

ISO/IEC 29500-2:201x

**Office Open XML File Formats — Open Packaging
Conventions**

Working DRAFT WD3.3

2018-01-09

Contents

Foreword	vii
Introduction	ix
1 Scope	1
2 Conformance	2
3 Normative References	3
4 Terms and Definitions	5
5 Notational Conventions	11
6 General Description	12
7 Overview	13
8 Package Model	15
8.1 General	15
8.2 Parts	15
8.2.1 General	15
8.2.2 Part Names	15
8.2.3 Media types	17
8.2.4 Growth Hint	17
8.2.5 XML Usage	17
8.3 Part Addressing	18
8.3.1 General	18
8.3.2 Pack Scheme	19
8.3.3 Resolving a Pack IRI to a Resource	20
8.3.4 Composing a Pack IRI	21
8.3.5 Equivalence	22
8.4 Resolving Relative References	22
8.4.1 General	22
8.4.2 Base IRIs	22
8.4.3 Examples	23
8.5 Relationships	24
8.5.1 General	24
8.5.2 Relationships Part	25
8.5.3 Relationship Markup	26
8.5.4 Examples	28
8.5.5 Support for Versioning and Extensibility	32
9 Physical Package	33
9.1 General	33
9.2 Physical Mapping Guidelines	33
9.2.1 General	33
9.2.2 Mapped Components	33
9.2.3 Mapping Media Types to Parts	34

9.2.4	Mapping Part Names to Physical Package Item Names.....	38
9.2.5	Interleaving	40
9.3	Mapping to a ZIP Archive	42
9.3.1	General.....	42
9.3.2	Mapping Part Data	43
9.3.3	ZIP Item Names	43
9.3.4	Mapping Part Names to ZIP Item Names	43
9.3.5	Mapping ZIP Item Names to Part Names	43
9.3.6	ZIP Package Limitations.....	44
9.3.7	Mapping the Media Types Stream.....	44
9.3.8	Mapping the Growth Hint	44
9.3.9	Late Detection of ZIP Items Unfit for Streaming Consumption	45
9.3.10	ZIP Format Clarifications for Packages	45
10	Core Properties.....	46
10.1	General	46
10.2	Core Properties Part.....	47
10.3	Core Properties Markup.....	47
10.3.1	General.....	47
10.3.2	coreProperties element	48
10.3.3	Property elements from Dublin Core Metadata Element Set, Version 1.1.....	48
10.3.4	Property Elements from DCMI Metadata Terms	49
10.3.5	Property Elements defined in this Document.....	49
10.4	Support for Versioning and Extensibility.....	52
11	Thumbnails.....	58
12	Digital Signatures.....	59
12.1	General	59
12.2	Choosing Content to Sign	59
12.3	Digital Signature Parts	59
12.3.1	General.....	59
12.3.2	Digital Signature Origin Part.....	60
12.3.3	Digital Signature XML Signature Part	60
12.3.4	Digital Signature Certificate Part.....	60
12.4	Digital Signature Markup.....	61
12.4.1	General	61
12.4.2	Signature Element.....	62
12.4.3	SignedInfo Element	62
12.4.4	CanonicalizationMethod Element.....	62
12.4.5	SignatureMethod Element.....	62
12.4.6	Reference Element.....	62
12.4.7	Transforms Element	63
12.4.8	Transform Element.....	63
12.4.9	DigestMethod Element	64
12.4.10	Object Element.....	64
12.4.11	KeyInfo Element	64
12.4.12	Manifest Element	64
12.4.13	SignatureProperties Element	65

12.4.14 SignatureProperty Element	65
12.4.15 SignatureTime Element	65
12.4.16 Format Element	65
12.4.17 Value Element	65
12.4.18 RelationshipReference Element	65
12.4.19 RelationshipsGroupReference Element	66
12.4.20 Relationships Transform Algorithm	66
12.5 Digital Signature Example	67
12.6 Generating Signatures	69
12.7 Validating Signatures	70
12.7.1 General	70
12.7.2 Signature Validation and Streaming Consumption	71
12.8 Support for Versioning and Extensibility	71
12.8.1 General	71
12.8.2 Using Relationship Types	71
12.8.3 Markup Compatibility Namespace for Package Digital Signatures	72
Annex A (normative) Preprocessing for Generating Relative References	73
Annex B (normative) ZIP Appnote.txt Clarifications	75
B.1 General	75
B.2 Archive File Header Consistency	75
B.3 Data Descriptor Signature	75
B.4 Table Key	75
Annex C (normative) Schemas - W3C XML Schema	86
C.1 General	86
C.2 Media Types Stream	86
C.3 Core Properties Part	87
C.4 Digital Signature XML Signature Markup	88
C.5 Relationships Part	89
Annex D (informative) Schemas - RELAX NG	90
D.1 General	90
D.2 Media Types Stream	90
D.3 Core Properties Part	91
D.4 Digital Signature XML Signature Markup	91
D.5 Relationships Part	93
D.6 Additional Resources	93
D.6.1 XML	93
D.6.2 XML Digital Signature Core	93
Annex E (normative) Standard Namespaces and Media Types	94
Annex F (informative) Physical Model Design Considerations	96
F.1 General	96
F.2 Access Styles	97
F.2.1 General	97
F.2.2 Direct Access Consumption	97
F.2.3 Streaming Consumption	97

F.2.4	Streaming Creation	97
F.2.5	Simultaneous Creation and Consumption	97
F.3	Layout Styles.....	97
F.3.1	General.....	97
F.3.2	Simple Ordering.....	98
F.3.3	Interleaved Ordering.....	98
F.4	Communication Styles.....	98
F.4.1	General	98
F.4.2	Sequential Delivery	98
F.4.3	Random Access.....	98
Annex G (informative)	Guidelines for Meeting Conformance	99
G.1	General	99
G.2	Package Model	99
G.3	Physical Packages	105
G.4	ZIP Physical Mapping.....	110
G.5	Core Properties.....	114
G.6	Thumbnail.....	114
G.7	Digital Signatures.....	115
G.8	Pack URI.....	120
Annex H (informative)	Differences Between ISO/IEC 29500 and ECMA-376:2006	122
H.1	General	122
H.2	XML Elements.....	122
H.3	XML Attributes.....	122
H.4	XML Enumeration Values	122
H.5	XML Simple Types.....	122
Bibliography		123

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.
- **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 29500-2:201x(E)

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, D, and F form a normative part of this Part of ISO/IEC 29500. Annexes E, G, H, and I 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 I 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 Conformance

Each conformance requirement is given a unique ID comprised of a letter (M – MANDATORY; S – SHOULD; O – OPTIONAL), an identifier for the topic to which it relates, and a unique ID within that topic. (Producers and consumers might use these IDs to report error conditions.) Mandatory requirements are those stated with the normative terms “shall”, “shall not”, or any of their normative equivalents. Should requirements are those stated with the normative terms “should”, “should not”, or any of their normative equivalents. Optional requirements are those stated with the normative terms “can”, “cannot”, “might”, “might not”, or any of their normative equivalents.

[*Example*: Package implementers shall not map logical item name(s) mapped to the Media Types stream in a ZIP archive to a part name. [M3.11] *end example*]

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.

Commented [JH2]: Delete if/when Appendix G references are removed

3 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*.

Dublin Core Element Set v1.1. <http://purl.org/dc/elements/1.1/>

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

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>.

W3C NOTE 19980827, *Date and Time Formats*, Wicksteed, Charles, and Misha Wolf, 1997, <http://www.w3.org/TR/1998/NOTE-datetime-19980827>.

ISO/IEC 29500-2:201x(E)

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, 27 June 2001.

XML Schema Part 1: Structures, W3C Recommendation, 28 October 2004.

XML Schema Part 2: Datatypes, W3C Recommendation, 28 October 2004.

XML-Signature Syntax and Processing, W3C Recommendation, 12 February 2002.

.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*]

4 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.

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .1~~

~~**access style**~~

~~style in which local access or networked access is conducted~~

Commented [MM3]: Should be deleted, since it is used mainly in informational Appendix F.

4.1

behavior

external appearance or action

4.2

behavior, implementation-defined

behavior, application-defined

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

4.3

behavior, unspecified

behavior where this Open Packaging specification imposes no requirements

4.4

byte

sequence of 8 bits treated as a unit

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .6~~

~~**communication style**~~

~~style in which package contents are delivered by a producer or received by a consumer~~

Commented [MM4]: Do we need this term in this clause? It is mainly used in informative Appendix F.

4.5

consumer

software or a device that reads packages through a package implementer

4.6

device

hardware, such as a personal computer, printer, or scanner, that performs a single function or set of functions

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .9~~

format consumer

consumer that consumes packages conforming to a format designer's specification

Commented [MM5]: Do we really need this term? It appears in informative Appendix G only

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .10~~

format designer

author of a particular file format specification built on this Open Packaging Conventions specification **derived format**

Commented [MM6]: I think that we should define "format" rather than "formant designer" and make clear that OPC-based formats may introduce additional restrictions.

a particular format built on this Open Packaging Conventions specification by imposing additional restrictions

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .11~~

format producer

producer that produces packages conforming to a format designer's specification

Commented [MM7]: Do we really need this term? It appears in informative Appendix G only. The only exception is in 9.2.3.2.1, but it is clearly a mistake. It should be "producer".

Commented [JH8]: Do a full pass over the document to clean up or remove uses of producer/consumer/implementer/designer.

4.7

growth hint

suggested number of bytes to reserve for a part to grow in-place

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .13~~

id

a name from an identification scheme

Commented [MM9]: What does this mean?

4.8

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

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .15~~

layout style

style in which the collection of parts in a physical package is laid out

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .16~~

local access

access architecture in which a pipe carries data directly from a producer to a consumer on a single device

Commented [MM10]: Do we need this term? It appears in Appendix F only.

4.9

logical item

either a non-interleaved part or a piece of an interleaved part

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .18~~

networked access

access architecture in which a consumer and the producer communicate over a protocol, such as across a process boundary, or between a server and a desktop computer

Commented [MM11]: Do we need this term? It appears in Appendix F only.

4.10

pack URI

IRI that conforms to the pack URI scheme

pack URI scheme

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

4.12

package

logical entity that holds a collection of parts

4.13

package implementer

software that implements the physical input-output operations to a package according to the requirements and recommendations of this Open Packaging specification

4.14

package model

package abstraction that holds a collection of parts

4.15

part

stream of bytes with a MIME media type and associated common properties

4.16

part name

path component of a pack URI

4.17

physical model

description of the capabilities of a particular physical format

4.18

physical package format

specific file format, or other persistence or transport mechanism that can represent all of the capabilities of a package

4.19

piece

portion of a part.

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .29~~

pipe

~~communication mechanism that carries data from the producer to the consumer~~

Commented [WD3.212]: At the Feb/Mar 2016 meeting, decide whether to use pack URI or pack IRI, and change usage globally.

Note, after doing a search of the 2012 spec in Prague, we found no uses of pack IRI.

Commented [rcj13]: Check existing uses of pack URI that should be pack URI scheme.

Commented [WD314]: Is this correct/adequate?

Commented [MM15R14]: Not quite correct. See 8.2.2.

Commented [MM16]: Here, the word "format" is used differently.

Commented [MM17]: Do we really need this term? It appears mainly in informative Appendix F.

4.20

producer

software or a device that writes packages through a package implementer

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .31~~

~~**random access**~~

~~style of communication between the producer and the consumer of the package~~

Commented [MM18]: This term appears only in informative Appendix F.

4.21

relationship

a package relationship or a part relationship

4.22

relationship, package

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

4.23

relationship, part

connection from a part to another part in a package, or to an external resource

4.24

relationship type

absolute IRI for specifying the role of a relationship

4.25

Relationships part

part containing an XML representation of relationships

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。 .37~~

~~**sequential delivery**~~

~~communication style in which all of the physical bits in the package are delivered in the order they appear in the package~~

Commented [MM19]: This term appears only in informative Appendix F.

4.26

signature policy

format-defined policy that specifies what configuration of parts and relationships shall or might be included in a signature for that format and what additional behaviors that producers and consumers of that format shall follow when applying or verifying signatures following that format's signature policy

4.27

simple ordering

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

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。40~~

~~**simultaneous creation and consumption**~~

~~style of access between a producer and a consumer in highly pipelined environments where streaming creation and streaming consumption occur simultaneously~~

Commented [MM20]: Should be dropped since this term appears only in Appendix F.

4.28

source

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

4.29

stream

linearly ordered sequence of bytes

4.30

streaming consumption

access style in which parts of a physical package can be processed by a consumer before all of the bits of the package have been delivered through the pipe

Commented [MM21]: This "part" is different from OPC parts.

Commented [MM22]: Since "process" is not defined, this entire definition is meaningless. In the context of XPS, streaming consumption means that a printer can start to print the first page even before the end of the XPS package is received.

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。44~~

~~**streaming creation**~~

~~production style in which a producer dynamically adds parts to a package after other parts have been added without modifying those parts~~

Commented [MM23]: Should be dropped since this term appears only in Appendix F.

4.31

target

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

4.32

thumbnail

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

~~エラー! [ホーム] タブを使用して、ここに表示する文字列に Heading 1 を適用してください。47~~

~~**unique identifier**~~

~~a unique name from an identification scheme~~

Commented [MM24]: Should we drop this again?

4.33

XSD

W3C XML Schema

4.34

ZIP archive

ZIP file as defined in the ZIP file format specification

ISO/IEC 29500-2:201x(E)

4.35

ZIP item

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

5 Notational Conventions

5.1 Document 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 §4 (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*]

6 General Description

Commented [rcj25]: Needs to be checked once all edits are completed.

This Open Packaging specification is divided into the following subdivisions:

- 1) Front matter (Clauses 1–6);
- 2) Overview (Clause 7);
- 3) Main body (Clauses 8–12);
- 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 Open Packaging specification.

The following form the normative part of this Open Packaging specification:

- Introduction
- Clauses 1–6, and 8–12
- Annex A–Annex C
- Annex E

The following form the informative part of this Open Packaging specification:

- Clause 7
- Annex D
- Annex F–Annex H
- All notes
- All examples

Conformance requirements written as requirements for package implementers (e.g., M1.1) are document conformance requirements.

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*]
- 4) [*Guidance: narrative ... end guidance*]

7 Overview

Commented [WD3.226]: Caroline wrote:
4. I think that clause 7 (Overview) would benefit from some general informative statements (and maybe examples) about implementation-dependent possibilities, such as additional constraints (e.g. needing relationship elements to avoid parts being "unknown), new relationship types, etc.

This clause is informative.

This Open Packaging specification describes an abstract model (§8) and physical format conventions (§9) for the use of XML, Unicode, ZIP, and other openly 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. The specification is written for developers who are building systems that process package content.

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

This Part of ISO/IEC 29500 specifies a set of conventions used by Office Open XML documents to define the structure and functionality of a *package* in terms of a package model and a physical model.

The *package model* is a package abstraction that holds a collection of *parts*. 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 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 Part of ISO/IEC 29500.

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

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

This Part of ISO/IEC 29500 specifies requirements for documents, producers, and consumers. Conformance requirements are identified throughout the text of this Part of ISO/IEC 29500. A formal conformance statement is given in §2. An informative summary of requirements relevant to particular classes of developers is given in Annex G.

A primary goal is to ensure the interoperability of independently created software and hardware systems that produce or consume package content and use common services. This Open Packaging specification defines the formal requirements that producers and consumers must satisfy in order to achieve interoperability.

ISO/IEC 29500-2:201x(E)

Various XML-based building blocks within a package make use of the conventions described in Part 3 to facilitate future enhancement and extension of XML markup. That Part must be cited explicitly by any markup specification that bases its versioning and extensibility strategy on Markup Compatibility elements and attributes.

End of informative text.

8 Package Model

8.1 General

This clause specifies an abstract model for a package. The requirements for mapping these concepts to a physical format, including specifically to ZIP files, are given in §9.

A *package* is a container that holds a collection of parts. The purpose of the package is to aggregate constituent components of a document (or other type of content) into a single object. [*Example*: A package holding a document with a picture might contain two parts: an XML markup part representing the document and another part representing the picture. *end example*] The package is also capable of storing relationships between parts.

The package provides a convenient way to distribute documents with all of their constituent components, such as images, fonts, and data. Although this Open Packaging specification defines a single-file package format, the package model allows for the future definition of other physical package representations. [*Example*: A package could be represented physically in a collection of loose files, in a database, or ephemerally in transit over a network connection. *end example*]

This Open Packaging specification also defines a URI scheme, the *pack URI* scheme, that allows URIs to be used as a uniform mechanism for addressing parts within a package.

8.2 Parts

8.2.1 General

A *part* is a stream of bytes with the properties listed in Table 8–1. A *stream* is a linearly ordered sequence of bytes. [*Note*: Parts are analogous to a file in a file system or to a resource on an HTTP server. *end note*]

Table 8–1. Part properties

Name	Description	Required/Optional
Name	The name of the part	Required
Media type	The type of content stored in the part	Required
Growth Hint	A suggested number of bytes to reserve for the part to grow in-place	Optional

8.2.2 Part Names

8.2.2.1 General

Each part shall have a name. *Part names* shall refer to parts within a 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 for the naming of package Relationships parts as specified in subclause 8.5.2.2. Part names in which the second last I18N segment is equivalent to `'_rels'` and the final segment is equivalent to any string ending with `'.rels'` shall be reserved for the naming of part Relationships parts as specified in subclause 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 package model, and no directory of folders exists in the package model. *end note*]

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

8.2.2.3 Part Name Equivalence and Integrity in a Package

Equivalence of part names is determined by ASCII case-insensitive matching. [ASCII case-insensitive matching compares a sequence of code points as if all ASCII code points in the range 0x41 to 0x5A \(A to Z\) were mapped to the corresponding code points in the range 0x61 to 0x7A \(a to z\).](#) [See CharMod2.]

The names of two different parts within a package shall not be equivalent, and the result of applying Unicode Normalization Form C (NFC) to the two names should not be equivalent.

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].

Commented [MM27]: Add a bibliography.

Commented [BCN28]: Merge the following text somehow into the marked paragraph:

Comparison of a character sequence as if all ASCII characters in the range 0x41 to 0x5A (A to Z) were mapped to the corresponding code points in the range 0x61 to 0x7A (a to z).

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 to N1 is equivalent to the result of applying NFC to N2[S].

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

[*Example:* If a package contains a part named `"/a"`, another part in that package must not have `"/a"` or `"/A"` in its name. If a package contains a part named `"/segment1/segment2/.../segmentn"`, other parts in that package must not have names such as `"/segment1"`, `"/SEGMENT1"`, `"/segment1/segment2"`, `"/segment1/SEGMENT2"`, or `"/segment1/segment2/.../segmentn-1"`. If a package contains a part named `"/Å"` where Å is 'ANGSTROM SIGN' (U+212B), another part in that package should not have in its name `"/Ā"` where Ā 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*]

[*Example:* Given N[s] equal to `"/a/b"` where N is `"/a"` and S is `"b"`, then `"/a/b"` is derivable from `"/a"`. A part named `"/é/a"`, where é is 'LATIN SMALL LETTER E' (U+0065) followed by 'COMBINING ACUTE ACCENT' (U+0301) is weakly derivable from `"/ê"`, where ê is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9). *end example*]

[*Note:* Some implementations of the directory structure always apply NFC or NFD normalization. *end note*]

8.2.3 Media types

Each part shall have a 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.

Derived formats might restrict the usage of parameters for media types. [O1.2]

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

8.2.4 Growth Hint

Sometimes a part is modified after it is placed in a package. Depending on the nature of the modification, the part might need to grow. For some physical package formats, this could be an expensive operation and could damage an otherwise efficiently interleaved package. Ideally, the part should be allowed to grow in-place, moving as few bytes as possible.

To support these scenarios, a package implementer can associate a growth hint with a part. [O1.1] The *growth hint* identifies the number of bytes by which the producer predicts that the part might grow. In a mapping to a particular physical format, this information might be used to reserve space to allow the part to grow in-place. This number serves as a hint only. The package implementer might ignore the growth hint or adhere only loosely to it when specifying the physical mapping. [O1.3] If the package implementer specifies a growth hint, it is set when a part is created, and the package implementer shall not change the growth hint after the part has been created. [M1.16]

8.2.5 XML Usage

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

Commented [JH29]: Add example for `"/é == /É/f`

- 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. Package implementers shall enforce this requirement upon creation and retrieval of the XML content. [M1.17]
- 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 Open Packaging specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content and shall treat the presence of DTD declarations as an error. [M1.18]
- 3) Any extensions in the XML content are removed by an MCE processor as specified in ISO/IEC 29500-3 before applying validation and subsequent processing. [M1.19]
- 4) XML content shall be valid against the corresponding XSD schema defined in this Open Packaging specification. 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. Package implementers shall enforce this requirement upon creation and retrieval of the XML content. [M1.20]
- 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 Open Packaging specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content. [M1.21]

8.3 Part Addressing

8.3.1 General

This subclause is informative.

This part of ISO/IEC 29500 defines a way to use IRIs (RFC 3987) to reference part resources inside a package. In particular, the scheme "pack" is introduced in accordance with the guidelines in RFC 4395.

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.

~~The following terms are used in accordance with RFC 3986: authority, fragment, path, pchar, pct-encoded characters, query, reserved characters, resource scheme, segment, sub-delims, and unreserved characters.~~

End of informative subclause.

8.3.2 Pack Scheme

This part of ISO/IEC 29500 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. [M7.2]

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]

ISO/IEC 29500-2:201x(E)

[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%2c%2c%2cwww.my.com%2cpackages.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 resource and a part name.

In order to be suitable for creating a pack IRI, the IRI of a package resource 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 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:

ISO/IEC 29500-2:201x(E)

/a/fooxml

The pack IRI is:

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

end example]

8.3.5 Equivalence

The package implementer shall consider pack IRIs 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 per RFC 3986); *and*
- 3) The path components are equivalent part names, as specified in §8.2.2 [M7.3]

[*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 3987, as extended in RFC 3987, §6.5.

This document introduces no changes to the resolution procedure, but Annex A introduces a preprocessing for generating relative references.

8.4.2 Base IRIs

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

[*Note:* Section 5.1 of RFC 3986 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 (§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/fooxml 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.

This subclause shows examples of resolving relative references when the base is a pack IRI.

Example 1: Leading slash: `/b/bar.xml`

1) `pack://http%3c,example.com,foo.opc/a/foo.xml`

Since this relative reference begins with the slash character, the path component (`/a/foo.xml`) of the base IRI 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,example.com,foo.opc/b/bar.xml`

2) `pack://http%3c,example.com,foo.opc/`

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

Example 2: No leading slash: `bar.xml`

1) `pack://http%3c,example.com,foo.opc/a/foo.xml`

Since this relative reference does not begin with the slash character, the path component (`/a/foo.xml`) of the base IRI and that (`bar.xml`) of the relative reference are merged. The "merge" routine in 5.2.3 first removes "foo.xml" from the path component of the base IRI, and emits `/a/bar.xml`". Thus, the resulting IRI is a pack IRI "pack://http%3c,example.com,foo.opc/a/bar.xml".

2) `pack://http%3c,example.com,foo.opc/`

Since the relative reference does not begin with the slash character, the path component (`/`) of the base IRI and that (`bar.xml`) of the relative reference are merged. The "merge" routine emits `/bar.xml`". Thus, the resulting IRI is a pack IRI "pack://http%3c,example.com,foo.opc/bar.xml".

Example 3: Dot segment: `./bar.xml`

1) `pack://http%3c,example.com,foo.opc/a/foo.xml`

ISO/IEC 29500-2:201x(E)

As in the previous case, the "merge" routine in 5.2.3 removes "foo.xml" from the path component of the base IRI, and emits "/a./bar.xml". But the "remove_dot_segments" routine further removes "./" and emits "/a/bar.xml". Thus, the resulting IRI is a pack IRI

pack://http%3c,example.com,foo.opc/a/bar.xml

2) pack://http%3c,example.com,foo.opc/

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,example.com,foo.opc/bar.xml

Example 4: Dot segment: ../bar.xml

1) pack://http%3c,example.com,foo.opc/a/foo.xml

This case is similar to the previous case, but the "remove_dot_segments" routine removes "a/.". Thus, the resulting IRI is a pack IRI "pack://http%3c,example.com,foo.opc/bar.xml".

2) pack://http%3c,example.com,foo.opc/

The "merge" routine emits "./bar.xml", but the "remove_dot_segments" routine replaces ""./" by "/". Thus, the resulting IRI is a pack IRI pack://http%3c,example.com,foo.opc/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 consumers 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. *Relationships* represent connections from a source part or source package to a target part or target resource. Relationships from parts are called part relationships, while those from packages are called package relationships. 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.

Relationships have identifiers such that they may be distinguished from one another and may be referred to within a source part. Occasionally, it is necessary to associate a relationship with a specific point in a source part. In this case, the identifier of the relationship is embedded at that point in the source part.

Commented [WD330]: In Prague, we agreed to "Reformat these as a series of points, so they are easier to read."

Commented [MM31]: Caroline's comment: Schema and published Part 2 require IDs for relationships.

Commented [MM32R31]: Right, but this subclause is not about the XML representation.

Commented [MM33]: 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 [MM34R33]: 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.

A relationship has a relationship type, 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. A Relationships part is either a package Relationships part or a part Relationships part. A package Relationships part contains all relationships from the package, while a part Relationships part contains all relationships from a specific part.

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 (see §8.5.3.3) 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/`

Commented [MM35]: I do not remember why we dropped "A relationships part shall not be the source of another relationships part."

Caroline thinks that a Relationships part cannot be the target of some relationship. I'm not sure.

ISO/IEC 29500-2:201x(E)

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 (see §8.5.3.3) 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.)

Commented [WD337]: 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.)

Attributes	Description
TargetMode	<p>This attribute specifies whether the target describes a part inside the package or a resource outside the package.</p> <p>Permissible values of the TargetMode attribute are Internal1 and External1.</p> <p>This attribute is optional [O1.5] and the default value is Internal1.</p> <p>The possible values for this attribute are defined by the ST_TargetMode simple type (§8.5.3.3).</p>
Target	<p>This attribute specifies the target of a relationship.</p> <p>This attribute is required.</p> <p>If the value of the TargetMode attribute is Internal1, the Target attribute shall be a relative reference to a part. If the value of the TargetMode attribute is External1, the Target attribute shall be a relative reference or an absolute IRI. Base IRIs for resolving relative references are defined in §8.4.</p> <p>The possible values for this attribute are defined by the xsd:anyURI simple type of the W3C Recommendation “XML Schema Part 2: Datatypes.”.</p>

Attributes	Description
Type	<p>This attribute specifies the role of a relationship, <i>relationship type</i>.</p> <p>This attribute is required.</p> <p>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.</p> <p>[Example: Type="http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/signature" end example]</p> <p>The possible values for this attribute are defined by the xsd:anyURI simple type of the W3C Recommendation "XML Schema Part 2: Datatypes."</p>
Id	<p>This attribute specifies the identifier of a relationship. The value of the Id attribute shall be unique within the Relationships part.</p> <p>This attribute is required. [M1.26]</p> <p>[Example: Id="A5FFC797514BC" end example]</p> <p>The possible values for this attribute are defined by the xsd:ID simple type of the W3C Recommendation "XML Schema Part 2: Datatypes."</p>

Commented [WD338]: Consider adding the essence of the following to this sentence: The package implementer shall require the Type attribute to be a URI that defines the role of the relationship and the format designer shall specify such a Type. [M1.27]

Commented [MM39R38]: I do not think the sentence above any values

[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 Relationship 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 an 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 mode of the first relationship is Internal (default). Thus, the base IRI for resolving ".a.xml" is the pack URI (pack://http%3c,,www.example.com,ex.opc) created from the IRI of the package (http://www.example.com/ex.opc). By resolving ".a.xml", we have 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 mode of the second relationship 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".

Commented [WD340]: Reformat this as a series of points, so it's easier to read.

Commented [WD341]: Reformat this as a series of points, so it's easier to read.

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 an 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".

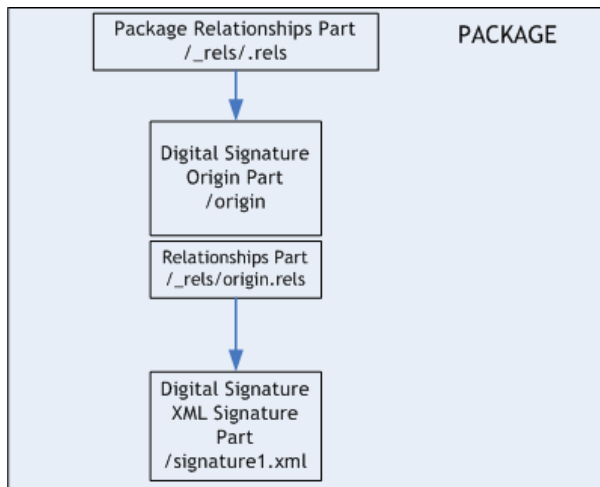
- The mode of the first relationship is Internal (default). Thus, the base IRI ("pack://http%3c,,www.example.com,ex.opc/foo/test.xml") for resolving ".b.xml" is the pack URI created from the IRI (http://www.example.com/ex.opc) of the package and the part name "/foo/test.xml". By resolving ".b.xml", we have "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 mode of the second relationship is External. Thus, the base IRI for resolving ".b.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/b.xml. The relationship type of this relationship is "http://example.com/relTypeExt2". The identifier of this relationship is "IDE2".

Commented [WD342]: Reformat this as a series of points, so it's easier to read.

Commented [WD343]: Reformat this as a series of points, so it's easier to read.

8.5.4.3 Relationships Part around Digital Signature

The figure below shows a Digital Signature Origin part and a Digital Signature XML Signature part. 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 of 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 unique Id values. 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"

```

ISO/IEC 29500-2:201x(E)

```
    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 Part 3 to incorporate elements and attributes drawn from other XML namespaces. [O1.7]

Commented [MM44]: This clause is not very useful, since 8.2.5 already allows the use of MCE in relationships parts

Commented [WD345]: Consider rewriting as a requirement on a document.

9 Physical Package

9.1 General

In contrast to the package model that describes the contents of a package in an abstract way, the physical package refers to a package that is stored in a particular physical file format. This includes the physical model and physical mapping considerations. This clause specifies the requirements for mapping the abstract package model concepts in §8 to a physical format, including specifically to ZIP files as described in §9.3.

The *physical model* abstractly describes the capabilities of a particular physical format, and how producers and consumers can use a package implementer to interact with that physical package format.

[Note: See Annex F for additional discussion of the physical model. *end note*]

Physical mappings describe the manner in which the package contents are mapped to the features of that specific physical format. Details of how package components are mapped are described, as well as common mapping patterns and mechanisms for storing part media types. This Open Packaging specification describes both the specific considerations for physical mapping to a ZIP archive as well as generic physical mapping considerations applicable to any physical package format.

9.2 Physical Mapping Guidelines

9.2.1 General

Whereas the package model defines a package abstraction, an *instance* of a package is based on a physical representation. A *physical package format* is a particular physical representation of the package contents in a file.

Many physical package formats have features that partially match the packaging model components. In defining mappings from the package model to a physical package format, it is advisable to take advantage of any similarities in capabilities between the package model and the physical package medium while using layers of mapping to provide additional capabilities not inherently present in the physical package medium. [Example: Some physical package 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*]

Designers of physical package formats face some common mapping problems. [Example: Associating arbitrary media types with parts and supporting part interleaving. *end example*] Package implementers might use the common mapping solutions defined in this Open Packaging specification. [O2.3]

9.2.2 Mapped Components

The package implementer shall define a physical package format with a mapping for the following required components. [M2.2] [Note: Not all physical package formats support the part-growth hint. *end note*]

Table 9–1. Mapped components

Name	Description	Required/Optional
Package	URI-addressable resource that identifies a package as a whole unit	Required. The package implementer shall provide a physical mapping for the package. [M2.2]
Part name	Names a part	Required. The package implementer shall provide a physical mapping for each part's name. [M2.2]
Part media type	Identifies the kind of content stored in the part	Required. The package implementer shall provide a physical mapping for each part's media type. [M2.2]
Part contents	Stores the actual content of the part	Required. The package implementer shall provide a physical mapping for each part's contents. [M2.2]
Part-growth hint	Number of additional bytes to reserve for possible growth of the part	Optional. The package implementer might provide a physical mapping for a growth hint that might be specified by a producer. [O2.2]

9.2.3 Mapping Media Types to Parts

9.2.3.1 General

The package implementer shall define a format mapping with a mechanism for associating media types with parts. [M2.3]

Some physical package formats 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 packages, the package implementer should use the native mechanism to map part media types to parts. [S2.1]

For all other physical package formats, the package should include an XML stream called the *Media Types stream*. [S2.2] The Media Types stream shall not be mapped to a part by the package implementer. [M2.1] This stream is therefore not 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 contained in the Media Types stream shall have a top-level Types element, and one or more Default and Override child elements. Default elements define default mappings from the extensions of part

names to media types. Override elements specify media types on parts that are not covered by, or are not consistent with, the default mappings. Package producers can use pre-defined Default elements to reduce the number of Override elements on a part, but are not required to do so. [O2.4]

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. [M2.4]

The package implementer shall require that there not be more than one Default element for any given extension, and there not be more than one Override element for any given part name. [M2.5]

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

If the package is intended for streaming consumption:

- The Media Types stream should not have Default elements; consequently, there should be 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.

[S2.3]

The package implementer can define Default media type mappings even though no parts use them. [O2.5]

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 has 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	<p>This attribute shall specify a string as a file extension.</p> <p>This attribute is required.</p> <p>A Default element shall match any part whose name ends with a period (".") followed by the value of this attribute.</p> <p>The possible values for this attribute are defined by the ST_Extension simple type shown in §C.2.</p>

Attributes	Description
ContentType	<p>This attribute shall specify a media type using the syntax defined in RFC 7231 §3.1.1.1.</p> <p>This attribute is required.</p> <p>The specified media type shall apply to any matching parts (unless overridden by Override elements).</p> <p>The possible values for this attribute are defined by the ST_ContentType simple type shown in §C.2.</p>

[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 media types on parts that are not covered by, or are not consistent with, the default mappings.

Attributes	Description
ContentType	<p>This attribute shall specify a media type using the syntax defined in RFC 7231 §3.1.1.1.</p> <p>This attribute is required.</p> <p>The specified media type shall apply to any matching parts.</p> <p>The possible values for this attribute are defined by the ST_ContentType simple type shown in in §C.2.</p>
PartName	<p>This attribute shall specify a part name.</p> <p>This attribute is required.</p> <p>An Override element shall match a part whose name is equal to the value of this attribute.</p> <p>The possible values for this attribute are defined by the xsd:anyURI simple type of the W3C Recommendation “XML Schema Part 2: Datatypes.</p>

[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 [M2.8]:

- 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.
 - 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.

Commented [JH46]: Should this be changed?

- 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.

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 [M2.9]:

- 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.
- 5) If neither Override nor Default elements with matching attributes are found for the specified part name, the implementation shall not map this part name to a part.

9.2.3.6 Support for Versioning and Extensibility

The package implementer shall not use the versioning and extensibility mechanisms defined in Part 3 to incorporate elements and attributes drawn from other XML-namespaces into the Media Types stream markup. [M2.10]

9.2.4 Mapping Part Names to Physical Package Item Names

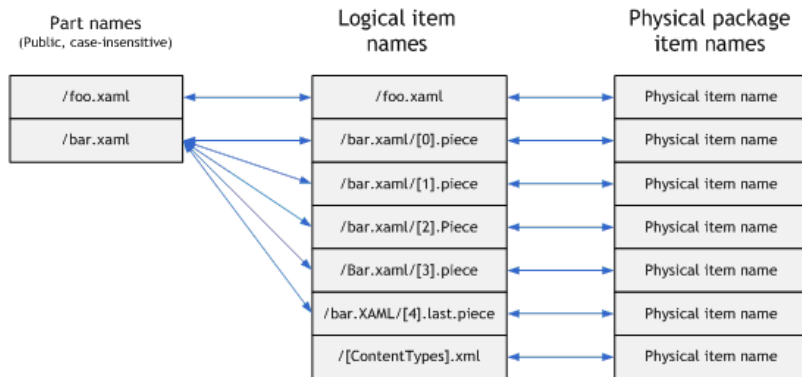
9.2.4.1 General

The mapping of part names to the names of items in the physical package uses an intermediate *logical item name* abstraction. This logical item name abstraction allows package implementers to manipulate physical data items consistently regardless of whether those data items can be mapped to parts or not or whether the package is laid out with simple ordering or interleaved ordering. See §9.2.5 for interleaving details.

[Example:

Figure 9–1 illustrates the relationship between part names, logical item names, and physical package item names.

Figure 9–1. Part names and logical item names



Commented [JH47]: Remove [ContentTypes].xml box. Agreed in Beijing/

end example]

9.2.4.2 Logical Item Names

Logical item names have the following syntax:

```

LogicalItemName = PrefixName [SuffixName]
PrefixName      = *AChar
AChar           = %x20-7E
SuffixName      = "/" "[" PieceNumber "]" [".last"] ".piece"
PieceNumber     = "0" | NonZeroDigit [1*Digit]
Digit           = "0" | NonZeroDigit
NonZeroDigit    = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

```

[Note: Piece numbers identify the individual pieces of an interleaved part. *end note*]

The package implementer shall compare prefix names by ASCII case-insensitive matching. [M2.12]

The package implementer shall compare suffix names by ASCII case-insensitive matching. [M2.13]

Logical item names are considered equivalent if their prefix names and suffix names are equivalent. The package implementer shall not allow packages that contain equivalent logical item names. [M2.14] The package implementer shall not allow packages that contain logical items with equivalent prefix names and with equal piece numbers, where piece numbers are treated as integer decimal values. [M2.15]

Logical item names that use suffix names 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.

9.2.4.3 Mapping Part Names to Logical Item Names

Non-interleaved part names are mapped to logical item names that have an equivalent prefix name and no suffix name.

Interleaved part names are mapped to the complete sequence of logical item names with an equivalent prefix name.

[*Note:* Prefix names mapped to part names correspond to the part names grammar (§8.2.2). In particular, prefix names can hold percent-encoded characters. For example, a logical name of “%C3%B1.ext” results in a ZIP item name of “%C3%B1.ext”, not “ñ.ext” (interpreted as a 2-byte UTF-8 sequence). *end note*]

9.2.4.4 Mapping Logical Item Names and Physical Package Item Names

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

9.2.4.5 Mapping Logical Item Names to Part Names

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

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

The package implementer might allow a package that contains 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 or the logical item does not have an associated media type. [O2.7] The package implementer shall not map logical items to parts if the logical item names violate the part naming rules. [M2.16]

The package implementer shall consider naming collisions within the set of part names mapped from logical item names to be an error. [M2.17]

9.2.5 Interleaving

Not all physical packages natively support interleaving of the data streams of parts. The package implementer should use the mechanism described in this Open Packaging specification to allow interleaving when mapping to the physical package for layout scenarios that support streaming consumption. [S2.4]

The interleaving mechanism breaks the data stream of a part into *pieces*, which can be interleaved with pieces of other parts or with whole parts. Pieces are named using a unique mapping from the part name, defined in §9.2.4. This enables a consumer to join the pieces together in their original order, forming the data stream of the part.

The individual pieces of an interleaved part exist only in the physical package and are not addressable in the packaging model. A piece might be empty.

An individual part shall be stored either in an interleaved or non-interleaved fashion. The package implementer shall not mix interleaving and non-interleaving for an individual part. [M2.11] A derived format specifies whether the use of interleaving is allowed or not. [O2.1]

The grammar for deriving piece names from a given part name is defined by the logical item name grammar as defined in §9.2.4.2. A suffix name is mandatory.

The package implementer should store pieces in their natural order for optimal efficiency. [S2.5] The package implementer might create a physical package containing interleaved parts and non-interleaved parts. [O2.6]

[Example:

Example 9–2. ZIP archive contents

A ZIP archive might contain the following 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
```

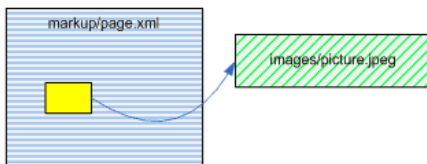
end example]

Under certain scenarios, interleaved ordering can provide important performance benefits, as demonstrated in the following example.

[Example:

Example 9–3. 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 consumer is unable to display the image until it has received *all* of the page part *and* the image part. 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 on the consumer's system.

ISO/IEC 29500-2:201x(E)



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 consumer to begin processing the image as soon as it encounters the reference.



end example]

9.3 Mapping to a ZIP Archive

9.3.1 General

This Open Packaging specification defines a mapping for the ZIP archive format. Future versions of this Open Packaging specification might provide additional mappings.

A *ZIP archive* is a ZIP file as defined in the ZIP file format specification excluding all elements of that specification related to encryption, decryption, or digital signatures. A ZIP archive contains *ZIP items*. [Note: ZIP items become files when the archive is unzipped. 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 9–2, shows the various components of the package model and their corresponding physical representation in a ZIP archive.

Table 9–2. Package model components and their physical representations

Package model component	Physical representation
Package	ZIP archive file
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.

Package model component	Physical representation
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

In a ZIP archive, the data associated with a part is represented as one or more items.

A package implementer shall store a non-interleaved part as a single ZIP item. [M3.1] When interleaved, a package implementer shall represent a part as one or more pieces, using the method described in §9.2.5. [M2.18] Pieces are named using the specified pattern, making it possible to rebuild the entire part from its constituent pieces. Each piece is stored within a ZIP archive as a single ZIP item.

In the ZIP archive, the chunk of bits that represents an item is stored contiguously. A package implementer might intentionally order the sequence of ZIP items in the archive to enable an efficient organization of the part data in order to achieve correct and optimal interleaving. [O3.1]

9.3.3 ZIP Item Names

ZIP item names are case-sensitive ASCII strings. Package implementers shall create ZIP item names that conform to ZIP archive-file name grammar. [M3.2] Package implementers shall create item names that are unique within a given archive. [M3.3]

9.3.4 Mapping Part Names to ZIP Item Names

To map part names to ZIP item names the package implementer shall perform, in order, the following steps [M3.4]:

- 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.
- 2) 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.

The package implementer shall not map a logical item name or complete sequence of logical item names sharing a common prefix to a part name if the logical item prefix has no corresponding media type. [M3.5]

9.3.5 Mapping ZIP Item Names to Part Names

To map ZIP item names to part names, the package implementer shall perform, in order, the following steps [M3.6]:

- 1) Map the ZIP item names to logical item names by adding a forward slash ("/") to each of the ZIP item names.
- 2) Map the obtained logical item names to part names. For more information, see §9.2.4.5.

9.3.6 ZIP Package Limitations

The package implementer shall map all ZIP items to parts except MS-DOS ZIP items, as defined in the ZIP specification, that are not MS-DOS files. [M3.7]

[*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*]

In ZIP archives, the package implementer shall not exceed 65,535 bytes for the combined length of the item name, Extra field, and Comment fields. [M3.8] Accordingly, part names stored in ZIP archives are limited to 65,535 characters, subtracting the size of the Extra and Comment fields.

Package implementers should restrict part naming to accommodate file system limitations when naming parts to be stored as ZIP items. [S3.1]

[*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. Package implementers shall enforce this restriction. [M3.9]

The compression algorithm supported is DEFLATE, as described in the .ZIP specification. The package implementer shall not use any compression algorithm other than DEFLATE.

9.3.7 Mapping the Media Types Stream

In ZIP archives, the Media Types stream shall be stored in an item with the prefix name “[Content_Types].xml” or, in the interleaved case, in the complete sequence of logical items with that prefix name.

Package implementers shall not map logical item name(s) mapped to the Media Types stream in a ZIP archive to a part name. [M3.11] [*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

In a ZIP archive, the growth hint is used to reserve additional bytes that can be used to allow an item to grow in-place. The padding is stored in the Extra field, as defined in the ZIP file format specification. If a growth hint is

used for an interleaved part, the package implementer should store the Extra field containing the growth hint padding with the item that represents the first piece of the part. [S3.2]

The format of the ZIP item's Extra field, when used for growth hints, is shown in Table 9–3, Structure of the Extra field for growth hints below.

Table 9–3. 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 the producer when the item is created
<padding>	[Padding Length]	Should be filled with NULL characters

9.3.9 Late Detection of ZIP Items Unfit for Streaming Consumption

Several substantial conditions that represent a package unfit for streaming consumption might be detected mid-processing by a streaming package implementer. These include:

- A duplicate ZIP item name is detected the moment the second ZIP item with that name is encountered. Duplicate ZIP item names are not allowed. [M3.3]
- In interleaved packages, an incomplete sequence of ZIP items is detected when the last ZIP item is received. Because one of the interleaved pieces is missing, the entire sequence of ZIP items cannot be mapped to a part and is therefore invalid. [M2.16]
- An inconsistency between the local ZIP item headers and the ZIP central directory file headers is detected at the end of package consumption, when the central directory is processed.
- A ZIP item that is not a file, according to the file attributes in the ZIP central directory, is detected at the end of package consumption, when the central directory is processed. Only a ZIP item that is a file shall be mapped to a part in a package.

When any of these conditions are detected, the streaming package implementer shall generate an error, regardless of any processing that has already taken place. Package implementers shall not generate a package containing any of these conditions when generating a package intended for streaming consumption. [M3.13]

9.3.10 ZIP Format Clarifications for Packages

The ZIP format includes a number of features that packages do not support. Some ZIP features are clarified in the package context. See Annex B for package-specific ZIP information.

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

<u>Property</u>	<u>Specification</u>	<u>Description</u>
<u>category</u>	<u>Open Packaging Conventions</u>	<u>A categorization of the content of this package.</u>
<u>contentStatus</u>	<u>Open Packaging Conventions</u>	<u>The status of the content.</u>
<u>created</u>	<u>Dublin Core</u>	<u>Date of creation of the resource.</u>
<u>creator</u>	<u>Dublin Core</u>	<u>An entity primarily responsible for making the content of the resource.</u>
<u>description</u>	<u>Dublin Core</u>	<u>An explanation of the content of the resource.</u>
<u>identifier</u>	<u>Dublin Core</u>	<u>An unambiguous reference to the resource within a given context.</u>
<u>keywords</u>	<u>Open Packaging Conventions</u>	<u>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.</u>
<u>language</u>	<u>Dublin Core</u>	<u>The language of the intellectual content of the resource. [Note: IETF RFC 3066 provides guidance on encoding to represent languages. end note]</u>
<u>lastModifiedBy</u>	<u>Open Packaging Conventions</u>	<u>The user who performed the last modification. The identification is environment-specific.</u>
<u>lastPrinted</u>	<u>Open Packaging Conventions</u>	<u>The date and time of the last printing.</u>
<u>modified</u>	<u>Dublin Core</u>	<u>Date on which the resource was changed.</u>
<u>revision</u>	<u>Open Packaging Conventions</u>	<u>The revision number.</u>

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

Editor's note: [Caroline wrote: If it would be permitted under ISO rules, I would like the group to consider whether putting links to the DC elements into the table makes sense. An example is <http://purl.org/dc/terms/subject>]

Commented [rcj48]:

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. The requirements specified in 8.2.5 apply.

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 Dublin Core Metadata Element Set, Version 1.1 as core property elements.

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

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 Dublin Core Metadata Element Set, Version 1.1.

```
<dc:creator>Alan Shen</dc:creator>
<dc:title>OPC Core Properties</dc:title>
```

Commented [O49]: Introduce a table and normative references.

```
<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" 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.

[Note: 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]

Commented [O50]: REFEREMCEW3c DTF

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]

It is recommended that this value be as concise as possible.

This element has 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]

Commented [O51]: Get rid of this meaningless sentence.

Commented [O52]: Shall have no

Commented [O53]: Introduce time

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 Open Packaging specification does not provide any requirements or guidelines for new parts or relationship types that are used to extend core properties. end note]

Editor's note: Add a reference to the extensibility TR and have the TR in a bibliography.

Commented [O54]: Drop it.

Commented [O55R54]: Add a reference to the TR (part 1)

Commented [rcj56]:

10—Core Properties

10.1—General

Core properties enable users to get and set well-known and common sets of property metadata within packages. The core properties and the Standard that describes them are shown in Table 10–1, “Core properties”. The namespaces for the properties in this table in the Open Packaging Conventions domain are defined in Annex E.

Core property elements are non-repeatable. They can be empty or omitted. The Core Properties Part can be omitted if no core properties are present.

Table 10–1. Core properties

Property	Domain	Description
category	Open Packaging Conventions	A categorization of the content of this 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]
contentStatus	Open Packaging Conventions	The status of the content. [Example: Values might include “Draft”, “Reviewed”, and “Final”. end-example]
created	Dublin-Core	Date of creation of the resource
creator	Dublin-Core	An entity primarily responsible for making the content of the resource
description	Dublin-Core	An explanation of the content of the resource. [Example: Values might include an abstract, table of contents, reference to a graphical representation of content, and a free-text account of the content. end-example]
identifier	Dublin-Core	An unambiguous reference to the resource within a given context

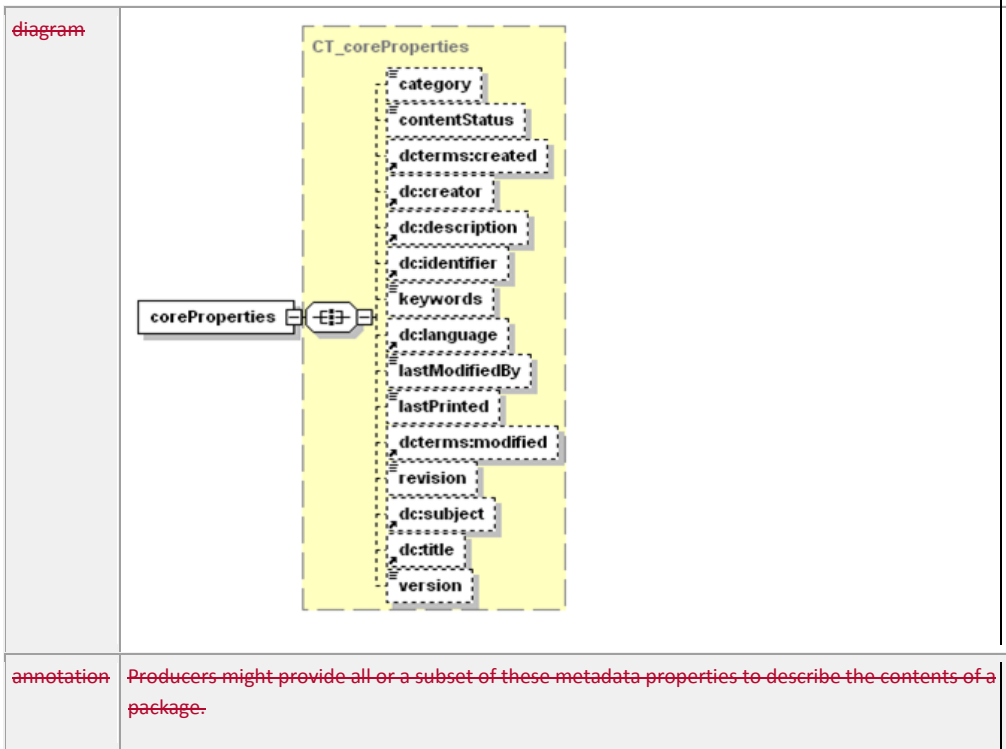
Property	Domain	Description
keywords	Open Packaging Conventions	<p>A delimited set of keywords to support searching and indexing. This is typically a list of terms that are not available elsewhere in the properties.</p> <p>The definition of this element uniquely allows for:</p> <ul style="list-style-type: none"> • Use of the <code>xml:lang</code> attribute to identify languages • A mixed content model, such that keywords can be flagged individually <p><i>[Example: The following instance of the keywords element has keywords in English (Canada), English (U.S.), and French (France):</i></p> <pre><keywords xml:lang="en-US"> color <value xml:lang="en-CA">colour</value> <value xml:lang="fr-FR">couleur</value> </keywords></pre> <p><i>end-example]</i></p>
language	Dublin-Core	The language of the intellectual content of the resource. <i>[Note: IETF RFC 3066 provides guidance on encoding to represent languages. end note]</i>
lastModifiedBy	Open Packaging Conventions	The user who performed the last modification. The identification is environment specific. <i>[Example: A name, email address, or employee ID. end-example]</i> It is recommended that this value be as concise as possible.
lastPrinted	Open Packaging Conventions	The date and time of the last printing
modified	Dublin-Core	Date on which the resource was changed
revision	Open Packaging Conventions	The revision number. <i>[Example: This value might indicate the number of saves or revisions, provided the application updates it after each revision. end-example]</i>
subject	Dublin-Core	The topic of the content of the resource
title	Dublin-Core	The name given to the resource
version	Open Packaging Conventions	The version number. This value is set by the user or by the application.

10.2 Core Properties Part

Core properties are stored in XML in the Core Properties part. Core properties are stored in an XML document in the Core Properties part. The requirements (including MCE processing before validation and subsequent processing) specified in 8.2.5 apply.

The Core Properties part media type is defined in Annex E.

The structure of the CoreProperties element is shown in the following diagram:



[Example:

Example 10-1. Core properties markup

An example of a core properties part is illustrated by this example:

```
<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:subject>Spec defines the schema for OPC Core Properties and their
location within the package</dc:subject>
<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>


```

end example}

10.3—Location of Core Properties Part

The location of the Core Properties part within the package is determined by traversing a well-defined package relationship, as listed in Annex E. The format designer shall specify and the format producer shall create at most one core properties relationship for a package. A format consumer shall consider more than one core properties relationship for a package to be an error. If present, the relationship shall target the Core Properties part. [M4.1]

10.4—Support for Versioning and Extensibility

The format designer shall not specify and the format producer shall not create Core Properties that use the Markup Compatibility namespace as defined in Annex E. A format consumer shall consider the use of the Markup Compatibility namespace to be an error. [M4.2] Instead, 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 Open Packaging specification does not provide any requirements or guidelines for new parts or relationship types that are used to extend core properties.

10.5—Schema Restrictions for Core Properties

The following restrictions apply to every XML document instance that contains Open Packaging Conventions core properties:

- 1) ~~Producers shall not create a document element that contains refinements to the Dublin Core elements, except for the two specified in the schema: <dcterms:created> and <dcterms:modified>. Consumers shall consider a document element that violates this constraint to be an error. [M4.3]~~
- 2) ~~Producers shall not create a document element that contains the xml:lang attribute at any other location than on the keywords or value elements. Consumers shall consider a document element that violates this constraint to be an error. [M4.4] For Dublin Core elements, this restriction is enforced by applications.~~
- 3) ~~Producers shall not create a document element that contains the xsi:type attribute, except for a <dcterms:created> or <dcterms:modified> element where the xsi:type attribute shall be present and~~

shall hold the value `dc:terms:W3CDTF`, where `dc:terms` is the namespace prefix of the Dublin Core namespace. Consumers shall consider a document element that violates this constraint to be an error. [M4.5]

11 Thumbnails

A derived format might allow images, called *thumbnails*, to be used to help end-users identify parts of a package or a package as a whole. These images can be generated by the producer and stored as parts. [O5.1]

Thumbnail parts shall be identified by either a part relationship or a package relationship. The producer shall build the package accordingly. [M5.1] For information about the relationship type for Thumbnail parts, see Annex E.

12 Digital Signatures

12.1 General

A derived format might allow a package to include digital signatures, which identify the parts of a package that have been signed and the process for validating the signatures. This clause describes how the package digital signature framework applies the W3C Recommendation “XML-Signature Syntax and Processing” (referred to here as the “XML Digital Signature specification”) and XML Advanced Electronic Signatures (XAdES). In addition to complying with the XML Digital Signature specification, producers and consumers also apply the modifications specified in §1.1.1.

12.2 Choosing Content to Sign

Any part or relationship in a package can be signed, including Digital Signature XML Signature parts themselves. An entire Relationships part or a subset of relationships can be signed. By signing a subset, other relationships can be added, removed, or modified without invalidating the signature.

Because applications use the package format to store various types of content, application designers that include digital signatures should define signature policies that are meaningful to their users. A signature policy specifies which portions of a package should not change in order for the content to be considered intact. To ensure validity, some clients require that *all* of the parts and relationships in a package be signed. Others require that *selected* parts or relationships be signed and validated to indicate that the content has not changed. The digital signature infrastructure in packages provides flexibility in defining the content to be signed, while allowing parts of the package to remain changeable.

12.3 Digital Signature Parts

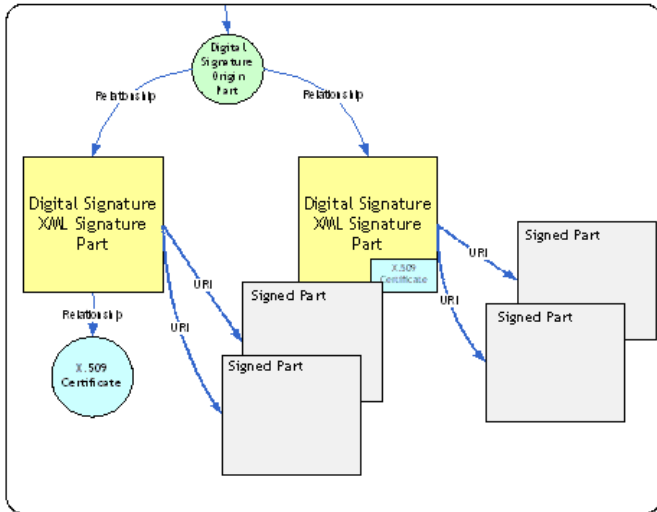
12.3.1 General

The digital signature parts consist of the Digital Signature Origin part, Digital Signature XML Signature parts, and Digital Signature Certificate parts. Relationship names and media types relating to the use of digital signatures in packages are specified in Annex E.

[Example:

Figure 12–1 shows a signed package with signature parts, signed parts, and an X.509 certificate. The example Digital Signature Origin part references 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]

12.3.2 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 root. [M6.1] This part shall exist if the package contains any Digital Signature XML Signature parts, and is optional otherwise. [O6.2] Relationships to the Digital Signature XML Signature parts are defined in the Relationships part. No content should exist in the Digital Signature Origin part itself. [S6.1]

12.3.3 Digital Signature XML Signature Part

A Digital Signature XML Signature part contains digital signature markup. 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. [M6.3] One or more of these parts may exist in a package. [O6.4]

12.3.4 Digital Signature Certificate Part

If present, the Digital Signature Certificate part contains an X.509 certificate for validating the signature.

Alternatively, ~~instead of using a Digital Signature Certificate part,~~ the certificate may exist as a separate part in the package, may be embedded within the Digital Signature XML Signature part itself, or may be excluded from the package if certificate data is known or can be obtained from a local or remote certificate store. [O6.5]

The package digital signature infrastructure supports X.509 certificate technology for signer authentication.

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. [M6.4] The part containing the certificate may be signed. [O6.6] The media types 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. [O6.7] A Digital Signature Certificate part should be the target of at least one Digital Signature Certificate relationship from a Digital Signature XML Signature part. [S6.2]

12.4 Digital Signature Markup

12.4.1 General

The markup described here includes additional requirements on some elements and attributes from the XML Digital Signature specification and some package-specific markup. For a complete example of a digital signature, see §1.1.

~~Drafting Note: Introduce the concept of the package specific Object element (12.4.11.1) here? Overview example?~~

~~Drafting note: The requirements (including MCE processing before validation and subsequent processing) specified in 8.2.5 apply.~~

12.4.2 Modifications to the XML Digital Signature Specification

~~The package modifications to the XML Digital Signature specification are summarized below.~~

~~{Note: All modifications to XML Digital Signature markup occur in locations where the XML Signature schema allows any namespace. Therefore, package digital signature XML is valid against the XML Signature schema. end note}~~

- ~~1) Reference elements within a SignedInfo element shall reference elements only within the same Signature element. Reference elements within a SignedInfo element shall not reference any resources outside the same Signature element. [M6.5] Reference elements within a SignedInfo element should reference an Object element. [S6.5] Packages shall not contain references to a package specific Object element that contains a transform other than a canonicalization transform. [M6.6]~~
- ~~2) The Signature element shall contain only one package specific Object element. [M6.7] {Note: A signature may contain other Object elements that are not package specific. end note}~~
- ~~3) Package specific Object elements shall contain exactly one Manifest element and exactly one SignatureProperties element. Package specific Object elements shall not contain other types of elements. [M6.8] The following constraints on the Manifest element, the SignatureProperties element, and their descendants apply:

 - ~~a) Reference elements within a Manifest element shall reference with their URI attributes only parts within the package. [M6.9] Relative references to these local parts shall have query components that specify the part media type as described in §12.4.7. The relative reference excluding the query component shall conform to the part name grammar. [M6.10] Reference~~~~

Commented [JH57]: 1. Update all references to sections in XMLDSIG spec if the normative reference is updated. Or remove them?

2. Unify pluralization and capitalization of "Relationship(s) (T/t)ransform"

Commented [MM58R57]: In this revision, we do not update references to XMLDSIG.

Commented [MM59]: The next subclause "Modifications to the XML Digital Signature Specification" was deleted, since it is neither an informative summary nor a complete list of requirements. Rather, it was a mixture. I thus moved normative requirements in that subclause to somewhere else in 12.4 and then entirely deleted 12.4.2.

Commented [BCN60]:

Commented [JH61]: This subclause might be redesigned?

Commented [MM62]: Some requirements in this subclause are summaries of detailed requirements specified elsewhere, while others are specified nowhere in this document.

~~elements shall have query components that specify in a case-sensitive manner the media type of the referenced part. [M6.11]~~

- ~~b) Reference elements within a Manifest element shall not contain transforms other than the canonicalization transform and relationships transform. [M6.12]~~
- ~~c) If an optional Relationships transform is used, it shall be followed by a canonicalization transform. [M6.13]~~
- ~~d) Exactly one SignatureProperty element with the Id attribute value set to idSignatureTime shall exist for a given signature. The Target attribute value of this element shall be either empty or contain a fragment reference to the value of the Id attribute of the root Signature element. A SignatureProperty element shall contain exactly one SignatureTime child element. [M6.14]~~

Commented [MM63]: Should we move these requirements to Subclause 12.4.9

~~12.4.3~~ 12.4.2 Signature Element

The structure of a Signature element is defined in §4.1 of XML-Signature Syntax and Processing.

A Signature element shall contain exactly one local-data, package-specific Object element and zero or more application-defined Object elements. [M6.15]

~~12.4.4~~ 12.4.3 SignedInfo Element

The structure of a SignedInfo element is defined in §4.3 of XML-Signature Syntax and Processing.

A SignedInfo element shall contain exactly one reference to the package-specific Object element. [M6.16]

~~12.4.5~~ 12.4.4 CanonicalizationMethod Element

The structure of a CanonicalizationMethod element is 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)

They shall be supported for validation.

[M6.34]

~~12.4.6~~ 12.4.5 SignatureMethod Element

The structure of a SignatureMethod element is defined in §4.3.2 of XML-Signature Syntax and Processing.

~~Producers shall support DSA and RSA algorithms to produce signatures. Consumers shall support DSA and RSA algorithms to validate signatures. [M6.17]~~

Commented [BCN64]: Needs work. Should OPC restrict the set of allowed algorithms?

Eventually, remove mention of producer/consumer.

12.4.6 Reference Element

~~-as a Child of a Manifest Element~~

Commented [MM65R64]: I believe that this paragraph adds nothing new. It merely repeats what is described in XMLDSig. Should we drop it or convert it to a note?

The structure of a Reference element is defined in §4.3.3 of XML-Signature Syntax and Processing, [but this document introduces further requirements](#).

12.4.6.1 Reference Element as a Child of a SignedInfo Element

[Reference elements within a SignedInfo element shall reference elements only within the same Signature element. Reference elements within a SignedInfo element shall not reference any resources outside the same Signature element. \[M6.5\] Reference elements within a SignedInfo element should reference an Object element. \[S6.5\]](#)

12.4.6.2 Reference Element as a Child of a Manifest Element

Each Reference element that is a child of a Manifest element shall contain a URI attribute whose value contains a part name without a fragment identifier. [M6.18]

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.

~~[Note: See §12.4.2 for additional requirements on Reference elements. end note]~~

[Example:

Example 12–4. 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]

12.4.7 Transforms Element

The structure of a Transforms element is defined in §4.3.3.4 of XML-Signature Syntax and Processing.

12.4.8 Transform Element

The structure of a Transform element is defined in §4.3.3.4 of XML-Signature Syntax and Processing.

~~Only one of the following transform algorithms shall be used specified:~~

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

[These algorithms shall be supported for validation](#)

Commented [BCN66]: Needs work. Should OPC restrict the set of allowed algorithms?

Commented [MM67R66]: Give up for this revision. Possibly in the next amendment.

Commented [BCN68]: Clarify which version of these algorithms we mean.

Relationships transforms shall only be used when the Transform element is a descendant element of a Manifest element and followed by a canonicalization transform [M6.19]

A Relationships transform describes how the Relationship elements from the Relationships XML are filtered using ID and/or Type attribute values. For algorithm details, see §12.4.20.

The URI for a Relationships transform is:

`http://schemas.openxmlformats.org/package/2005/06/RelationshipTransform`

12.4.9 DigestMethod Element

The structure of a DigestMethod element is defined in §4.3.3.5 of XML-Signature Syntax and Processing.

RSA-SHA1 algorithms shall be used specified by a DigestMethod element. [M6.17] They shall be supported for validation.

12.4.10 Object Element

The structure of an Object element is defined in §4.2.5 of XML-Signature Syntax and Processing, but this document introduces further requirements.

~~The An~~ Object element ~~can shall~~ be either package-specific or application-defined.

12.4.10.1 Package-Specific Object Element

~~The A~~ package-specific Object element ~~shall contain s the~~ Manifest element followed by a and SignatureProperties element, and no other elements that are package specific. Each Signature element shall have exactly one package-specific Object. [M6.15] The Id attribute of the package-specific Object element shall be ~~specified~~ specified, and its value shall be `"_idPackageObject_"`.

12.4.10.2 Application-Defined Object Element

~~The An~~ application-defined Object element specifies application-defined information. A derived format ~~The format designer~~ might permit one or more application-defined Object elements. ~~If allowed by the format designer, signatures may contain one or more application-defined Object elements. [O6.8] Such elements shall contain XML-compliant data. [M6.20] Derived f~~ Formats designers might not apply package-specific restrictions regarding URIs and Transform elements to application-defined Object elements. [O6.9]

12.4.11 KeyInfo Element

The structure of a KeyInfo element is defined in §4.4 of XML-Signature Syntax and Processing.

The certificate embedded in the Digital Signature XML Signature part shall be used when it is specified. [M6.21]

12.4.12 Manifest Element

The structure of a Manifest element is defined in §4.5.4.1 of XML-Signature Syntax and Processing.

Commented [MM69]: Changed so that MS word becomes conformant.

Commented [BCN70]: Should OPC restrict the set of allowed algorithms?

Commented [MM71R70]: Give up for this revision. Possibly in the next amendment. As in the previous edition, restricted to SHA1.

The Manifest element within a package-specific Object element ~~contains~~ shall contain references to the signed parts of the package. ~~Such a Manifest element and~~ shall not reference any data outside of the package. [M6.22]

12.4.13 SignatureProperties Element

The structure of a SignatureProperties element is defined in §5.2 of XML-Signature Syntax and Processing.

12.4.14 SignatureProperty Element as a Child of a package-specific Object Element

12.4.14.1 SignatureProperty Element within a Package-Specific Object Element

A SignatureProperty element within a package-specific Object element shall specify "idSignatureTime" by the Id attribute, and shall only have one child element, which shall be 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.

12.4.14.2 SignatureProperty Element within an Application-Defined Object Element

Derived formats may introduce requirements on SignatureProperty elements within package-specific Object elements.

12.4.13 12.4.15 SignatureTime Element

The SignatureTime element holds the date/time stamp for the signature. This element is package-specific, and its namespace is shown in Annex E. A SignatureTime element shall only occur as a child element of a SignatureProperty element. The schema definition for the SignatureTime element is specified in Annex C.4.

12.4.14 12.4.16 Format Element

The Format element specifies the format of the date/time stamp. This element is package-specific, and its namespace is shown in Annex E. The date/time format shall conform to the syntax described in the W3C Note "Date and Time Formats". [M6.23] The schema definition for the Format element is specified in Annex C.4.

12.4.15 12.4.17 Value Element

The Value element specifies the value of the date/time stamp. This element is package-specific, and its namespace is shown in Annex E. The value shall conform to the format specified in the Format element. [M6.24] The schema definition for the Value element is specified in Annex C.4.

12.4.16 12.4.18 RelationshipReference Element

The RelationshipReference element specifies the Relationship element with the specified Id value is to be signed. This element is package-specific, and its namespace is shown in Annex E. A RelationshipsReference element shall only occur as a child element of a Transform element (§12.4.8) that is a Relationships Transform. The schema definition for the RelationshipReference element is specified in Annex C.4.

Attributes	Description
SourceId (Reference to Relationship)	Specifies the value of the Id attribute of the referenced Relationship element within the Relationships part specified by the URI attribute of the Reference element containing this Relationships Transform.

~~12.4.17~~12.4.19 RelationshipsGroupReference Element

The RelationshipsGroupReference element specifies that the group of Relationship elements with the specified value for the Type attribute is to be signed. [This element is package-specific, and its namespace is shown in Annex E.](#) A RelationshipsGroupReference element shall only occur as a child element of the Transform element (§12.4.8) that is a Relationships Transform. The schema definition for the RelationshipsGroupReference element is specified in Annex C.

~~Format designers~~ [A derived format](#) might ~~permit producers to allow sign~~ individual relationships in a package or the Relationships part as a whole [to be signed](#). [O6.10] To sign or validate a subset of relationships, the package-specific Relationships Transform shall be used. [M6.25] To filter signed relationships based on their IDs, a RelationshipReference element with the corresponding SourceID attribute is added to the Relationships Transform element (§12.4.8). To filter signed relationships based on their type, a RelationshipGroupReference element with the corresponding SourceType attribute is added to the Relationships Transform element. Only one relationship transform shall be specified for a particular Relationships part. [M6.35]

A canonicalization transform shall immediately follow a Relationships Transform. [M6.26]

Attributes	Description
SourceType (Relationship Type)	Specifies the value of the Type attribute of Relationship elements within the Relationships part specified by the URI attribute of the Reference element containing this Relationships Transform.

~~12.4.18~~12.4.20 Relationships Transform Algorithm

The relationships transform takes the XML document from the Relationships part and converts it to another XML document.

The Relationships part might contain content from several namespaces, along with versioning instructions as defined in Part 3, “Markup Compatibility and Extensibility”. [O6.11]

The relationships transform algorithm is as follows:

Step 1: Process versioning instructions

Process the Relationships part as specified in Part 3, §9, where the only understood namespace is the Relationships namespace.

Step 2: Sort and filter 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.
- 4) Remove all Relationship elements that do not have either an Id value that matches any SourceId value or a Type value that matches any SourceType value, among the SourceId and SourceType values specified in the transform definition. Values shall be compared as case-sensitive Unicode strings.
[M6.27] The resulting XML document holds all Relationship elements that either have an Id value that matches a SourceId value or a Type value that matches a SourceType value specified in the transform definition.

Step 3: Prepare for canonicalization

- 1) Remove all characters between the Relationships start tag and the first Relationship start tag.
- 2) Remove any contents of the Relationship element.
- 3) Remove all characters between the last Relationship end tag and the Relationships end tag.
- 4) If there are no Relationship elements, remove all characters between the Relationships start tag and the Relationships end tag.
- 5) Remove comments from the Relationships XML content.
- 6) Add a TargetMode attribute with its default value, if this optional attribute is missing from the Relationship element.
- 7) Relationship elements can be specified as start-tag/end-tag pairs with empty content or as empty elements. A canonicalization transform, applied immediately after the Relationships Transform, converts all XML elements into start-tag/end-tag pairs.

12.5 Additional Requirements for Use of XAdES

This subclause specifies additional requirements for XAdES elements used in OPC.

- ~~The SignedSignatureProperties element shall contain a SigningCertificate child element.~~
- ~~A SigningTime element **[[shall/should]]** be present.~~
- ~~If **[[...]]**, the time stamp information shall be specified as an EncapsulatedTimeStamp element, containing DER encoded ASN.1 Data.~~
- ~~If the signature contains **[[time stamps on?]]** references to validation data, the SigAndRefsTimestamp element shall be used.~~
- ~~A Reference element specifying the digest of the SignedProperties element shall be present. This Reference element **[[shall/should]]** be a descendent of the SignedInfo element.~~
- ~~If the SignaturePolicyIdentifier element is used, the SignaturePolicyId element should be used.~~

12.6.12.5 Digital Signature Example

[Example: Digital signature markup for packages is illustrated in this example. For information about namespaces used in this example, see Annex E.

Commented [JH72]: Need to incorporate pending new version of XAdES
Need to say something about using old vs. new (e.g., allow old, recommend new)

Commented [JH73]: John TBD: Why does MSO require this for OOXML files?

Commented [JH74]: ODF: If any timestamp elements of type XAdESTimeStampType are present, such as ...,

MS-OFFCRYPTO: If the information as specified in [XAdES] contains a time stamp as specified by the requirements for XAdES-T

What scope should we target, just -T or -T/-C/-A or other?
XAdESTimeStampType is a ComplexType for:
- SignatureTimeStamp (-T)
- SigAndRefsTimeStamp (-C)
- RefsOnlyTimeStamp (-C)
- ArchiveTimeStamp (-A)

Commented [JH75]: ODF: "If references to validation data are present "
MS-OFF: "If ... contains time stamps on references to validation data "

Commented [JH76]: [JP] Miyachi-san prefers we require CertificatesValues/RevocationValues (-X-L)

Commented [BCN77]: ODF says "shall"
MSO accommodates a version of MSO 2007, which does something different

Commented [JH78]: Need one (partial?) with XAdES, or add XAdES to this and comment it as optional?

Commented [MM79R78]: Give up for this revision.

```

<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>
    </Reference>
  </Manifest>

```

```

</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/2005/06/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 example]

12.7.12.6 Generating Signatures

The steps for signing package contents follow the algorithm outlined in §3.1 of the W3C Recommendation “XML-Signature Syntax and Processing,” with some modification for package-specific constructs.

The steps below might not be sufficient for generating signatures that contain application-defined Object elements. **Format-Derived format designers** that utilize application-defined Object elements shall also define the additional steps that shall be performed to sign the application-defined Object elements.

Commented [JH80]: Does anything need to change here for XAdES?

Commented [MM81R80]: Give up for this revision.

ISO/IEC 29500-2:201x(E)

To generate references:

- 1) For each package part being signed:
 - a) Apply the transforms to the contents of the part. [Note: Relationships Transforms are applied only to Relationship parts. When applied, the Relationships Transform filters the subset of relationships within the entire Relationship part for purposes of signing. end note]
 - b) Calculate the digest value using the resulting contents of the part.
- 2) Create a Reference element that includes the reference of the part with the query component matching the media type of the target part, necessary Transform elements, the DigestMethod element and the DigestValue element.
- 3) Construct the package-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.
- 4) Create a reference to the resulting package-specific Object element.

When signing Object element data, package implementers shall follow the generic reference creation algorithm described in §3.1 of the W3C Recommendation “XML-Signature Syntax and Processing”. [M6.28]

To generate signatures:

- 1) Create the SignedInfo element with a SignatureMethod element, a CanonicalizationMethod element, and at least one Reference element.
- 2) Canonicalize the data and then calculate the SignatureValue element using the SignedInfo element based on the algorithms specified in the SignedInfo element.
- 3) Construct a Signature element that includes SignedInfo, Object, and SignatureValue elements. If a certificate is embedded in the signature, the package implementer shall also include the KeyInfo element.

12.8.12.7 Validating Signatures

12.8.12.7.1 General

Signature validation follows the steps described in §3.2 of the W3C Recommendation “XML-Signature Syntax and Processing.” When validating digital signatures, the media type and the digest contained in each Reference descendant element of the SignedInfo element shall be verified and the signature calculated using the SignedInfo element shall be validated. [M6.29]

The steps below might not be sufficient to validate signatures that contain application-defined Object elements. **Format Derived format designers** that utilize application-defined Object elements shall also define the additional steps that shall be performed to validate the application-defined Object elements.

To validate references:

Commented [JH82]: Does anything need to change here for XAdES?

Commented [MM83R82]: Give up for this revision.

- 1) Canonicalize the SignedInfo element based on the CanonicalizationMethod element specified in the SignedInfo element.
- 2) For each Reference element in the SignedInfo element:
 - a) Obtain the Object element to be digested.
 - b) For the package-specific Object element, validate references to signed parts stored in the Manifest element. References are invalid if there is a missing part. [M6.9] *[Note: If a Relationships Transform is specified for a signed Relationships part, only the specified subset of relationships within the entire Relationships part are validated. end note]*
 - c) For the package-specific Object element, validation of Reference elements includes verifying the media type of the referenced part and the media type specified in the reference query component. References are invalid if these two values are different. The string comparison shall be case-sensitive and locale-invariant. [M6.11]
 - d) Digest the obtained Object element using the DigestMethod element specified in the Reference element.
 - e) Compare the generated digest value against the DigestValue element in the Reference element of the SignedInfo element. References are invalid if there is any mismatch. [M6.30]

To validate signatures:

- 1) Obtain the public key information from the KeyInfo element or from an external source.
- 2) Obtain the canonical form of the SignatureMethod element using the CanonicalizationMethod element. The result and the previously obtained KeyInfo element are used to confirm the SignatureValue element stored in the SignedInfo element. The SignatureValue element shall be decrypted using the public key prior to comparison.

~~12.8.2~~ 12.7.2 Signature Validation and Streaming Consumption

Streaming consumers that maintain signatures shall be able to cache the parts necessary for detecting and processing signatures. [M6.31]

~~12.9~~ 12.8 Support for Versioning and Extensibility

~~12.9.1~~ 12.8.1 General

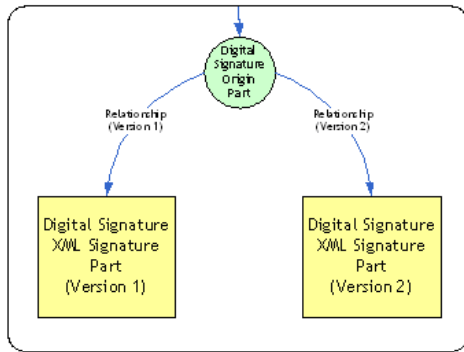
The package digital signature infrastructure supports the exchange of signed packages between current and future package clients.

~~12.9.2~~ 12.8.2 Using Relationship Types

Future versions of the package format might specify distinct relationship types for revised signature parts. Using these relationships, producers would be able to store separate signature information for current and previous versions. Consumers would be able to choose the signature information they know how to validate.

Figure 12–2, “Part names and logical item names”, illustrates this versioning capability that might be available in future versions of the package format.

Figure 12–2. A package containing versioned signatures



12.9.3 12.8.3 Markup Compatibility Namespace for Package Digital Signatures

The Markup Compatibility namespace, as specified in Annex E, shall not be used within the package-specific Object element. [M6.32]

Derived from Format **designers** might specify an application-defined package part format that allows for the embedding of versioned or extended content that might not be fully understood by all present and future implementations. Producers might create such embedded versioned or extended content and consumers might encounter such content. [O6.12] *[Example: An XML package part format might rely on Markup Compatibility elements and attributes to embed such versioned or extended content. end example]*

If an application allows for a single part to contain information that might not be fully understood by all implementations, then the **derived** format **designer** shall **be** carefully designed **so that** the signing and verification policies **to** account for the possibility of different implementations being used for each action in the sequence of content creation, content signing, and signature verification. Producers and consumers shall account for this possibility in their signing and verification processing. [M6.33]

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 implementations support only some of them.

- 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/ü.xml	/a/ü.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

Commented [rcj84]: Need to reword this.

Commented [MM85R84]: We might want to use "package implementers" as the subject of some sentences.

Unicode string	IRI	URI	Part name
./a.xml	./a.xml	./a.xml	/a.xml
./u.xml	./u.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 [rcj86]: I (Murata-san) removed some obviously unnecessary rows and columns, but I have to remove more.

Annex B (normative) ZIP Appnote.txt Clarifications

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 Open Packaging specification. Package producers and consumers shall adhere to the requirements noted below.

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 archive to be a physical layer for a package, the package implementer shall ensure that the ZIP archive 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. [M3.14]

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 archive containing this record or field, however, support might mean ignoring. [M3.15] 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. [M3.16]
- “No” — A “No” value for a field in a table in Annex B indicates the package implementer should not use this record or field. [M3.17]
- “Optional” — An “Optional” value for a record in a table in Annex B indicates that package implementers might write this record during production. [O3.2]
- “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. [M3.18]

- “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 archive. [M3.19]

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)
	Last mod file time (Pass through, no interpretation)	Yes	Yes	Yes

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	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)
	Number of this disk	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	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
	Total number of entries in the central directory	Yes	Yes	Yes

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	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 Apname.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
5.0	File is encrypted using DES	No	No	No
5.0	File is encrypted using 3DES	No	No	No

Version	Feature	Supported on Consumption	Supported on Production	Pass through on editing
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
8	The file is Deflated	Yes	Yes	Yes
9	Enhanced Deflating using Deflate64™	No	No	No

Code	Method	Supported on Consumption	Supported on Production	Pass through on editing
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	<table border="1"> <thead> <tr> <th>Bit 2</th> <th>Bit 1</th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Normal (-en) compression option was used.</td> </tr> <tr> <td>0</td> <td>1</td> <td>Maximum (-exx/-ex) compression option was used.</td> </tr> <tr> <td>1</td> <td>0</td> <td>Fast (-ef) compression option was used.</td> </tr> <tr> <td>1</td> <td>1</td> <td>Super Fast (-es) compression option was used.</td> </tr> </tbody> </table>	Bit 2	Bit 1		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.	Yes	Yes	Yes
	Bit 2	Bit 1																	
	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	Ziplt Macintosh	No	No	Yes

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x2705	Ziplt Macintosh 1.3.5+	No	No	Yes
0x2805	Ziplt 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. [M3.20]

The package implementer shall ensure that all fields that contain “number of entries” do not exceed 2, 147, 483, 647. [M3.21]

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 follow below, and they also reside in an accompanying file named OpenPackagingConventions-XMLSchema.zip, which is distributed in electronic form.

C.2 Media Types Stream

```

1 <xs:schema xmlns="http://schemas.openxmlformats.org/package/2006/content-types"
2   xmlns:xs="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://schemas.openxmlformats.org/package/2006/content-types"
4   elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
5     <xs:element name="Types" type="CT_Types"/>
6     <xs:element name="Default" type="CT_Default"/>
7     <xs:element name="Override" type="CT_Override"/>
8     <xs:complexType name="CT_Types">
9       <xs:choice minOccurs="0" maxOccurs="unbounded">
10        <xs:element ref="Default"/>
11        <xs:element ref="Override"/>
12      </xs:choice>
13    </xs:complexType>
14    <xs:complexType name="CT_Default">
15      <xs:attribute name="Extension" type="ST_Extension" use="required"/>
16      <xs:attribute name="ContentType" type="ST_ContentType" use="required"/>
17    </xs:complexType>
18    <xs:complexType name="CT_Override">
19      <xs:attribute name="ContentType" type="ST_ContentType" use="required"/>
20      <xs:attribute name="PartName" type="xs:anyURI" use="required"/>
21    </xs:complexType>
22    <xs:simpleType name="ST_ContentType">
23      <xs:restriction base="xs:string">
24        <xs:pattern value="(((([\p{IsBasicLatin}-
25 [\p{Cc}&#127;\(\)&lt;&gt;@,;:\&quot;\[\]\?=\{\}\s\t]])+)))/(((([\p{IsBasicLatin}-
26 [\p{Cc}&#127;\(\)&lt;&gt;@,;:\&quot;\[\]\?=\{\}\s\t]])+))(\s+)*;(\s+)*(((([\p{IsBasicLatin}-
27 [\p{Cc}&#127;\(\)&lt;&gt;@,;:\&quot;\[\]\?=\{\}\s\t]])+))=(((([\p{IsBasicLatin}-
28 [\p{Cc}&#127;\(\)&lt;&gt;@,;:\&quot;\[\]\?=\{\}\s\t]])+)|(\&quot;(([\p{IsLatin-
29 1Supplement}\p{IsBasicLatin}-[\p{Cc}&#127;&quot;\n\r]](\s+))|\\[\p{IsBasicLatin}]])*&quot;)))))*"/>
30      </xs:restriction>
31    </xs:simpleType>
32    <xs:simpleType name="ST_Extension">
33      <xs:restriction base="xs:string">
34        <xs:pattern value="(![\&'\(\)\*\+\,;:=]|(%[0-9a-fA-F][0-9a-fA-F])|[:@]|[a-zA-Z0-9\-\_~])+"/>

```



```

35     </xs:restriction>
36   </xs:simpleType>
37 </xs:schema>

```

C.3 Core Properties Part

```

1 <xs:schema targetNamespace="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
2   xmlns="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
3   xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:dc="http://purl.org/dc/elements/1.1/"
4   xmlns:dcterms="http://purl.org/dc/terms/" elementFormDefault="qualified" blockDefault="#all">
5   <xs:import namespace="http://purl.org/dc/elements/1.1/"
6     schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dc.xsd"/>
7   <xs:import namespace="http://purl.org/dc/terms/"
8     schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dcterms.xsd"/>
9   <xs:import id="xml" namespace="http://www.w3.org/XML/1998/namespace"/>
10  <xs:element name="coreProperties" type="CT_CoreProperties"/>
11  <xs:complexType name="CT_CoreProperties">
12    <xs:all>
13      <xs:element name="category" minOccurs="0" maxOccurs="1" type="xs:string"/>
14      <xs:element name="contentStatus" minOccurs="0" maxOccurs="1" type="xs:string"/>
15      <xs:element ref="dcterms:created" minOccurs="0" maxOccurs="1"/>
16      <xs:element ref="dc:creator" minOccurs="0" maxOccurs="1"/>
17      <xs:element ref="dc:description" minOccurs="0" maxOccurs="1"/>
18      <xs:element ref="dc:identifier" minOccurs="0" maxOccurs="1"/>
19      <xs:element name="keywords" minOccurs="0" maxOccurs="1" type="CT_Keywords"/>
20      <xs:element ref="dc:language" minOccurs="0" maxOccurs="1"/>
21      <xs:element name="lastModifiedBy" minOccurs="0" maxOccurs="1" type="xs:string"/>
22      <xs:element name="lastPrinted" minOccurs="0" maxOccurs="1" type="xs:dateTime"/>
23      <xs:element ref="dcterms:modified" minOccurs="0" maxOccurs="1"/>
24      <xs:element name="revision" minOccurs="0" maxOccurs="1" type="xs:string"/>
25      <xs:element ref="dc:subject" minOccurs="0" maxOccurs="1"/>
26      <xs:element ref="dc:title" minOccurs="0" maxOccurs="1"/>
27      <xs:element name="version" minOccurs="0" maxOccurs="1" type="xs:string"/>
28    </xs:all>
29  </xs:complexType>
30  <xs:complexType name="CT_Keywords" mixed="true">
31    <xs:sequence>
32      <xs:element name="value" minOccurs="0" maxOccurs="unbounded" type="CT_Keyword"/>
33    </xs:sequence>
34    <xs:attribute ref="xml:lang" use="optional"/>
35  </xs:complexType>
36  <xs:complexType name="CT_Keyword">
37    <xs:simpleContent>
38      <xs:extension base="xs:string">
39        <xs:attribute ref="xml:lang" use="optional"/>
40      </xs:extension>
41    </xs:simpleContent>
42  </xs:complexType>
43 </xs:schema>

```

C.4 Digital Signature XML Signature Markup

```

1 <xsd:schema xmlns="http://schemas.openxmlformats.org/package/2006/digital-signature"
2   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://schemas.openxmlformats.org/package/2006/digital-signature"
4   elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
5     <xsd:element name="SignatureTime" type="CT_SignatureTime"/>
6     <xsd:element name="RelationshipReference" type="CT_RelationshipReference"/>
7     <xsd:element name="RelationshipsGroupReference" type="CT_RelationshipsGroupReference"/>
8     <xsd:complexType name="CT_SignatureTime">
9       <xsd:sequence>
10        <xsd:element name="Format" type="ST_Format"/>
11        <xsd:element name="Value" type="ST_Value"/>
12      </xsd:sequence>
13    </xsd:complexType>
14    <xsd:complexType name="CT_RelationshipReference">
15      <xsd:simpleContent>
16        <xsd:extension base="xsd:string">
17          <xsd:attribute name="SourceId" type="xsd:string" use="required"/>
18        </xsd:extension>
19      </xsd:simpleContent>
20    </xsd:complexType>
21    <xsd:complexType name="CT_RelationshipsGroupReference">
22      <xsd:simpleContent>
23        <xsd:extension base="xsd:string">
24          <xsd:attribute name="SourceType" type="xsd:anyURI" use="required"/>
25        </xsd:extension>
26      </xsd:simpleContent>
27    </xsd:complexType>
28    <xsd:simpleType name="ST_Format">
29      <xsd:restriction base="xsd:string">
30        <xsd:pattern value="(YYYY)|(YYYY-MM)|(YYYY-MM-DD)|(YYYY-MM-DDThh:mmTZD)|(YYYY-MM-
31          DDThh:mm:ssTZD)|(YYYY-MM-DDThh:mm:ss.sTZD)"/>
32      </xsd:restriction>
33    </xsd:simpleType>
34    <xsd:simpleType name="ST_Value">
35      <xsd:restriction base="xsd:string">
36        <xsd:pattern value="((([0-9][0-9][0-9][0-9])|((([0-9][0-9][0-9][0-9])-(0[1-
37          9])|(1(0|1|2))))|((([0-9][0-9][0-9][0-9])-(0[1-9])|(1(0|1|2))))-(0[1-9])|(1[0-9])|(2[0-
38          9])|(3(0|1))))|((([0-9][0-9][0-9][0-9])-(0[1-9])|(1(0|1|2))))-(0[1-9])|(1[0-9])|(2[0-
39          9])|(3(0|1)))T((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-
40          9])|(5[0-9]))((\+|-)((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-
41          9])|(4[0-9])|(5[0-9]))|Z)|(((0[0-9][0-9][0-9][0-9])-(0[1-9])|(1(0|1|2)))-((0[1-9])|(1[0-9])|(1[0-
42          9])|(2[0-9])|(3(0|1)))T((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-
43          9])|(4[0-9])|(5[0-9])):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))((\+|-
44          )(0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
45          9]))|Z)|(((0[0-9][0-9][0-9][0-9])-(0[1-9])|(1(0|1|2)))-((0[1-9])|(1[0-9])|(1[0-9])|(2[0-
46          9])|(3(0|1)))T((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-
47          9])|(5[0-9])):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))\.[0-9])((\+|-
48          )(0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
49          9]))|Z)"/>
50      </xsd:restriction>
51    </xsd:simpleType>

```

52 </xsd:schema>

C.5 Relationships Part

```

1 <xsd:schema xmlns="http://schemas.openxmlformats.org/package/2006/relationships"
2   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://schemas.openxmlformats.org/package/2006/relationships"
4   elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
5   <xsd:element name="Relationships" type="CT_Relationships"/>
6   <xsd:element name="Relationship" type="CT_Relationship"/>
7   <xsd:complexType name="CT_Relationships">
8     <xsd:sequence>
9       <xsd:element ref="Relationship" minOccurs="0" maxOccurs="unbounded"/>
10    </xsd:sequence>
11  </xsd:complexType>
12  <xsd:complexType name="CT_Relationship">
13    <xsd:simpleContent>
14      <xsd:extension base="xsd:string">
15        <xsd:attribute name="TargetMode" type="ST_TargetMode" use="optional"/>
16        <xsd:attribute name="Target" type="xsd:anyURI" use="required"/>
17        <xsd:attribute name="Type" type="xsd:anyURI" use="required"/>
18        <xsd:attribute name="Id" type="xsd:ID" use="required"/>
19      </xsd:extension>
20    </xsd:simpleContent>
21  </xsd:complexType>
22  <xsd:simpleType name="ST_TargetMode">
23    <xsd:restriction base="xsd:string">
24      <xsd:enumeration value="External"/>
25      <xsd:enumeration value="Internal"/>
26    </xsd:restriction>
27  </xsd:simpleType>
28 </xsd:schema>

```

Annex D (informative) Schemas - RELAX NG

This clause is informative.

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 follow below; they also 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

```

1 default namespace =
2   "http://schemas.openxmlformats.org/package/2006/content-types"
3
4 start = Types
5 Types = element Types { CT_Types }
6 Default = element Default { CT_Default }
7 Override = element Override { CT_Override }
8 CT_Types = (Default | Override)*
9 CT_Default =
10   attribute Extension { ST_Extension },
11   attribute ContentType { ST_ContentType }
12 CT_Override =
13   attribute ContentType { ST_ContentType },
14   attribute PartName { xsd:anyURI }
15 ST_ContentType =
16   xsd:string {
17     pattern =
18       '((((([\p{IsBasicLatin}-[\p{Cc}\x{127}\(\)<>@,;:\\"/[\]\?=\{\}\s\t]]+)))/(((([\p{IsBasicLatin}-
19 [\p{Cc}\x{127}\(\)<>@,;:\\"/[\]\?=\{\}\s\t]]+)))(\s+)*;(\s+)*(((([\p{IsBasicLatin}-
20 [\p{Cc}\x{127}\(\)<>@,;:\\"/[\]\?=\{\}\s\t]]+))=([([\p{IsBasicLatin}-
21 [\p{Cc}\x{127}\(\)<>@,;:\\"/[\]\?=\{\}\s\t]]+)|"([\p{IsLatin-1Supplement}\p{IsBasicLatin}-
22 [\p{Cc}\x{127}"\n\r]](\s+))|([[\p{IsBasicLatin}]])*)))))*'
23   }
24 ST_Extension =
25   xsd:string {
26     pattern =
27       "(!$&'\\(\)\*\+, :=]|(%[0-9a-fA-F][0-9a-fA-F])|[:@]|[a-zA-Z0-9\-\_~])+"
28   }

```

D.3 Core Properties Part

```

1 default namespace =
2   "http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
3 namespace dc = "http://purl.org/dc/elements/1.1/"
4 namespace dcterms = "http://purl.org/dc/terms/"
5 namespace xsi = "http://www.w3.org/2001/XMLSchema-instance"
6 include "xml.rnc"
7
8 start = coreProperties
9 coreProperties = element coreProperties { CT_CoreProperties }
10 CT_CoreProperties =
11   element category { xsd:string }?
12   & element contentStatus { xsd:string }?
13   & element dcterms:created {
14     attribute xsi:type { xsd:QName "dcterms:W3CDTF" }, xml_lang?, W3CDTF
15   }?
16   & element dc:creator { SimpleLiteral }?
17   & element dc:description { SimpleLiteral }?
18   & element dc:identifier { SimpleLiteral }?
19   & element keywords { CT_Keywords }?
20   & element dc:language { SimpleLiteral }?
21   & element lastModifiedBy { xsd:string }?
22   & element lastPrinted { xsd:dateTime }?
23   & element dcterms:modified {
24     attribute xsi:type { xsd:QName "dcterms:W3CDTF" }, xml_lang?, W3CDTF
25   }?
26   & element revision { xsd:string }?
27   & element dc:subject { SimpleLiteral }?
28   & element dc:title { SimpleLiteral }?
29   & element version { xsd:string }?
30 CT_Keywords =
31   mixed {
32     xml_lang?,
33     element value { CT_Keyword }*
34   }
35 CT_Keyword = xsd:string, xml_lang?
36 SimpleLiteral = xml_lang?, xsd:string
37 W3CDTF = xsd:gYear | xsd:gYearMonth | xsd:date | xsd:dateTime

```

D.4 Digital Signature XML Signature Markup

```

1 default namespace =
2   "http://schemas.openxmlformats.org/package/2006/digital-signature"
3 namespace ds = http://www.w3.org/2000/09/xmldsig#
4
5
6 include "security.any.rnc"
7 include "xmldsig-core-schema.rnc" {
8
9
10 ds CanonicalizationMethodType =
11 attribute Algorithm {

```

```

12 xsd:anyURI "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"
13 | xsd:anyURI http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
14 }
15 ds SignatureMethodType =
16 attribute Algorithm {
17 xsd:anyURI "http://www.w3.org/2000/09/xmldsig#dsa-sha1"
18 | xsd:anyURI "http://www.w3.org/2000/09/xmldsig#rsa-sha1" }
19 ds SignaturePropertyType =
20 SignatureTime,
21 attribute Id { xsd:ID }?,
22 attribute Target { xsd:anyURI }
23 ds TransformType =
24 ds CanonicalizationMethodType
25 | (attribute Algorithm {
26 xsd:anyURI "http://schemas.openxmlformats.org/package/2006/RelationshipTransform" },
27 (RelationshipReference | RelationshipsGroupReference)*),
28 ds ObjectType = application defined ObjectType | package specific ObjectType
29 }
30
31 start = ds SignatureSignaturePropertyType =
32 SignatureTime,
33 attribute Id { xsd:ID }?,
34 attribute Target { xsd:anyURI }
35
36 TransformType =
37 element ds:XPath { xsd:string }?,
38 (RelationshipReference | RelationshipsGroupReference)*,
39 attribute Algorithm { xsd:anyURI }
40 }
41
42 SignatureTime = element SignatureTime { CT_SignatureTime }
43 RelationshipReference =
44 element RelationshipReference { CT_RelationshipReference }
45 RelationshipsGroupReference =
46 element RelationshipsGroupReference { CT_RelationshipsGroupReference }
47 CT_SignatureTime =
48 element Format { ST_Format },
49 element Value { ST_Value }
50 CT_RelationshipReference =
51 xsd:string,
52 attribute SourceId { xsd:string }
53 CT_RelationshipsGroupReference =
54 xsd:string,
55 attribute SourceType { xsd:anyURI }
56 ST_Format =
57 xsd:string {
58 pattern =
59 "(YYYY)|(YYYY-MM)|(YYYY-MM-DD)|(YYYY-MM-DDThh:mmTZD)|(YYYY-MM-DDThh:mm:ssTZD)|(YYYY-MM-
60 DDThh:mm:ss.sTZD)"
61 }
62 ST_Value =
63 xsd:string {
64 pattern =

```

```

65      "(([\0-9][\0-9][\0-9][\0-9]))|(([\0-9][\0-9][\0-9][\0-9])-(\0[1-9])|(1(0|1|2))))|(([\0-9][\0-9][\0-9][\0-9])-
66      9)-(\0[1-9])|(1(0|1|2)))-((\0[1-9])|(1[0-9])|(2[0-9])|(3(0|1))))|(([\0-9][\0-9][\0-9][\0-9])-(\0[1-
67      9])|(1(0|1|2)))-(\0[1-9])|(1[0-9])|(2[0-9])|(3(0|1)))T((\0[0-9])|(1[0-9])|(2(0|1|2|3))):((\0[0-
68      9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))((\+|-)(\0[0-9])|(1[0-9])|(2(0|1|2|3))):((\0[0-
69      9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))|Z)|(([\0-9][\0-9][\0-9][\0-9])-(\0[1-9])|(1(0|1|2)))-
70      ((\0[1-9])|(1[0-9])|(2[0-9])|(3(0|1)))T((\0[0-9])|(1[0-9])|(2(0|1|2|3))):((\0[0-9])|(1[0-9])|(2[0-
71      9])|(3[0-9])|(4[0-9])|(5[0-9])):(\0[0-9])|(\0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))((\+|-)(\0[0-
72      9])|(1[0-9])|(2(0|1|2|3))):((\0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))|Z)|(([\0-9][\0-
73      9][\0-9][\0-9])-(\0[1-9])|(1(0|1|2)))-((\0[1-9])|(1[0-9])|(2[0-9])|(3(0|1)))T((\0[0-9])|(1[0-
74      9])|(2(0|1|2|3))):((\0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9])):(\0[0-9])|(1[0-9])|(2[0-
75      9])|(3[0-9])|(4[0-9])|(5[0-9]))\.[0-9]((\+|-)(\0[0-9])|(1[0-9])|(2(0|1|2|3))):((\0[0-9])|(1[0-
76      9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))|Z))"
77      }

```

D.5 Relationships Part

```

1  default namespace =
2      "http://schemas.openxmlformats.org/package/2006/relationships"
3
4  start = Relationships
5  Relationships = element Relationships { CT_Relationships }
6  Relationship = element Relationship { CT_Relationship }
7  CT_Relationships = Relationship*
8  CT_Relationship =
9      xsd:string,
10     attribute TargetMode { ST_TargetMode }?,
11     attribute Target { xsd:anyURI },
12     attribute Type { xsd:anyURI },
13     attribute Id { xsd:ID }
14  ST_TargetMode = string "External" | string "Internal"

```

D.6 Additional Resources

D.6.1 XML

```

1  xml_lang = attribute xml:lang { xsd:language | xsd:string "" }
2  xml_space = attribute xml:space { "default" | "preserve" }
3  xml_base = attribute xml:base { xsd:anyURI }
4  xml_id = attribute xml:id { xsd:ID }
5  xml_specialAttrs = xml_base?, xml_lang?, xml_space?, xml_id?

```

D.6.2 XML Digital Signature Core

xmldsig-core-schema.rnc (a RELAX NG schema in the compact syntax) can be created from xmldsig-core-schema.rng (a RELAX NG schema in the XML syntax), which is available at <http://www.w3.org/Signature/2002/07/xmldsig-core-schema.rng>.

End of informative text.

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

Package implementers ~~and format designers~~ shall not create media types with parameters for the package-specific parts defined in this Open Packaging specification and shall treat the presence of parameters in these media types as an error. [M1.22]

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
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

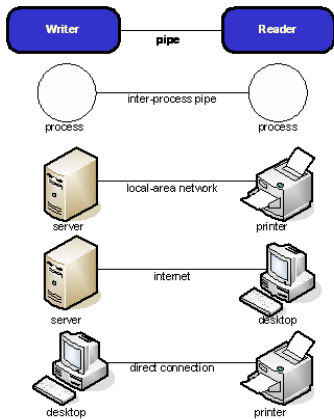
Annex F (informative) Physical Model Design Considerations

This annex is informative.

F.1 General

The physical 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 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 package 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 package 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.

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 models. Regardless of how the physical 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 package 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.

End of informative text.

Annex G (informative) Guidelines for Meeting Conformance

This annex is informative.

G.1 General

This annex summarizes best practices for producers and consumers implementing the Open Packaging Conventions. It is intended as a convenience; the text in the referenced clause or subclause is considered normative in all cases.

The top-level topics and their identifiers are described as follows:

- 1) Package Model requirements
- 2) Physical Packages requirements
- 3) ZIP Physical Mapping requirements
- 4) Core Properties requirements
- 5) Thumbnail requirements
- 6) Digital Signatures requirements
- 7) Pack URI requirements

Additionally, these tables identify, as does the referenced text, who is burdened with enforcing or supporting the requirement:

G.2 Package Model

Table G–1. Package model conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.1	Error! Reference source not found. Error! Reference source not found. Error! Reference source not found.	8.2, Error! Reference source not found. , Error! Reference source not found.	×			
M1.2	Error! Reference source not found.	8.2	×	×		

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.3	Error! Reference source not found. Error! Reference source not found. Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			
M1.4	Error! Reference source not found. Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			
M1.5	Error! Reference source not found. Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			
M1.6	Error! Reference source not found. Error! Reference source not found..	Error! Reference source not found., Error! Reference source not found.	×			
M1.7	Error! Reference source not found. Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			
M1.8	Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.9	Error! Reference source not found. Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			
M1.10	Error! Reference source not found. Error! Reference source not found.	Error! Reference source not found., Error! Reference source not found.	×			
M1.11	Error! Reference source not found.	0	×			
M1.12	Error! Reference source not found.	Error! Reference source not found.	×			
M1.13	Error! Reference source not found.	0	×	×		
M1.14	Error! Reference source not found.	0	×	×		
M1.15	Error! Reference source not found.	0	×	×		
M1.16	If the package implementer specifies a growth hint, it is set when a part is created, and the package implementer shall not change the growth hint after the part has been created.	8.2.4	×		×	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.17	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. Package implementers shall enforce this requirement upon creation and retrieval of the XML content.	8.2.5	x			
M1.18	DTD declarations shall not be used in the XML markup defined in this Open Packaging specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content and shall treat the presence of DTD declarations as an error.	8.2.5	x			
M1.19	Any extensions in the XML content are removed by an MCE processor as specified in ISO/IEC 29500-3 before applying validation and subsequent processing. [M1.19]	8.2.5	x			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.20	XML content shall be valid against the corresponding XSD schema defined in this Open Packaging specification. 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. Package implementers shall enforce this requirement upon creation and retrieval of the XML content.	8.2.5	x			
M1.21	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 Open Packaging specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content.	8.2.5	x			
M1.22	Package implementers shall not create media types with parameters for the package-specific parts defined in this Open Packaging specification and shall treat the presence of parameters in these media types as an error.	Annex E	x	x		
M1.23	Error! Reference source not found.	8.3				x
M1.24	Error! Reference source not found.Error! Reference source not found.	8.3				x

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.25	Error! Reference source not found.	8.5.2	×			
M1.26	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.	8.5.3	×			
M1.27	Error! Reference source not found.	8.5.3.3	×	×		
M1.28	Error! Reference source not found.	8.5.3.3	×			
M1.29	Error! Reference source not found.	8.5.3.3	×			
M1.30	Error! Reference source not found.	0	×			
M1.31	Error! Reference source not found.	8.5.5			×	×
M1.32	Error! Reference source not found.	8.5.3.3	×			
M1.33	Error! Reference source not found.	Error! Reference source not found.			×	×
M1.34	Error! Reference source not found.	Error! Reference source not found.				×

Table G-2. Package model optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O1.1	Error! Reference source not found.	8.2, 8.2.4	×			
O1.2	Derived formats might restrict the usage of parameters for media types.	0		×		
O1.3	The package implementer might ignore the growth hint or adhere only loosely to it when specifying the physical mapping.	8.2.4	×			
O1.4	Error! Reference source not found.	8.3		×	×	×
O1.5	Error! Reference source not found.	8.5.3.3	×			
O1.6	Error! Reference source not found.	8.5.3.3		×		
O1.7	Relationships parts might contain the versioning and extensibility mechanisms defined in Part 3 to incorporate elements and attributes drawn from other XML namespaces.	8.5.5			×	

G.3 Physical Packages

Table G–3. Physical packages conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.1	The Media Types stream shall not be mapped to a part by the package implementer.	Error! Reference source not found.	x ^A			
M2.2	The package implementer shall define a physical package format with a mapping for the following required components.	9.2.2	×			
M2.3	The package implementer shall define a format mapping with a mechanism for associating media types with parts.	Error! Reference source not found.	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.4	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.	9.2.3.2	x ^A			
M2.5	The package implementer shall require that there not be more than one Default element for any given extension, and there not be more than one Override element for any given part name.	9.2.3.2	x ^A			
M2.6	Error! Reference source not found.Error! Reference source not found.	0	x ^A	x ^A		
M2.7	Error! Reference source not found.Error! Reference source not found.	0	x ^A	x ^A		
M2.8	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 steps described in §9.2.3.4.	9.2.3.4	x ^A			
M2.9	To get the media type of a part, the package implementer shall perform the steps described in §9.2.3.5.	9.2.3.5	x ^A			
M2.10	The package implementer shall not use the versioning and extensibility mechanisms defined in Part 3 to incorporate elements and attributes drawn from other XML-namespaces into the Media Types stream markup.	9.2.3.6	x ^A			
M2.11	The package implementer shall not mix interleaving and non-interleaving for an individual part.	9.2.5	x ^B			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.12	The package implementer shall compare prefix names by ASCII case-insensitive matching.	9.2.4.2	x			
M2.13	The package implementer shall compare suffix names by ASCII case-insensitive matching.	9.2.4.2	x ^B			
M2.14	The package implementer shall not allow packages that contain equivalent logical item names.	9.2.4.2	x			
M2.15	The package implementer shall not allow packages that contain logical items with equivalent prefix names and with equal piece numbers, where piece numbers are treated as integer decimal values.	9.2.4.2	x ^B			
M2.16	The package implementer shall not map logical items to parts if the logical item names violate the part naming rules.	9.2.4.5	x			
M2.17	The package implementer shall consider naming collisions within the set of part names mapped from logical item names to be an error.	9.2.4.5	x			
M2.18	When interleaved, a package implementer shall represent a part as one or more pieces, using the method described in §9.2.5.	9.3.2	x ^B			

Notes:

A: Only relevant if using the media type mapping strategy specified in the Open Packaging Conventions.

B: Only relevant if supporting the interleaving strategy specified in the Open Packaging Conventions.

Table G–4. Physical packages recommendations

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S2.1	Some physical package formats have a native mechanism for associating media types with parts. For such packages, the package implementer should use the native mechanism to map part media types to parts.	Error! Reference source not found.	x			
S2.2	If no native method of mapping a media type to a part exists, the package should include an XML stream called the Media Types stream	Error! Reference source not found.	x			
S2.3	If the package is intended for streaming consumption: The Media Types stream should not have Default elements; consequently, there should be 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.	9.2.3.2	x ^A		x ^A	
S2.4	The package implementer should use the mechanism described in this Open Packaging specification to allow interleaving when mapping to the physical package for layout scenarios that support streaming consumption.	9.2.5	x ^B			
S2.5	The package implementer should store pieces in their natural order for optimal efficiency.	9.2.5	x ^B			

Notes:

A: Only relevant if using the media type mapping strategy specified in the Open Packaging Conventions.

B: Only relevant if supporting the interleaving strategy specified in the Open Packaging Conventions.

Table G–5. Physical packages optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O2.1	A derived format specifies whether the use of interleaving is allowed or not.	9.2.5		x		
O2.2	Optional. The package implementer might provide a physical mapping for a growth hint that might be specified by a producer.	9.2.2	x			
O2.3	Package implementers might use the common mapping solutions defined in this Open Packaging specification.	9.2	x			
O2.4	Package producers can use pre-defined Default elements to reduce the number of Override elements on a part, but are not required to do so.	9.2.3.2			x ^A	
O2.5	The package implementer can define Default media type mappings even though no parts use them.	9.2.3.2	x ^A			
O2.6	The package implementer might create a physical package containing interleaved parts and non-interleaved parts.	9.2.5	x			
O2.7	The package implementer might allow a package that contains 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 or the logical item does not have an associated media type.	9.2.4.5	x ^B			

Notes:

A: Only relevant if using the media type mapping strategy specified in the Open Packaging Conventions.

B: Only relevant if supporting the interleaving strategy specified in the Open Packaging Conventions.

G.4 ZIP Physical Mapping

The requirements in Table G–6, Table G–7, and Table G–8 are only relevant when mapping to the ZIP physical package format.

Table G–6. ZIP physical mapping conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.1	A package implementer shall store a non-interleaved part as a single ZIP item.	9.3.2	x			
M3.2	ZIP item names are case-sensitive ASCII strings. Package implementers shall create ZIP item names that conform to ZIP archive-file name grammar.	9.3.3	x			
M3.3	Package implementers shall create item names that are unique within a given archive.	9.3.3	x			
M3.4	To map part names to ZIP item names the package implementer shall perform, in order, the steps described in §9.3.4.	9.3.4	x			
M3.5	The package implementer shall not map a logical item name or complete sequence of logical item names sharing a common prefix to a part name if the logical item prefix has no corresponding media type.	9.3.4	x			
M3.6	To map ZIP item names to part names, the package implementer shall perform, in order, the steps described in §9.3.5.	9.3.5	x			
M3.7	The package implementer shall map all ZIP items to parts except MS-DOS ZIP items, as defined in the ZIP specification, that are not MS-DOS files.	9.3.6	x			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.8	<p>The package implementer shall map all ZIP items to parts except MS-DOS ZIP items, as defined in the ZIP specification, that are not MS-DOS files. [M3.7]</p> <p><i>[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]</i></p> <p>In ZIP archives, the package implementer shall not exceed 65,535 bytes for the combined length of the item name, Extra field, and Comment fields.</p>	9.3.6	×			
M3.9	ZIP-based packages shall not include encryption as described in the ZIP specification. Package implementers shall enforce this restriction.	9.3.6	×			
M3.10	Error! Reference source not found.	9.3.7	×			
M3.11	Package implementers shall not map logical item name(s) mapped to the Media Types stream in a ZIP archive to a part name.	9.3.7	×			
M3.13	Several substantial conditions that represent a package unfit for streaming consumption might be detected mid-processing by a streaming package implementer, described in §9.3.9. When any of these conditions are detected, the streaming package implementer shall generate an error, regardless of any processing that has already taken place. Package implementers shall not generate a package containing any of these conditions when generating a package intended for streaming consumption.	9.3.9	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.14	For a ZIP archive to be a physical layer for a package, the package implementer shall ensure that the ZIP archive 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.	Annex B	×			
M3.15	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 archive containing this record or field, however, support might mean ignoring.	Annex B	×			
M3.16	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.	Annex B	×			
M3.17	A "No" value for a field in a table in Annex B indicates the package implementer should not use this record or field.	Annex B	×			
M3.18	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.	Annex B	×			
M3.19	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 archive.	Annex B	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.20	The package implementer shall ensure that all 64-bit stream record sizes and offsets have the high-order bit = 0.	Annex B	x			
M3.21	The package implementer shall ensure that all fields that contain "number of entries" do not exceed 2, 147, 483, 647.	Annex B	x			

Notes:

A: Only relevant if supporting the interleaving strategy specified in the Open Packaging Conventions.

Table G–7. ZIP physical mapping recommendations

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S3.1	Package implementers should restrict part naming to accommodate file system limitations when naming parts to be stored as ZIP items.	9.3.6	x			
S3.2	If a growth hint is used for an interleaved part, the package implementer should store the Extra field containing the growth hint padding with the item that represents the first piece of the part.	9.3.8	x			

Table G–8. ZIP physical mapping optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O3.1	A package implementer might intentionally order the sequence of ZIP items in the archive to enable an efficient organization of the part data in order to achieve correct and optimal interleaving.	9.3.2	x			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O3.2	An “Optional” value for a record in a table in Annex B indicates that package implementers might write this record during production.	Annex B	×			

G.5 Core Properties

The requirements in Table G–9 are only relevant if using the core properties feature.

Table G–9. Core properties conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M4.1	.	1.1		×	×	×
M4.2		1.1		×	×	×
M4.3		1.1			×	×
M4.4		1.1			×	×
M4.5		1.1			×	×

G.6 Thumbnail

The requirements in Table G–10 and Table G–11 are only relevant if using the thumbnail feature.

Table G–10. Thumbnail conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M5.1	Thumbnail parts shall be identified by either a part relationship or a package relationship. The producer shall build the package accordingly.	11		×	×	

Table G–11. Thumbnail optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O5.1	A derived format might allow images, called thumbnails, to be used to help end-users identify parts of a package or a package as a whole. These images can be generated by the producer and stored as parts.	11		x	x	

G.7 Digital Signatures

The requirements in Table G–12, Table G–13, and Table G–14 are only relevant if using the digital signatures feature.

Table G–12. Digital Signatures conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.1	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 root.	12.3.2	x			
M6.2	This part shall exist if the package contains any Digital Signature XML Signature parts,	12.3.2	x			
M6.3	Error! Reference source not found.	12.3.2			x	x
M6.4	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.	0			x	x
M6.5		1.1.1			x	x
M6.6		1.1.1			x	x
M6.7		1.1.1			x	x
M6.8		1.1.1			x	x

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.9		1.1.1			x	x
M6.10		1.1.1			x	x
M6.11		1.1.1			x	x
M6.12		1.1.1			x	x
M6.13		1.1.1			x	x
M6.14		1.1.1			x	x
M6.15	A Signature element shall contain exactly one local-data, package-specific Object element and zero or more application-defined Object elements.	12.4.2			x	x
M6.16	A SignedInfo element shall contain exactly one reference to the package-specific Object element.	12.4.3			x	x
M6.17	RSA-SHA1 algorithms shall be specified by a DigestMethod element .	12.4.5			x	x
M6.18	Each Reference element that is a child of a Manifest element shall contain a URI attribute whose value contains a part name without a fragment identifier.	12.4.6			x	x
M6.19	Error! Reference source not found.	12.4.7			x	x
M6.20		12.4.10.2			x	x
M6.21	The certificate embedded in the Digital Signature XML Signature part shall be used when it is specified. The certificate embedded in the Digital Signature XML Signature part shall be used when it is specified.	12.4.11			x	x
M6.22	_and shall not reference any data outside of the package.	12.4.12			x	x
M6.23	Error! Reference source not found.	12.4.16			x	x
M6.24	Error! Reference source not found.	12.4.17			x	x

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.25	Error! Reference source not found.	12.4.19			x	x
M6.26	Error! Reference source not found.	12.4.19			x	x
M6.27	When applying a relationships transform for digital signatures, the Remove all Relationship elements that do not have either an Id value that matches any SourceId value or a Type value that matches any SourceType value, among the SourceId and SourceType values specified in the transform definition. Values shall be compared as case-sensitive Unicode strings.	12.4.20			x	x
M6.28	When signing Object element data, package implementers shall follow the generic reference creation algorithm described in §3.1 of the W3C Recommendation "XML-Signature Syntax and Processing".	12.6	x			
M6.29	When validating digital signatures, the media type and the digest contained in each Reference descendant element of the SignedInfo element shall be verified and the signature calculated using the SignedInfo element shall be validated.	12.7				x
M6.30	Compare the generated digest value against the DigestValue element in the Reference element of the SignedInfo element. References are invalid if there is any mismatch.	12.7	x			
M6.31	Streaming consumers that maintain signatures shall be able to cache the parts necessary for detecting and processing signatures.	12.7.2				x

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.32	The Markup Compatibility namespace, as specified in Annex E, shall not be used within the package-specific Object element.	12.8.3	x			
M6.33	If an application allows for a single part to contain information that might not be fully understood by all implementations, then the derived format shall be carefully designed so that the signing and verification policies account for the possibility of different implementations being used for each action in the sequence of content creation, content signing, and signature verification. Producers and consumers shall account for this possibility in their signing and verification processing.	12.8.3		x	x	x
M6.34	Packages shall use only the following canonicalization methods: XML Canonicalization (c14n) XML Canonicalization with Comments (c14n with comments) They shall be supported for validation.	12.4.4			x	x
M6.35	A producer shall not specify more than one relationship transform for a particular relationships part. A consumer shall treat the presence of more than one relationship transform for a particular relationships part as an error.	12.4.19			x	x

Table G–13. Digital signatures recommendations

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S6.1	No content should exist in the Digital Signature Origin part itself.	12.3.2			x	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S6.2	A Digital Signature Certificate part should be the target of at least one Digital Signature Certificate relationship from a Digital Signature XML Signature part.	0			x	
S6.3	For digital signatures, Error! Reference source not found.	12.4.4			x	x
S6.4	Error! Reference source not found.	12.4.4			x	x
S6.5		1.1.1			x	

Table G–14. Digital signatures optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.1	<i>[Rule deleted]</i>	12		x	x	
O6.2	and is optional otherwise.	12.3.2			x	
O6.4	[M6.3] One or more of these parts may exist in a package.	12.3.3			x	
O6.5	Alternatively, the certificate may exist as a separate part in the package, may be embedded within the Digital Signature XML Signature part itself, or may be excluded from the package if certificate data is known or can be obtained from a local or remote certificate store.	0			x	
O6.6	The part containing the certificate may be signed.	0			x	
O6.7	A Digital Signature Certificate part may be used to create more than one signature.	0			x	
O6.8	A derived format might permit one or more application-defined Object elements. .	12.4.10.2		x	x	
O6.9	Derived formats might not apply package-specific restrictions regarding URIs and Transform elements to application-defined Object elements.	12.4.10.2		x	x	
O6.10		12.4.19		x	x	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.11	The Relationships part might contain content from several namespaces, along with versioning instructions as defined in Part 3, “Markup Compatibility and Extensibility”.	12.4.20	×			
O6.12	Derived formats might specify an application-defined package part format that allows for the embedding of versioned or extended content that might not be fully understood by all present and future implementations. Producers might create such embedded versioned or extended content and consumers might encounter such content.	12.8.3		×	×	×

G.8 Pack URI

Table G–15. Pack URI conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M7.1	Error! Reference source not found.	Error! Reference source not found.	×			
M7.2	Error! Reference source not found.	Error! Reference source not found.	×			
M7.3	Error! Reference source not found.	Error! Reference source not found.	×			
M7.4	Error! Reference source not found.	Error! Reference source not found.	×			

Table G–16. Pack URI optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O7.1	Error! Reference source not found.	Error! Reference source not found.				×

End of informative text.

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

This annex is informative.

H.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.

H.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)

H.3 XML Attributes

No changes.

H.4 XML Enumeration Values

No changes.

H.5 XML Simple Types

No changes.

End of informative text.

Bibliography

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

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