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

**Part 2:
Open Packaging Conventions**

*Technologies de l'information — Description des documents et
langages de traitement — Formats de fichier "Office Open XML" —*

Partie 2: Conventions de paquetage ouvert



Reference number
ISO/IEC 29500-2:2012(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2012



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Table of Contents

Foreword	vii
Introduction	viii
1. Scope	1
2. Conformance	2
3. Normative References	3
4. Terms and Definitions	5
5. Notational Conventions	9
5.1 Document Conventions.....	9
5.2 Diagram Notes.....	9
6. Acronyms and Abbreviations	11
7. General Description	12
8. Overview	13
9. Package Model	14
9.1 Parts.....	14
9.1.1 Part Names.....	14
9.1.2 Content Types	17
9.1.3 Growth Hint.....	18
9.1.4 XML Usage.....	18
9.2 Part Addressing	19
9.2.1 Relative References.....	19
9.3 Relationships	20
9.3.1 Relationships Part.....	20
9.3.2 Relationship Markup	20
9.3.3 Representing Relationships.....	24
9.3.4 Support for Versioning and Extensibility.....	26
10. Physical Package	27
10.1 Physical Mapping Guidelines.....	27
10.1.1 Mapped Components.....	28
10.1.2 Mapping Content Types	28
10.1.3 Mapping Part Names to Physical Package Item Names.....	33
10.1.4 Interleaving	35
10.2 Mapping to a ZIP Archive	37
10.2.1 Mapping Part Data	37
10.2.2 ZIP Item Names	37
10.2.3 Mapping Part Names to ZIP Item Names.....	38
10.2.4 Mapping ZIP Item Names to Part Names.....	38
10.2.5 ZIP Package Limitations.....	38
10.2.6 Mapping Part Content Type	39
10.2.7 Mapping the Growth Hint	39

10.2.8	Late Detection of ZIP Items Unfit for Streaming Consumption	40
10.2.9	ZIP Format Clarifications for Packages	40
11.	Core Properties.....	41
11.1	Core Properties Part	42
11.2	Location of Core Properties Part	44
11.3	Support for Versioning and Extensibility	44
11.4	Schema Restrictions for Core Properties	44
12.	Thumbnails.....	46
12.1	Thumbnail Parts.....	46
13.	Digital Signatures.....	47
13.1	Choosing Content to Sign	47
13.2	Digital Signature Parts	47
13.2.1	Digital Signature Origin Part.....	48
13.2.2	Digital Signature XML Signature Part	48
13.2.3	Digital Signature Certificate Part.....	49
13.2.4	Digital Signature Markup	49
13.3	Digital Signature Example.....	59
13.4	Generating Signatures.....	61
13.5	Validating Signatures.....	62
13.5.1	Signature Validation and Streaming Consumption	63
13.6	Support for Versioning and Extensibility.....	63
13.6.1	Using Relationship Types	63
13.6.2	Markup Compatibility Namespace for Package Digital Signatures.....	63
Annex A.	(normative) Resolving Unicode Strings to Part Names	65
A.1	Creating an IRI from a Unicode String	65
A.2	Creating a URI from an IRI	65
A.3	Resolving a Relative Reference to a Part Name	66
A.4	String Conversion Examples	66
Annex B.	(normative) Pack URI	67
B.1	Pack URI Scheme	67
B.2	Resolving a Pack URI to a Resource	69
B.3	Composing a Pack URI	69
B.4	Equivalence	70
Annex C.	(normative) ZIP Appnote.txt Clarifications.....	71
C.1	Archive File Header Consistency	71
C.2	Data Descriptor Signature	71
C.3	Table Key	71
Annex D.	(normative) Schemas - W3C XML Schema	82
D.1	Content Types Stream	82
D.2	Core Properties Part	83
D.3	Digital Signature XML Signature Markup	84
D.4	Relationships Part.....	85
Annex E.	(informative) Schemas - RELAX NG	86

E.1	Content Types Stream	86
E.2	Core Properties Part	87
E.3	Digital Signature XML Signature Markup	87
E.4	Relationships Part.....	88
E.5	Additional Resources.....	89
E.5.1	XML.....	89
E.5.2	XML Digital Signature Core.....	89
Annex F. (normative)	Standard Namespaces and Content Types.....	90
Annex G. (informative)	Physical Model Design Considerations	92
G.1	Access Styles	93
G.1.1	Direct Access Consumption.....	93
G.1.2	Streaming Consumption.....	93
G.1.3	Streaming Creation	93
G.1.4	Simultaneous Creation and Consumption	93
G.2	Layout Styles.....	93
G.2.1	Simple Ordering.....	93
G.2.2	Interleaved Ordering.....	94
G.3	Communication Styles.....	94
G.3.1	Sequential Delivery	94
G.3.2	Random Access.....	94
Annex H. (informative)	Guidelines for Meeting Conformance.....	95
H.1	Package Model	95
H.2	Physical Packages	103
H.3	ZIP Physical Mapping	108
H.4	Core Properties.....	112
H.5	Thumbnail.....	114
H.6	Digital Signatures.....	114
H.7	Pack URI.....	125
Annex I. (informative)	Differences Between ISO/IEC 29500 and ECMA-376:2006	127
I.1	XML Elements.....	127
I.2	XML Attributes.....	127
I.3	XML Enumeration Values	127
I.4	XML Simple Types.....	127
Annex J. (informative)	Index.....	128

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 third edition cancels and replaces the second edition (ISO/IEC 29500-2:2011), which has been technically revised by incorporation of the Technical Corrigendum ISO/IEC 29500-2:2011/Cor.1:2012.

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, I, and J are for information only.

This Part of ISO/IEC 29500 includes two annexes (Annex D and Annex E) 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.

The following organizations have participated in the creation of ISO/IEC 29500 and their contributions are gratefully acknowledged:

Apple, Barclays Capital, BP, The British Library, Essilor, Intel, Microsoft, NextPage, Novell, Statoil, Toshiba, and the United States Library of Congress

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 specifies a set of conventions that are 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 content types and are uniquely identified using the well-defined naming rules provided in this Part of ISO/IEC 29500.

The *physical mapping* 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 H.

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 items 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 Content Types stream in a ZIP archive to a part name. [M3.11] *end example*]

Each Part of this multi-part standard has its own conformance clause, as appropriate. The term *conformance class* is used to disambiguate conformance within different Parts of this multi-part standard. This Part of ISO/IEC 29500 has only one conformance class, *OPC* (that is, Open Packaging Conventions).

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 2616 *Hypertext Transfer Protocol — HTTP/1.1*, The Internet Society, Berners-Lee, T., R. Fielding, H. Frystyk, J. Gettys, P. Leach, L. Masinter, and J. Mogul, 1999, <http://www.ietf.org/rfc/rfc2616.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>.

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 C. *end note*]

4. Terms and Definitions

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

The terms *base URI* and *relative reference* are used in accordance with RFC 3986.

access style — The style in which local access or networked access is conducted. The access styles are as follows: streaming creation, streaming consumption, simultaneous creation and consumption, and direct access consumption.

behavior — External appearance or action.

behavior, implementation-defined — Unspecified behavior where each implementation shall document that behavior, thereby promoting predictability and reproducibility within any given implementation. (This term is sometimes called “application-defined behavior”.)

behavior, unspecified — Behavior where this Open Packaging specification imposes no requirements.

byte — A sequence of 8 bits treated as a unit.

communication style — The style in which package contents are delivered by a producer or received by a consumer. Communication styles include random access and sequential delivery.

consumer — Software or a device that reads packages through a package implementer. A consumer is often designed to consume packages only for a specific physical package format.

content type — Describes the content stored in a part. Content types define a media type, a subtype, and an optional set of parameters, as defined in RFC 2616.

Content Types stream — A specially-named stream that defines mappings from part names to content types. The content types stream is not itself a part, and is not URI addressable.

device — Hardware, such as a personal computer, printer, or scanner, that performs a single function or set of functions.

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

format designer — The author of a particular file format specification built on this Open Packaging Conventions specification.

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

growth hint — A suggested number of bytes to reserve for a part to grow in-place.

id — In some XML-related technologies, the term *id* implies use of the xsd:ID data type. In this international standard, this term is used to refer to a variety of different identification schemes. See *unique identifier*.

interleaved ordering — The layout style of a physical package where parts are broken into pieces and “mixed-in” with pieces from other parts. When delivered, interleaved packages can help improve the performance of the consumer processing the package.

layout style — The style in which the collection of parts in a physical package is laid out: either simple ordering or interleaved ordering.

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

logical item name — An abstraction that 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.

networked access — The 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.

pack URI — A URI scheme that allows URIs to be used as a uniform mechanism for addressing parts within a package. Pack URIs are used as Base URIs for resolving relative references among parts in a package.

package — A logical entity that holds a collection of parts.

package implementer — Software that implements the physical input-output operations to a package according to the requirements and recommendations of this Open Packaging specification. A package implementer is used by a producer or consumer to interact with a physical package. A package implementer can be either a stand-alone API or can be an integrated component of a producer, consumer application, or device.

package model — A package abstraction that holds a collection of parts.

package relationship — A relationship whose target is a part and whose source is the package as a whole. Package relationships are found in the package relationships part named “/_rels/.rels”.

part — A stream of bytes with a MIME content type and associated common properties. Typically corresponds to a file [*Example: on a file system end example*], a stream [*Example: in a compound file end example*], or a resource [*Example: in an HTTP URI end example*].

part name — The path component of a pack URI. Part names are used to refer to a part in the context of a package, typically as part of a URI.

physical model — A description of the capabilities of a particular physical format.

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

piece — A portion of a part. Pieces of different parts can be interleaved together. The individual pieces are named using a unique mapping from the part name. Piece name grammar is not equivalent to the part name grammar. Pieces are not addressable in the package model.

pipe — A communication mechanism that carries data from the producer to the consumer.

producer — Software or a device that writes packages through a package implementer. A producer is often designed to produce packages according to a particular physical package format specification.

random access — A style of communication between the producer and the consumer of the package. Random access allows the consumer to reference and obtain data from anywhere within a package.

relationship — A connection between a source part and a target part in a package. (See also Package Relationships.)

relationship type — An absolute IRI for identifying a relationship.

relationships part — A part containing an XML representation of relationships.

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

signature policy — A 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.

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

simultaneous creation and consumption — A style of access between a producer and a consumer in highly pipelined environments where streaming creation and streaming consumption occur simultaneously.

source part — The part from which a connection is established by a relationship. [*Example:* Picture a SpreadsheetML file with a chart part, “drawing1.xml”, and a sheet part “sheet1.xml”, with the relationship between the chart and the sheet defined in the “sheet1.xml.rels” part as “rld1”. The source part of that relationship is sheet1.xml, because it is inside sheet1.xml that the relationship rld1 is actually referenced. The target part for the relationship is the value of the “Target” attribute of the relationship - drawing1.xml, in this case. *end example*].

stream — A linearly ordered sequence of bytes.

streaming consumption — An 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.

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

target part — The part referenced by the “Target” attribute of a relationship.

thumbnail — A small image that is a graphical representation of a part or the package as a whole.

unique identifier — In some XML-related technologies, the term *unique identifier* implies use of the xsd:ID data type. In this international standard, this term is used to refer to a variety of different identification schemes. See *id*.

XSD — W3C XML Schema

ZIP archive — A ZIP file as defined in the ZIP file format specification. A ZIP archive contains ZIP items.

ZIP item — A ZIP item is an atomic set of data in a ZIP archive that becomes a file when the archive is uncompressed. When a user unzips a ZIP-based package, the user sees an organized set of files and folders.

5. Notational Conventions

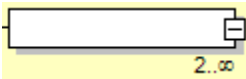
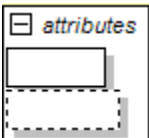
5.1 Document Conventions

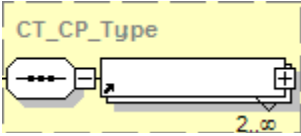
The following typographical conventions are used in ISO/IEC 29500:

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. Acronyms and Abbreviations

This clause is informative.

The following acronyms and abbreviations are used throughout this Part of ISO/IEC 29500:

IEC — the International Electrotechnical Commission

ISO — the International Organization for Standardization

W3C — World Wide Web Consortium

End of informative text.

7. General Description

This Open Packaging specification is divided into the following subdivisions:

1. Front matter (clauses 1–7);
2. Overview (clause 8);
3. Main body (clauses 9–13);
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 1–5, 7, and 9–13
- Annex A–Annex D
- Annex F

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

- Clauses 6 and 8
- Annex E
- Annex G–Annex J
- 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*]

8. Overview

This clause is informative.

This Open Packaging specification describes an abstract model and physical format conventions 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. It is intended to support the content types and organization for various applications and 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.

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.

9. Package Model

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.

9.1 Parts

A *part* is a stream of bytes with the properties listed in Table 9–1. 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 9–1. Part properties

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

9.1.1 Part Names

Each part has a name. *Part names* refer to parts within a package. [Example: The part name “/hello/world/doc.xml” contains three segments: “hello”, “world”, and “doc.xml”. The first two segments in the sample represent levels in the logical hierarchy and serve to organize the parts of the package, whereas the

third contains actual content. Note that segments are not explicitly represented as folders in the package model, and no directory of folders exists in the package model. *end example*]

9.1.1.1 Part Name Syntax

A Part name shall be an IRI and shall be encoded as either a Part IRI or a Part URI. A Part IRI is a physical representation that permits direct use of Unicode characters. A Part URI is a physical representation that uses a percent-encoding for non-ASCII Unicode characters.

[*Note*: Not all versions of the ZIP specification support a Part name represented as a Part IRI. To preserve interoperability, implementers are encouraged to use the currently more prevalent Part URI representation. *end note*]

9.1.1.1.1 Part IRI Syntax

The part IRI grammar is defined as follows:

```
part-IRI = 1*( "/" isegment )
isegment = 1*( ipchar )
```

ipchar is defined in RFC 3987:

```
ipchar      = iunreserved / pct-encoded / sub-delims / ":" / "@"
iunreserved = ALPHA / DIGIT / "-" / "." / "_" / "~" / ucschar
ucschar     = %xA0-D7FF / %xF900-FDCF / %xFDF0-FFEF
              / %x10000-1FFFD / %x20000-2FFFD / %x30000-3FFFD
              / %x40000-4FFFD / %x50000-5FFFD / %x60000-6FFFD
              / %x70000-7FFFD / %x80000-8FFFD / %x90000-9FFFD
              / %xA0000-AFFFD / %xB0000-BFFFD / %xC0000-CFFFD
              / %xD0000-DFFFD / %xE1000-EFFFD
pct-encoded = "%" HEXDIG HEXDIG
sub-delims  = "!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="
```

The part IRI grammar implies the following constraints. The package implementer shall neither create any part that violates these constraints nor retrieve any data from a package as a part if the purported part IRI violates these constraints.

- A part IRI shall not be empty. [M1.1]
- A part IRI shall not have empty isegments. [M1.3]
- A part IRI shall start with a forward slash ("/") character. [M1.4]
- A part IRI shall not have a forward slash as the last character. [M1.5]
- An isegment shall not hold any characters other than ipchar characters. [M1.6]

Part IRI isegments have the following additional constraints. The package implementer shall neither create any part with a part IRI comprised of an isegment that violates these constraints nor retrieve any data from a package as a part if the purported part IRI contains an isegment that violates these constraints.

- An isegment shall not contain percent-encoded forward slash ("/"), or backward slash ("\") characters. [M1.7]
- An isegment shall not contain percent-encoded unreserved characters. [M1.8]
- An isegment shall not end with a dot (".") character. [M1.9]
- An isegment shall include at least one non-dot character. [M1.10]

9.1.1.1.2 Part URI Syntax

The part URI grammar is defined as follows:

```
part-URI  = 1*( "/" segment )
segment   = 1*( pchar )
```

pchar is defined in RFC 3986:

```
pchar      = unreserved / pct-encoded / sub-delims / ":" / "@"
unreserved = ALPHA / DIGIT / "-" / "." / "_" / "~"
pct-encoded = "%" HEXDIG HEXDIG
sub-delims = "!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "="
```

The part URI grammar implies the following constraints. The package implementer shall neither create any part that violates these constraints nor retrieve any data from a package as a part if the purported part URI violates these constraints.

- A part URI shall not be empty. [M1.1] [*Note: The Mx.x notation is discussed in §2. end note*]
- A part URI shall not have empty segments. [M1.3]
- A part URI shall start with a forward slash ("/") character. [M1.4]
- A part URI shall not have a forward slash as the last character. [M1.5]
- A segment shall not hold any characters other than pchar characters. [M1.6]

Part URI segments have the following additional constraints. The package implementer shall neither create any part with a part URI comprised of a segment that violates these constraints nor retrieve any data from a package as a part if the purported part URI contains a segment that violates these constraints.

- A segment shall not contain percent-encoded forward slash ("/"), or backward slash ("\") characters. [M1.7]
- A segment shall not contain percent-encoded unreserved characters. [M1.8]
- A segment shall not end with a dot (".") character. [M1.9]
- A segment shall include at least one non-dot character. [M1.10]

[Example:

Example 9–1. A part name

```
/a/%D1%86.xml
/xml/item1.xml
```

Example 9–2. An invalid part name

```
//xm1/.
```

end example]

9.1.1.2 Part IRI and Part URI Mapping

A Part IRI can be converted to a Part URI by converting ucschar characters to percent-encoded triplets, as defined in Step 2 in §3.1 of RFC 3987.

A Part URI can be converted to a Part IRI by converting percent-encoded triplets to ucschar characters, as defined in §3.2 of RFC 3987.

9.1.1.3 Part Name Equivalence

Part names shall be mapped to either the Part IRI or Part URI form for comparison. Part names represented in different forms cannot be compared.

[*Note:* Equivalence rules for the Part IRI and Part URI forms guarantee uniformity of the comparison result for Part Names converted either to Part IRI or to Part URI form. *end note*]

Packages shall not contain equivalent part names, and package implementers shall neither create nor recognize packages with equivalent part names. [M1.12]

9.1.1.3.1 Part IRI Equivalence

Part IRI equivalence is determined by comparing part IRIs character-by-character:

- pct-encoded and ALPHA characters as case-insensitive ASCII
- ucschar characters as case-sensitive Unicode

9.1.1.3.2 Part Name Equivalence

Part URI equivalence is determined by comparing part URIs as case-insensitive ASCII strings.

9.1.1.4 Part Naming

A package implementer shall neither create nor recognize a part with a part name derived from another part name by appending segments to it. [M1.11] [*Example:* If a package contains a part named “/segment1/segment2/.../segmentn”, then other parts in that package shall not have names such as: “/segment1”, “segment1/segment2”, or “/segment1/segment2/.../segmentn-1”. *end example*]

9.1.2 Content Types

Every part has a *content type*, which identifies the type of content that is stored in the part. Content types define a media type, a subtype, and an optional set of parameters. Package implementers shall only create and only recognize parts with a content type; format designers shall specify a content type for each part included in the format. Content types for package parts shall fit the definition and syntax for media types as specified in RFC 2616, §3.7. [M1.13] This definition is as follows:

`media-type = type "/" subtype *( ";" parameter )`

where parameter is expressed as

`attribute "=" value`

The type, subtype, and parameter attribute names are case-insensitive. Parameter values might be case-sensitive, depending on the semantics of the parameter attribute name.

The value of the content type is permitted to be the empty string.

Content types shall not use linear white space either between the type and subtype or between an attribute and its value. Content types also shall not have leading or trailing white space. Package implementers shall create only such content types and shall require such content types when retrieving a part from a package; format designers shall specify only such content types for inclusion in the format. [M1.14]

The package implementer shall require a content type that does not include comments, and the format designer shall specify such a content type. [M1.15]

Format designers might restrict the usage of parameters for content types. [O1.2]

Content types for package-specific parts are defined in Annex F.

9.1.3 Growth Hint

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

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

9.1.4 XML Usage

All XML content defined in this Open Packaging specification shall conform to the following validation rules:

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]

9.2 Part Addressing

Parts often contain references to other parts. [*Example: A package might contain two parts: an XML markup file and an image. The markup file holds a reference to the image so that when the markup file is processed, the associated image can be identified and located. end example.*]

9.2.1 Relative References

A relative reference is expressed so that the address of the referenced part is determined relative to the part containing the reference.

Relative references from a part are interpreted relative to the base URI of that part. By default, the base URI of a part is derived from the name of the part, as defined in §B.3.

If the format designer permits it, parts can contain Unicode strings representing references to other parts. If allowed by the format designer, format producers can create such parts, and format consumers shall consume them. [O1.4] In particular, XML markup might contain Unicode strings referencing other parts as values of the `xsd:anyURI` data type. Format consumers shall convert these Unicode strings to URIs, as defined in Annex A before resolving them relative to the base URI of the part containing the Unicode string. [M1.23]

Some types of content provide a way to override the default base URI by specifying a different base in the content. [*Example: XML Base or HTML end example*]. In the presence of one of these overrides, format consumers shall use the specified base URI instead of the default. [M1.24]

[*Example:*

Example 9–3. Part names and relative references

A package includes parts with the following names:

- /markup/page.xml
- /images/picture.jpg
- /images/other_picture.jpg

If /markup/page.xml contains a reference to ../images/picture.jpg, then this reference is interpreted as referring to the part name /images/picture.jpg.

end example]

9.3 Relationships

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 content type of the part; that is, in arbitrary markup or an application-defined encoding. This effectively hides the internal and external links between parts from consumers that do not understand the content 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 provide 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*]

9.3.1 Relationships Part

Each set of relationships sharing a common source is represented by XML stored in a *Relationships part*. The Relationships part is URI-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 content type of the Relationships part is defined in Annex F.

9.3.2 Relationship Markup

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

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]

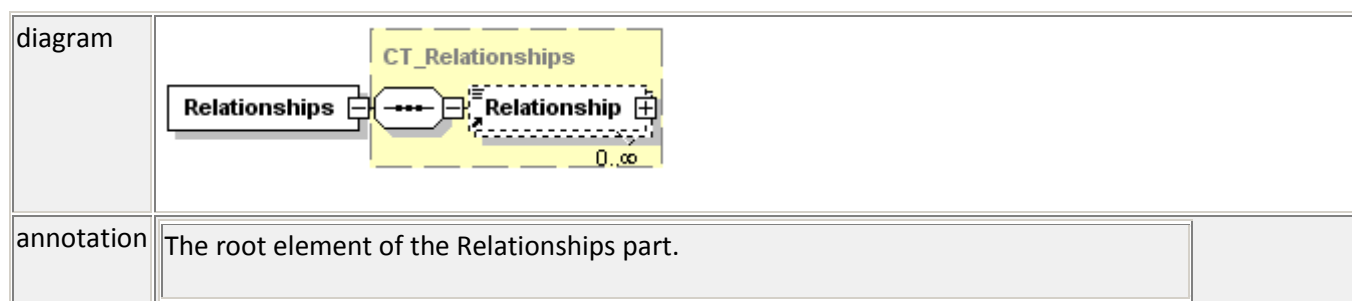
The nature of a Relationship element is identified by the Type attribute. The value of this attribute shall be a relationship type. By using types patterned after the Internet domain-name space, non-coordinating parties can safely create non-conflicting relationship types.

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.

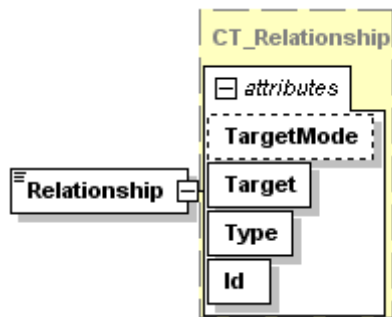
9.3.2.1 Relationships Element

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



9.3.2.2 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 Annex B. 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. <i>end note</i>] 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]

9.3.3 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 F.

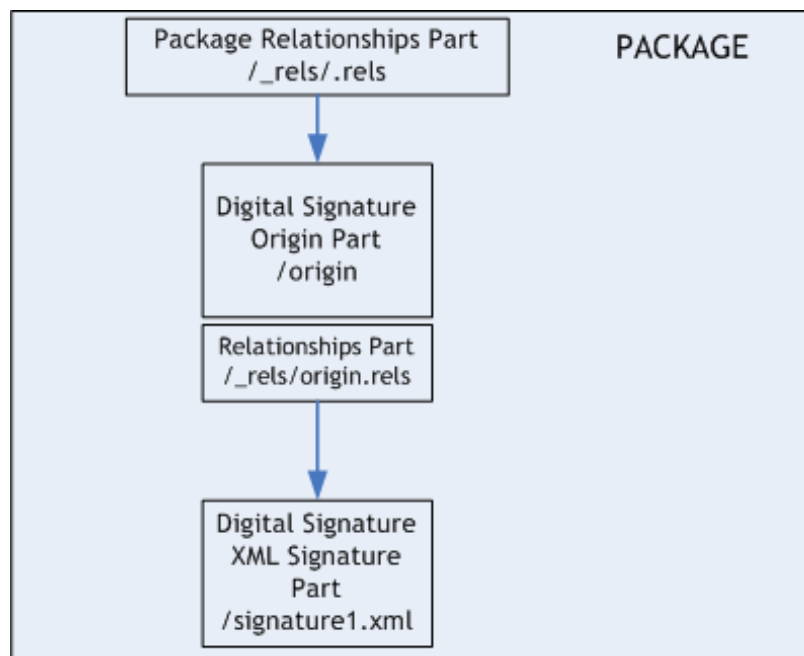
A special naming convention is used for the Relationships part. First, the Relationships part for a part in a given folder in the name hierarchy is stored in a sub-folder called “_rels”. Second, the name of the Relationships part is formed by appending “.rels” to the name of the original part. Package 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 content type for a Relationships part. [M1.30]

[Example:

Example 9–4. 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 9–5. 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 9–6. 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]

9.3.4 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]

10. Physical Package

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.

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*, or 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*, or how part contents are physically stored within the package. The layout style can either be *simple ordering*, where the parts are arranged contiguously as atomic blocks 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 G 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 content 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.

10.1 Physical Mapping Guidelines

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 content 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]

10.1.1 Mapped Components

The package implementer shall define a physical package format with a mapping for the required components package, part name, part content type and part contents. [M2.2] [*Note: Not all physical package formats support the part growth hint. end note*]

Table 10–1. Mapped components

Name	Description	Required/Optional
Package	URI-addressable resource that identifies 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 content type	Identifies the kind of content stored in the part	Required. The package implementer shall provide a physical mapping for each part's content 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 part	Optional. The package implementer might provide a physical mapping for a growth hint that might be specified by a producer. [O2.2]

10.1.2 Mapping Content Types

Methods for mapping part content types to a physical format are described below.

10.1.2.1 Identifying the Part Content Type

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

Some physical package formats have a native mechanism for representing content types. [*Example: the content type header in MIME. end example*] For such packages, the package implementer should use the native mechanism to map the content type for a part. [S2.1]

For all other physical package formats, the package implementer should include a specially-named XML stream in the package called the *Content Types stream*. [S2.2] The Content 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.

10.1.2.2 Content Types Stream Markup

The Content Types stream identifies the content type for each package part. The Content 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 content types. Override elements are used to specify content 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 (§9.3.1) and the Content Types part itself, the Content 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 Content Types stream is not significant.

If the package is intended for streaming consumption:

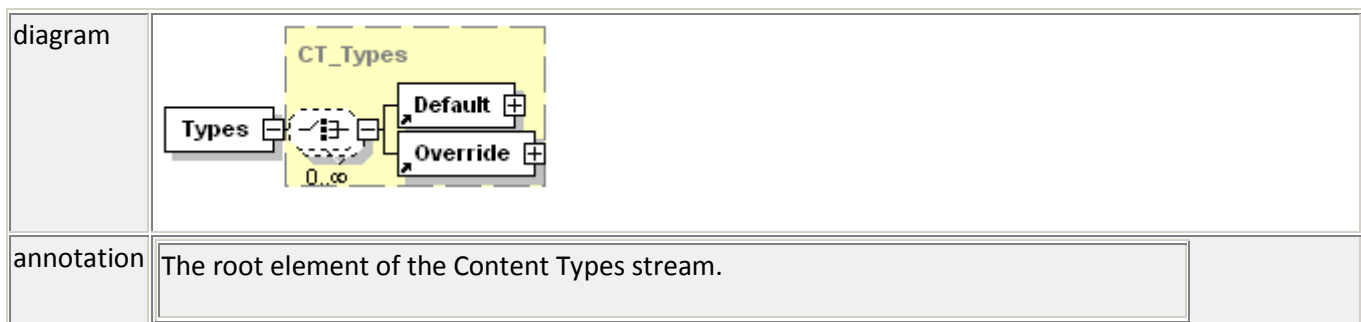
- The package implementer should not allow Default elements; as a consequence, there should be one Override element for each part in the package.
- The format producer should write the Override elements to the package so they appear before the parts to which they correspond, or in close proximity to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

[S2.3]

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

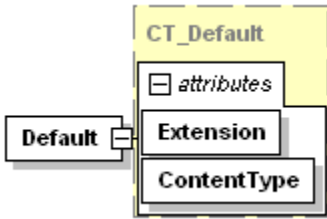
10.1.2.2.1 Types Element

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



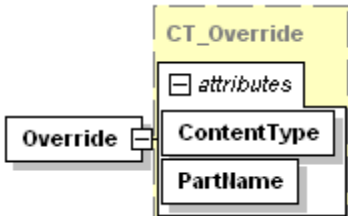
10.1.2.2.2 Default Element

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 content type as defined in RFC 2616. Indicates the content type of any matching parts (unless overridden). The package implementer shall require a content type in a Default element and the format designer shall specify the content type. [M2.6]
annotation	Defines default mappings from the extensions of part names to content types.					

10.1.2.2.3 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 content type as defined in RFC 2616. Indicates the content type of the matching part. The package implementer shall require a content type and the format designer shall specify the content type in an Override element. [M2.7]
	PartName	xs:anyURI	required			A part name (§9.1.1). 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 content types on parts that are not covered by, or are not consistent with, the default mappings.					

10.1.2.2.4 Content Types Stream Markup Example

[Example:

Example 10–7. Content 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 not a container for generic types, but specifically for content types to be used within the package.

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

Part name	Content type
/a/b/sample1.txt	text/plain
/a/b/sample2.jpg	image/jpeg

Part name	Content type
/a/b/sample3.picture	image/gif
/a/b/sample4.picture	image/jpeg

end example]

10.1.2.3 Setting the Content Type of a Part

When adding a new part to a package, the package implementer shall ensure that a content type for that part is specified in the Content 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 Content Types stream.
3. Compare the resulting extension with the values specified for the Extension attributes of the Default elements in the Content Types stream. The comparison shall be case-insensitive ASCII.
4. If there is a Default element with a matching Extension attribute, then the content 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 2616, or it might follow the grammar of RFC 2616.
 - a. If the content types match, no further action is required.
 - b. If the content types do not match, a new Override element shall be added to the Content 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 Content Types stream.

10.1.2.4 Getting the Content Type of a Part

To get the content 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 Content 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.

10.1.2.5 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 Content Types stream markup. [M2.10]

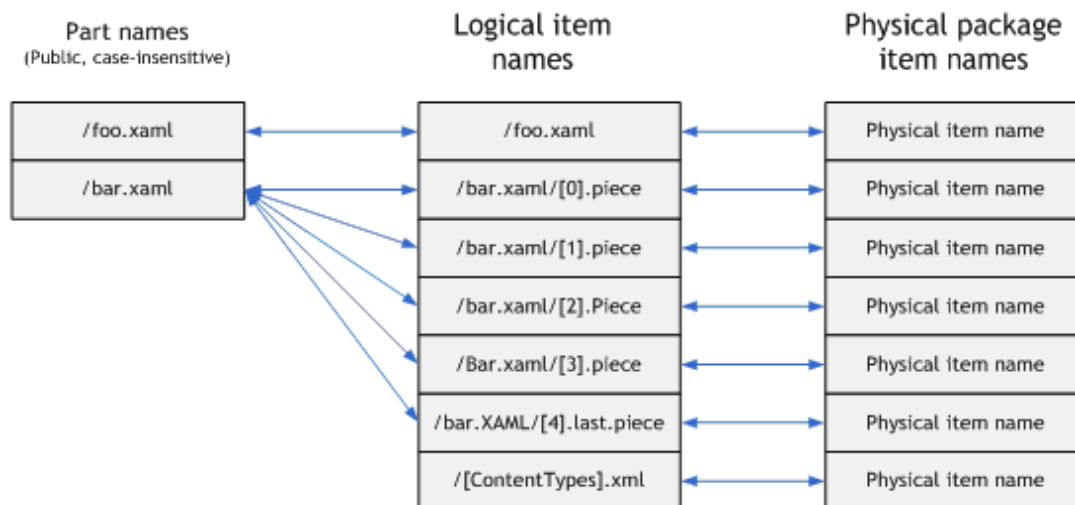
10.1.3 Mapping Part Names to Physical Package Item Names

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 §10.1.4 for interleaving details.

[Example:

Figure 10–1 illustrates the relationship between part names, logical item names, and physical package item names.

Figure 10–1. Part names and logical item names



end example]

10.1.3.1 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.

10.1.3.2 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 (§9.1.1). 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*]

10.1.3.3 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.

10.1.3.4 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.