ISO/IEC 29500-2:201x

Office Open XML File Formats — Open Packaging Conventions

Working DRAFT WD2.1

2015-10-19

Contents

Foreword

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

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

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

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

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

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

The major changes from the previous edition include:

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

Commented [rcj1]:

The major changes in the third edition include:

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

There were no major changes in the second edition.

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

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

Annexes A, B, C, 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*]

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

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.

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/

Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, 04 February 2004.

Namespaces in XML 1.1, W3C Recommendation, 4 February 2004.

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 4234 *Augmented BNF for Syntax Specifications: ABNF*, The Internet Society, Crocker, D., (editor), 2005, http://www.ietf.org/rfc/rfc4234.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.

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 Path Language (XPath), Version 1.0, W3C Recommendation, 16 November 1999.

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

Commented [JH3]: Remove unnecessary terms

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* and *relative reference* are used in accordance with RFC 3986. The term *media type* is used in accordance with RFC 2046.

4.1
access style
style in which local access or networked access is conducted

4.2
behavior
external appearance or action

4.3
behavior, implementation-defined
behavior, application-defined
unspecified behavior where each implementation shall document that behavior, thereby promoting predictability and reproducibility within any given implementation

4.4
behavior, unspecified
behavior where this Open Packaging specification imposes no requirements

4.5
byte
sequence of 8 bits treated as a unit

4.6
communication style
style in which package contents are delivered by a producer or received by a consumer

4.7
consumer
software or a device that reads packages through a package implementer

4.8
device
hardware, such as a personal computer, printer, or scanner, that performs a single function or set of functions

4.9
format consumer
consumer that consumes packages conforming to a format designer's specification

4.10
format designer
author of a particular file format specification built on this Open Packaging Conventions specification

4.11
format producer
producer that produces packages conforming to a format designer's specification

Commented [JH4]: Do a full pass over the document to clean up or remove uses of producer/consumer/implementer/designer.

4.12
growth hint
suggested number of bytes to reserve for a part to grow in-place

4.13
id
a name from an identification scheme

4.14
interleaved ordering
layout style of a physical package where parts are broken into pieces and “mixed-in” with pieces from other parts

4.15
layout style
style in which the collection of parts in a physical package is laid out

4.16
local access
access architecture in which a pipe carries data directly from a producer to a consumer on a single device

4.17
logical item
either a non-interleaved part or a piece of an interleaved part

4.18
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

4.19
pack URI
URI scheme that allows URIs to be used as a uniform mechanism for addressing parts within a package

4.20
package
logical entity that holds a collection of parts

4.21
package implementer
software that implements the physical input-output operations to a package according to the requirements and recommendations of this Open Packaging specification

4.22
package model
package abstraction that holds a collection of parts

4.23
package relationship
relationship whose target is a part and whose source is the package as a whole

4.24
part
stream of bytes with a MIME media type and associated common properties

4.25
part name
path component of a pack URI

4.26
physical model
description of the capabilities of a particular physical format

4.27
physical package format
specific file format, or other persistence or transport mechanism that can represent all of the capabilities of a package

4.28
piece
portion of a part.

4.29
pipe
communication mechanism that carries data from the producer to the consumer

4.30
producer
software or a device that writes packages through a package implementer

4.31
random access
style of communication between the producer and the consumer of the package

4.32
relationship
connection between a source part and a target part in a package

4.33
relationship type
absolute IRI for identifying a relationship

4.34
relationships part
part containing an XML representation of relationships

4.35
sequential delivery
communication style in which all of the physical bits in the package are delivered in the order they appear in the package

4.36
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

4.37
simple ordering
defined ordering for laying out the parts in a package in which all the bits comprising each part are stored contiguously

4.38
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

4.39
source part
part from which a connection is established by a relationship

4.40
stream
linearly ordered sequence of bytes

4.41
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

4.42
streaming creation
production style in which a producer dynamically adds parts to a package after other parts have been added without modifying those parts

4.43
target part
part to which a connection is established by a relationship

4.44
thumbnail
small image that is a graphical representation of a part or the package as a whole

4.45
unique identifier
a unique name from an identification scheme

4.46
XSD
W3C XML Schema

4.47
ZIP archive
ZIP file as defined in the ZIP file format specification

4.48
ZIP item
an atomic set of data in a ZIP archive that becomes a file when the archive is uncompressed

5 Notational Conventions

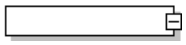
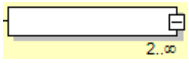
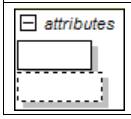
5.1 Document Conventions

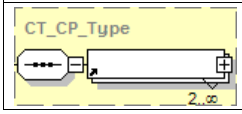
The following typographical conventions are used in ISO/IEC 29500:

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*]

5.2 Diagram Notes

In some cases, markup semantics are described using diagrams. The diagrams place the parent element on the left, with attributes and child elements to the right. The symbols are described below.

Symbol	Description
	Required element: This box represents an element that shall appear exactly once in markup when the parent element is included. The "+" and "–" symbols on the right of these boxes have no semantic meaning.
	Optional element: This box represents an element that shall appear zero or one times in markup when the parent element is included.
	Range indicator: These numbers indicate that the designated element or choice of elements can appear in markup any number of times within the range specified.
	Attribute group: This box indicates that the enclosed boxes are each attributes of the parent element. Solid-border boxes are required attributes; dashed-border boxes are optional attributes.

Symbol	Description
	Sequence symbol: The element boxes connected to this symbol shall appear in markup in the illustrated sequence only, from top to bottom.
	Choice symbol: Only one of the element boxes connected to this symbol shall appear in markup.
	Complex Type indicator: The elements within the dashed box are of the complex type indicated.

6 General Description

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

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.

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*, 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. Parts are analogous to a file in a file system or to a resource on an HTTP server.

Table 8–1. Part properties

Name	Description	Required/Optional
Name	The name of the part	Required. The package implementer shall require a part name. [M1.1]
Media type	The type of content stored in the part	Required. The package implementer shall require a media type and the format designer shall specify the media type. [M1.2]

Name	Description	Required/Optional
Growth Hint	A suggested number of bytes to reserve for the part to grow in-place	Optional. The package implementer might allow a growth hint to be provided by a producer. [O1.1]

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 rule in the ABNF syntax defined in RFC 2234, where isegment-nz is defined in RFC 3987

part_name = 1*("/" isegment-nz)

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 contain percent-encoded "!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "=", ":" , or "@" [Drafting note: Where does this come from? Should we drop this? The published Part 2 does not have this restriction.]
- No I18N segments shall end with a dot (“.”) character.

Commented [m5]: Why?

Commented [rcj6]:

where an I18N segment is a Unicode string that matches the non-terminal isegment-nz.

[*Example:* The part name “/hello/world/doc.xml” contains three path segments, namely, “hello”, “world”, and “doc.xml”. The first two path segments represent levels in the logical hierarchy and serve to organize the parts of the package, whereas the third contains actual content. *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.

Drafting Note: We might want to disallow the asterisk (“*”) and colon (“:”) part names. See the last example in “10.2.5 ZIP Package Limitations”

Commented [rcj7]:

Drafting note: RFC 3986 allows sub-delims ("!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / 8182"="), ":" , and "@" to occur in path segments, but they are not unreserved characters. In other words, they are expected to

have special semantics imposed by particular URI schemes. If OPC does not need special semantics for them, we might want to disallow them.

Drafting note: NTFS disallows "?" , '"' (double quotation mark), "/", "\", "<", ">", "*", " | ", and ":"

Drafting note: FAT disallows ".",'"' (double quotation mark), "/", "\", "[", "]", ":", ";", "|", "=", and ","

8.2.2.3 Part Name Integrity in a Package

Equivalence of part names is determined by ASCII case-insensitive matching. 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].

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].

[*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/…/segment*n*", other parts in that package must not have names such as "/segment1", "/SEGMENT1", "/segment1/segment2", "/segment1/SEGMENT2", or "/segment1/segment2/…/segment*n*-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 "/Å" where Å is 'LATIN CAPITAL LETTER A WITH RING ABOVE' (U+00C5) because U+212B and U+00C5 are normalized to the same character sequence. *end example*]

[*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 "/é", where é 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.

Format designers 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

Commented [rcj8]:

Commented [rcj9]:

Commented [rcj10]:

Commented [BCN11]: 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).

Commented [JH12]: Add example for /é == /É/f

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:

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) If the XML content contains the Markup Compatibility namespace, as described in Part 3, it shall be processed by the package implementer to remove Markup Compatibility elements and attributes, ignorable namespace declarations, and ignored elements and attributes before applying subsequent validation rules. [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]

Commented [BCN13]: Are we really going to allow MCE to be used in Digital Signature parts?

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.

[*Note:* Schemes are the prefix in an IRI before the colon. A well-known example is "http". *end note*]

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

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

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*.

The pack scheme is a provisional scheme in the IANA-maintained registry of schemes located at ???. A provisional registration does not have an expiration date. Further information on provisional registrations can be found at RFC 4395.

Commented [BCN14]: How much do we really want to say here?

The syntax of pack IRIs is defined by the EBNF (see RFC 2234) as follows:

```
pack_IRI  = "pack://" iauthority [ "/" | ipath ]
iauthority      = *( iunreserved | sub-delims | pct-encoded )
ipath     = 1*( "/" isegment )
isegment  = 1*( ipchar )
```

`sub-delims` and `pct-encoded` are defined in RFC 3986 and `iunreserved` and `ipchar` are defined in RFC 3987.

The authority component contains an embedded IRI that points to a package. The authority component shall not reference a package embedded in another package. The package implementer shall create an embedded IRI that meets the requirements defined in RFC 3987 for a valid IRI. [M7.1] describes the rules for composing pack IRIs by combining the IRI of an entire package resource with a part name.

The package implementer shall not create an authority component with an unescaped colon (:) character. [M7.4] Consumer applications, based on the obsolete URI specification RFC 2396, might tolerate the presence of an unescaped colon character in an authority component. [O7.1]

The optional path component identifies a particular part within the package. The package implementer shall only create path components that conform to the part naming rules. When the path component is missing, the resource identified by the pack IRI is the package as a whole. [M7.2]

In order to be able to embed the IRI of the package in the pack IRI, it is necessary either to replace or to percent-encode occurrences of certain characters in the embedded IRI. For example, forward slashes ("/") are replaced with commas (","). The rules for these substitutions are described in §8.3.4.

The optional query component in a pack IRI is ignored when resolving the IRI to a part.

A pack IRI might have a fragment identifier as specified in RFC 3987. If present, this fragment applies to whatever resource the pack IRI identifies.

[*Example:*

Example 8–. Using the pack IRI to identify a part

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

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

end example]

[*Example:*

Example 8–. 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:*

Example 8–. 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 is the IRI for the package as a whole.
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:*

Example –. Resolving a pack IRI to a resource

Given the pack IRI:

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

The components:

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

are converted to the package IRI:

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

and the path:

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

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

end example]

8.3.4 Composing a Pack IRI

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

In order to be suitable for creating a pack IRI, the IRI reference 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:*

Example –. Composing a pack IRI

Given the package IRI:

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

and the part name:

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

The pack IRI is:

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

end example]

8.3.5 Equivalence

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 defined in [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*]

8.3.6 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 ways for establishing base IRIs for resolving relative references. The procedure in this subclause provides the second way (5.1.2) dedicated to OPC packages.

Note: Base IRIs determined by the procedure in this subclause may be overridden by ways 3 or 4 in RFC 3986.

Commented [BCN15]: Update text that refers to base URIs

Case 1: Within a non-relationship part

The base IRI within a non-relationship part shall be the pack IRI created from the IRI of the package and the part name.

[*Example:*

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

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

The base IRI is

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

end example]

Case 2: Within a relationship part for some part

The base IRI within a relationship part shall be the pack IRI created from the IRI of the package and the source part name.

[*Example:*

Consider a relationship part /a/b/_rels/foo.xml.rels 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]

Case 3: Within a relationship part /_rels/.rels of the entire package

The base IRI within a relationship part shall be the pack IRI created from the IRI of the package.

[*Example:*

Consider a relationship part of a package available at http://www.mysite.com/my.package.

The base IRI is

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

end example]

8.4 Resolving Relative References

This subclause is informative.

Relative references in parts are resolved as specified in RFC 3987. With the exception of optional preprocessing (see), this part of ISO/IEC 29500 introduces no changes to the resolution procedure.

Commented [rcj16]: Is this referring to Part 3?

This subclause shows examples of resolving relative references to pack IRIs in relative to two pack IRIs. One is a pack IRI "pack://http%3c,example.com,foo.opc/a/foo.xml" for a part /a/foo.xml, while the other is a pack IRI "pack://http%3c,example.com,foo.opc/" for an entire package.

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 is 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

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

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.

The package introduces a higher-level mechanism to describe references from parts to other internal or external resources, namely, relationships. *Relationships* represent the type of connection between a source part and a target resource. They 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 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*]

8.5.2 Relationships Part

Each set of relationships sharing a common source is represented by XML stored in a *Relationships part*. The Relationships part is IRI-addressable and it can be opened, read, and deleted. The Relationships part shall not have relationships to any other part. Package implementers shall enforce this requirement upon the attempt to create such a relationship and shall treat any such relationship as invalid. [M1.25]

The media type of the Relationships part is defined in Annex E.

8.5.3 Relationship Markup

8.5.3.1 General

Relationships are represented using Relationship elements nested in a single Relationships element. These elements are defined in the Relationships namespace, as specified in Annex E. The W3C XML Schema for relationships is described in Annex C.5.

After the removal of any extensions using the mechanisms in ISO/IEC 29500-3, a Relationships part shall be a schema-valid XML document against opc-relationships.xsd.

The package implementer shall require that every Relationship element has an Id attribute, the value of which is unique within the Relationships part, and that the Id datatype is xsd:ID, the value of which conforms to the naming restrictions for xsd:ID as described in the W3C Recommendation "XML Schema Part 2: Datatypes." [M1.26]

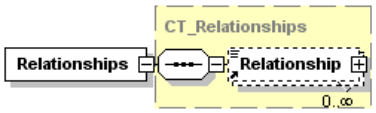
The nature of a Relationship element is identified by the Type attribute. The value of this attribute shall be a relationship type. By using types patterned after the Internet domain-name space, non-coordinating parties can safely create non-conflicting relationship types.

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.

The Target attribute of the Relationship element holds a URI that points to a target resource. Where the URI is expressed as a relative reference, it is resolved against the base URI of the Relationships source part. The xml:base attribute shall not be used to specify a base URI for relationship XML content.

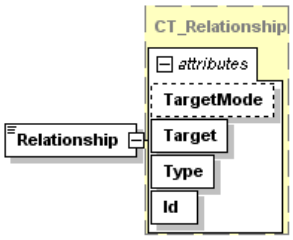
8.5.3.2 Relationships Element

The structure of a Relationships element is shown in the following diagram:

diagram	
annotation	The root element of the Relationships part.

8.5.3.3 Relationship Element

The structure of a Relationship element is shown in the following diagram:

diagram	

attributes

Name	Type	Use	Default	Fixed	Annotation
TargetMode	ST_TargetMode	optional			The package implementer might allow a TargetMode to be provided by a producer. [O1.5] The TargetMode indicates whether or not the target describes a resource inside the package or outside the package. The valid values, in the Relationships schema, are `Internal` and `External`. The default value is `Internal`. When set to `Internal`, the Target attribute shall be a relative reference and that reference is interpreted relative to the “parent” part. For package relationships, the package implementer shall resolve relative references in the Target attribute

						against the pack URI that identifies the entire package resource. [M1.29] For more information, see **Error! eference source not found.**. When set to `External`, the Target attribute can be a relative reference or a URI. If the Target attribute is a relative reference, then that reference is interpreted relative to the location of the package.
	Target	xsd:anyURI	required			The package implementer shall require the Target attribute to be a URI reference pointing to a target resource. The URI reference shall be a URI or a relative reference. [M1.28] [*Note:* The target is a reference to a part, not a part name, and thus is not restricted to the syntax requirements for part names. *end note*] Target attribute values are dependent on the TargetMode attribute value.
	Type	xsd:anyURI	required			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]
	Id	xsd:ID	required			The package implementer shall require a valid XML identifier. [M1.26] The Id type is xsd:ID and it shall conform to the naming restrictions for xsd:ID as specified in the W3C Recommendation “XML Schema Part 2: Datatypes.” The value of the Id attribute shall be unique within the

						Relationships part.
annotation	Represents a single relationship.					

A format designer might allow fragment identifiers in the value of the Target attribute of the Relationship element. [O1.6] If a fragment identifier is allowed in the Target attribute of the Relationship element, a package implementer shall not resolve the URI to a scope less than an entire part. [M1.32]

8.5.4 Representing Relationships

Relationships are represented in XML in a Relationships part. Each part in the package that is the source of one or more relationships can have 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.

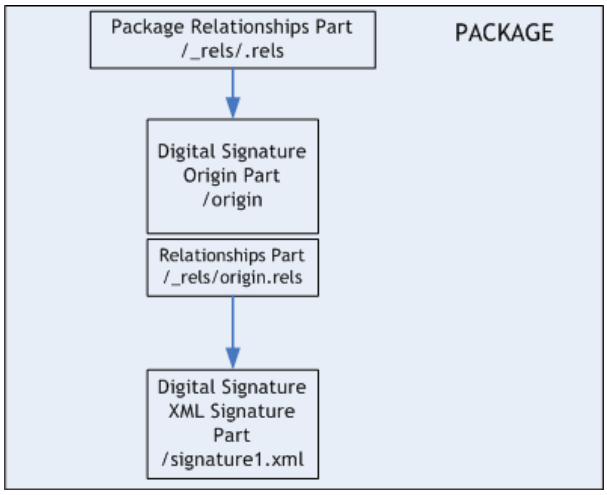
A special naming convention is used for the Relationships part. First, the Relationships part for a part in a given folder in the name hierarchy is stored in a sub-folder called "_rels". Second, the name of the Relationships part is formed by appending ".rels" to the name of the original part. Package relationships are found in the package relationships part named "/_rels/.rels".

The package implementer shall name relationship parts according to the special relationships part naming convention and require that parts with names that conform to this naming convention have the media type for a Relationships part. [M1.30]

[*Example:*

Example 8–3. Sample relationships and associated markup

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 relationship.



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

The Relationships part associated with the Digital Signature Origin 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>
```

end example]

[*Example:*

Example 8–4. Targeting 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 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.custom.com/images/pic1.jpg"
       Type="http://www.custom.com/external-resource"/>
    <Relationship
       TargetMode="External"
       Id="A5EFC797514BC"
       Target="./images/my_house.jpg"
       Type="http://www.custom.com/external-resource"/>
 </Relationships>
```

end example]

[*Example:*

Example 8–5. Re-using attribute values

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"
       Id="B5F32797CC4B7"
       Type="http://www.custom.com/internal-resource"/>
 </Relationships>
```

end example]

8.5.5 Support for Versioning and Extensibility

Producers might generate relationship markup that uses the versioning and extensibility mechanisms defined in Part 3 to incorporate elements and attributes drawn from other XML namespaces. [O1.7]

Consumers shall process relationship markup in a manner that conforms to Part 3. [M1.31]

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. The physical model includes the *access style*—the manner in which package input-output is conducted—as well as the *communication style*, which describes the method of interaction between producers and consumers across a communications *pipe*. The physical model also includes the *layout style*; that is, how part contents are physically stored within the package. The layout style either can be *simple ordering*, where the parts are arranged contiguously each as an atomic block of data, or *interleaved ordering*, where the parts are broken into individual pieces and the pieces are stored as interleaved blocks of data in an optimized fashion. The performance of a physical package design is reliant upon the physical model capabilities.

[*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 conform to the following markup specification.

The Media Types stream contains XML with 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:

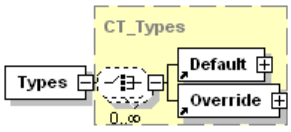
- The package implementer should not allow Default elements; consequently, there should be one Override element for each part in the package.
- The format producer should write the Override elements to the package, so they appear before the part to which they correspond, or in close proximity to the part to which they correspond.

[S2.3]

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

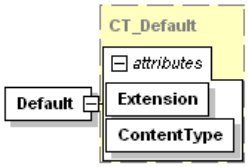
9.2.3.2.2 Types Element

The structure of a Types element is shown in the following diagram:

diagram	
annotation	The root element of the Media Types stream.

9.2.3.2.3 Default Element

The structure of a Default element is shown in the following diagram:

diagram	

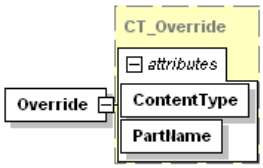
attributes

Name	Type	Use	Default	Fixed	Annotation
Extension	ST_Extension	required			A part-name extension. A Default element matches any part whose name ends with a period (“.”) followed by the value of this attribute. The package implementer shall require a non-empty extension in a Default element. [M2.6]
ContentType	ST_ContentType	required			A media type specified using the syntax defined in RFC 7231 §3.1.1.1. Indicates the media type of any matching parts (unless overridden). The package implementer shall require a media type in a Default element and the format designer shall specify the media type. [M2.6]

annotation	Defines default mappings from the extensions of part names to media types.

9.2.3.2.4 Override Element

The structure of an Override element is shown in the following diagram:

diagram	

attributes

Name	Type	Use	Default	Fixed	Annotation
ContentType	ST_ContentType	required			A media type specified using the syntax defined in RFC 7231 §3.1.1.1. Indicates the media type of the part referenced by the PartName attribute. The package implementer shall require a media type and the format designer shall specify the media type in an Override element. [M2.7]
PartName	xs:anyURI	required			A part name (§8.2.2). An Override element matches the part whose name is equal to the value of this attribute. The package implementer shall require a part name. [M2.7]

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

9.2.3.2.5 Media Types Stream Markup Example

[*Example:*

Example 9–6. 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]

9.2.3.3 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 case-insensitive ASCII.
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.

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

Commented [JH17]: Should this be changed?

9.2.3.4 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 case-insensitive ASCII.
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 case-insensitive ASCII.

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.

9.2.3.5 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

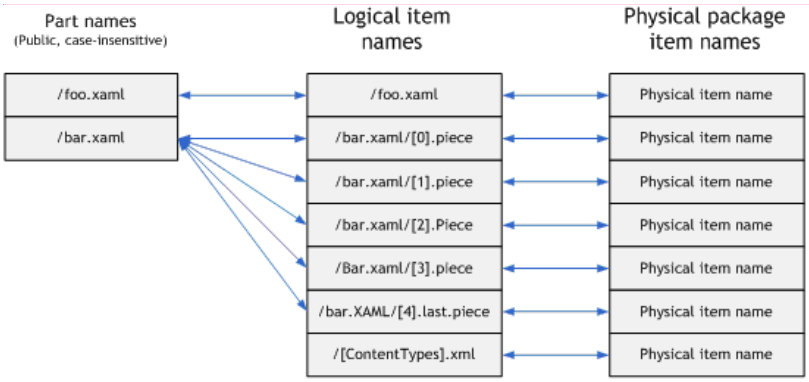
9.2.4.1 General

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 [JH18]: 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 as case-insensitive ASCII strings. [M2.12]

The package implementer shall compare suffix names as case-insensitive ASCII strings. [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] The format designer specifies whether that format might use interleaving. [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 9–7. 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
```

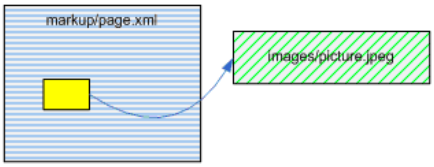
end example]

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

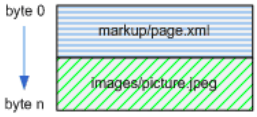
[*Example:*

Example 9–8. 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.



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, Package model components and their physical representations, 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 §**Error! eference source not found.**. See §9.3.7 for details about the ZIP item name.
Growth hint	Padding reserved in the ZIP Extra field in the local header that precedes the item. See §9.3.8 for a detailed description of the data structure.

9.3.2 Mapping Part Data

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

Field	Size	Value
<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

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

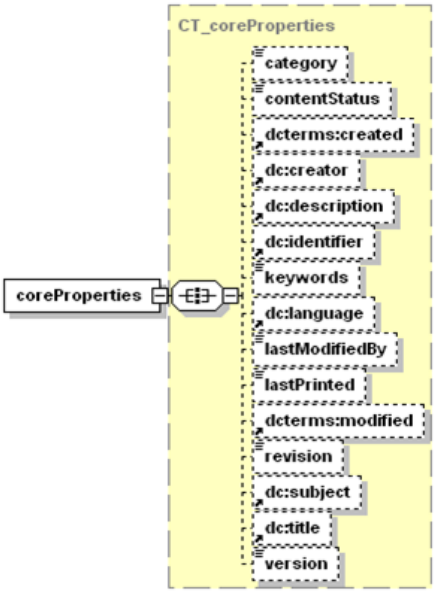
keywords	Open Packaging Conventions	A delimited set of keywords to support searching and indexing. This is typically a list of terms that are not available elsewhere in the properties.
		
		The definition of this element uniquely allows for:
		• Use of the xml:lang attribute to identify languages
		• A mixed content model, such that keywords can be flagged 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]

language	Dublin Core	The language of the intellectual content of the resource. [*Note*: IETF RFC 3066 provides guidance on encoding to represent languages. *end note*]
lastModifiedBy	Open Packaging Conventions	The user who performed the last modification. The identification is environment-specific. [*Example*: A name, email address, or employee ID. *end example*] 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. [*Example:* This value might indicate the number of saves or revisions, provided the application updates it after each revision. *end example*]
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. The Core Properties part media type is defined in Annex E.

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

diagram	
annotation	Producers might provide all or a subset of these metadata properties to describe the contents of a package.

[*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 dcterms:W3CDTF, where `dcterms` 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

The format designer 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]

The format designer shall specify thumbnail parts that are 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

Format designers 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 §12.4.2.

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

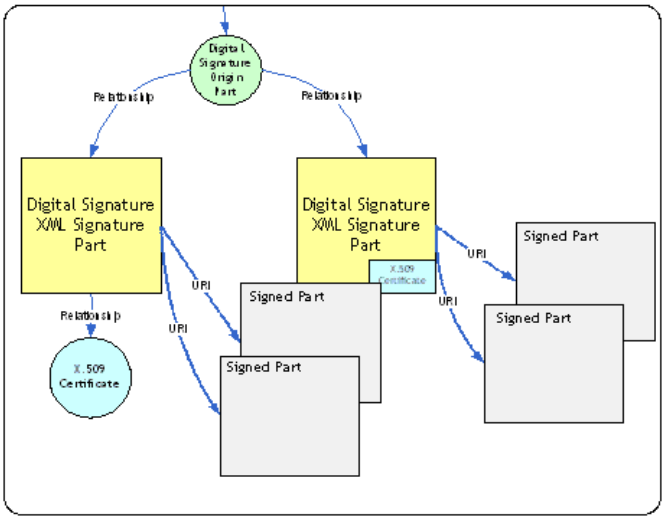
12.3.1 General

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

Commented [JH19]: 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"

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 §12.5.

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

Commented [BCN20]:

12.4.2 Modifications to the XML Digital Signature Specification

Commented [JH21]: This subclause might be redesigned?

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 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].

12.4.3 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 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 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)

[M6.34]

12.4.6 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 [BCN22]: Needs work. Should OPC restrict the set of allowed algorithms?

Eventually, remove mention of producer/consumer.

12.4.7 Reference Element as a Child of a Manifest Element

The structure of a Reference element is defined in §4.3.3 of XML-Signature Syntax and Processing.

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 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 media type of the targeted part.

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

[*Example:*

Example 12–2. 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]

12.4.8 Transforms Element

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

12.4.9 Transform Element

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

Only the following transform algorithms shall be used:

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

Relationships transforms shall only be used when the Transform element is a descendant element of a Manifest element [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
```

12.4.10 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. [M6.17]

Commented [BCN23]: Needs work. Should OPC restrict the set of allowed algorithms?

Commented [BCN24]: Clarify which version of these algorithms we mean.

Commented [BCN25]: Should OPC restrict the set of allowed algorithms?

12.4.11 Object Element

The structure of an Object element is defined in §4.2 of XML-Signature Syntax and Processing.

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

12.4.11.1 Package-Specific Object Element

The package-specific Object element contains the Manifest and SignatureProperties elements that are package-specific. Each Signature element shall have exactly one package-specific Object. [M6.15] The Id attribute shall be specified and its value shall be `idPackageObject`.

12.4.11.2 Application-Defined Object Element

The application-defined Object element specifies application-defined information. 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] Format designers might not apply package-specific restrictions regarding URIs and Transform elements to application-defined Object elements. [O6.9]

12.4.12 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]

12.4.13 Manifest Element

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

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

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

A SignatureProperty element within a package-specific Object element shall only have one child element, which shall be a SignatureTime element.

12.4.15 SignatureTime Element

The SignatureTime element holds the date/time stamp for the signature. 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.

12.4.16 Format Element

The Format element specifies the format of the date/time stamp. 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.

12.4.17 Value Element

The Value element specifies the value of the date/time stamp. 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.

12.4.18 RelationshipReference Element

The RelationshipReference element specifies the Relationship element with the specified Id value is to be signed. A RelationshipsReference element shall only occur as a child element of a Transform element (§12.4.9) 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.

12.4.19 RelationshipsGroupReference Element

The RelationshipsGroupReference element specifies that the group of Relationship elements with the specified value for the Type attribute is to be signed. A RelationshipsGroupReference element shall only occur as a child element of the Transform element (§12.4.9) that is a Relationships Transform. The schema definition for the RelationshipsGroupReference element is specified inAnnex C.

Format designers might permit producers to sign individual relationships in a package or the Relationships part as a whole. [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 (§12.4.9). 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.

12.4.20 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 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.

```
<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>
```

Commented [JH26]: Need to incorporate pending new version of XAdES
Need to say something about using old vs. new (e.g., allow old, recommend new)

Commented [JH27]: John TBD: Why does MSO requrie this for OOXML files?

Commented [JH28]: 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 [JH29]: ODF: "If references to validation data are present "
MS-OFF: "If … contains time stamps on references to validation data "

Commented [JH30]: [JP] Miyachi-san prefers we require CertificatesValues/RevocationValues (-X-L)

Commented [BCN31]: ODF says "shall"
MSO accommodates a version of MSO 2007, which does something different

Commented [JH32]: Need one (partial?) with XAdES, or add XAdES to this and comment it as optional?

```
</SignedInfo>
<SignatureValue>…</SignatureValue>

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

<Object Id="idPackageObject" xmlns:pds="http://schemas.openxmlformats.org/
   package/2006/digital-signature">
   <Manifest>
      <Reference URI="/document.xml?ContentType=application/
         vnd.ms-document+xml">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/2001/
               REC-xml-c14n-20010315"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>…</DigestValue>
      </Reference>
      <Reference
         URI="/_rels/document.xml.rels?ContentType=application/
            vnd.openxmlformats-package.relationships+xml">
         <Transforms>
            <Transform Algorithm="http://schemas.openxmlformats.org/
               package/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 Generating Signatures

Commented [JH33]: Does anything need to change here for XAdES?

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 designers that utilize application-defined Object elements shall also define the additional steps that shall be performed to sign the application-defined Object elements.

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 SignatureMethodelement, 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 Validating Signatures

Commented [JH34]: Does anything need to change here for XAdES?

12.8.1 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 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:

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 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 Support for Versioning and Extensibility

12.9.1 General

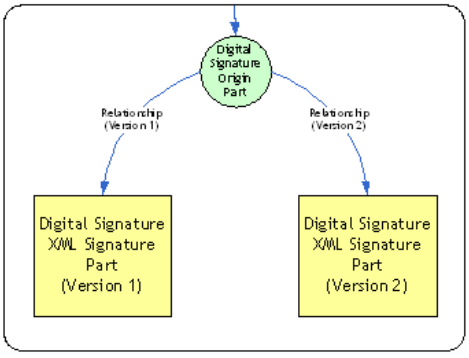
The package digital signature infrastructure supports the exchange of signed packages between current and future package clients.

12.9.2 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 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]

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 format designer shall carefully design 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

Although relative references within packages can reference parts, Unicode strings that are similar to but are not strictly relative references are 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.

Commented [rcj35]: Need to reword this.

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

[*Example:*

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

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

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

end example]

Commented [rcj36]: 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 Appnote.txt.

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

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

<table>
<tr><th>Bit</th><th>Feature</th><th>Supported on Consumption</th><th>Supported on Production</th><th>Pass through on editing</th></tr>
<tr><td>0</td><td>If set, indicates that the file is encrypted.</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>1, 2</td><td><table><tr><th>Bit 2</th><th>Bit 1</th><th></th></tr><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></table></td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>3</td><td>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.</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>4</td><td>Reserved for use with method 8, for enhanced deflating</td><td>No</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>5</td><td>If this bit is set, this indicates that the file is compressed patched data. (Requires PKZIP version 2.70 or greater.)</td><td>No</td><td>Bits set to 0</td><td>Yes</td></tr>
</table>

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

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

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

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

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

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

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

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

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

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

```
<xs:schema xmlns="http://schemas.openxmlformats.org/package/2006/content-types"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.openxmlformats.org/package/2006/content-types"
  elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
       <xs:element name="Types" type="CT_Types"/>
       <xs:element name="Default" type="CT_Default"/>
       <xs:element name="Override" type="CT_Override"/>
    <xs:complexType name="CT_Types">
       <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="Default"/>
          <xs:element ref="Override"/>
       </xs:choice>
    </xs:complexType>
    <xs:complexType name="CT_Default">
       <xs:attribute name="Extension" type="ST_Extension" use="required"/>
       <xs:attribute name="ContentType" type="ST_ContentType" use="required"/>
    </xs:complexType>
    <xs:complexType name="CT_Override">
       <xs:attribute name="ContentType" type="ST_ContentType" use="required"/>
       <xs:attribute name="PartName" type="xs:anyURI" use="required"/>
    </xs:complexType>
    <xs:simpleType name="ST_ContentType">
       <xs:restriction base="xs:string">
       <xs:pattern value=" (((([\p{IsBasicLatin}-
[\p{Cc}&#127;\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]])+))/((([\p{IsBasicLatin}-
[\p{Cc}&#127;\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]])+))((\s+)*;(\s+)*(((([\p{IsBasicLatin}-
[\p{Cc}&#127;\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]])+))=((([\p{IsBasicLatin}-
[\p{Cc}&#127;\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]])+)|(&quot;(([\p{IsLatin-
1Supplement}\p{IsBasicLatin}-[\p{Cc}&#127;&quot;\n\r]]|(\s+))|(\\[\p{IsBasicLatin}]))*&quot;))))*)"/>
       </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ST_Extension">
       <xs:restriction base="xs:string">
       <xs:pattern value=" ([!$&amp;'\(\)\*\+,:=]|(%[0-9a-fA-F][0-9a-fA-F])|[:@]|[a-zA-Z0-9\-_~])+"/>
```

```
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

C.3 Core Properties Part

```
<xs:schema targetNamespace="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
  xmlns="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/" elementFormDefault="qualified" blockDefault="#all">
    <xs:import namespace="http://purl.org/dc/elements/1.1/"
      schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dc.xsd"/>
    <xs:import namespace="http://purl.org/dc/terms/"
      schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dcterms.xsd"/>
    <xs:import id="xml" namespace="http://www.w3.org/XML/1998/namespace"/>
        <xs:element name="coreProperties" type="CT_CoreProperties"/>
    <xs:complexType name="CT_CoreProperties">
        <xs:all>
            <xs:element name="category" minOccurs="0" maxOccurs="1" type="xs:string"/>
            <xs:element name="contentStatus" minOccurs="0" maxOccurs="1" type="xs:string"/>
            <xs:element ref="dcterms:created" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="dc:creator" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="dc:description" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="dc:identifier" minOccurs="0" maxOccurs="1"/>
        <xs:element name="keywords" minOccurs="0" maxOccurs="1" type="CT_Keywords"/>
            <xs:element ref="dc:language" minOccurs="0" maxOccurs="1"/>
            <xs:element name="lastModifiedBy" minOccurs="0" maxOccurs="1" type="xs:string"/>
            <xs:element name="lastPrinted" minOccurs="0" maxOccurs="1" type="xs:dateTime"/>
            <xs:element ref="dcterms:modified" minOccurs="0" maxOccurs="1"/>
            <xs:element name="revision" minOccurs="0" maxOccurs="1" type="xs:string"/>
            <xs:element ref="dc:subject" minOccurs="0" maxOccurs="1"/>
            <xs:element ref="dc:title" minOccurs="0" maxOccurs="1"/>
            <xs:element name="version" minOccurs="0" maxOccurs="1" type="xs:string"/>
        </xs:all>
    </xs:complexType>
    <xs:complexType name="CT_Keywords" mixed="true">
        <xs:sequence>
        <xs:element name="value" minOccurs="0" maxOccurs="unbounded" type="CT_Keyword"/>
        </xs:sequence>
        <xs:attribute ref="xml:lang" use="optional"/>
    </xs:complexType>
    <xs:complexType name="CT_Keyword">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute ref="xml:lang" use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

C.4 Digital Signature XML Signature Markup

```
<xsd:schema xmlns="http://schemas.openxmlformats.org/package/2006/digital-signature"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.openxmlformats.org/package/2006/digital-signature"
  elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
        <xsd:element name="SignatureTime" type="CT_SignatureTime"/>
        <xsd:element name="RelationshipReference" type="CT_RelationshipReference"/>
        <xsd:element name="RelationshipsGroupReference" type="CT_RelationshipsGroupReference"/>
    <xsd:complexType name="CT_SignatureTime">
        <xsd:sequence>
        <xsd:element name="Format" type="ST_Format"/>
        <xsd:element name="Value" type="ST_Value"/>
</xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CT_RelationshipReference">
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="SourceId" type="xsd:string" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:complexType name="CT_RelationshipsGroupReference">
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="SourceType" type="xsd:anyURI" use="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:simpleType name="ST_Format">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="(YYYY)|(YYYY-MM)|(YYYY-MM-DD)|(YYYY-MM-DDThh:mmTZD)|(YYYY-MM-
              DDThh:mm:ssTZD)|(YYYY-MM-DDThh:mm:ss.sTZD)"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ST_Value">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="(([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])-((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[1-9])|(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[0-9])|(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[0-9])|(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)))-((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[0-9])|(3[0-
              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]))(((\+|-
              )((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-
              9])))|Z))|(([0-9][0-9][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[0-9])|(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[0-9])|(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[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
              9])))|Z))"/>
        </xsd:restriction>
    </xsd:simpleType>
```

```
</xsd:schema>
```

C.5 Relationships Part

```
<xsd:schema xmlns="http://schemas.openxmlformats.org/package/2006/relationships"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.openxmlformats.org/package/2006/relationships"
  elementFormDefault="qualified" attributeFormDefault="unqualified" blockDefault="#all">
       <xsd:element name="Relationships" type="CT_Relationships"/>
       <xsd:element name="Relationship" type="CT_Relationship"/>
    <xsd:complexType name="CT_Relationships">
       <xsd:sequence>
          <xsd:element ref="Relationship" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CT_Relationship">
       <xsd:simpleContent>
          <xsd:extension base="xsd:string">
       <xsd:attribute name="TargetMode" type="ST_TargetMode" use="optional"/>
              <xsd:attribute name="Target" type="xsd:anyURI" use="required"/>
              <xsd:attribute name="Type" type="xsd:anyURI" use="required"/>
              <xsd:attribute name="Id" type="xsd:ID" use="required"/>
          </xsd:extension>
       </xsd:simpleContent>
    </xsd:complexType>
    <xsd:simpleType name="ST_TargetMode">
       <xsd:restriction base="xsd:string">
          <xsd:enumeration value="External"/>
          <xsd:enumeration value="Internal"/>
       </xsd:restriction>
    </xsd:simpleType>
</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

```
default namespace =
  "http://schemas.openxmlformats.org/package/2006/content-types"

start = Types
Types = element Types { CT_Types }
Default = element Default { CT_Default }
Override = element Override { CT_Override }
CT_Types = (Default | Override)*
CT_Default =
  attribute Extension { ST_Extension },
  attribute ContentType { ST_ContentType }
CT_Override =
  attribute ContentType { ST_ContentType },
  attribute PartName { xsd:anyURI }
ST_ContentType =
  xsd:string {
    pattern =
      '(((([\p{IsBasicLatin}-[\p{Cc}\x{127}\(\)<>@,;:\\"/\[\]\?=\{\}\s\t]])+))/((([\p{IsBasicLatin}-
[\p{Cc}\x{127}\(\)<>@,;:\\"/\[\]\?=\{\}\s\t]])+))((\s+)*;(\s+)*((((([\p{IsBasicLatin}-
[\p{Cc}\x{127}\(\)<>@,;:\\"/\[\]\?=\{\}\s\t]])+))=((((([\p{IsBasicLatin}-
[\p{Cc}\x{127}\(\)<>@,;:\\"/\[\]\?=\{\}\s\t]])+))|("(([\p{IsLatin-1Supplement}\p{IsBasicLatin}-
[\p{Cc}\x{127}"\n\r]]|(\s+))|(\\[\p{IsBasicLatin}]))*"))))*)'
  }
ST_Extension =
  xsd:string {
    pattern =
      "([!$&'\(\)\*\+,:=]|(%[0-9a-fA-F][0-9a-fA-F])|[:@]|[a-zA-Z0-9\-_~])+"
  }
```

D.3 Core Properties Part

```
default namespace =
  "http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
namespace dc = "http://purl.org/dc/elements/1.1/"
namespace dcterms = "http://purl.org/dc/terms/"
namespace xsi = "http://www.w3.org/2001/XMLSchema-instance"
include "xml.rnc"

start = coreProperties
coreProperties = element coreProperties { CT_CoreProperties }
CT_CoreProperties =
  element category { xsd:string }?
  & element contentStatus { xsd:string }?
  & element dcterms:created {
      attribute xsi:type { xsd:QName "dcterms:W3CDTF" }, xml_lang?, W3CDTF
    }?
  & element dc:creator { SimpleLiteral }?
  & element dc:description { SimpleLiteral }?
  & element dc:identifier { SimpleLiteral }?
  & element keywords { CT_Keywords }?
  & element dc:language { SimpleLiteral }?
  & element lastModifiedBy { xsd:string }?
  & element lastPrinted { xsd:dateTime }?
  & element dcterms:modified {
      attribute xsi:type { xsd:QName "dcterms:W3CDTF" }, xml_lang?, W3CDTF
    }?
  & element revision { xsd:string }?
  & element dc:subject { SimpleLiteral }?
  & element dc:title { SimpleLiteral }?
  & element version { xsd:string }?
CT_Keywords =
  mixed {
    xml_lang?,
    element value { CT_Keyword }*
  }
CT_Keyword = xsd:string, xml_lang?
SimpleLiteral = xml_lang?,  xsd:string
W3CDTF = xsd:gYear | xsd:gYearMonth | xsd:date | xsd:dateTime
```

D.4 Digital Signature XML Signature Markup

```
default namespace =
  "http://schemas.openxmlformats.org/package/2006/digital-signature"
namespace ds = "http://www.w3.org/2000/09/xmldsig#"

include "xmldsig-core-schema.rnc" {

SignaturePropertyType =
  SignatureTime,
  attribute Id { xsd:ID }?,
  attribute Target { xsd:anyURI }

```

```
TransformType =
  element ds:XPath { xsd:string }?,
  (RelationshipReference | RelationshipsGroupReference)*,
  attribute Algorithm { xsd:anyURI }
}

SignatureTime = element SignatureTime { CT_SignatureTime }
RelationshipReference =
  element RelationshipReference { CT_RelationshipReference }
RelationshipsGroupReference =
  element RelationshipsGroupReference { CT_RelationshipsGroupReference }
CT_SignatureTime =
  element Format { ST_Format },
  element Value { ST_Value }
CT_RelationshipReference =
  xsd:string,
  attribute SourceId { xsd:string }
CT_RelationshipsGroupReference =
  xsd:string,
  attribute SourceType { xsd:anyURI }
ST_Format =
  xsd:string {
    pattern =
      "(YYYY)|(YYYY-MM)|(YYYY-MM-DD)|(YYYY-MM-DDThh:mmTZD)|(YYYY-MM-DDThh:mm:ssTZD)|(YYYY-MM-
DDThh:mm:ss.sTZD)"
  }
ST_Value =
  xsd:string {
    pattern =
      "(([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])-((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-
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-
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-
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)))-
((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-
9])|(3[0-9])|(4[0-9])|(5[0-9])):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9]))(((\+|-)((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[0-9])))|Z))|(([0-9][0-
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-
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-
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-
9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9])))|Z))"
  }
```

D.5 Relationships Part

```
default namespace =
  "http://schemas.openxmlformats.org/package/2006/relationships"

start = Relationships
Relationships = element Relationships { CT_Relationships }
Relationship = element Relationship { CT_Relationship }
CT_Relationships = Relationship*
CT_Relationship =
```

```
  xsd:string,
  attribute TargetMode { ST_TargetMode }?,
  attribute Target { xsd:anyURI },
  attribute Type { xsd:anyURI },
  attribute Id { xsd:ID }
ST_TargetMode = string "External" | string "Internal"
```

D.6 Additional Resources

D.6.1 XML

```
xml_lang = attribute xml:lang { xsd:language | xsd:string "" }
xml_space = attribute xml:space { "default" | "preserve" }
xml_base = attribute xml:base { xsd:anyURI }
xml_id = attribute xml:id { xsd:ID }
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
Markup Compatibility	http://schemas.openxmlformats.org/markup-compatibility/2006

Commented [JH37]: Should this be specified here, or just reference Part 3?

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

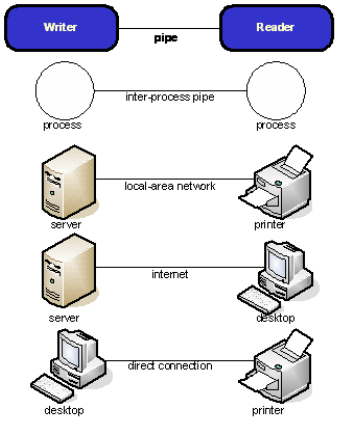
Annex F
(informative)
Physical Model Design Considerations

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.

[Drafting Note: Should we delete all tables this annex? (1) They add nothing new but merely duplicate information, (2) they do not capture some requirements in the body, (3) they make the revision process difficult, and (4) some columns are not about conformance. It was agreed to postpone this issue until we have a better idea about the introduction of XAdES into OPC.]

Commented [rcj38]: I have not been updating this Annex. If it is retained, it will have to be updated to reflect the final content of the rest of the spec.

Commented [rcj39]:

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	The package implementer shall require a part name. **Error! Reference source not found. Error! Reference source not ound.**	8.2, **Error! eference source not found.**, **Error! Reference source not found.**	×			
M1.2	The package implementer shall require a media type and the format designer shall specify the media type.	8.2	×	×		
M1.3	**Error! Reference source not ound. Error! Reference source not found.Error! Reference source not found.**	**Error! eference source not found.**, **Error! Reference source not found.**	×			
M1.4	**Error! Reference source not ound. Error! Reference source not found.**	**Error! eference source not found.**, **Error! Reference source not found.**	×			
M1.5	**Error! Reference source not ound.Error! Reference source not found.**	**Error! eference source not found.**, **Error! Reference source not found.**	×			
M1.6	**Error! Reference source not ound.Error! Reference source not found.**.	**Error! eference source not found.**, **Error! Reference source not found.**	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.7	**Error! Reference source not ound.Error! Reference source not found.**	**Error! eference source not found., Error! Reference source not found.**	×			
M1.8	**Error! Reference source not ound.**	**Error! eference source not found., Error! Reference source not found.**	×			
M1.9	**Error! Reference source not ound.Error! Reference source not found.**	**Error! eference source not found., Error! Reference source not found.**	×			
M1.10	**Error! Reference source not ound. Error! Reference source not found.**	**Error! eference source not found., Error! Reference source not found.**	×			
M1.11	**Error! Reference source not ound.**	0	×			
M1.12	**Error! Reference source not ound.**	**Error! eference source not found.**	×			
M1.13	**Error! Reference source not ound.**	0	×	×		
M1.14	**Error! Reference source not ound.**	0	×	×		
M1.15	**Error! Reference source not ound.**	0	×	×		

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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	×		×	
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	×			
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	×			
M1.19	If the XML content contains the Markup Compatibility namespace, as described in Part 3, it shall be processed by the package implementer to remove Markup Compatibility elements and attributes, ignorable namespace declarations, and ignored elements and attributes before applying subsequent validation rules.	8.2.5	×			

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	×			
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	×			
M1.22	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.	Annex E	×	×		
M1.23	**Error! Reference source not ound.**	8.3				×
M1.24	**Error! Reference source not ound.Error! Reference source not found.**	8.3				×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.25	The Relationships part shall not have relationships to any other part. Package implementers shall enforce this requirement upon the attempt to create such a relationship and shall treat any such relationship as invalid.	8.5.2	×			
M1.26	After the removal of any extensions using the mechanisms in ISO/IEC 29500-3, a Relationships part shall be a schema-valid XML document against opc-relationships.xsd. The package implementer shall require that every Relationship element has an Id attribute, the value of which is unique within the Relationships part, and that the Id datatype is xsd:ID, the value of which conforms to the naming restrictions for xsd:ID as described in the W3C Recommendation "XML Schema Part 2: Datatypes."	8.5.3	×			
M1.27	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.	8.5.3.3	×	×		
M1.28	The package implementer shall require the Target attribute to be a URI reference pointing to a target resource. The URI reference shall be a URI or a relative reference.	8.5.3.3	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.29	When set to Internal, the Target attribute shall be a relative reference and that reference is interpreted relative to the "parent" part. For package relationships, the package implementer shall resolve relative references in the Target attribute against the pack URI that identifies the entire package resource.	8.5.3.3	×			
M1.30	The package implementer shall name relationship parts according to the special relationships part naming convention and require that parts with names that conform to this naming convention have the media type for a Relationships part	8.5.4	×			
M1.31	Consumers shall process relationship markup in a manner that conforms to Part 3.	8.5.5			×	×
M1.32	If a fragment identifier is allowed in the Target attribute of the Relationship element, a package implementer shall not resolve the URI to a scope less than an entire part.	8.5.3.3	×			
M1.33	**Error! Reference source not ound.**	**Error! eference source not found.**			×	×
M1.34	**Error! Reference source not ound.**	**Error! eference source not found.**				×

Table G–2. Package model optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O1.1	The package implementer might allow a growth hint to be provided by a producer.	8.2, 8.2.4	×			
O1.2	Format designers 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	The package implementer might allow a TargetMode to be provided by a producer.	8.5.3.3	×			
O1.6	A format designer might allow fragment identifiers in the value of the Target attribute of the Relationship element.	8.5.3.3		×		
O1.7	Producers might generate relationship markup that uses 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! eference source not found.**	×[A]			
M2.2	The package implementer shall define a physical package format with a mapping for the following required components.	9.2.2	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.3	The package implementer shall define a format mapping with a mechanism for associating media types with parts.	**Error! eference source not found.**	×			
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	×[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	×[A]			
M2.6	The package implementer shall require a non-empty extension in a Default element. The package implementer shall require a media type in a Default element and the format designer shall specify the media type.	9.2.3.2.3	×[A]	×[A]		
M2.7	The package implementer shall require a media type and the format designer shall specify the media type in an Override element. The package implementer shall require a part name.	9.2.3.2.4	×[A]	×[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 §9.2.3.3.	9.2.3.3	×[A]			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.9	To get the media type of a part, the package implementer shall perform the steps described in §9.2.3.4.	9.2.3.4	×[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.	9.2.3.5	×[A]			
M2.11	The package implementer shall not mix interleaving and non-interleaving for an individual part.	9.2.5	×[B]			
M2.12	The package implementer shall compare prefix names as case-insensitive ASCII strings.	9.2.4.2	×			
M2.13	The package implementer shall compare suffix names as case-insensitive ASCII strings.	9.2.4.2	×[B]			
M2.14	The package implementer shall not allow packages that contain equivalent logical item names.	9.2.4.2	×			
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	×[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	×			
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	×			
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	×[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! eference source not found.**	×			
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! eference source not found.**	×			
S2.3	If the package is intended for streaming consumption: The package implementer should not allow Default elements; consequently, there should be one Override element for each part in the package. The format producer should write the Override elements to the package, so they appear before the part to which they correspond, or in close proximity to the part to which they correspond.	9.2.3.2	x[A]		x[A]	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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	×[B]			
S2.5	The package implementer should store pieces in their natural order for optimal efficiency.	9.2.5	×[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	The format designer specifies whether that format might use interleaving.	9.2.5		×		
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	×			
O2.3	Package implementers might use the common mapping solutions defined in this Open Packaging specification.	9.2	×			
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			×[A]	
O2.5	The package implementer can define Default media type mappings even though no parts use them.	9.2.3.2	×[A]			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O2.6	The package implementer might create a physical package containing interleaved parts and non-interleaved parts.	9.2.5	×			
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	×[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	×			
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	×			
M3.3	Package implementers shall create item names that are unique within a given archive.	9.3.3	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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	×			
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	×			
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	×			
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	×			
M3.8	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.	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	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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	×			
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	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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	×			
M3.20	The package implementer shall ensure that all 64-bit stream record sizes and offsets have the high-order bit = 0.	Annex B	×			
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	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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	×			

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	×			
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	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.	10.3		×	×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M4.2	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.	10.4		×	×	×
M4.3	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.	10.5			×	×
M4.4	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.	10.5			×	×
M4.5	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 dcterms:W3CDTF, where `dcterms` is the namespace prefix of the Dublin Core namespace. Consumers shall consider a document element that violates this constraint to be an error.	10.5			×	×

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	The format designer shall specify thumbnail parts that are 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	The format designer 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		×	×	

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	×			
M6.2	This part shall exist if the package contains any Digital Signature XML Signature parts,	12.3.2	×			
M6.3	**Error! Reference source not ound.**	12.3.2			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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			×	×
M6.5	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.	12.4.2			×	×
M6.6	Packages shall not contain references to a package-specific Object element that contains a transform other than a canonicalization transform.	12.4.2			×	×
M6.7	The Signature element shall contain only one package-specific Object element.	12.4.2			×	×
M6.8	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.	12.4.2			×	×
M6.9	Reference elements within a Manifest element shall reference with their URI attributes only parts within the package.	12.4.2			×	×
M6.10	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.	12.4.2			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.11	Reference elements shall have query components that specify in a case-sensitive manner the media type of the referenced part.	12.4.2			×	×
M6.12	Reference elements within a Manifest element shall not contain transforms other than the canonicalization transform and relationships transform.	12.4.2			×	×
M6.13	If an optional Relationships transform is used, it shall be followed by a canonicalization transform.	12.4.2			×	×
M6.14	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.	12.4.2			×	×
M6.15	A Signature element shall contain exactly one local-data, package-specific Object element and zero or more application-defined Object elements.	12.4.3			×	×
M6.16	A SignedInfo element shall contain exactly one reference to the package-specific Object element.	12.4.4			×	×
M6.17	RSA-SHA1 algorithms shall be used.	12.4.6			×	×
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.	12.4.7			×	×
M6.19	**Error! Reference source not ound.Error! Reference source not found.**	12.4.8			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.20	Such elements shall contain XML-compliant data.	12.4.11.2			×	×
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.	12.4.12			×	×
M6.22	Such a Manifest element shall not reference any data outside of the package.	12.4.13			×	×
M6.23	**Error! Reference source not ound.**	12.4.16			×	×
M6.24	**Error! Reference source not ound.**	12.4.17			×	×
M6.25	**Error! Reference source not ound.**	12.4.19			×	×
M6.26	**Error! Reference source not ound.**	12.4.19			×	×
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			×	×
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”.	12.7	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
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.	12.8				×
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.	12.8	×			
M6.31	Streaming consumers that maintain signatures shall be able to cache the parts necessary for detecting and processing signatures.	12.8.2				×
M6.32	The Markup Compatibility namespace, as specified in Annex E, shall not be used within the package-specific Object element.	12.9.3	×			
M6.33	If an application allows for a single part to contain information that might not be fully understood by all implementations, then the format designer shall carefully design 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.	12.9.3		×	×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.34	Packages shall use only the following canonicalization methods: XML Canonicalization (c14n) XML Canonicalization with Comments (c14n with comments)	12.4.5			×	×
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			×	×

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			×	
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			×	
S6.3	For digital signatures, **Error! eference source not found.**	12.4.5			×	×
S6.4	**Error! Reference source not found.**	12.4.5			×	×
S6.5	Reference elements within a SignedInfo element should reference an Object element.	12.4.2			×	

Table G–14. Digital signatures optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.1	*[Rule deleted]*	12		×	×	
O6.2	and is optional otherwise.	12.3.2			×	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.4	[M6.3] One or more of these parts may exist in a package.	12.3.3			×	
O6.5	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.	0			×	
O6.6	The part containing the certificate may be signed.	0			×	
O6.7	A Digital Signature Certificate part may be used to create more than one signature.	0			×	
O6.8	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.	12.4.11.2		×	×	
O6.9	Format designers might not apply package-specific restrictions regarding URIs and Transform elements to application-defined Object elements.	12.4.11.2		×	×	
O6.10		12.4.19		×	×	
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	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.12	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.	12.9.3		×	×	×

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! eference source not found.**	×			
M7.2	**Error! Reference source not found.**	**Error! eference source not found.**	×			
M7.3	**Error! Reference source not found.**	**Error! eference source not found.**	×			
M7.4	**Error! Reference source not found.**	**Error! eference 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! eference 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.