

end example]

[*Note*: The W3C XML Schema definition of this element's content model (CT_Keywords) is located in §D.2. *end note*]

10.3.5.4 value Element

A value element specifies a keyword for the content of the package.

A value element shall have an optional attribute xml:lang, as defined by XML 1.0.

[*Note*: The W3C XML Schema definition of this element's content model (CT_Keyword) is located in §D.2. *end note*]

10.3.5.5 lastModifiedBy Element

A lastModifiedBy element specifies who modified the content of the content.

[*Example*: A name, email address, or employee ID. *end example*]

This element shall have no attributes.

The content of this element is defined by the xsd:string simple type.

[*Note*: The W3C XML Schema definition of this element is located in §D.2. *end note*]

[*Example*: The example in 10.3.1 contains

```
<lastModifiedBy>Alan Shen</lastModifiedBy>
```

end example]

10.3.5.6 lastPrinted Element

A lastPrinted element specifies when the content was printed last time.

This element shall have no attributes.

The content of this element is defined by the xsd:dateTime simple type.

[*Note*: The W3C XML Schema definition of this element is located in §D.2. *end note*]

[*Example*: The example in §10.3.1 contains

```
<lastPrinted>2017-01-01</lastPrinted>
```

Another example is

```
<lastPrinted>2017-04-17T14:20:10+09:00</lastPrinted>
```

end example]

10.3.5.7 revision Element

A revision element specifies the revision number of the content of the package.

This element shall have no attributes.

The content of this element is defined by the xsd:string simple type.

[*Note*: The W3C XML Schema definition of this element is located in §D.2. *end note*]

[*Example*:

```
<revision>4</revision>
```

end example]

10.3.5.8 version Element

A version element specifies the version of the content of the package.

This element shall have no attributes.

The content of this element is defined by the xsd:string simple type.

[*Note*: The W3C XML Schema definition of this element is located in §D.2. *end note*]

[*Example*:

```
<version>1.0</version>
```

end example]

10.4 Support for Versioning and Extensibility

A Core Properties part shall not contain elements or attributes of the Markup Compatibility namespace as defined in Annex E.

[*Note*: Versioning and extensibility functionality is accomplished by creating a new part and using a relationship with a new type to point from the Core Properties part to the new part. This document does not provide any requirements or guidelines for new parts or relationship types that are used to extend core properties. ISO/IEC TR 30114-1 [4] provides such a guideline. *end note*]

11 Thumbnails

Thumbnail parts shall be image parts identified by either a part relationship or a package relationship. This relationship shall have a relationship type for Thumbnail parts, as defined in Annex E.

[*Note*: Thumbnail parts may be used to help end-users identify parts of a package or a package as a whole. *end note*]

12 Digital Signatures

12.1 General

A package may include markup specifying that parts of a package have been signed. This clause describes how OPC applies the W3C Recommendation “XML-Signature Syntax and Processing” in the construction of this markup.

12.2 Overview of OPC-Specific Restrictions and Extensions to “XML-Signature Syntax and Processing”

This subclause is informative.

OPC represent digital signatures as separate OPC parts. In other words, digital signatures are detached from the content to be signed.

OPC introduces markup for specifying when a signature is created. This markup appears in an Object element.

OPC introduces markup (see §12.5.6.2) and a transform algorithm ("Relationships transform", see §12.6) for flexibly defining the relationships to be signed.

OPC mandates the use of the Manifest element as a child of an Object element for enumerating parts to be signed.

End of informative text.

12.3 Choosing Content to Sign

It is assumed that there is a signature policy to determine which parts and relationships to sign.

This clause provides flexibility in defining the content to be signed, thus allowing other contents to be mutable. For further information on how to define which content is to be signed, see §12.5.5 and §12.5.6.2.

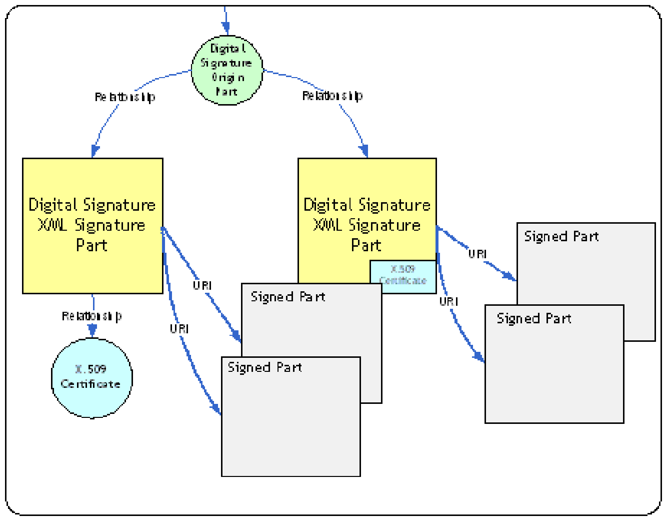
12.4 Digital Signature Parts

12.4.1 General

Digital signatures in packages use the Digital Signature Origin part, Digital Signature XML Signature parts, and Digital Signature Certificate parts. Relationship types and media types relating to the use of digital signatures in packages are specified in Annex E. [*Note*: An example relationship from the Digital Signature Origin part to a Digital Signature XML Signature part is shown in §8.5.4.3. *end note*]

[*Example*: Figure 12–1 shows a signed package with signature parts, signed parts, and an X.509 certificate part. The example Digital Signature Origin part has relationships to two Digital Signature XML Signature parts, each containing a signature. The signatures relate to the signed parts.

Figure 12–1. A signed package



end example]

12.4.2 Digital Signature Origin Part

The Digital Signature Origin part is the starting point for navigating through the signatures in a package. No more than one Digital Signature Origin part shall exist in a package and that part shall be the target of a Digital Signature Origin relationship, as specified in Annex E, from the package. This part shall exist if the package contains any Digital Signature XML Signature parts, and is optional otherwise. The content of the Digital Signature Origin part shall be empty.

12.4.3 Digital Signature XML Signature Part

A Digital Signature XML Signature part shall contain digital signature markup (see §12.5). Each Digital Signature XML Signature part shall be the target of a Digital Signature relationship, as specified in Annex E, from the Digital Signature Origin part. A package may contain more than one Digital Signature XML Signature part.

[*Note*: Future versions of this document might specify distinct relationship types for revised signature parts. Using these relationships, packages would be able to contain separate signature information for current and previous versions. For reference validation and signature validation it would be possible to choose the appropriate XML digital signatures. *end note*]

12.4.4 Digital Signature Certificate Part

An X.509 certificate is used to validate a signature and can be contained either within a Digital Signature XML Signature part, or as a separate Digital Signature Certificate part, or stored outside the package.

If the certificate is represented as a separate part within the package, that certificate shall be the target of a Digital Signature Certificate part relationship, as specified in Annex E, from the appropriate Digital Signature XML Signature part. The part containing the certificate may be signed. The media type of the Digital Signature Certificate part and the relationship targeting it from the Digital Signature XML Signature part are defined in Annex E. A Digital Signature Certificate part may be used to create more than one signature. A Digital Signature Certificate part should be the target of at least one Digital Signature Certificate relationship from a Digital Signature XML Signature part.

12.5 Digital Signature Markup

12.5.1 General

The following subclauses cover OPC-specific restrictions and extensions to “XML-Signature Syntax and Processing”. An element with restrictions or extensions has a subclause; elements without restrictions or extensions (such as X509Certificate) are allowed and have no subclause.

OPC-specific elements belong to the namespace for Digital Signatures (see Table E-1 in Annex E). Their schema definitions are in Annex C.4.

[*Note*: For a general example of XML digital signature markup, see Section 2 of “XML-Signature Syntax and Processing”. For a complete example of an OPC-specific digital signature, see §12.712.7. *end note*]

12.5.2 Signature Element

This document introduces further requirements to those defined in §4.1 of “XML-Signature Syntax and Processing”.

A Signature element shall contain exactly one OPC-specific Object element and zero or more application-defined Object elements.

12.5.3 SignedInfo Element

This document introduces further requirements to those defined in §4.3 of “XML-Signature Syntax and Processing”

A SignedInfo element shall contain exactly one reference to an OPC-specific Object element.

12.5.4 CanonicalizationMethod Element

This document introduces further requirements to those defined in §4.3.1 of “XML-Signature Syntax and Processing”.

Packages shall use only the following canonicalization methods:

- XML Canonicalization (c14n)
- XML Canonicalization with Comments (c14n with comments)

12.5.5 Reference Element

12.5.5.1 General

This document introduces further requirements to those defined in §4.3.3 of "XML-Signature Syntax and Processing".

12.5.5.2 Reference Element as a Child of a SignedInfo Element

Reference elements within a SignedInfo element shall reference elements only within the same Signature element, and should reference an Object element.

12.5.5.3 Reference Element as a Child of a Manifest Element

Each Reference element that is a child of a Manifest element shall only reference parts in the package. The value of the URI attribute shall be a part name without a fragment identifier.

References to package parts shall include the part media type as a query component. The syntax of the relative reference is as follows:

```
/page1.xml?ContentType=value
```

where *value* is the (case-insensitive) media type of the targeted part.

[*Example*:

Example 12–3. Part reference with query component

In the following example, the media type is "application/vnd.openxmlformats-package.relationships+xml":

```
URI="/_rels/document.xml.rels?ContentType=application/vnd.openxmlformats-
package.relationships+xml"
```

end example]

12.5.6 Transform Element

12.5.6.1 General

This document introduces further requirements to those defined in §4.3.3.4 of "XML-Signature Syntax and Processing".

One of the following transform algorithms shall be specified:

- XML Canonicalization (c14n)

- XML Canonicalization with Comments (c14n with comments)
- Relationships transform (OPC-specific)

12.5.6.2 Transform Element Representing a Relationships Transform

A Transform element represents a Relationships transform if the value of its attribute "Algorithm" is:

```
http://schemas.openxmlformats.org/package/2006/RelationshipTransform
```

Such a Transform element shall:

- contain one or more RelationshipReference or RelationshipsGroupReference elements,
- be a descendant element of a Manifest element,
- be followed by a Transform element specifying either XML Canonicalization (c14n) or XML Canonicalization with Comments (c14n with comments)

A Relationships transform describes how the Relationship elements from the Relationships part are selected for signing. Only one Relationships transform shall be specified for a particular Relationships part. For algorithm details, see §12.6.

12.5.7 RelationshipReference Element

The RelationshipReference element specifies which Relationship element is signed, and shall only occur as a child element of a Transform element representing a Relationships transform (§12.5.6.2). This element is OPC-specific.

Attributes	Description
SourceId (Reference to Relationship)	The value of the Id attribute of the referenced Relationship element within the given Relationships part

12.5.8 RelationshipsGroupReference Element

The RelationshipsGroupReference element specifies that the group of Relationship elements with the specified value for the Type attribute is signed. This element shall only occur as a child element of a Transform element representing a Relationships transform (§12.5.6.2). This element is OPC-specific.

Attributes	Description
SourceType (Relationship Type)	The value of the Type attribute of the Relationship elements within the given Relationships part

12.5.9 DigestMethod Element

This document introduces further requirements to those defined in §4.3.3.5 of “XML-Signature Syntax and Processing”.

The RSA-SHA1 algorithm shall be specified by a DigestMethod element.

12.5.10 Object Element

12.5.10.1 General

This document introduces further requirements to those defined in §4.5 of "XML-Signature Syntax and Processing". An Object element shall be either OPC-specific or application-defined.

12.5.10.2 OPC-specific Object Element

An OPC-specific Object element shall contain a Manifest element followed by a SignatureProperties element, and no other elements. The Id attribute of the OPC-specific Object element shall be specified, and its value shall be "`idPackageObject`".

The Markup Compatibility namespace, as specified in Annex E, shall not be used within the OPC-specific Object element.

12.5.10.3 Application-Defined Object Element

An application-defined Object element specifies application-defined information The Id attribute of the application-defined Object element shall be absent or have a value other than "idPackageObject".

The Markup Compatibility namespace, as specified in Annex E, shall not be used within the OPC-specific Object element.

12.5.11 Manifest Element

This document introduces further requirements to those defined in §4.4 of "XML-Signature Syntax and Processing" only when a Manifest element occurs as a child of an OPC-specific Object element. Reference elements in such a Manifest element shall satisfy requirements defined in §12.5.5.3.

12.5.12 SignatureProperty Element

This document introduces further requirements to those defined in §5.2 of "XML-Signature Syntax and Processing" only when a SignatureProperty element is a child of a child SignatureProperties element of an OPC-specific Object element. Such a SignatureProperty element shall specify the Id attribute to have the value "`idSignatureTime`", and shall contain a SignatureTime element and no other elements. The Target attribute value of such a SignatureProperty element shall be either empty or contain a fragment reference to the value of the Id attribute of the root Signature element.

12.5.13 SignatureTime Element

The SignatureTime element specifies the date/time stamp for the signature. This element is OPC-specific.

12.5.14 Format Element

The Format element specifies the format of the date/time stamp. This element is OPC-specific. The date/time format shall conform to the syntax described in the W3C Note "Date and Time Formats" [2].

12.5.15 Value Element

The Value element specifies the value of the date/time stamp. This element is OPC-specific. The value shall conform to the format specified in the Format element.

12.5.16 XPath Element

Although the XPath element is allowed in “XML-Signature Syntax and Processing”, it is disallowed in this document. [*Note*: The XPath element is only for XPath filtering, which is disallowed in OPC. *end note*]

12.6 Relationships Transform Algorithm

The Relationships transform takes the XML document from the specified Relationships part and transforms it to another XML document. This transform shall be supported in generating and validating signatures. [*Note*: The output XML document is subsequently canonicalized by the specified canonicalization algorithm. *end note*]

The Relationships part might contain content from several namespaces, along with versioning instructions as defined in Part 3, “Markup Compatibility and Extensibility”.

The Relationships transform algorithm has the following steps:

Step 1: Process versioning instructions

Process the Relationships part as specified in Part 3, §9, where the markup configuration is empty and the application configuration contains the Relationships namespace only.

Step 2: Sort and select signed relationships

1) Remove all namespace declarations except the Relationships namespace declaration.
2) Remove the Relationships namespace prefix, if it is present.
3) Sort relationship elements by Id value in lexicographical order, considering Id values as case-sensitive Unicode strings.
   Keep only those Relationship elements which either have an Id value that matches a SourceId value or have a Type value that matches a SourceType value specified in the Relationships transform. Matching is ASCII case-insensitive.

[*Example*: Consider a Relationships part

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rlsps:Relationships
xmlns:rlsps="http://schemas.openxmlformats.org/package/2006/relationships"
xmlns:foo="http://example.com/foo">
    <rlsps:Relationship Id="rId6" Type="http://../relationships/footnotes"
Target="footnotes.xml"/>
    <rlsps:Relationship Id="rId8" Type="http://../relationships/header"
Target="header1.xml"/>
```

```
    <rlsps:Relationship Id="rId32" Type="http://../relationships/image"
Target="media/image1.png"/>
    <rlsps:Relationship Id="rId3" Type="http://../relationships/styles"
Target="styles.xml"/>
    <rlsps:Relationship Id="rId21" Type="http://../relationships/image"
Target="media/image2.jpeg"/>
    <rlsps:Relationship Id="rId12" Type="http://../relationships/header"
Target="header1.xml"/>
</rlsps:Relationships>
```

Given Id="rId6" and Type=http://../relationships/image, Step 2 constructs

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships
xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId21" Type="http://../relationships/image"
Target="media/image2.jpeg"/>
    <Relationship Id="rId32" Type="http://../relationships/image"
Target="media/image1.png"/>
    <Relationship Id="rId6" Type="http://../relationships/footnotes"
Target="footnotes.xml"/>
</Relationships>
```

end example]

Step 3: Prepare for canonicalization

1) Remove all text nodes and comments within the document.
2) If the TargetMode attribute is missing from a Relationship element, add it with the default value "Internal".

12.7 Digital Signature Example

This subclause is informative.

Digital signature markup for packages is illustrated in this example. For information about namespaces used in this example, see Annex E. Note that the namespace prefix pds refer to the namespace for OPC-specific elements in digital signatures.

There are two Object elements in this example. The first Object element is OPC-specific since the value of its Id attribute is "idPackageObject". The second Object element (at the very end of this example) is application-dependent since the value of its Id attribute is not "idPackageObject".

The OPC-specific Object element contains a Manifest element followed by a SignatureProperties element. The Manifest element specifies a list of parts by its Reference child elements. The first Reference element references a part /document.xml via the value of the URI attribute. The second Reference element references a Relationships part /_rels/document.xml.rels, the source part of which is /document.xml.

Children of these Reference elements specify which transform and digest method is used and also specify obtained digest values. Note that the first transform for the Relationships part is a Relationships transform.

The SignedInfo element (at the beginning of this example) references the two Object elements. The OPC-specific Object element including its Manifest and SignatureProperties child elements are canonicalized and then signed. The application-defined Object element is also signed.

The SignatureValue element contains a signature, while the KeyInfo element contains an X509 certificate.

```
<Signature Id="SignatureId" xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
         REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <Reference
         URI="#idPackageObject"
         Type="http://www.w3.org/2000/09/xmldsig#Object">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/2001/
               REC-xml-c14n-20010315"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>…</DigestValue>
      </Reference>
      <Reference
         URI="#Application"
         Type="http://www.w3.org/2000/09/xmldsig#Object">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/2001/
               REC-xml-c14n-20010315"/>
         </Transforms>
         <DigestMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>…</DigestValue>
      </Reference>
   </SignedInfo>
   <SignatureValue>…</SignatureValue>

   <KeyInfo>
      <X509Data>
         <X509Certificate>…</X509Certificate>
      </X509Data>
   </KeyInfo>

   <Object Id="idPackageObject" xmlns:pds="http://schemas.openxmlformats.org/
      package/2006/digital-signature">
      <Manifest>
         <Reference URI="/document.xml?ContentType=application/
            vnd.ms-document+xml">
            <Transforms>
               <Transform Algorithm="http://www.w3.org/TR/2001/
                  REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>…</DigestValue>
         </Reference>
         <Reference
            URI="/_rels/document.xml.rels?ContentType=application/
               vnd.openxmlformats-package.relationships+xml">
            <Transforms>
               <Transform Algorithm="http://schemas.openxmlformats.org/
                  package/2006/RelationshipTransform">
                  <pds:RelationshipReference SourceId="B1"/>
                  <pds:RelationshipReference SourceId="A1"/>
                  <pds:RelationshipReference SourceId="A11"/>
                  <pds:RelationshipsGroupReference SourceType=
                     "http://schemas.example.com/required-resource"/>
               </Transform>
               <Transform Algorithm="http://www.w3.org/TR/2001/
                  REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>…</DigestValue>
         </Reference>
      </Manifest>
      <SignatureProperties>
         <SignatureProperty Id="idSignatureTime" Target="#SignatureId">
            <pds:SignatureTime>
               <pds:Format>YYYY-MM-DDThh:mmTZD</pds:Format>
               <pds:Value>2003-07-16T19:20+01:00</pds:Value>
            </pds:SignatureTime>
         </SignatureProperty>
      </SignatureProperties>
   </Object>
```

```
    <Object Id="Application">…</Object>
  </Signature>
```

End of informative text.

12.8 Generating Signatures

Digitally signed packages can be generated by reference generation and signature generation, as described in §3.1 of "XML-Signature Syntax and Processing", with some modification for OPC-specific constructs.

[*Note*: The steps below do not apply to the generation of signatures that contain application-defined Object elements. *end note*]

The signature policy determines which parts and relationships to sign and the transforms and digest methods that are applicable in each case.

Reference generation:

1) For each part being signed, create a Reference element following the steps in §3.1.1 of "XML-Signature Syntax and Processing".
2) Construct the OPC-specific Object element containing a Manifest element with both the child Reference elements obtained from the preceding step and a child SignatureProperties element, which, in turn, contains a child SignatureTime element.
3) Create a reference to the resulting OPC-specific Object element following the steps in §3.1.1 of "XML-Signature Syntax and Processing".

Signature generation:

Follow the steps in §3.1.2 of "XML-Signature Syntax and Processing".

12.9 Validating Signatures

Digitally signed packages can be validated by reference validation and signature validation, as described in §3.2 of "XML-Signature Syntax and Processing", with some modification for OPC-specific constructs.

[*Note*: The steps below do not apply to the validation of signatures that contain application-defined Object elements. *end note*]

The certificate embedded in the KeyInfo element in the Digital Signature XML Signature part shall be used when it is specified.

Reference validation:

First, validate the reference to the OPC-specific Object element following the steps in §3.2.2 of "XML-Signature Syntax and Processing".

Second, for each reference in the Manifest element:

1) validate the reference following the steps in §3.2.2 of “XML-Signature Syntax and Processing”.
2) validate the media type of the referenced part against the media type specified in the reference query component. References are invalid if these two values are different. The string comparison shall be case-insensitive.

Signature validation:

Follow the steps in §3.2.2 of “XML-Signature Syntax and Processing”.

Annex A (normative) Preprocessing for Generating Relative References

Relative references are available for referencing parts. However, in reality, Unicode strings that are similar to but are not strictly relative references are also used to reference parts. [*Example*: "\a.xml" is not a relative reference since the backslash character is disallowed in RFC 3986/3987.] This annex specifies a preprocessing for the conversion of such Unicode strings to relative references.

This preprocessing is neither required nor recommended.

This preprocessing has eight steps. Some package implementers support only some of them.

Commented [rcj14]: Need to reword this.

Commented [MM15R14]: We might want to use "package implementers" as the subject of some sentences.

1) Percent-encode each open bracket ("[") and close bracket ("]").
2) Percent-encode each percent ("%") character that is not followed by a hexadecimal notation of an octet value.
3) Un-percent-encode each percent-encoded unreserved character.
4) Un-percent-encode each forward slash ("/") and back slash ("\").
5) Convert all back slashes to forward slashes.
6) If present in a segment containing non-dot (".") characters, remove trailing dot (".") characters from each segment.
7) Replace each occurrence of multiple consecutive forward slashes ("/") with a single forward slash.
8) If a single trailing forward slash ("/") is present, remove that trailing forward slash.
9) Remove complete segments that consist of three or more dots.

[*Example*:

Examples of Unicode strings converted to IRIs, URIs, and part names are shown below:

Unicode string	IRI	URI	Part name
/a/b.xml	/a/b.xml	/a/b.xml	/a/b.xml
/a/ц.xml	/a/ц.xml	/a/%D1%86.xml	/a/%D1%86.xml
/%41/%61.xml	/%41/%61.xml	/%41/%61.xml	/A/a.xml
/%25XY.xml	/%25XY.xml	/%25XY.xml	/%25XY.xml
/%XY.xml	/%XY.xml	/%25XY.xml	/%25XY.xml
/%2541.xml	/%2541.xml	/%2541.xml	/%2541.xml

Unicode string	IRI	URI	Part name
/../a.xml	/../a.xml	/../a.xml	/a.xml
/./ц.xml	/./ ц.xml	/./%D1%86.xml	/%D1%86.xml
/%2e/%2e/a.xml	/%2e/%2e/a.xml	/%2e/%2e/a.xml	/a.xml
\a.xml	%5Ca.xml	%5Ca.xml	/a.xml
\%41.xml	%5C%41.xml	%5C%41.xml	/A.xml
/%D1%86.xml	/%D1%86.xml	/%D1%86.xml	/%D1%86.xml
\%2e/a.xml	%5C%2e/a.xml	%5C%2e/a.xml	/a.xml

end example]

Commented [rcj16]: I (Murata-san) removed some obviously unnecessary rows and columns, but I have to remove more.

Annex B (normative) Constraints and Clarifications on the use of ZIP Features

B.1 General

The ZIP specification includes a number of features that packages do not support. Some ZIP features are clarified in the context of this document.

B.2 Archive File Header Consistency

Data describing files stored in the archive is substantially duplicated in the Local File Headers and Data Descriptors, and in the File headers within the Central Directory Record. For a ZIP file to be a physical layer for a package, the package implementer shall ensure that the ZIP file holds equal values in the appropriate fields of every File Header within the Central Directory and the corresponding Local File Header and Data Descriptor pair, when the Data Descriptor exists, except as described in Table B–5 for bit 3 of general-purpose bit flags.

B.3 Data Descriptor Signature

Packages may contain a 4-byte signature value 0x08074b50 at the beginning of Data Descriptors, immediately before the crc-32 field. Package implementers should be able to read packages, whether or not a signature exists.

B.4 Table Key

- "Yes" — During consumption of a package, a "Yes" value for a field in a table in Annex B indicates a package implementer shall support reading the ZIP file containing this record or field, however, support might mean ignoring. During production of a package, a "Yes" value for a field in a table in Annex B indicates that the package implementer shall write out this record or field.
- "No" — A "No" value for a field in a table in Annex B indicates the package implementer should not use this record or field.
- "Optional" — An "Optional" value for a record in a table in Annex B indicates that package implementers might write this record during production.
- "Partially, details below" — A "Partially, details below" value for a record in a table in Annex B indicates that the record contains fields that might not be supported by package implementers during production or consumption. See the details in the corresponding table to determine requirements.

- “Only used when needed” — The value “Only used when needed” associated with a record in a table in Annex C indicates that the package implementer shall use the record only when needed to store data in the ZIP file.

Table B–1 specifies the requirements for package production, consumption, and editing in regard to particular top-level records or fields described in the ZIP Appnote.txt. [*Note*: In this context, editing means in-place modification of individual records. A format specification can require editing applications to instead modify content in-memory and re-write all parts and relationships on each save in order to maintain more rigorous control of ZIP record usage. *end note*]

Table B–1. Support for records

Record name	Supported on Consumption	Supported on Production	Pass through on editing
Local File Header	Yes (partially, details below)	Yes (partially, details below)	Yes
File data	Yes	Yes	Yes
Data descriptor	Yes	Optional	Optional
Archive decryption header	No	No	No
Archive extra data record	No	No	No
Central directory structure: File header	Yes (partially, details below)	Yes (partially, details below)	Yes
Central directory structure: Digital signature	Yes (ignore the signature data)	Optional	Optional
Zip64 end of central directory record V1 (from spec version 4.5)	Yes (partially, details below)	Yes (partially, details below, used only when needed)	Optional
Zip64 end of central directory record V2 (from spec version 6.2)	No	No	No
Zip64 end of central directory locator	Yes (partially, details below)	Yes (partially, details below, used only when needed)	Optional
End of central directory record	Yes (partially, details below)	Yes (partially, details below, used only when needed)	Yes

Table B–2 specifies the requirements for package production, consumption, and editing in regard to individual record components described in the ZIP Appnote.txt.

Table B–2. Support for record components

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
Local File Header	Local file header signature	Yes	Yes	Yes
	Version needed to extract	Yes (partially, see Table B–3)	Yes (partially, see Table B–3)	Yes (partially, see Table B–3)
	General purpose bit flag	Yes (partially, see Table B–5)	Yes (partially, see Table B–5)	Yes (partially, see Table B–5)
	Compression method	Yes (partially, see Table B–4)	Yes (partially, see Table B–4)	Yes (partially, see Table B–4)
	Last mod file time	Yes	Yes	Yes
	Last mod file date	Yes	Yes	Yes
	Crc-32	Yes	Yes	Yes
	Compressed size	Yes	Yes	Yes
	Uncompressed size	Yes	Yes	Yes
	File name length	Yes	Yes	Yes
	Extra field length	Yes	Yes	Yes
	File name (variable size)	Yes	Yes	Yes
	Extra field (variable size)	Yes (partially, see Table B–6)	Yes (partially, see Table B–6)	Yes (partially, see Table B–6)
Central directory structure: File header	Central file header signature	Yes	Yes	Yes
	version made by: high byte	Yes	Yes (0 = MS-DOS is default publishing value)	Yes
	Version made by: low byte	Yes	Yes	Yes
	Version needed to extract (see Table B–3 for details)	Yes (partially, see Table B–3)	Yes (1.0, 1.1, 2.0, 4.5)	Yes
	General purpose bit flag	Yes (partially, see Table B–5)	Yes (partially, see Table B–5)	Yes (partially, see Table B–5)
	Compression method	Yes (partially, see Table B–4)	Yes (partially, see Table B–4)	Yes (partially, see Table B–4)

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	Last mod file time (Pass through, no interpretation)	Yes	Yes	Yes
	Last mod file date (Pass through, no interpretation)	Yes	Yes	Yes
	Crc-32	Yes	Yes	Yes
	Compressed size	Yes	Yes	Yes
	Uncompressed size	Yes	Yes	Yes
	File name length	Yes	Yes	Yes
	Extra field length	Yes	Yes	Yes
	File comment length	Yes	Yes (always set to 0)	Yes
	Disk number start	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Internal file attributes	Yes	Yes	Yes
	External file attributes (Pass through, no interpretation)	Yes	Yes (MS DOS default value)	Yes
	Relative offset of local header	Yes	Yes	Yes
	File name (variable size)	Yes	Yes	Yes
	Extra field (variable size)	Yes (partially, see Table B–6)	Yes (partially, see Table B–6)	Yes (partially, see Table B–6)
	File comment (variable size)	Yes	Yes (always set to empty)	Yes
Zip64 end of central directory V1 (from spec version 4.5, only used when needed)	Zip64 end of central directory signature	Yes	Yes	Yes
	Size of zip64 end of central directory	Yes	Yes	Yes
	Version made by: high byte (Pass through, no interpretation)	Yes	Yes (0 = MS-DOS is default publishing value)	Yes
	Version made by: low byte	Yes	Yes (always 4.5 or above)	Yes
	Version needed to extract (see Table B–3 for details)	Yes (4.5)	Yes (4.5)	Yes (4.5)

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	Number of this disk	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Number of the disk with the start of the central directory	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Total number of entries in the central directory on this disk	Yes	Yes	Yes
	Total number of entries in the central directory	Yes	Yes	Yes
	Size of the central directory	Yes	Yes	Yes
	Offset of start of central directory with respect to the starting disk number	Yes	Yes	Yes
	Zip64 extensible data sector	Yes	No	Yes
Zip64 end of central directory locator (only used when needed)	Zip64 end of central dir locator signature	Yes	Yes	Yes
	Number of the disk with the start of the zip64 end of central directory	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Relative offset of the zip64 end of central directory record	Yes	Yes	Yes
	Total number of disks	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
End of central directory record	End of central dir signature	Yes	Yes	Yes
	Number of this disk	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Number of the disk with the start of the central directory	Yes (partial — no multi disk archive)	Yes (always 1 disk)	Yes (partial — no multi disk archive)
	Total number of entries in the central directory on this disk	Yes	Yes	Yes

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	Total number of entries in the central directory	Yes	Yes	Yes
	Size of the central directory	Yes	Yes	Yes
	Offset of start of central directory with respect to the starting disk number	Yes	Yes	Yes
	ZIP file comment length	Yes	Yes	Yes
	ZIP file comment	Yes	No	Yes

Table B–3 specifies the detailed production, consumption, and editing requirements for the Extract field, which is fully described in the ZIP Appnote.txt.

Table B–3. Support for Version Needed to Extract field

Version	Feature	Supported on Consumption	Supported on Production	Pass through on editing
1.0	Default value	Yes	Yes	Yes
1.1	File is a volume label	Yes (do not interpret as a part)	No	(rewrite/remove)
2.0	File is a folder (directory)	Yes (do not interpret as a part)	No	(rewrite/remove)
2.0	File is compressed using Deflate compression	Yes	Yes	Yes
2.0	File is encrypted using traditional PKWARE encryption	No	No	No
2.1	File is compressed using Deflate64(tm)	No	No	No
2.5	File is compressed using PKWARE DCL Implode	No	No	No
2.7	File is a patch data set	No	No	No
4.5	File uses ZIP64 format extensions	Yes	Yes	Yes
4.6	File is compressed using BZIP2 compression	No	No	No

Version	Feature	Supported on Consumption	Supported on Production	Pass through on editing
5.0	File is encrypted using DES	No	No	No
5.0	File is encrypted using 3DES	No	No	No
5.0	File is encrypted using original RC2 encryption	No	No	No
5.0	File is encrypted using RC4 encryption	No	No	No
5.1	File is encrypted using AES encryption	No	No	No
5.1	File is encrypted using corrected RC2 encryption	No	No	No
5.2	File is encrypted using corrected RC2-64 encryption	No	No	No
6.1	File is encrypted using non-OAEP key wrapping	No	No	No
6.2	Central directory encryption	No	No	No

Table B–4 specifies the detailed production, consumption, and editing requirements for the Compression Method field, which is fully described in the ZIP Appnote.txt.

Table B–4. Support for Compression Method field

Code	Method	Supported on Consumption	Supported on Production	Pass through on editing
0	The file is stored (no compression)	Yes	Yes	Yes
1	The file is Shrunk	No	No	No
2	The file is Reduced with compression factor 1	No	No	No
3	The file is Reduced with compression factor 2	No	No	No
4	The file is Reduced with compression factor 3	No	No	No
5	The file is Reduced with compression factor 4	No	No	No
6	The file is Imploded	No	No	No
7	Reserved for Tokenizing compression algorithm	No	No	No

Code	Method	Supported on Consumption	Supported on Production	Pass through on editing
8	The file is Deflated	Yes	Yes	Yes
9	Enhanced Deflating using Deflate64™	No	No	No
10	PKWARE Data Compression Library Imploding	No	No	No
11	Reserved by PKWARE	No	No	No

Table B–5 specifies the detailed production, consumption, and editing requirements when utilizing these general-purpose bit flags within records.

Table B–5. Support for modes/structures defined by general-purpose bit flags

Bit	Feature	Supported on Consumption	Supported on Production	Pass through on editing
0	If set, indicates that the file is encrypted.	No	No	No
1, 2	**Bit 2** / **Bit 1**	Yes	Yes	Yes
	0 / 0: Normal (-en) compression option was used.			
	0 / 1: Maximum (-exx/-ex) compression option was used.			
	1 / 0: Fast (-ef) compression option was used.			
	1 / 1: Super Fast (-es) compression option was used.			
3	If this bit is set, the fields crc-32, compressed size, and uncompressed size are set to zero in the local header. The correct values are put in the data descriptor immediately following the compressed data.	Yes	Yes	Yes
4	Reserved for use with method 8, for enhanced deflating	No	Bits set to 0	Yes
5	If this bit is set, this indicates that the file is compressed patched data. (Requires PKZIP version 2.70 or greater.)	No	Bits set to 0	Yes

Bit	Feature	Supported on Consumption	Supported on Production	Pass through on editing
6	Strong encryption. If this bit is set, you should set the version needed to extract value to at least 50 and you shall set bit 0. If AES encryption is used, the version needed to extract value shall be at least 51.	No	Bits set to 0	Yes
7	Currently unused	No	Bits set to 0	Yes
8	Currently unused	No	Bits set to 0	Yes
9	Currently unused	No	Bits set to 0	Yes
10	Currently unused	No	Bits set to 0	Yes
11	Currently unused	No	Bits set to 0	Yes
12	Unused	No	Bits set to 0	Yes
13	Used when encrypting the Central Directory to indicate selected data values in the Local Header are masked to hide their actual values. See the section describing the Strong Encryption Specification for details.	No	Bits set to 0	Yes
14	Unused	No	Bits set to 0	Yes
15	Unused	No	Bits set to 0	Yes

Table B–6 specifies the detailed production, consumption, and editing requirements for the Extra field entries reserved by PKWARE and described in the ZIP Appnote.txt.

Table B–6. Support for Extra field (variable size), PKWARE-reserved

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x0001	ZIP64 extended information extra field	Yes	Yes	Optional
0x0007	AV Info	No	No	Yes

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x0008	Reserved for future Unicode file name data (PFS)	No	No	Yes
0x0009	OS/2	No	No	Yes
0x000a	NTFS	No	No	Yes
0x000c	OpenVMS	No	No	Yes
0x000d	Unix	No	No	Yes
0x000e	Reserved for file stream and fork descriptors	No	No	Yes
0x000f	Patch Descriptor	No	No	Yes
0x0014	PKCS#7 Store for X.509 Certificates	No	No	Yes
0x0015	X.509 Certificate ID and Signature for individual file	No	No	Yes
0x0016	X.509 Certificate ID for Central Directory	No	No	Yes
0x0017	Strong Encryption Header	No	No	Yes
0x0018	Record Management Controls	No	No	Yes
0x0019	PKCS#7 Encryption Recipient Certificate List	No	No	Yes
0x0065	IBM S/390 (Z390), AS/400 (I400) attributes — uncompressed	No	No	Yes
0x0066	Reserved for IBM S/390 (Z390), AS/400 (I400) attributes — compressed	No	No	Yes
0x4690	POSZIP 4690 (reserved)	No	No	Yes

Table B–7 specifies the detailed production, consumption, and editing requirements for the Extra field entries reserved by third parties and described in the ZIP Appnote.txt.

Table B–7. Support for Extra field (variable size), third-party extensions

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x07c8	Macintosh	No	No	Yes
0x2605	ZipIt Macintosh	No	No	Yes

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x2705	ZipIt Macintosh 1.3.5+	No	No	Yes
0x2805	ZipIt Macintosh 1.3.5+	No	No	Yes
0x334d	Info-ZIP Macintosh	No	No	Yes
0x4341	Acorn/SparkFS	No	No	Yes
0x4453	Windows NT security descriptor (binary ACL)	No	No	Yes
0x4704	VM/CMS	No	No	Yes
0x470f	MVS	No	No	Yes
0x4b46	FWKCS MD5 (see below)	No	No	Yes
0x4c41	OS/2 access control list (text ACL)	No	No	Yes
0x4d49	Info-ZIP OpenVMS	No	No	Yes
0x4f4c	Xceed original location extra field	No	No	Yes
0x5356	AOS/VS (ACL)	No	No	Yes
0x5455	extended timestamp	No	No	Yes
0x554e	Xceed unicode extra field	No	No	Yes
0x5855	Info-ZIP Unix (original, also OS/2, NT, etc)	No	No	Yes
0x6542	BeOS/BeBox	No	No	Yes
0x756e	ASi Unix	No	No	Yes
0x7855	Info-ZIP Unix (new)	No	No	Yes
0xa220	Padding, Microsoft	No	Optional	Optional
0xfd4a	SMS/QDOS	No	No	Yes

The package implementer shall ensure that all 64-bit stream record sizes and offsets have the high-order bit = 0.

The package implementer shall ensure that all fields that contain "number of entries" do not exceed `2,147,483,647`.

Annex C
(normative)
Schemas - W3C XML Schema

C.1 General

This Part of ISO/IEC 29500 includes a family of schemas defined using the W3C XML Schema 1.0 syntax. The normative definitions of these schemas reside in an accompanying file named OpenPackagingConventions-XMLSchema.zip, which is distributed in electronic form.

C.2 Media Types Stream

This schema is available in the file opc-contentTypes.xsd. The schema documentation is also available.

C.3 Core Properties Part

This schema is available in the file opc-coreProperties.xsd. The schema documentation is also available.

C.4 Digital Signature XML Signature Markup

This schema is available in the file opc-digSig.xsd. The schema documentation is also available.

C.5 Relationships Part

This schema is available in the file opc-relationships.xsd. The schema documentation is also available.

Annex D
(informative)
Schemas - RELAX NG

D.1 General

This Part of ISO/IEC 29500 includes a family of schemas defined using the RELAX NG syntax. The definitions of these schemas reside in an accompanying file named OpenPackagingConventions-RELAXNG.zip, which is distributed in electronic form.

If discrepancies exist between the RELAX NG version of a schema and its corresponding XML Schema, the XML Schema is the definitive version.

D.2 Media Types Stream

This schema is available in the file opc-contentTypes.rnc.

D.3 Core Properties Part

The schema is available in the file opc-coreProperties.rnc.

D.4 Digital Signature XML Signature Markup

This schema is available in the file opc-digSig.rnc.

D.5 Relationships Part

This schema is available in the file opc-relationships.rnc.

D.6 Additional Resources

D.6.1 XML

This schema is available in the file xml.rnc.

D.6.2 XML Digital Signature Core

The schema in D.4 relies on two schemas from XML Security RELAX NG Schemas [3], security_any.rnc and xmldsig-core-schema.rnc.

Annex E (normative) Standard Namespaces and Media Types

The namespaces available for use in a package are listed in Table E–1, Package-wide namespaces

Table E–1. Package-wide namespaces

Description	Namespace URI
Media Types stream	http://schemas.openxmlformats.org/package/2006/content-types
Core Properties	http://schemas.openxmlformats.org/package/2006/metadata/core-properties
Digital Signatures	http://schemas.openxmlformats.org/package/2006/digital-signature
Relationships	http://schemas.openxmlformats.org/package/2006/relationships

The media types for the parts defined in this specification a package are listed in Table E–2, Package-wide media types

Table E–2. Package-wide media types

Description	Media type
Core Properties part	application/vnd.openxmlformats-package.core-properties+xml
Digital Signature Certificate part	application/vnd.openxmlformats-package.digital-signature-certificate
Digital Signature Origin part	application/vnd.openxmlformats-package.digital-signature-origin
Digital Signature XML Signature part	application/vnd.openxmlformats-package.digital-signature-xmlsignature+xml
Relationships part	application/vnd.openxmlformats-package.relationships+xml

The relationship types available for use in a package are listed in Table E–3, Package-wide relationship types.

Table E–3. Package-wide relationship types

Description	Relationship Type
Core Properties	http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties

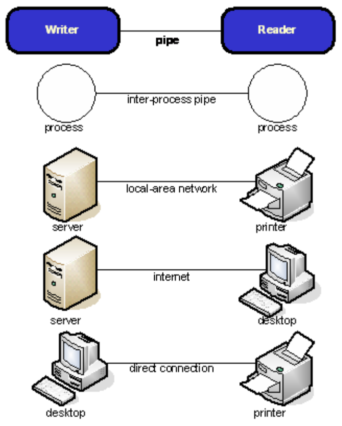
Description	Relationship Type
Digital Signature	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/signature
Digital Signature Certificate	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/certificate
Digital Signature Origin	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/origin
Thumbnail	http://schemas.openxmlformats.org/package/2006/relationships/metadata/thumbnail

Annex F (informative) Physical Package Model Design Considerations

F.1 General

The physical package model defines the ways in which packages are produced and consumed. This model is based on three components: a producer, a consumer, and a pipe between them.

Figure F–1. Components of the physical package model



A *producer* is software or a device that *writes* packages. A *consumer* is software or a device that *reads* packages. A *device* is hardware, such as a printer or scanner that performs a single function or set of functions. Data is carried from the producer to the consumer by a *pipe*.

In *local access*, the pipe carries data directly from a producer to a consumer on a single device.

In *networked access* the consumer and the producer communicate with each other over a protocol. The significant communication characteristics of this pipe are speed and request latency. For example, this communication might occur across a process boundary or between a server and a desktop computer.

In order to maximize performance, designers of physical formats consider access style, layout style, and communication style.

F.2 Access Styles

F.2.1 General

The *access style* in which local access or networked access is conducted determines the simultaneity possible between processing and input-output operations.

F.2.2 Direct Access Consumption

Direct access consumption allows consumers to request the specific portion of the package desired, without sequentially processing the preceding parts of the package. For example, a byte-range request. This is the most common access style.

F.2.3 Streaming Consumption

Streaming consumption allows consumers to begin processing parts before the entire package has arrived. Physical formats should be designed to allow consumers to begin interpreting and processing the data they receive before all of the bits of the package have been delivered through the pipe.

The earlier editions of this document defined requirements for streaming consumption. This edition dropped them since different applications of OPC require different requirements on streaming consumption.

However, to allow streaming consumption, it is recommended that the Media Types stream have no Default elements and should have one Override element for each part in the package. Each Override element should appear before or in close proximity to the part to which it corresponds.

F.2.4 Streaming Creation

Streaming creation allows producers to begin writing parts to the package without knowing in advance all of the parts that are to be written. For example, when an application begins to build a print spool file package, it might not know how many pages the package contains. Likewise, a program that is generating a report might not know initially how long the report is or how many pictures it has.

In order to support streaming creation, the package implementer should allow a producer to add parts after other parts have already been added. A Consumer shall not require a producer to state how many parts they might create when they start writing. The package implementer should allow a producer to begin writing the contents of a part without knowing the ultimate length of the part.

F.2.5 Simultaneous Creation and Consumption

Simultaneous creation and consumption allows streaming creation and streaming consumption to happen at the same time on a package. Because of the benefits that can be realized within pipelined architectures that use it, the package implementer should support simultaneous creation and consumption in the physical package.

F.3 Layout Styles

F.3.1 General

The style in which parts are ordered within a package is referred to as the *layout style*. Parts can be arranged in one of two styles: simple ordering or interleaved ordering.

F.3.2 Simple Ordering

With *simple ordering*, parts are arranged contiguously. When a package is delivered sequentially, all of the bytes for the first part arrive first, followed by all of the bytes for the second part, and so on. When such a package uses simple ordering, all of the bytes for each part are stored contiguously.

F.3.3 Interleaved Ordering

With *interleaved ordering*, pieces of parts are interleaved, allowing optimal performance in certain scenarios. For example, interleaved ordering improves performance for multi-media playback, where video and audio are delivered simultaneously and inline resource referencing, where a reference to an image occurs within markup.

By breaking parts into pieces and interleaving those pieces, it is possible to optimize performance while allowing easy reconstruction of the original contiguous part.

Because of the performance benefits it provides, package implementers should support interleaving in the physical package. The package implementer might handle the internal representation of interleaving differently in different physical package models. Regardless of how the physical package model handles interleaving, a part that is broken into multiple pieces in the physical file is considered one logical part; the pieces themselves are not parts and are not addressable.

F.4 Communication Styles

F.4.1 General

The style in which a package and its parts are delivered by a producer or accessed by a consumer is referred to as the *communication style*. Communication can be based on sequential delivery of or random access to parts. The communication style used depends on the capabilities of both the pipe and the physical format.

F.4.2 Sequential Delivery

With *sequential delivery*, all of the physical bits in the package are delivered in the order they appear in the. Generally, all pipes support sequential delivery.

F.4.3 Random Access

Random access allows consumers to request the delivery of a part out of sequential physical order. Some pipes are based on protocols that can enable random access. For example, HTTP 1.1 with byte-range support. In order to maximize performance, the package implementer should support random access in both the pipe and the physical package. In the absence of this support, consumers need to wait until the parts they need are delivered sequentially.

Annex G (informative) Differences Between ISO/IEC 29500 and ECMA-376:2006

G.1 General

This annex documents the syntactic differences between the versions of the Open Packaging Specification defined in ISO/IEC 29500 and ECMA-376:2006.

G.2 XML Elements

The following XML elements are included in ISO/IEC 29500 but are not included in ECMA-376:2006:

- The value element (in the Core Properties Part schema in §C.3)

The following XML elements are included in ECMA-376:2006 but are not included in ISO/IEC 29500:2011:

- The contentType element (in the Core Properties Part schema in §C.3)

G.3 XML Attributes

No changes.

G.4 XML Enumeration Values

No changes.

G.5 XML Simple Types

No changes.

Annex H (informative) Primer

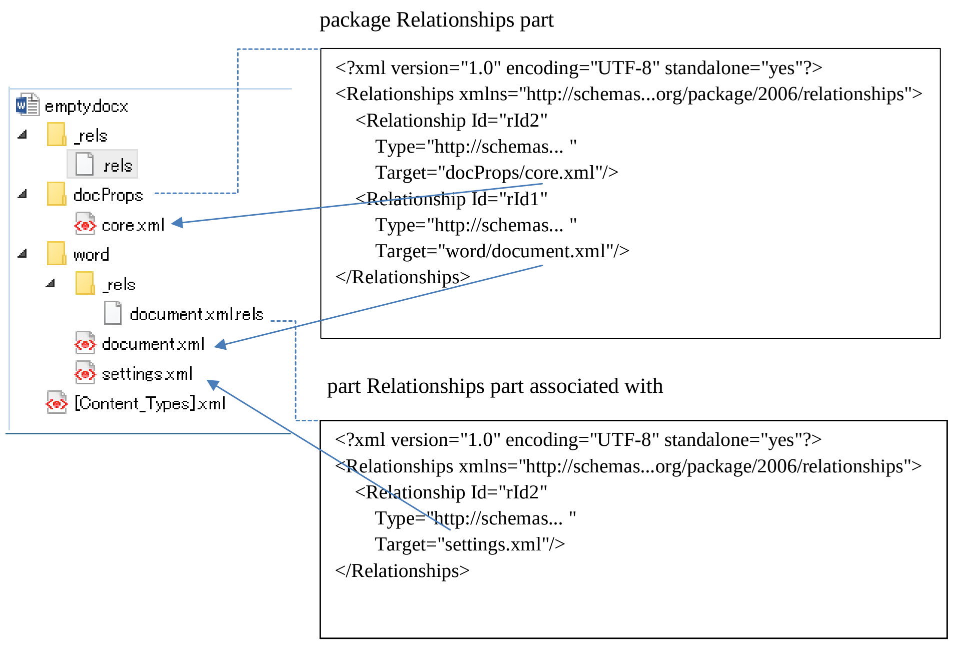
H.1 General

This annex depicts an abstract package and a physical package representing a WordprocessingML document.

H.2 Abstract Package

This abstract package contains five parts: /_rels/.rels, /docProps/core.xml, /word/_rels/document.xml.rels, /word/document.xml, and /word/settings.xml.

Figure H–1. An example abstract package



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas...org/package/2006/relationships">
   <Relationship Id="rId2"
      Type="http://schemas... "
      Target="docProps/core.xml"/>
   <Relationship Id="rId1"
      Type="http://schemas... "
      Target="word/document.xml"/>
</Relationships>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas...org/package/2006/relationships">
   <Relationship Id="rId2"
      Type="http://schemas... "
      Target="settings.xml"/>
</Relationships>
```

Two of these parts are Relationships parts (/_rels/.rels and /word/_rels/document.xml.rels) and three of them are non-Relationships parts (/word/document.xml, /docProps/core.xml, and /word/settings.xml), where _rels/.rels is a package Relationships part and /word/_rels/document.xml.rels is a part Relationships part associated with /word/document.xml. The package Relationships part contains two relationships from the package to /docProps/core.xml and /word/document.xml, respectively. The part Relationships part contains a relationship from /word/document.xml to /word/settings.xml.

H.3 Physical Package

This physical package (empty.docx) is a ZIP file. The ZIP items in this ZIP file are _rels/.rels, docProps/core.xml, word/_rels/document.xml.rels, word/document.xml, and word/settings.xml, and [Content_Types].xml.

Except [Content_Types].xml, these ZIP items represent parts. Note that part names have "/" as the first character. [Content_Types].xml represents the Media Types stream.

Bibliography

The following documents are useful references for implementers and users of this International Standard, in addition to the Normative References:

[1] *Character Model for the World Wide Web: String Matching and Searching*, W3C Working Draft 07 April 2016, https://www.w3.org/TR/2016/WD-charmod-norm-20160407/

[2] *Date and Time Formats*, W3C NOTE 19980827, 1997, http://www.w3.org/TR/1998/NOTE-datetime-19980827

[3] *XML Security RELAX NG Schemas*, W3C Working Group Note 11 April 2013, https://www.w3.org/TR/xmlsec-rngschema/

[4] ISO/IEC TR 30114-1, *Information technology -- Extensions of Office Open XML file formats -- Part 1: Guidelines*