Ecma/TC45/2021/004
Ecma/GA/2021/120

ECMA-376, 5th Edition

Office Open XML File Formats — Open Packaging Conventions

December 2021

Contents

Foreword

This edition cancels and replaces the previous edition (ECMA-376-2:2012), which has been technically revised.

The main changes compared to the previous edition are as follows:

- Where appropriate, normative references have been updated to use undated or more recent versions of other standards.
- Clause 3 (Terms and definitions) has been revised by removing terms not used by any normative clauses and then reorganizing terms into groups.
- The subclause for diagram notes (5.1 in the preceding editions) has been removed, since core properties are now defined by prose and schemas rather than by diagrams.
- The clause for acronyms and abbreviations (Clause 6 in the preceding editions) has been removed, since it does not make sense for an ISO/IEC standard to define "ISO" and "IEC".
- Clause 6 (Abstract package model, Clause 8 in the previous edition) has been completely rewritten. In particular, (1) pack IRIs have been defined in this clause rather than in an annex, (2) a new subclause, "Resolving relative references", has been added; (3) part Relationships parts and package Relationships parts have been distinguished; (4) base IRIs have been clearly defined; and (5) handling of non-ASCII characters in part names has been clarified on the basis of RFC 3987.
- The option for media type to be an empty string has been removed, as this conflicts with the definition of media type in RFC 2046 and the existing regular expression defined in the schema referenced by C.2.
- Clause 7 (Physical package model, Clause 9 in the previous edition) has been slightly revised. Interleaving has been introduced before logical item names. Percent-encoding and un-percent encoding of non-ASCII characters have been explicitly introduced in 7.3.4 and 7.3.5.
- Clause 8 (Core properties, Clause 10 in the previous edition) has been rewritten by using prose and schemas rather than diagrams.
- Clause 10 (Digital signatures, Clause 12 in the previous edition) has been thoroughly revised. In particular, this clause now makes clear a convention for the choice of algorithms for signature and digest methods, which reflects the ongoing development of algorithms since the first edition of this document.
- Annex A has been made informative.
- The normative annex that defined pack IRIs (Annex B in the preceding editions) has been dropped. Pack IRIs are now defined in Clause 6.
- Annex C and Annex D (Annexes D and E in the preceding editions) no longer define schemas but reference externally defined schemas.
- Guidelines for meeting conformance requirements (Annex H in the preceding editions) have been dropped.
- Requirements around streaming consumption have been dropped.
- Wherever possible, requirements on programs have been rewritten as those on data.

- Annex H has been added to depict an example package.
- The Index (Annex J in the preceding editions) has been deleted.
- Bibliography has been added.

Introduction

ECMA-376 (all Parts) 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[1] documents.

This document includes two annexes (Annex C and Annex D) that refer to data files provided in electronic form.

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

This fifth edition preserves all previous functionality and adds no new functionality.

[1] This information is given for the convenience of users of this document and does not constitute an endorsement by Ecma of the product named. Equivalent products may be used if they can be shown to lead to the same results.

1 Scope

This document defines a set of conventions for packaging one or more interrelated byte streams (parts) as a single resource (package). These conventions are applicable not only to Office Open XML specifications as described in ECMA-376-1 and ECMA-376-4, but also to other markup specifications.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

ANSI/INCITS 4-1986 [R2017] - *Information Systems - Coded Character Sets - 7-Bit American National Standard Code For Information Interchange (7-Bit ASCII)*, American National Standards Institute (ANSI). 2017

FIPS 186-4, *Digital Signature Standard (DSS)*, National Institute of Standards and Technology, US Department of Commerce, July 2013

ECMA-376-3, *Information technology — Document description and processing languages — Office Open XML File Formats, Part 3: Markup Compatibility and Extensibility*

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

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

ISO 15836-2, *Information and documentation — The Dublin Core metadata element set — Part 2: DCMI Properties and classes*

RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, The Internet Society, November 1996, N. Freed and N. Borenstein. Available at https://www.rfc-editor.org/info/rfc2046

RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, The Internet Society, January 2005, Berners-Lee, T., R. Fielding, and L. Masinter. Available at https://www.rfc-editor.org/info/rfc3986

RFC 3987, *Internationalized Resource Identifiers (IRIs)*, The Internet Society, January 2005, Duerst, M. and M. Suignard. Available at https://www.rfc-editor.org/info/rfc3987

RFC 5234, *Augmented BNF for Syntax Specifications: ABNF*, The Internet Society, January 2008, D. Crocker and P.Overell, (editors). Available at https://www.rfc-editor.org/info/rfc5234

RFC 6931, *Additional XML Security Uniform Resource Identifiers (URIs)*, The Internet Society, April 2013, D. Eastlake 3rd. Available at https://www.rfc-editor.org/info/rfc6931

RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, The Internet Society, June 2014, R. Fielding and J. Reschke. Available at https://www.rfc-editor.org/info/rfc7231

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

The XML 1.0 specification, *Extensible Markup Language (XML) 1.0, Fourth Edition*. World Wide Web Consortium, 2006, Tim Bray, Jean Paoli, Eve Maler, C. M. Sperberg-McQueen, and François Yergeau (editors). Available at http://www.w3.org/TR/2006/REC-xml-20060816/[2]

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

XML Base, *XML Base (Second Edition)*, World Wide Web Consortium , 28 January 2009. Jonathan Marsh and Richard Tobin (editors). Available at https://www.w3.org/TR/2009/REC-xmlbase-20090128/

W3C XML Schema Structures, *XML Schema Part 1: Structures (Second Edition)*, World Wide Web Consortium, 28 October 2004, Henry Thompson, David Beech, Murray Maloney and Noah Mendelsohn (editors). Available at https://www.w3.org/TR/xmlschema-1/

W3C XML Schema Datatypes, *XML Schema Part 2: Datatypes (Second Edition)*, World Wide Web Consortium, 28 October 2004, Paul Biron and Ashok Malhotra (editors). Available at https://www.w3.org/TR/xmlschema-2/

XML-Signature Syntax and Processing, World Wide Web Consortium, 12 February 2002, Donald Eastlake, Joseph Reagle and David Solo (editors). Available at http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

ZIP Appnote, *ZIP File Format Specification* Version 6.2.0, PKWARE Inc., 2004. Available at http://www.pkware.com/documents/APPNOTE/APPNOTE_6.2.0.txt

[2]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 document also makes normative reference, and which also depend upon XML, such as XML Namespaces and XML Base, are all aligned with the 5th Edition.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1 Basics

3.1.1
byte
sequence of 8 bits treated as a unit

3.1.2
stream
linearly ordered sequence of *bytes* (3.1.1)

3.2 Abstract package model

3.2.1
part
stream (3.1.2) with a name, a MIME media type and associated common properties

3.2.2
abstract package
logical entity that holds a collection of *parts* (3.2.1) and *relationships* (3.2.3)

3.2.3
relationship
package relationship (3.2.4) or *part relationship* (3.2.5)

3.2.4
package relationship
connection from a package to a specific *part* (3.2.1) in the same package, or to an external resource

3.2.5
part relationship
connection from a *part* (3.2.1) in a package to another part in the same package, or to an external resource

3.2.6
source
part (3.2.1) or package from which a connection is established by a *relationship* (3.2.3)

3.2.7
target
part (3.2.1) or external resource to which a connection is established by a *relationship* (3.2.3)

3.2.8
relationship type
absolute IRI for specifying the role of a *relationship* (3.2.3)

3.2.9
Relationships part
part (3.2.1) containing an XML representation of *relationships* (3.2.3)

3.2.10
abstract package model
abstract model that defines *abstract packages* (3.2.2)

3.2.11
growth hint
suggested number of *bytes* (3.1.1) to reserve for a *part* (3.2.1) to grow in place

3.2.12
pack scheme
URI scheme that allows IRIs to be used as a uniform mechanism for addressing *parts* (3.2.1) within a package

3.2.13
pack IRI
IRI that conforms to the *pack scheme* (3.2.12)

3.2.14
part name
string that uniquely identifies a *part* (3.2.1) within a package

3.2.15
relationship identifier
string that consists of XML name characters and uniquely identifies a *relationship* (3.2.3) among those from the same *source* (3.2.6)

3.2.16
target mode
mode of resolution of relative references to *targets* (3.2.7)

3.2.17
I18N segment
Unicode string in a *part name* (3.2.14)

Note 1 to entry: The constraints on the value of the Unicode string shall be stated when the term is used in 6.2.2.2.

3.3 Physical package model

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

3.3.2
physical package
result of mapping an *abstract package* (3.2.2) to a *physical format* (3.3.1)

3.3.3
physical package model
pair of a *physical format* (3.3.1) and a mapping between the *abstract package model* (3.2.10) and that physical format

3.3.4
piece
portion of a *part* (3.2.1)

3.3.5
logical item
non-interleaved *part* (3.2.1), non-interleaved *Media Types stream* (3.3.12), *piece* (3.3.4) of an interleaved part, or piece of an interleaved Media Types stream

3.3.6
physical package item
atomic set of data in a *physical package* (3.3.2)

3.3.7
ZIP item
atomic set of data in a *ZIP file* (3.3.8) that becomes a file when the archive is uncompressed

3.3.8
ZIP file
file as defined in the ZIP Appnote

3.3.9
simple ordering
defined ordering for laying out the *parts* (3.2.1) in a package in which all the bits comprising each part are stored contiguously

3.3.10
interleaved ordering
defined ordering for laying out the *parts* (3.2.1) in a package in which parts are broken into *pieces* (3.3.4) and “mixed-in” with pieces from other parts

3.3.11
ASCII case-insensitive matching
comparing a sequence of code points as if all ASCII code points in the range 0x41 to 0x5A (A to Z) were mapped to the corresponding code points in the range 0x61 to 0x7A (a to z)

Note 1 to entry: The ASCII code points shall be as defined by ANSI/INCITS 4-1986.

3.3.12
Media Types stream
stream (3.1.2) in a *physical package* (3.3.2) representing an XML document that specifies the media type of each *part* (3.2.1) in the package

3.4 Digital signature and thumbnail

3.4.1
signature policy
specification of what *parts* (3.2.1) and *relationships* (3.2.3) are included in a signature and what additional behaviors are required for generating and validating signatures

3.4.2
thumbnail
small image that is a graphical representation of a *part* (3.2.1) or the package as a whole

3.5 Implementations

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

3.6 Core properties

3.6.1
core property
property of a package

4 Conformance

A package is of conformance class OPC if it obeys all syntactic constraints specified in this document.

OPC conformance is purely syntactic.

5 Overview

This document describes an abstract package model (Clause 6) and a physical package model (Clause 7) for the use of XML, Unicode, ISO/IEC 10646 (see Reference [7]), ZIP, and other relevant technologies and specifications to organize the content and resources of a document within a package. The package structure is intended to support the organization of constituent resources for various applications and categories of content. An example package is shown in Annex H.

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

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

This document also describes other features, including core properties for package metadata, a thumbnail for graphical representation of a package, and digital signatures of package contents. This document relies on ECMA-376-3 to allow future extensions of OPC without introducing compatibility problems.

This document specifies requirements for packages. Conformance requirements are identified throughout this document. A formal conformance statement is given in Clause 4.

6 Abstract package model

6.1 General

This clause introduces abstract packages in terms of parts (6.2) and relationships (6.5). It also introduces the pack scheme (6.3.2).

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

An example abstract package is shown in H.2.

6.2 Parts

6.2.1 General

A part is analogous to a file in a file system or to a resource on an HTTP server.

6.2.2 Part names

6.2.2.1 General

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

6.2.2.2 Syntax

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

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

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

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

The part name "`/_rels/.rels`" shall be reserved (6.5.2.2). Part names in which the second-to-last I18N segment is equivalent to "`_rels`" and the final segment is equivalent to any string ending with "`.rels`" shall be reserved (6.5.2.3).

EXAMPLE 1 The part name "`/hello/world/doc.xml`" contains three path segments, namely, "`hello`", "`world`", and "`doc.xml`".

EXAMPLE 2 The part name "`/é`" contains a path segment "`é`" where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9).

NOTE Path segments are not explicitly represented as folders in the abstract package model, and no directory of folders exists in the abstract package model.

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

6.2.2.3 Part name equivalence and integrity in an abstract package

Equivalence of part names shall be determined by ASCII case-insensitive matching. Such matching compares a sequence of code points as if all ASCII code points in the range 0x41–0x5A (A–Z) were mapped to the corresponding code points in the range 0x61–0x7A (a–z). See Reference [1].

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

EXAMPLE 1 If an abstract package contains a part named "`/a`", the name of another part in that abstract package cannot be "`/a`" or "`/A`".

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

EXAMPLE 2 "`/a/b`" is derivable from "`/a`", where N is "`/a`" and S is "`b`".

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

EXAMPLE 3 Suppose that an abstract package contains a part named "`/segment1/segment2/…/segmentn`". For it not to be derivable, other parts in that abstract package cannot have names such as "`/segment1`", "`/SEGMENT1`", "`/segment1/segment2`", "`/segment1/SEGMENT2`", or "`/segment1/segment2/…/segmentn-1`".

This subclause further introduces recommendations, so that Unicode Normalization Form C (NFC) and Unicode Normalization Form D (NFD) of part names do not cause part-name collisions. Note that some implementations of directory structures always apply NFD normalization.

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

If an abstract package contains a part named "`/é`", where é is 'LATIN SMALL LETTER E' (U+0065) followed by 'COMBINING ACUTE ACCENT' (U+0301), the name of another part in that abstract package should not

be "/é", where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9), or "/É", where É is 'LATIN CAPITAL LETTER E WITH ACUTE '(U+00C9).

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

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

EXAMPLE 4 Consider a part name "/é", where é is 'LATIN SMALL LETTER E WITH ACUTE' (U+00E9). Another part name "/é/a", where é is 'LATIN SMALL LETTER E' (U+0065) followed by 'COMBINING ACUTE ACCENT' (U+0301) is weakly derivable from "/é". Another part name "/É/a", where É is 'LATIN CAPITAL LETTER E' (U+0045) followed by 'COMBINING ACUTE ACCENT' (U+0301) is also weakly derivable.

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

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

6.2.3 Media types

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

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

6.2.4 Growth hint

A part may have a growth hint.

Sometimes a part in a physical package is modified and needs to become larger. For some physical formats, creating a new physical package that contains the larger part is an expensive operation. To allow the part to grow in place, moving as few bytes as possible, the growth hint may be used to reserve space in a mapping to a particular physical format.

6.2.5 XML usage

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

a) 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.
b) The XML 1.0 specification allows for the usage of Document Type Definitions (DTDs), which enable Denial of Service attacks, typically through the use of an internal entity expansion technique. As mitigation for this potential threat, DTD declarations shall not be used in the XML markup defined in this document.
c) XML documents shall conform to XML Namespaces.
d) XML content shall be schema-valid, as defined by W3C XML Schema Structures and W3C XML Schema Datatypes, with respect to the corresponding XSD schema defined in Annex C of this document. In particular, the XML content shall not contain elements or attributes drawn from namespaces that are not explicitly defined in the corresponding XSD schema unless the XSD schema allows elements or attributes drawn from any namespace to be present in particular locations in the XML markup.
e) XML content shall not contain elements or attributes drawn from “xml” or “xsi” namespaces unless they are explicitly defined in the XSD schema or by other means described in this document.

6.3 Part addressing

6.3.1 General

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

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

EXAMPLE An example of an IRI in the pack scheme is:

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

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

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

6.3.2 Pack scheme

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

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

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

```
iunreserved = <iunreserved, see [RFC3987], Section 2.2>
sub-delims = <sub-delims, see [RFC3986], Section 2.2>
pct-encoded = <pct-encoded, see [RFC3986], Section 2.1>
```

The authority component (`iauthority`) contains an embedded IRI that points to a package. (See 6.3.4 for the procedure for transforming the IRI for the package to permit embedding in the pack IRI as the authority component.) The authority component shall not reference a package embedded in another package.

NOTE The definition of the authority component requires that the colon character (:) be escaped as %3c. However, in the proposed registration of the pack scheme, an unescaped colon (:) character was mistakenly used. Due to this mistake, the provisional pack scheme was registered by IANA as a historical scheme. The pack scheme can be inspected in the IANA-maintained registry of schemes (see Reference [8]).

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

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

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

EXAMPLE 1 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"
```

EXAMPLE 2 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/"
```

EXAMPLE 3 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.containe
r/c/d/bar.xml"
```

6.3.3 Resolving a pack IRI to a resource

The following algorithm shall be used to resolve a pack IRI to a resource (either a package or a part):

a) Parse the pack IRI into the potential three components: scheme, authority, path, as well as any fragment identifier.

b) In the authority component, replace all commas (“,”) with forward slashes (“/”).
c) Un-percent-encode ASCII characters in the resulting authority component.
d) The resultant authority component shall be a valid IRI for the package as a whole. If it is not, the pack IRI is invalid.
e) If the path component is empty, the pack IRI resolves to the package as a whole and the resolution process is complete.
f) A non-empty path component shall be a valid part name. If it is not, the pack IRI is invalid.
g) The pack IRI resolves to the part with this part name in the package identified by the authority component.

EXAMPLE Resolving a pack IRI to a resource

Given the pack IRI:

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

The components:

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

are converted to the package IRI:

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

and the path:

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

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

6.3.4 Composing a pack IRI

The following algorithm shall be used to compose a pack IRI from the absolute IRI of an entire package and a part name:

a) Remove the fragment identifier from the absolute package IRI, if present.
b) Percent-encode all percent signs (“%”), question marks (“?”), at signs (“@”), colons (“:”) and commas (“,”) in the package IRI.
c) Replace all forward slashes (“/”) with commas (“,”) in the resulting string.
d) Append the resulting string to the string "`pack://`".
e) Append a forward slash (“/”) to the resulting string. The constructed string represents a pack IRI with a blank path component.
f) 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.

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"
```

6.3.5 Equivalence

Two pack IRIs shall be treated as equivalent if:

a) The scheme components are octet-by-octet identical after they are both converted to lowercase; and
b) The IRIs, decoded as described in 6.3.3 from the authority components, are equivalent (the equivalency rules by scheme), as specified in RFC 3986; and
c) The path components are equivalent part names, as specified in 6.2.2.

NOTE In some scenarios, such as caching or writing parts to a package, it is necessary to determine if two pack IRIs are equivalent without resolving them.

6.4 Resolving relative references

6.4.1 General

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

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

6.4.2 Base IRIs

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

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

The base IRI depends on where that reference occurs within the package. This subclause covers the case where a relative reference occurs in a part that is not a Relationships part. 6.5.2 covers the case where a relative reference occurs in a Relationships part.

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

EXAMPLE

Consider a part /a/b/foo.xml in a package located at

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

The base IRI is

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

6.4.3 Examples

6.4.3.1 General

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

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

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

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

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

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

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

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

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

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

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

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

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

6.4.3.3 No leading slash: "bar.xml"

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

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

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

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

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

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

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

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

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

6.4.3.4 Dot segment: "./bar.xml"

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

As in 6.4.3.3, the merge routine removes "foo.xml" from the path component of the base IRI, and emits "/a/b/./bar.xml". But the remove_dot_segments routine in 5.2.4 of RFC 3986 removes "./" and emits "/a/b/bar.xml". Thus, the resulting IRI is:

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

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

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

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

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

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

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

6.4.3.5 Dot segment: "../bar.xml"

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

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

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

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

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

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

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

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

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

6.5 Relationships

6.5.1 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 applications that do not understand the media types of the parts containing such references.

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

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

Relationships have relationship identifiers. These identifiers allow relationships to be distinguished from one another. An identifier can also be used to associate the target of a relationship with a specific point in a source part (for example, to represent a hyperlink), by embedding the relationship identifier at that point.

A relationship has a relationship type.

Relationships are represented in XML in Relationships parts. If the package itself or any part in the package is the source of one or more relationships, there is an associated Relationships part. This part holds the list of relationships for the source. The Relationships namespace and relationship types for parts defined in this document are listed in Annex E.

Relationships have a second important function: providing additional information about parts without modifying their content. Note that 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.

6.5.2 Relationships part

6.5.2.1 Relationships part

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

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

A Relationships part shall be either a package Relationships part (6.5.2.2) or a part Relationships part (6.5.2.3).

6.5.2.2 Package Relationships part

A package Relationships part shall be a Relationships part containing package relationships and no other relationships.

The name of a package Relationships part shall be "`/_rels/.rels`".

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

EXAMPLE Consider the package Relationships part for a package located at "`http://www.mysite.com/my.package`".

If the target mode is external, the base IRI is

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

If the target mode is internal, the base IRI is

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

6.5.2.3 Part Relationships part

A part Relationships part shall be a Relationships part containing part relationships from the same source part and no other relationships.

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

EXAMPLE 1 If the source part name is "`/foo`", the part Relationships part name is "`/_rels/foo.rels`". Conversely, if the name of a part is "`/_rels/foo.rels`", it is a part Relationships part for the source part "`/foo`". If the source part name is "`/foo/bar.xml`", the part Relationships part name is "`/foo/_rels/bar.xml.rels`". Conversely, if the name of a part is "`/foo/_rels/bar.xml.rels`", it is a part Relationships part for the source part "`/foo/bar.xml`".

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

EXAMPLE 2 Consider a part Relationships part "`/a/b/_rels/foo.xml.rels`" in a package located at

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

If the target mode is external, the base IRI is

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

If the target mode is internal, the base IRI is

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

6.5.3 Relationship markup

6.5.3.1 General

The content of a Relationships part shall be an XML document. After the removal of any extensions by an MCE processor as specified in ECMA-376-3, a Relationships part shall be a schema-valid XML document against opc-relationships.xsd (C.5). For this MCE processing, the markup configuration shall be empty and the application configuration shall contain the Relationships namespace only.

The output document resulting from any MCE processing of the Relationships part shall not contain an `xml:base` attribute, as specified by XML Base.

6.5.3.2 Support for versioning and extensibility

Relationships parts may use the versioning and extensibility mechanisms defined in ECMA-376-3 to incorporate elements and attributes drawn from other XML namespaces.

6.5.3.3 `Relationships` element

A `Relationships` element is the root element of a Relationships part. It is the container for zero or more `Relationship` elements. It has no attributes. The W3C XML Schema definition of this element's content model is the complex type `CT_Relationships`, which is defined in the schema opc-relationships.xsd (C.5).

6.5.3.4 `Relationship` element

A `Relationship` element shall represent a relationship. The source of a relationship shall be either a package or part with which the Relationships part containing this `Relationship` element is associated.

Attributes	Description
`TargetMode`	This attribute specifies the target mode of a relationship. This attribute is optional, and the default value is `Internal`. The possible values for this attribute are `Internal` and `External`, as defined by the simple type `ST_TargetMode`, which is defined in the schema opc-relationships.xsd (C.5).
`Target`	This attribute specifies the target of a relationship. This attribute is required. If the value of the `TargetMode` attribute is `Internal`, the `Target` attribute shall be a relative reference to a part. If the value of the `TargetMode` attribute is `External`, the `Target` attribute shall be a relative reference or an absolute IRI. Base IRIs for resolving relative references are defined in 6.4. The range of values for this attribute shall be as defined by the `xsd:anyURI` simple type of W3C XML Schema Datatypes.
`Type`	This attribute specifies the relationship type of a relationship. This attribute is required. Relationship types can be compared to determine whether two `Relationship` elements are of the same type. This comparison is conducted in the same way as when comparing URIs that identify XML namespaces: the two URIs are treated as strings and considered identical if and only if the strings have the same sequence of characters. The comparison is case-sensitive, and no escaping is done or undone. EXAMPLE 1 `Type="http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/signature"` The range of values for this attribute shall be as defined by the `xsd:anyURI` simple type of W3C XML Schema Datatypes.

Attributes	Description
`Id`	This attribute specifies the identifier of a relationship. The value of the `Id` attribute shall be unique within the Relationships part. This attribute is required. EXAMPLE 2 `Id="A5FFC797514BC"` The range of values for this attribute shall be as defined by the `xsd:ID` simple type of W3C XML Schema Datatypes.

The W3C XML Schema definition of this element's content model is the complex type `CT_Relationship`, which is defined in the schema opc-relationships.xsd (C.5).

6.5.4 Examples

6.5.4.1 Relationships part associated with the entire package

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

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

```
<Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
   <Relationship
      Target="a.xml"
      Id="IDI1"
      Type="http://example.com/relTypeInt1"/>
   <Relationship
      Target="a.xml"
      TargetMode="External"
      Id="IDE1"
      Type="http://example.com/relTypeExt1"/>
</Relationships>
```

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

The first relationship:

- The target mode is Internal (default). Thus, the base IRI for resolving "`a.xml`" is the pack IRI ("`pack://http%3c,,www.example.com,ex.opc`") created from the IRI of the package ("`http://www.example.com/ex.opc`").

- The result of resolving "a.xml" is "pack://http%3c,,www.example.com,ex.opc/a.xml". The target of this relationship is thus the part "/a.xml" in this package.
- The relationship type of this relationship is "http://example.com/relTypeInt1".
- The identifier of this relationship is "IDI1".

The second relationship:

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

6.5.4.2 Relationships part associated with a part

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

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

```
<Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
   <Relationship
      Target="b.xml"
      Id="IDI2"
      Type="http://example.com/relTypeInt2"/>
   <Relationship
      Target="b.xml"
      TargetMode="External"
      Id="IDE2"
      Type="http://example.com/relTypeExt2"/>
</Relationships>
```

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

The first relationship:

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

- The relationship type of this relationship is "`http://example.com/relTypeInt2`".
- The identifier of this relationship is "IDI2".

The second relationship:

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

6.5.4.3 Relationships parts related to digital signature markup

The Digital Signature Origin part (10.4.2) is targeted by a package relationship, which is stored in the package Relationships part, "`/_rels/.rels`".

EXAMPLE 1

A Relationship element representing the package relationship to the Digital Signature Origin part:

```
<Relationship Id="rId4"
  Type="http://schemas.openxmlformats.org/package/2006/relationships/
  digital-signature/origin"
  Target="_xmlsignatures/origin.sigs"/>
```

The connection from the Digital Signature Origin to the Digital Signature XML Signature part is represented by a part relationship, which is stored in a part Relationships part, "`/_xmlsignatures/_rels/origin.sigs.rels`".

EXAMPLE 2

An XML document representing the content of "`/_xmlsignatures/_rels/origin.sigs.rels`":

```
<Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
   <Relationship
      Target="sig1.xml"
      Id="rId1"
      Type="http://schemas.openxmlformats.org/package/2006/relationships/
         digital-signature/signature"/>
</Relationships>
```

6.5.4.4 Relationships targeting external resources

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

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

6.5.4.5 Multiple relationships that have the same target

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

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

7 Physical package model

7.1 General

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

This clause further specifies general guidelines and common mechanisms for physical package models and defines a ZIP-based physical package model. The interleaving mechanism (see 7.2.4) is such a common mechanism.

An example physical package is described in H.3.

7.2 Physical mapping guidelines

7.2.1 Using features of physical formats

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

7.2.2 Mapped components

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

7.2.3 Mapping media types to parts

7.2.3.1 General

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

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

7.2.3.2 Media Types stream markup

7.2.3.2.1 General

The content of the Media Types stream shall be a schema-valid XML document against opc-contentTypes.xsd (C.2). This XML document shall have a top-level `Types` element, and one or more `Default` and `Override` child elements. `Default` elements shall define default mappings from the extensions of part names to media types. `Override` elements shall specify media types on parts that are not covered by, or are not consistent with, the default mappings. Note that `Default` elements can be used to reduce the number of `Override` elements on a part.

For all parts of the package other than Relationships parts (6.5.2), the Media Types stream shall specify either:

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

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

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

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

7.2.3.2.2 Support for versioning and extensibility

The Media Types stream shall not use the versioning and extensibility mechanisms defined in ECMA-376-3.

7.2.3.2.3 `Types` element

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

This element shall have no attributes.

The W3C XML Schema definition of this element's content model is the complex type `CT_Types`, which is defined in the schema opc-contentTypes.xsd (C.2).

7.2.3.2.4 `Default` element

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

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

The W3C XML Schema definition of this element's content model is the complex type `CT_Default`, which is defined in the schema opc-contentTypes.xsd (C.2).

7.2.3.2.5 `Override` element

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

Attributes	Description
`ContentType`	This attribute specifies a media type using the syntax defined in RFC 7231, 3.1.1.1. This attribute is required. The specified media type shall apply to the part named in the attribute `PartName`. The possible values for this attribute are defined by the simple type `ST_ContentType`, which is defined in the schema opc-contentTypes.xsd (C.2).
`PartName`	This attribute specifies a part name. This attribute is required. An `Override` element shall match a part whose name is equal to the value of this attribute. The range of values for this attribute shall be as defined by the `xsd:anyURI` simple type of W3C XML Schema Datatypes.

The W3C XML Schema definition of this element's content model is the complex type `CT_Override`, which is defined in the schema opc-contentTypes.xsd (C.2).

7.2.3.3 Media Types stream markup example

EXAMPLE 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 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

7.2.3.4 Setting a part media type in the Media Types stream

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

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) If a part name has no extension, a corresponding `Override` element shall be added to the Media Types stream.
c) Compare the resulting extension with the values specified for the `Extension` attributes of the `Default` elements in the Media Types stream. The comparison shall be ASCII case-insensitive matching.
d) 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 shall be case-insensitive and include every character regardless of the role it plays in the content-type grammar of RFC 7231.

    1) If the media types match, no further action is required.

2) If the media types do not match, a new `Override` element shall be added to the Media Types stream.

e) 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.

7.2.3.5 Determining a part media type from the Media Types stream

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

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

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) Check the `Default` elements of the Media Types stream, comparing the extension with the value of the `Extension` attribute. The comparison shall be ASCII case-insensitive matching.

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

NOTE Given a conformant package, either an Override element is found by step b) or a Default element is found by step c).

7.2.4 Interleaving

When mapping an abstract package to a physical package, the data stream of a part or the Media Types stream may be broken into pieces. Each piece shall represent a data stream, which may be empty. Pieces can later be joined together, forming the original stream, based on piece names, as specified in 7.2.5.2.

A physical package may contain both interleaved parts and non-interleaved parts. Interleaved parts shall be parts broken into pieces. Non-interleaved parts shall be parts not broken into pieces.

Pieces shall exist only in the physical package and shall not be addressable in the abstract package model. Pieces shall occur in their natural piece-number order and may be interleaved with pieces of other parts or with non-interleaved parts.

Because of the performance benefits it provides, package implementers should support interleaving but are not required to do so.

For further discussion of performance benefits of interleaving see F.3.

7.2.5 Mapping part names to physical package item names

7.2.5.1 General

A mapping from an abstract package to a physical package shall use logical items as intermediate objects in order to permit interleaving (7.2.4). If a part or the Media Types stream is interleaved, each piece constructed from it shall be a logical item; otherwise, the part or Media Types stream shall be a logical item. See Figure 1.

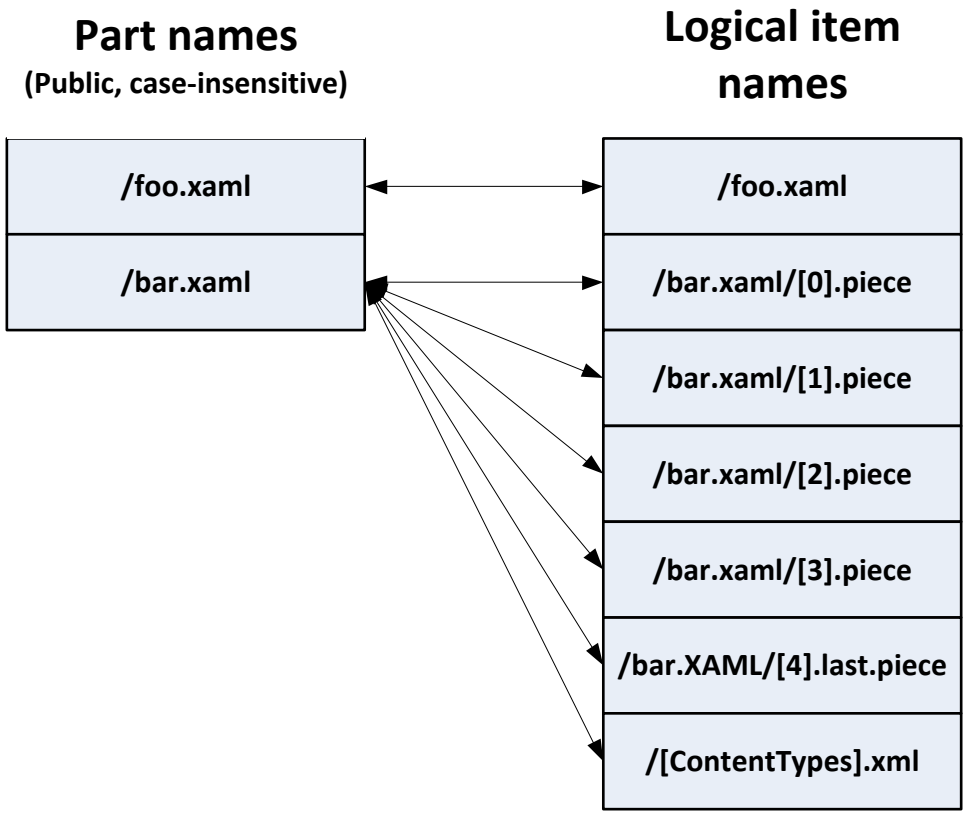

Figure 1 — Mapping Part names to logical item names

7.2.5.2 Logical item names

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

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

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

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

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

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

a) the prefix names of all logical item names in the sequence are equivalent, and
b) 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.

7.2.5.3 Mapping part names to logical item names

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

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

7.2.5.4 Mapping logical item names and physical package item names

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

7.2.5.5 Mapping logical item names to part names

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

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

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

EXAMPLE A logical item name "`/[trash]/0000.dat`" cannot be mapped to a part item. Thus, this logical item does not represent a part.

7.3 Mapping to a ZIP file

7.3.1 General

This document defines a mapping for the ZIP file format.

A ZIP file representing a physical package shall satisfy the requirements of Annex B and should follow the recommendations of Annex B.

Physical package items of ZIP files shall be ZIP items. Note that 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.

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

Table 1 — Abstract package model components and their physical representations

Abstract package model component	Physical representation
Package	ZIP file
Part	ZIP item
Part name	Stored in item header (and ZIP central directory as appropriate). See 7.3.4 for conversion rules.
Part media type	Stored in the ZIP item containing the Media Types stream described in 7.2.3.2. See 7.3.7 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 7.3.8 for a detailed description of the data structure.

7.3.2 Mapping part data

Each non-interleaved part shall be represented as a single ZIP item. Each piece of an interleaved part, as described in 7.2.4, shall be represented as a single ZIP item.

7.3.3 ZIP item names

ZIP item names shall conform to the ZIP Appnote. A mapping from an abstract package to a ZIP file shall only use ASCII ZIP item names. ZIP item names shall be unique within a given ZIP file.

EXAMPLE The following ZIP item names in a ZIP file are 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"
```

7.3.4 Mapping logical item names to ZIP item names

For each logical item, the process of mapping of logical item names to ZIP item names shall involve the following steps, in order:

a) 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.
b) Percent-encode every non-ASCII character.

7.3.5 Mapping ZIP item names to logical item names

The names of all ZIP items shall be mapped to logical item names, except for items that do not represent files.

NOTE For some file systems, the ZIP Appnote provides further information on ZIP items that are recognized as files.

For each ZIP item, the process of mapping of ZIP item names to logical item names shall involve the following steps, in order:

a) Un-percent-encode every non-ASCII character.
b) Add a forward slash (“/”).

7.3.6 ZIP package limitations

This document requires that a file header in the central directory structure within a ZIP file shall not exceed 65 535 bytes (see "F. Central directory structure" in the ZIP Appnote). Each file header contains a zip item name, Extra field (including bytes representing growth hint as specified in 6.2.4), File Comment, and 42 more bytes representing miscellaneous fields.

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

EXAMPLE Examples of these limitations are:

- On MS Windows® file systems, the asterisk (“*”) is 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; they cannot handle parts with longer names once the parts are unzipped.

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

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

7.3.7 Mapping the Media Types stream

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

NOTE Bracket characters "[" and "]" were chosen for the Media Types stream name specifically because these characters violate the part naming grammar, thus reinforcing the requirement that the ZIP item names constructed from the Media Types stream are always distinguishable from those constructed from part names.

7.3.8 Mapping the growth hint

The additional space suggested by growth hint is stored in the Extra field, as defined in the ZIP Appnote. If the growth hint is used for an interleaved part, the padding is stored in the Extra field of the ZIP item representing the first piece of the part.

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

Table 2 — Structure of the Extra field for growth hints

Field component	Size	Value
Header ID	2 bytes	0xA220
Length of Extra field	2 bytes	The length in bytes of the remaining components of the Extra field: Signature component length + Padding Initial Length component length + Padding component length
Signature (for verification)	2 bytes	0xA028
Padding Initial Length	2 bytes	The length in bytes of the Padding component set by a package implementer when the item is created
Padding	variable	Filled with 0x00 bytes

8 Core properties

8.1 General

Users can associate core properties with packages. Such core properties enable users to get and set well-known and common sets of property metadata to packages. The core properties and the specifications that describe them are shown in Table 3:

Table 3 — Core properties

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

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

8.2 Core Properties part

A package shall contain at most one Core Properties part.

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

The media type of a Core Properties part shall be the Core Properties part media type, as defined in Annex E.

8.3 Core properties markup

8.3.1 General

The content of the Core Properties part shall be a schema-valid XML document against opc-coreProperties.xsd (C.3).

Unless specified otherwise, elements representing a Core Properties part shall be of the namespace as defined in Annex E.

EXAMPLE

An example of a Core Properties part is shown below.

```
<coreProperties
   xmlns="http://schemas.openxmlformats.org/package/2006/metadata/
      core-properties"
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <dc:creator>Alan Shen</dc:creator>
   <dcterms:created xsi:type="dcterms:W3CDTF">
      2005-06-12
   </dcterms:created>

   <dc:title>OPC Core Properties</dc:title>
```

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

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

8.3.2 Support for versioning and extensibility

A Core Properties part shall not contain elements or attributes of the Markup Compatibility namespace as defined in ECMA-376-3.

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

8.3.3 `coreProperties` element

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

This element shall have no attributes.

Children of this element shall be core property elements, as defined in 8.3.4.

The content of this element is defined by the complex type `CT_CoreProperties`, which is defined in the schema opc-coreProperties.xsd (C.3).

8.3.4 Core property elements

8.3.4.1 General

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

8.3.4.2 Core property elements as defined by ISO 15836-1

This document allows `creator`, `description`, `identifier`, `language`, `subject`, and `title` elements as core property elements. If any of these elements are included, they shall be as specified by ISO 15836-1.

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

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

EXAMPLE

The example in 8.3.1 contains four elements from ISO 15836-1.

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

8.3.4.3 Core property elements as defined by ISO 15836-2

This document allows `created` and `modified` elements as core property elements. If either or both of these elements are included, they shall be as specified by ISO 15836-2.

NOTE These elements belong to the namespace "`http://purl.org/dc/terms/`".

This document introduces further requirements. These elements shall not have child elements and shall not have the `xml:lang` attribute. These elements shall have the `xsi:type` attribute whose value is "`dcterms:W3CDTF`" (see Reference [2]) and `dcterms` shall be declared as the prefix of the Dublin Core namespace "`http://purl.org/dc/terms/`".

EXAMPLE

The example in 8.3.1 contains two elements from DCMI Metadata Terms.

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

8.3.4.4 Core property elements defined in this document

8.3.4.4.1 `category` element

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

This element can have values such as “Resume”, “Letter”, “Financial Forecast”, “Proposal”, and “Technical Presentation”. This element shall have no attributes.

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

The W3C XML Schema definition of this element is in the schema opc-coreProperties.xsd (C.3).

EXAMPLE

A category element is in the example in 8.3.1.

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

8.3.4.4.2 `contentStatus` element

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

This element can have values such as “Draft”, “Reviewed”, and “Final”.This element shall have no attributes.

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

The W3C XML Schema definition of this element is in the schema opc-coreProperties.xsd (C.3).

EXAMPLE

The example in 8.3.1 contains

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

8.3.4.4.3 `keywords` element

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

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

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

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

The W3C XML Schema definition of this element's content model in the complex type `CT_Keywords`, which is defined in the schema opc-coreProperties.xsd (C.3).

8.3.4.4.4 `value` element

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

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

The W3C XML Schema definition of this element's content model is the complex type `CT_Keyword`, which is defined in the schema opc-coreProperties.xsd (C.3).

8.3.4.4.5 `lastModifiedBy` element

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

EXAMPLE 1 A name, email address, or employee ID.

This element shall have no attributes.

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

The W3C XML Schema definition of this element is the schema opc-coreProperties.xsd (C.3).

EXAMPLE 2 The example in 8.3.1 contains

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

8.3.4.4.6 `lastPrinted` element

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

This element shall have no attributes.

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

The W3C XML Schema definition of this element is the schema opc-coreProperties.xsd (C.3).

EXAMPLE 1 The example in 8.3.1 contains

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

EXAMPLE 2

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

8.3.4.4.7 `revision` element

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

This element shall have no attributes.

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

The W3C XML Schema definition of this element is the schema opc-coreProperties.xsd (C.3).

EXAMPLE

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

8.3.4.4.8 `version` element

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

This element shall have no attributes.

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

The W3C XML Schema definition of this element is the schema opc-coreProperties.xsd (C.3).

EXAMPLE

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

9 Thumbnails

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

NOTE Thumbnail parts can be used to help end-users identify parts of a package or a package as a whole.

10 Digital signatures

10.1 General

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

10.2 Overview of OPC-specific restrictions and extensions to "XML-Signature Syntax and Processing"

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

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

This document introduces markup (10.5.8.2) and a transform algorithm (10.6) for flexibly defining the relationships to be signed.

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

10.3 Choosing content to sign

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

This clause provides flexibility in defining the content to be signed, thus allowing other content to be mutable. For further information on how to define which content is to be signed, see 10.5.6 and 10.5.8.2.

10.4 Digital signature parts

10.4.1 General

Digital signatures in packages use the Digital Signature Origin part, Digital Signature XML Signature parts, and Digital Signature Certificate parts. Relationship types and media types relating to the use of digital signatures in packages are specified in Annex E. Note that an example relationship from the Digital Signature Origin part to a Digital Signature XML Signature part is provided in 6.5.4.3.

Figure 2 shows a signed package with signature parts, signed parts, and an X.509 certificate part. The example Digital Signature Origin part has relationships to two Digital Signature XML Signature parts, each containing a signature. The signatures relate to the signed parts.

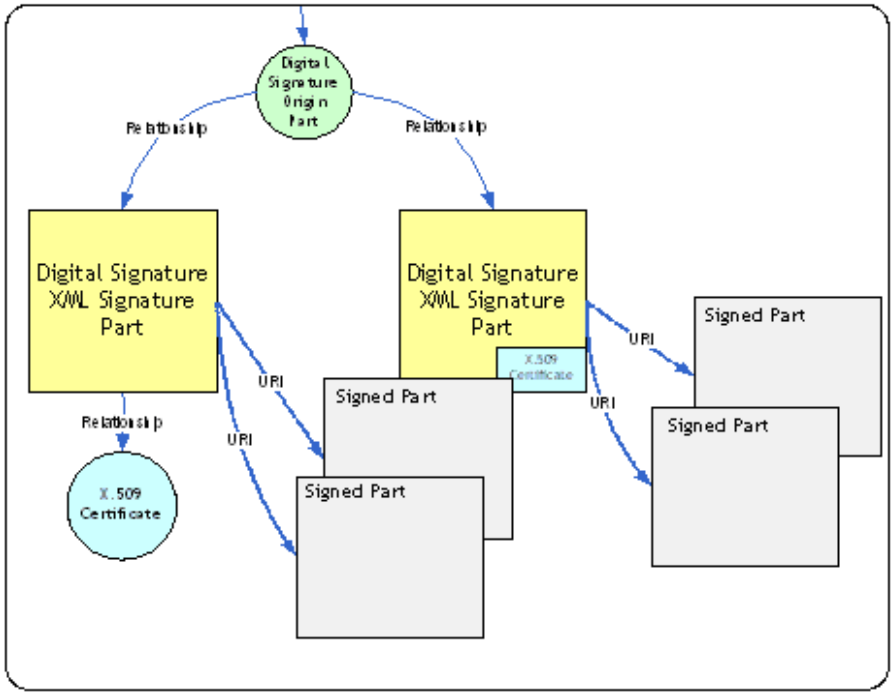

Figure 2 — A signed package

10.4.2 Digital Signature Origin part

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

10.4.3 Digital Signature XML Signature part

A Digital Signature XML Signature part shall contain a signature, represented by digital signature markup (see 10.5). Each Digital Signature XML Signature part shall be the target of a Digital Signature relationship, as specified in Annex E, from the Digital Signature Origin part. A package may contain more than one Digital Signature XML Signature part.

NOTE If future versions of this document specify distinct relationship types for revised signature parts, packages would be able to contain different signature information for different versions. For reference validation and signature validation it would be possible to choose the appropriate XML digital signatures.

10.4.4 Digital Signature Certificate part

The content of a Digital Signature Certificate part shall be a digital certificate as defined in X.509.

The X.509 certificate used for signature validation can:

— be contained within a Digital Signature XML Signature part;

— form a separate Digital Signature Certificate part; or

— be stored outside the package.

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

10.5 Digital signature markup

10.5.1 General

The content of a Digital Signature XML Signature part shall be an XML document. The requirements specified in 6.2.5 apply.

The content of each Digital Signature XML Signature part shall be a schema-valid XML document against xmldsig-core-schema.xsd, as specified in the W3C Recommendation "XML-Signature Syntax and Processing", and opc-digSig.xsd (see C.4). Algorithms shall be identified by URIs as shown in this W3C recommendation or RFC 6931.

10.5.2 to 10.5.18 cover OPC-specific restrictions and extensions to "XML-Signature Syntax and Processing". Subclauses are provided for elements defined for OPC-specific use or for which OPC introduces restrictions. Elements defined in "XML-Signature Syntax and Processing" (such as `X509Certificate`) for which no subclause is provided below are allowed in OPC packages without restriction.

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

NOTE For a general example of XML digital signature markup, see Section 2 of "XML-Signature Syntax and Processing". For a complete example of an OPC-specific digital signature, see 10.7.

10.5.2 Support for versioning and extensibility

A Digital Signature XML Signature part shall not contain elements or attributes of the Markup Compatibility namespace as defined in ECMA-376-3.

10.5.3 `Signature` element

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

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

10.5.4 `SignedInfo` element

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

A `SignedInfo` element shall contain exactly one `Reference` element referencing an OPC-specific `Object` element. The `SignedInfo` element may also contain one or more `Reference` elements referencing other data objects.

10.5.5 `CanonicalizationMethod` element

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

Packages shall use only the following canonicalization methods:

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

10.5.6 `SignatureMethod` element

This document introduces further requirements to those defined in 4.3.2 of "XML-Signature Syntax and Processing".

A `SignatureMethod` element should specify one of the following algorithms:

- http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512

The key length of RSA SHA algorithms shall be equal to or longer than 1 024 bits, and should be longer than or equal to 2 048 bits. ECDSA algorithms shown in NIST SP 800-56A Rev. 3, Appendix D (see Reference [6]), should be used. The maximum target security strength should be greater than or equal to 128.

This element should not specify

- http://www.w3.org/2000/09/xmldsig#dsa-sha1
- http://www.w3.org/2000/09/xmldsig#rsa-sha1
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1

This element may specify other algorithms.

10.5.7 `Reference` element

10.5.7.1 General

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

10.5.7.2 `Reference` element as a child of a `SignedInfo` element

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

10.5.7.3 `Reference` element as a child of a `Manifest` element

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

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

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

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

EXAMPLE 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"
```

10.5.8 `Transform` element

10.5.8.1 General

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

One of the following transform algorithms shall be specified:

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

10.5.8.2 `Transform` element representing a Relationships transform

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

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

Such a `Transform` element shall:

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

A Relationships transform describes how the `Relationship` elements from the Relationships part are selected for signing. Only one Relationships transform shall be specified for a particular Relationships part. For algorithm details, see 10.6.

10.5.9 `RelationshipReference` element

The `RelationshipReference` element specifies which `Relationship` element is signed, and shall only occur as a child element of a `Transform` element representing a Relationships transform (10.5.8.2). This element is OPC-specific.

Attributes	Description
`SourceId` (Reference to Relationship)	The value of the `Id` attribute of the referenced `Relationship` element within the given Relationships part. This attribute is required. The range of values for this attribute shall be as defined by the `xsd:string` simple type of W3C XML Schema Datatypes.

The W3C XML Schema definition of this element's content model is the complex type `CT_RelationshipReference`, which is defined in the schema opc-digSig.xsd (C.2).

10.5.10 `RelationshipsGroupReference` element

The `RelationshipsGroupReference` element specifies that the group of `Relationship` elements with the specified value for the `Type` attribute is signed. This element shall only occur as a child element of a Transform element representing a Relationships transform (10.5.8.2). This element is OPC-specific.

Attributes	Description
`SourceType` (Relationship Type)	The value of the `Type` attribute of the `Relationship` elements within the given Relationships part. This attribute is required. The range of values for this attribute shall be as defined by the `xsd:string` simple type of W3C XML Schema Datatypes.

The W3C XML Schema definition of this element's content model is the complex type `CT_RelationshipsGroupReference`, which is defined in the schema opc-digSig.xsd (C.2).

10.5.11 `DigestMethod` element

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

A `DigestMethod` element should specify one of the following algorithms:

- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmldsig-more#sha384
- http://www.w3.org/2001/04/xmlenc#sha512

This element should not specify:

- http://www.w3.org/2000/09/xmldsig#sha1

This element shall not specify:

- http://www.w3.org/2001/04/xmldsig-more#md5

This element may specify other algorithms.

10.5.12 `Object` element

10.5.12.1 General

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

10.5.12.2 OPC-specific `Object` element

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

10.5.12.3 Application-Defined `Object` element

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

Implementations should avoid values (such as "`idOfficeObject`") that are in widespread use.

10.5.13 `Manifest` element

This document introduces further requirements to those defined in 4.4 of "XML-Signature Syntax and Processing" only when a `Manifest` element occurs as a child of an OPC-specific `Object` element. `Reference` elements in such a `Manifest` element shall satisfy requirements defined in 10.5.7.3.

10.5.14 SignatureProperty element

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

10.5.15 SignatureTime element

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

This element has no attributes.

The W3C XML Schema definition of this element's content model is the complex type CT_SignatureTime, which is defined in the schema opc-digSig.xsd (C.2).

10.5.16 Format element

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

This element has no attributes.

The W3C XML Schema definition of this element's content model is ST_Format, which is defined in the schema opc-digSig.xsd (C.2).

10.5.17 Value element

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

This element has no attributes.

The W3C XML Schema definition of this element's content model is ST_Value, which is defined in the schema opc-digSig.xsd (C.2).

10.5.18 XPath element

The XPath element shall not be present. Note that the XPath element is only for XPath filtering, which is disallowed in OPC.

10.6 Relationships transform algorithm

The Relationships transform takes the XML document from the specified Relationships part and transforms it to another XML document. This transform shall be supported in generating and validating signatures.

Note that the output XML document is subsequently canonicalized by the specified canonicalization algorithm.

The Relationships transform shall have the following steps:

Step 1: Process versioning instructions

Process the Relationships part as specified in ECMA-376-3, where the markup configuration is empty, and the application configuration contains the Relationships namespace only.

Step 2: Sort and select signed relationships

a) Remove all namespace declarations except the Relationships namespace declaration.
b) Remove the Relationships namespace prefix, if it is present.
c) Sort `Relationship` elements by `Id` value in case-sensitive lexicographical order. Keep only those `Relationship` elements which either have an `Id` value that matches a `SourceId` value of a `RelationshipReference` element or have a `Type` value that matches a `SourceType` value of a `RelationshipGroupReference` element specified in the Relationships transform. Matching is ASCII case-insensitive.

EXAMPLE Consider a Relationships part

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rlsps:Relationships
xmlns:rlsps="http://schemas.openxmlformats.org/package/2006/relationships"
xmlns:foo="http://example.com/foo">
    <rlsps:Relationship Id="rId6" Type="http://../relationships/footnotes"
Target="footnotes.xml"/>
    <rlsps:Relationship Id="rId8" Type="http://../relationships/header"
Target="header1.xml"/>
    <rlsps:Relationship Id="rId32" Type="http://../relationships/image"
Target="media/image1.png"/>
    <rlsps:Relationship Id="rId3" Type="http://../relationships/styles"
Target="styles.xml"/>
    <rlsps:Relationship Id="rId21" Type="http://../relationships/image"
Target="media/image2.jpeg"/>
    <rlsps:Relationship Id="rId12" Type="http://../relationships/header"
Target="header1.xml"/>
</rlsps:Relationships>
```

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

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

```
    <Relationship Id="rId32" Type="http://../relationships/image"
Target="media/image1.png"/>
    <Relationship Id="rId6" Type="http://../relationships/footnotes"
Target="footnotes.xml"/>
</Relationships>
```

Step 3: Prepare for canonicalization

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

10.7 Digital signature example

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

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

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

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

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

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

```
<Signature Id="SignatureId" xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
         REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-
sha1"/>
      <Reference
```

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

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

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

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

10.8 Generating signatures

Generation of digitally signed packages shall use reference generation and signature generation as described in 3.1 of “XML-Signature Syntax and Processing”, with some modification for OPC-specific constructs as specified in this subclause.

NOTE The steps below do not apply to the generation of signatures that contain application-defined Object elements.

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

Reference generation:

a) For each part being signed, create a `Reference` element following the steps in 3.1.1 of “XML-Signature Syntax and Processing”.

b) Construct the OPC-specific `Object` element containing a `Manifest` element with both the child `Reference` elements obtained from the preceding step and a child `SignatureProperties` element, which, in turn, contains a child `SignatureTime` element.
c) Create a reference to the resulting OPC-specific `Object` element following the steps in 3.1.1 of "XML-Signature Syntax and Processing".

Reference generation shall support the following digest algorithms:

- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmldsig-more#sha384
- http://www.w3.org/2001/04/xmlenc#sha512

Reference generation should not support

- http://www.w3.org/2000/09/xmldsig#sha1

Reference generation shall not support

- http://www.w3.org/2001/04/xmldsig-more#md5

Reference generation may support other algorithms.

Signature generation:

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

Signature generation shall support the following algorithms:

- http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512

In the case of RSA SHA algorithms, signature generation shall support key lengths greater than or equal to 2 048. It should not support key lengths less than 2 048 bits and shall not support key lengths less than 1 024 bits. In the case of ECDSA algorithms, signature generation should support the elliptic curves defined in FIPS 186-4 as P-256, P-384, and P-521, but should not support P-224.

Signature generation should not support:

- http://www.w3.org/2000/09/xmldsig#dsa-sha1
- http://www.w3.org/2000/09/xmldsig#rsa-sha1
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1

Signature generation may support other algorithms.

10.9 Validating signatures

Validation of digitally signed packages shall use reference validation and signature validation as described in 3.2 of "XML-Signature Syntax and Processing", with some modification for OPC-specific constructs as specified in this subclause.

NOTE The steps below do not apply to the validation of signatures that contain application-defined Object elements.

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

Reference validation:

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

Second, for each reference in the `Manifest` element:

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

Reference validation shall support the following digest algorithms:

- http://www.w3.org/2001/04/xmlenc#sha256
- http://www.w3.org/2001/04/xmldsig-more#sha384
- http://www.w3.org/2001/04/xmlenc#sha512

Reference validation shall not support

- http://www.w3.org/2001/04/xmldsig-more#md5

Reference validation may support other algorithms including

- http://www.w3.org/2000/09/xmldsig#sha1

Signature validation:

Follow the steps in 3.2.2 of "XML-Signature Syntax and Processing".

Signature validation shall support the following algorithms specified by `SignatureMethod` elements:

- http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
- http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384

- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512

In the case of RSA SHA algorithms, signature validation shall support key lengths greater than or equal to 2 048 bits. They shall not support key lengths less than 1 024 bits. They may support key lengths greater than or equal to 1 024 bits and less than 2 048 bits. In the case of ECDSA algorithms, signature validation shall support the elliptic curves defined in FIPS 186-4 as P-256, P-384, and P-521, but should not support P-224.

Signature validation may support other algorithms including:

- http://www.w3.org/2000/09/xmldsig#dsa-sha1
- http://www.w3.org/2000/09/xmldsig#rsa-sha1
- http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1

Annex A
(informative)
Preprocessing for generating relative references

Relative references are available for referencing parts. Unicode strings that are similar to but are not strictly relative references are also used to reference parts. For example, "`\a.xml`" is not a relative reference since the backslash character is disallowed in RFC 3986 or RFC 3987.

Some implementations provide preprocessing of such Unicode strings to replace them with relative references. This preprocessing can involve some of (but is not limited to) the following actions:

- Percent-encode each open bracket ("[") and close bracket ("]").
- Percent-encode each space character (U+0020).
- Percent-encode each percent ("%") character that is not followed by a hexadecimal notation of an octet value.
- Un-percent-encode each percent-encoded unreserved character.
- Un-percent-encode each forward slash ("/") and back slash ("\").
- Convert all back slashes to forward slashes.
- If present in a segment containing non-dot (".") characters, remove trailing dot (".") characters from each segment.
- Replace each occurrence of multiple consecutive forward slashes ("/") with a single forward slash.
- If a single trailing forward slash ("/") is present, remove that trailing forward slash.
- Remove complete segments that consist of three or more dots.

Examples of Unicode strings converted to relative references are shown below:

Unicode string	Relative reference
`/A B.xml`	`/A%20B.xml`
`/%41/%61.xml`	`/A/a.xml`
`/%25XY.xml`	`/%25XY.xml`
`/%XY.xml`	`/%25XY.xml`
`/%2541.xml`	`/%2541.xml`
`/%2e/%2e/a.xml`	`/a.xml`
`\a.xml`	`/a.xml`
`\%41.xml`	`/A.xml`
`/%D1%86.xml`	`/%D1%86.xml`

Unicode string	Relative reference
\%2e/a.xml	/a.xml

Annex B
(normative)
Constraints and clarifications on the use of ZIP features

B.1 General

This annex specifies requirements and recommendations on features of the ZIP format. A ZIP file representing a physical package shall satisfy the specified requirements and should also follow the specified recommendations.

This annex is of particular relevance to (consuming, producing, or pass-through) physical package implementers that choose to use ZIP to represent physical packages.

B.2 Archive file header consistency

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

B.3 Data descriptor signature

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

B.4 Requirements on package implementers

The fields in the tables in this subclause contain the following values:

- “Yes” — During consumption of a package, a “Yes” value for a field in a table in this annex indicates a package implementer shall not fail to read the ZIP file containing this record or field; however, the field may be ignored. During production of a package, a “Yes” value for a field in a table in this annex indicates that the package implementer shall write out this record or field.
- “No” — A “No” value for a field in a table in this annex indicates the package implementer should not use this record or field.
- “Optional” — An “Optional” value for a record in a table in this annex indicates that package implementers may write this record during production.

- "Partially, details below" — A "Partially, details below" value for a record in a table in this annex indicates that the record contains fields for which support is not required by package implementers during production or consumption. See the details in the corresponding table to determine requirements.
- "Only used when needed" — The value "Only used when needed" associated with a record in a table in this annex indicates that the package implementer shall use the record only when needed to store data in the ZIP file.

Table B.1 specifies the requirements for package production, consumption, and editing in regard to particular top-level records or fields described in the ZIP Appnote. Note that 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.

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

Record name	Supported on consumption	Supported on production	Pass through on editing
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.

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

Record	Field	Supported on consumption	Supported on production	Pass through on editing
	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
	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,	Zip64 end of central directory signature	Yes	Yes	Yes
	Size of zip64 end of central directory	Yes	Yes	Yes

Record	Field	Supported on consumption	Supported on production	Pass through on editing
only used when needed)	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)
	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

Record	Field	Supported on consumption	Supported on production	Pass through on editing
	Number of this disk	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Number of the disk with the start of the central directory	Yes (partial — no multi disk 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
	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 Version Needed to Extract field, which is fully described in the ZIP Appnote.

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

Version	Feature	Supported on consumption	Supported on production	Pass through on editing
2.1	File is compressed using Deflate64™	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
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.

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

Code	Method	Supported on consumption	Supported on production	Pass through on editing
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
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 the general-purpose bit flags within records.

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

Bit	Feature	Supported on consumption	Supported on production	Pass through on editing
0	If set, indicates that the file is encrypted.	No	No	No
1,2	0 0: Normal (-en) compression option was used.	Yes	Yes	Yes
1,2	0 1: Maximum (-exx/-ex) compression option was used.	Yes	Yes	Yes
1,2	1 0: Fast (-ef) compression option was used.	Yes	Yes	Yes
1,2	1 1: Super Fast (-es) compression option was used.	Yes	Yes	Yes

Bit	Feature	Supported on consumption	Supported on production	Pass through on editing
3	If this bit is set, the fields crc-32, compressed size, and uncompressed size are set to zero in the local header. The correct values are put in the data descriptor immediately following the compressed data.	Yes	Yes	Yes
4	Reserved for use with method 8, for enhanced deflating	No	Bits set to 0	Yes
5	If this bit is set, this indicates that the file is compressed patched data. (Requires PKZIP version 2.70 or greater.)	No	Bits set to 0	Yes
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.

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

Field ID	Field description	Supported on consumption	Supported on production	Pass through on editing
0x0007	AV Info	No	No	Yes
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.

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

Field ID	Field description	Supported on consumption	Supported on production	Pass through on editing
0x2605	ZipIt Macintosh	No	No	Yes
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	No	No	Yes
0x4c41	OS/2 access control list (text ACL)	No	No	Yes
0x4d49	Info-ZIP OpenVMS	No	No	Yes
0x4f4c	Xceed original location extra field	No	No	Yes
0x5356	AOS/VS (ACL)	No	No	Yes
0x5455	extended timestamp	No	No	Yes
0x554e	Xceed unicode extra field	No	No	Yes
0x5855	Info-ZIP Unix (original, also OS/2, NT, etc)	No	No	Yes
0x6542	BeOS/BeBox	No	No	Yes
0x756e	ASi Unix	No	No	Yes
0x7855	Info-ZIP Unix (new)	No	No	Yes
0xa220	Padding, Microsoft	No	Optional	Optional
0xfd4a	SMS/QDOS	No	No	Yes

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

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

Annex C
(normative)
Schemas - W3C XML

C.1 General

This document is accompanied by a family of schemas defined using the syntax specified in "W3C XML Schema Structures" and "W3C XML Schema Datatypes." A ZIP file called OpenPackagingConventions-XMLSchema.zip containing all schemas accompanies this specification.

C.2 Media Types stream

See file opc-contentTypes.xsd.

C.3 Core Properties part

See file opc-coreProperties.xsd.

C.4 Digital signature XML signature markup

See file opc-digSig.xsd.

C.5 Relationships part

See file opc-relationships.xsd.

Annex D
(informative)
Schemas - RELAX NG

D.1 General

This document is accompanied by a family of schemas defined using the RELAX NG syntax. A ZIP file called OpenPackagingConventions-RELAXNG.zip containing all schemas accompanies this specification.

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

D.2 Media Types stream

See file opc-contentTypes.rnc.

D.3 Core Properties part

See file opc-coreProperties.rnc.

The schema is available at: http://standards.iso.org/iso-iec/29500/-2/ed-4/en/OpenPackagingConventions-RELAXNG/opc-coreProperties.rnc.

D.4 Digital signature XML signature markup

See files opc-digSig.rnc and security_any.rnc.

D.5 Relationships part

See file opc-relationships.rnc.

This schema is available at: http://standards.iso.org/iso-iec/29500/-2/ed-4/en/OpenPackagingConventions-RELAXNG/opc-relationships.rnc.

D.6 Additional resources

D.6.1 XML

See file xml.rnc.

D.6.2 XML digital signature core

See file xmldsig-core-schema.rnc.

Annex E
(normative)
Standard namespaces and media types

The namespaces available for use in a package are listed in Table E.1.

Table E.1 — Package-wide namespaces

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

The media types for the parts defined in this document are listed in Table E.2.

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"

Description	Media type
Relationships part	"application/vnd.openxmlformats-package.relationships+xml"

The relationship types available for use in a package are listed in Table E.3.

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 package model design considerations

F.1 General

The physical package model defines the ways in which packages are produced and consumed. This model is based on three components: a producer, a consumer, and a pipe between them, as shown in Figure F.1.

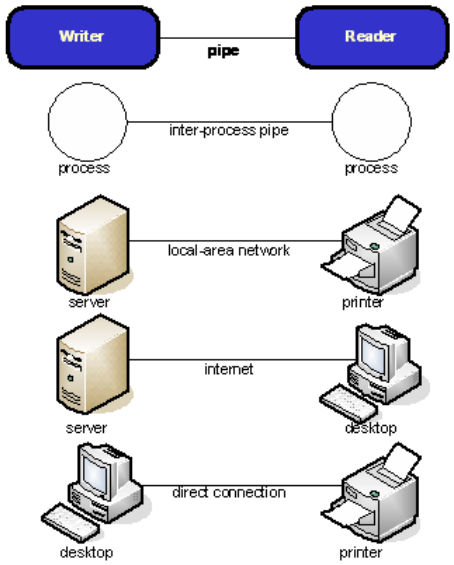

Figure F.1 — Components of the physical package model

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

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

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

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

F.2 Access styles

F.2.1 General

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

F.2.2 Direct access consumption

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

F.2.3 Streaming consumption

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

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

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

F.2.4 Streaming creation

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

EXAMPLE Performance bottleneck with simple ordering

Figure F.2 contains two parts: a page part (markup/page.xml) describing the contents of a page, and an image part (images/picture.jpeg) 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.

Figure F.2 illustrates this scenario. The image cannot be displayed until the entire page part and the image part have been received. In some circumstances, such as small packages on a high-speed network, this might be acceptable. In others, having to read through all of markup/page.xml to get to the image results in unacceptable performance or places unreasonable memory demands.

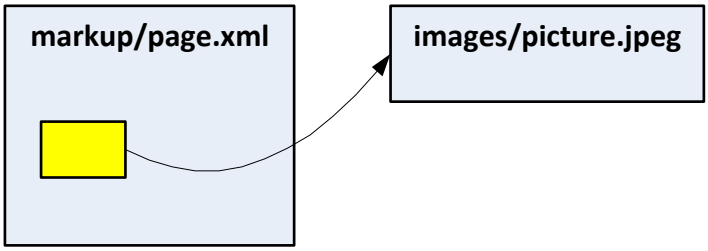

Figure F.2 — Page part and image part

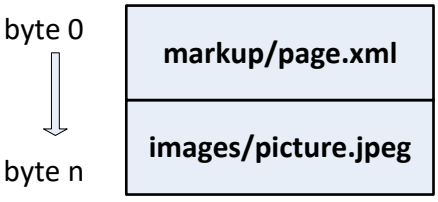

Figure F.3 — Delivery order of bytes with simple ordering

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. A part that is broken into multiple pieces in the physical file is considered one logical part; the pieces themselves are not parts and are not addressable.

F.4 Communication styles

F.4.1 General

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

F.4.2 Sequential delivery

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

F.4.3 Random access

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

Annex G
(informative)
Differences between ECMA-376-2021 and ECMA-376:2006

G.1 General

This annex documents the syntactic differences between the versions of the Open Packaging Conventions defined in this document and ECMA-376:2006 (see Reference [5]).

G.2 XML elements

The following XML element is included in this document but are not included in ECMA-376:2006:

- The `value` element (8.3.4.4.4)

The following XML element is included in ECMA-376:2006 but are not included in this document:

- The `contentType` element

G.3 XML attributes

No changes.

G.4 XML enumeration values

No changes.

G.5 XML simple types

No changes.

G.6 Part names

Non-ASCII part names are allowed by this document but are disallowed by ECMA-376:2006.

Annex H
(informative)
Package example

H.1 General

This annex depicts an abstract package and a physical package representing a Microsoft® Office document.

H.2 Abstract package

This abstract package contains five parts: "`/_rels/.rels, /docProps/core.xml`", "`/word/_rels/document.xml.rels`", "`/word/document.xml`", and "`/word/settings.xml`". See Figure H.1

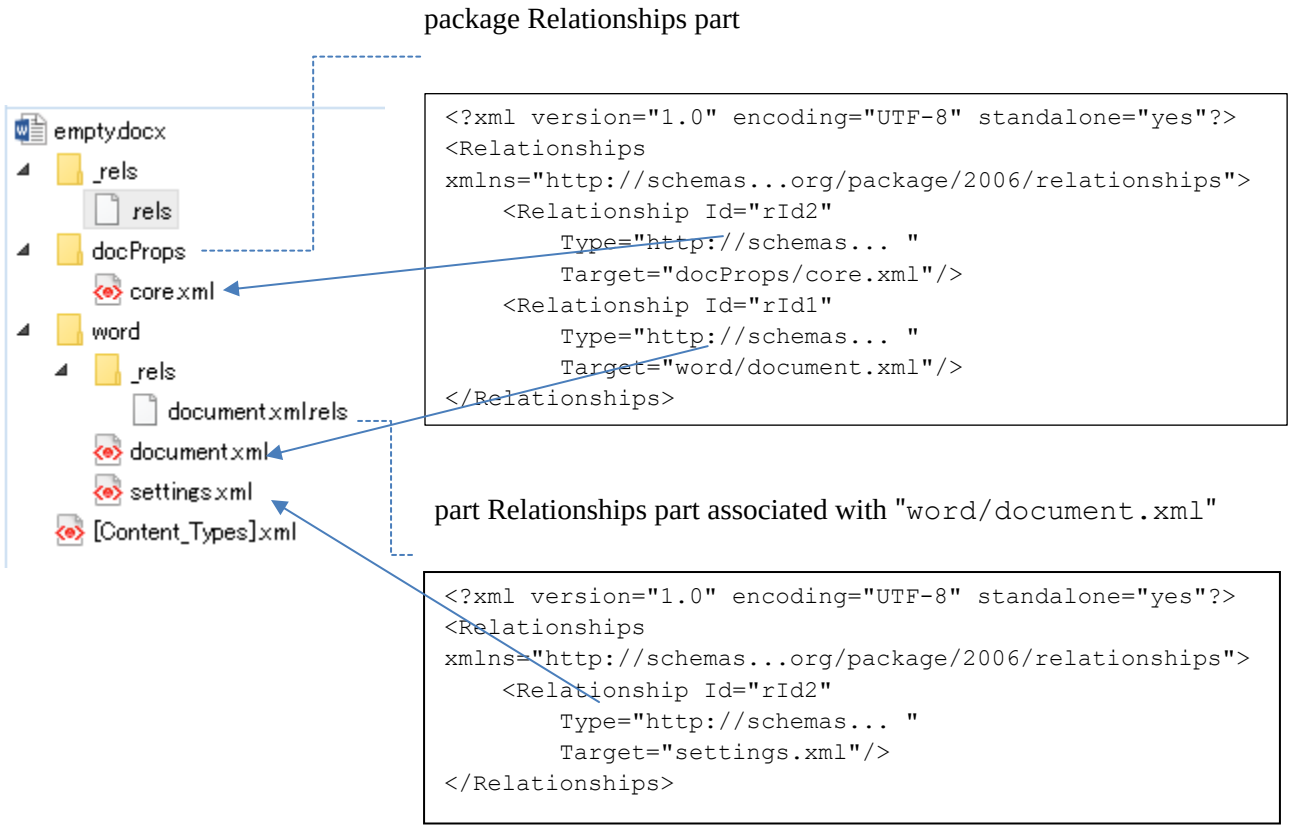

Figure H.1 — An example abstract package

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

H.3 Physical package

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

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

Bibliography

[1] *Character Model for the World Wide Web: String Matching*, W3C Working Group Note, 04 February 2019, available at https://www.w3.org/TR/2019/NOTE-charmod-norm-20190204/

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

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

[4] ISO/IEC TR 30114-1, *Information technology -- Extensions of Office Open XML file formats -- Part 1: Guidelines*, available at https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[5] ECMA-376:2006 *Office Open XML File Formats*, 1st edition (December 2006), available at https://www.ecma-international.org/publications-and-standards/standards/ecma-376/

[6] NIST SP 800-56A Rev. 3, *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*, available at https://doi.org/10.6028/NIST.SP.800-56Ar3, April 2018

[7] ISO/IEC 10646, *Information technology — Universal coded character set (UCS)*, available at https://standards.iso.org

[8] *Uniform Resource Identifier (URI) Schemes*, available at https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml