Office Open XML

Document Interchange Specification

Ecma TC45

Working Draft 1.4

Part 2: Open Packaging Conventions

Public Distribution

August 2006

The contents of this document reflect the work of Ecma TC45 as of August 2006, and are subject to change without notice.

Text highlighted like this indicates a placeholder for some TODO action.

Table of Contents

Introduction

This Standard is Part 2 of a multi-part standard covering Open XML-related technology.

- Part 1: "Fundamentals"
- **Part 2: "Open Packaging Conventions" (this document)**
- Part 3: "Primer"
- Part 4: "Markup Language Reference"
- Part 5: "Markup Compatibility"

1. Scope

This Standard specifies the structure and functionality of a *package* in terms of a package model and a physical model.

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

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

This Standard also describes certain features that might be supported in a package, including *core properties* for package metadata, a *thumbnail* for graphical representation of a package, and *digital signatures* of package contents.

Because this Standard will continue to evolve, packages are designed to accommodate extensions and support compatibility goals in a limited way. The versioning and extensibility mechanisms described in Part 4: "Markup Compatibility" support compatibility between software systems based on different versions of this Standard while allowing package creators to make use of new or proprietary features.

This Standard specifies requirements for package implementers, producers, and consumers.

2. Conformance to this Standard

Conformance to this Standard is of interest to the following audiences:

- Those designing, implementing, or maintaining Open Packaging Conventions consumers or producers.
- Governmental or commercial entities wishing to procure Open Packaging Conventions consumers or producers.
- Testing organizations wishing to provide an Open Packaging Conventions conformance test suite.
- Programmers wishing to interact programmatically with Open Packaging Conventions consumers or producers.
- Educators wishing to teach about Open Packaging Conventions consumers or producers.
- Authors wanting to write about Open Packaging Conventions consumers or producers.

As such, conformance is most important, and the bulk of this Standard is aimed at specifying the characteristics that make Open Packaging Conventions consumers or producers strictly conforming ones.

Use of the word “shall” indicates required behavior.

The text in this Standard is divided into normative and informative categories. Normative text is further broken into *mandatory* and *optional* subcategories. A mandatory feature shall be implemented as specified by this Standard. An optional feature need not be implemented; however, if it is supported, it shall be implemented as specified by this Standard. Unless stated otherwise, all features are mandatory. The text in this Standard that specifies requirements is considered mandatory. All other text in this specification is informative; that is, for information purposes only.

To conform to this Standard, an implementation shall provide the specified normative elements and meet the criteria of 2.1, A Conforming Implementation.

This Standard does not contain any unspecified behavior.

2.1 A Conforming Implementation

A *strictly conforming consumer or producer* shall use only those features of Open Packaging Conventions specified in this Standard as being mandatory. It shall not act in a manner that is dependent on any unspecified or implementation-defined behavior. A strictly conforming consumer shall accept any valid Open Packaging Conventions package. The Open Packaging Conventions packages generated by a strictly conforming producer shall be valid.

A strictly conforming consumer or producer shall interpret characters in conformance with ISO/IEC 10646-1 as required by the XML 1.0 Standard. A strictly conforming consumer or producer shall accept Unicode source files encoded with either the UTF-8 or UTF-16 encoding forms as required by the XML 1.0 Standard.

A strictly conforming consumer shall produce at least one diagnostic message if its input package is invalid. A package is invalid if any of its contents violate any rule of syntax or any negative requirement in this Standard.

A (non-strictly) *conforming consumer or producer* is one having capabilities that are a superset of those described in this Standard, provided these capabilities do not alter the behavior that is required by a strictly conforming consumer or producer. Conforming consumers and producers shall diagnose Open Packaging Conventions packages containing extensions that are outside the scope of this Standard. However, having done so, they are permitted to continue to consume or produce such packages.

A conforming consumer or producer shall be accompanied by a document that defines all implementation-defined characteristics and all extensions.

In order for any consumer to be considered conformant, it shall observe the following rules:

It shall not report errors when processing conforming instances of the documented formats except when forced to do so by resource exhaustion.

It should report errors when processing non-conforming instances of the documented formats when doing so does not pose an undue processing or performance burden.

In order for any producer to be considered conformant, it shall observe the following rules:

It shall not generate any new, non-conforming instances of a documented format.

It shall not introduce any non-conformance when modifying an instance of a documented format.

Editing applications shall observe all of the above rules.

Conformance requirements are documented inline in this specification, and each requirement is denoted with a rule number enclosed in brackets. For convenience, these rules are collected together in Annex J, "Conformance Requirements".

2.2 Verbal Forms for the Expression of Provisions

Specific verbal forms are used in the normative clauses of this Standard in order to distinguish among requirements for compliance, provisions allowing a freedom of choice, and recommendations. Those verbal forms are prescribed by ISO/IEC Directives, Part 2, "Rules for the structure and drafting of International Standards."

The following Table 2–1, "Verbal forms" summarizes the prescribed verbal forms and equivalent expressions used in this Standard.

Table 2–1. Verbal forms

Provision	Verbal form	Alternative expression

Provision	Verbal form	Alternative expression
A requirement on a producer or consumer, strictly to be followed for compliance to this Standard	shall shall not	is required to is, is to is not permitted is not allowed
A permission expressed by the Standard	might need not	is permitted to is allowed is not required to
A recommendation expressed by the Standard, it need not be followed	should should not	it is recommended that is recommended
A capability or possibility open to a producer or a consumer of the Standard	can cannot	is able to it is possible to is possible

3. Normative References

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this Standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

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

ISO/IEC 9594-8 *Public-key and attribute certificate frameworks* (x.509 Certificate).

ISO/IEC 10646 (all parts), *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*.

4. Definitions

For the purposes of this Standard, the following definitions apply. Other terms are defined where they appear in italic type. Terms explicitly defined in this Standard are not to be presumed to refer implicitly to similar terms defined elsewhere.

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

behavior — External appearance or action.

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

behavior, unspecified —Behavior where this Standard imposes no requirements.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Piece — A portion of a part. Pieces of different parts may be interleaved together. The individual pieces are named using a unique mapping from the part name. Piece name grammar does not conform with part name grammar. Pieces are not addressable in the package model.

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

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

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

Relationship —The kind of connection between a source part and a target part in a package. Relationships make the connections between parts directly discoverable without looking at the content in the parts, and without altering the parts themselves. (See also Package Relationships.)

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

Sequential delivery — A communication style in which all of part n is delivered to a consumer before part n+1.

Simple ordering — A defined ordering for laying out the parts in a package in which, when the package is delivered in a purely linear fashion, all of the bytes for the first part arrive first, then all of the bytes for the second part, and so on.

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

Stream — A linearly ordered sequence of bytes.

Streaming consumption — An access style in which parts of a physical package may be processed by a consumer before all of the bits of the package have been delivered through the pipe.

Streaming creation — A generating style in which a producer may dynamically add parts to a package after other parts have been added.

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

Well-known part — A part with a well-known relationship, which enables the part to be found without knowing the location of other parts.

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

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

5. Notational Conventions

5.1 Document Conventions

Except where otherwise noted, syntax descriptions are expressed in the ABNF format as defined in RFC 4234.

Glossary terms are formatted like *this*.

Syntax descriptions and code are formatted in `monospace` type.

Attributes names are formatted in Arial type.

Attribute values are formatted in `Lucida Console` type.

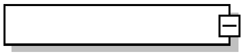
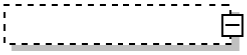
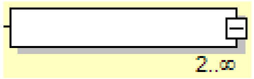
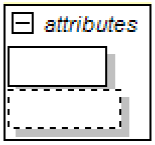
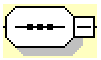
Elements are formatted in Arial type.

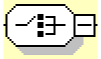
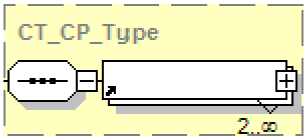
Examples are delimited by [*Example: … end example*] Examples are informative.

Notes are delimited by [*Note: … end note*] Notes are informative.

5.2 Diagram Notes

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

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

Symbol	Description
	Choice symbol: Only one of the element boxes connected to this symbol shall appear in markup.
	Type indicator: The elements within the dashed box are of the complex type indicated.

6. Acronyms and Abbreviations

This clause is informative.

The following acronyms and abbreviations are used throughout this Standard

IEC — the International Electrotechnical Commission

ISO — the International Organization for Standardization

W3C — World Wide Web Consortium

End of informative text.

7. General Description

This Standard is intended for use by implementers, academics, and application programmers. As such, it contains a considerable amount of explanatory material that, strictly speaking, is not necessary in a formal language specification.

This Standard is divided into the following subdivisions:

1. Front matter (clauses 1–7);
2. Overview and introductory material (clause 8);
3. Main body (clauses 9-13);
4. Annexes

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

The following form the normative part of this Standard:

- Clauses 1, "Scope“,” through 5, “Notational Conventions;” 7, “General Description;” and 9, “Package Model,” through 13, “Digital Signatures”
- Annex A, “Resolving Unicode Strings to Part Names,” through Annex H, “Standard Namespaces and Content Types”

The following form the informative part of this Standard:

- Introduction
- Clauses 6, “Acronyms and Abbreviations,” and 8, “Overview”
- Annex I, “Physical Model Design Considerations,” and Annex J, “Conformance Requirements”
- All notes
- All examples
- The bibliography

Whole clauses and annexes that are informative are identified as such. Informative text that is contained within normative text is identified as either an example, or a note as specified in 5.1, “Document Conventions.”

8. Overview

This clause is informative.

This specification describes an abstract model and physical format conventions for the use of XML, Unicode, ZIP, and other openly available technologies and specifications to organize the content and resources of a document within a package. It is intended to support the content types and organization for various applications and is written for developers who are building systems that process package content.

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

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

Various XML-based building blocks within a package make use of the conventions described in Part 5: "Markup Compatibility" to facilitate future enhancement and extension of XML markup. That specification shall be explicitly cited by any markup specification that bases its versioning and extensibility strategy on Markup Compatibility elements and attributes.

End of informative text.

9. Package Model

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

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

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

9.1 Parts

A *part* is a stream of bytes with the properties listed in Table 9–1. A *stream* is a linearly ordered sequence of bytes. Parts are analogous to a file in a file system or to a resource on an HTTP server.

Table 9–1. Part properties

Name	Description	Required/Optional
Name	The name of the part	Required. The package implementer shall require a part name. [M1.1]
Content Type	The type of content stored in the part	Required. The package implementer shall require a content type and the format designer shall specify the content type. [M1.2]
Growth Hint	A suggested number of bytes to reserve for the part to grow in-place	Optional. The package implementer might allow a growth hint to be provided by a producer. [O1.1]

9.1.1 Part Names

Each part has a name. *Part names* refer to parts within a package. *[Example:* The part name "/hello/world/doc.xml" contains three segments: "hello", "world", and "doc.xml". The first two segments in the sample represent levels in the logical hierarchy and serve to organize the parts of the package, whereas the third contains actual content. Note that segments are not explicitly represented as folders in the package model, and no directory of folders exists in the package model. *end example]*

9.1.1.1 Part Name Syntax

The part name grammar is defined as follows:

```
part_name = 1*( "/" segment )
segment   = 1*( pchar )
```

`pchar` is defined in RFC 3986.

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

- A part name shall not be empty. [M1.1]
- A part name shall not have empty segments. [M1.3]
- A part name shall start with a forward slash ("/") character. [M1.4]
- A part name shall not have a forward slash as the last character. [M1.5]
- A segment shall not hold any characters other than pchar characters. [M1.6]

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

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

[*Example:*

Example 9–1. A part name

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

end example]

9.1.1.2 Part Naming

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

9.1.1.3 Part Name Equivalence

Part name equivalence is determined by comparing part names as case-insensitive ASCII strings. Packages shall not contain equivalent part names and package implementers shall neither create nor recognize packages with equivalent part names. [M1.12]

9.1.2 Content Types

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

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

where parameter is expressed as

```
attribute "=" value
```

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

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

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

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

Content types for package-specific parts are defined in Annex H, “Standard Namespaces and Content Types.”

9.1.3 Growth Hint

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

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

9.1.4 XML Usage

All XML content of the parts defined in this specification shall conform to the following validation rules:

1. XML content shall be encoded using either UTF-8 or UTF-16. If any part includes an encoding declaration (as defined in §4.3.3 of the XML specification), that declaration shall not name any encoding other than UTF-8 or UTF-16. Package implementers shall enforce this requirement upon creation and retrieval of the XML content. [M1.17]
2. The XML 1.0 specification allows for the usage of Data 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 content shall not be used in the XML markup defined in this specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content and shall treat the presence of DTD content as an error. [M1.18]
3. If the XML content contains the Markup Compatibility namespace, as described in Part 4: "Markup Compatibility", it shall be processed by the package implementer to remove Markup Compatibility elements and attributes and ignorable namespaces before applying further validation rules below. [M1.19]
4. XML content shall be valid according to the corresponding XSD schema defined in this specification. In particular, the XML content shall not contain namespaces that are not explicitly defined in the corresponding XSD unless the XSD allows any namespace to be present in particular locations in the XML markup. Package implementers shall enforce this requirement upon creation and retrieval of the XML content. [M1.20]
5. XML content shall not contain elements or attributes drawn from “xml” or “xsi” namespaces unless they are explicitly defined in the XSD schema or by other means described in the specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content. [M1.21]

9.2 Part Addressing

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

9.2.1 Relative References

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

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

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

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

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

[*Example:*

Example 9–2. Part names and relative references

A package includes parts with the following names:

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

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

end example]

9.2.2 Fragments

Sometimes it is useful to address a portion of or a specific point in a part. In URIs, a fragment identifier is used for this purpose. (See RFC 3986.)

[Example: In an XML part a fragment identifier might identify a portion of the XML content using an XPath expression. *end example]*

9.3 Relationships

Parts often contain references to other parts in the package and to resources outside of the package. In general, these references are represented inside the referring part in ways that are specific to the content type of the part, that is, in arbitrary markup or an application-specific encoding. This effectively hides the internal and external links between parts from consumers that do not understand the content types of the parts containing such references.

The package introduces a higher-level mechanism to describe references from parts to other internal or external resources: relationships. *Relationships* represent the type of connection between a source part and a target resource. They make the connection directly discoverable without looking at the part contents, so they are independent of content-specific schemas and quick to resolve.

Relationships provide a second important function: relating parts without modifying their content. Sometimes relationships act as a label where the content type of the labeled part does not define a way to attach the given information. Some scenarios require information to be attached to an existing part without modifying that part, either because the part is encrypted and cannot be decrypted, or because it is digitally signed and changing it would invalidate the signature.

9.3.1 Relationships Part

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

The content type of the Relationships part is defined in Annex H, “Standard Namespaces and Content Types".

9.3.2 Package Relationships

A relationship whose source is a package as a whole is known as a *package relationship*. Package relationships are used to identify the “starting” parts in a package for a given context. This method avoids relying on naming conventions for finding parts in a package.

9.3.3 Relationship Markup

Relationships are represented using <Relationship> elements nested in a single <Relationships> element. These elements are defined in the Relationships namespace, as specified in Annex H, “Standard Namespaces and Content Types". The schema for relationships is described in Annex D, “Relationships Schema.”

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

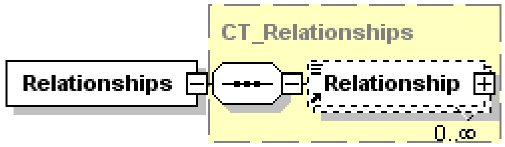
The nature of a <Relationship> element is identified by the Type attribute. Relationship Type is defined in the same way that namespaces are defined for XML namespaces. By using types patterned after the Internet domain-name space, non-coordinating parties can safely create non-conflicting relationship types.

Relationship types can be compared to determine whether two <Relationship> elements are of the same type. This comparison is conducted in the same way as when comparing URIs that identify XML namespaces: the two URIs are treated as strings and considered identical if and only if the strings have the same sequence of characters. The comparison is case-sensitive and no escaping is done or undone.

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

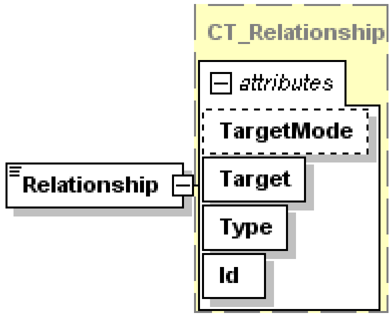
9.3.3.1 <Relationships>Element

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

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>annotation</td><td>The root element of the Relationships part.</td></tr>
</table>

9.3.3.2 <Relationship>Element

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

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

						implementer shall resolve relative references in the Target attribute against the pack URI that identifies the entire package resource. [M1.29] For more information, see Annex B, “Pack URI.” When set to External, the Target attribute may be a relative reference or a URI. If the Target attribute is a relative reference, then that reference is interpreted relative to the location of the package.
	Target	xsd:anyURI	required			The package implementer shall require the Target attribute to be a URI reference pointing to a target resource. The URI reference shall be a URI or a relative reference. [M1.28]
	Type	xsd:anyURI	required			The package implementer shall require the Type attribute to be a URI that defines the role of the relationship and the format designer shall specify such a Type. [M1.27]
	Id	xsd:ID	required			The package implementer shall require a valid XML identifier. [M1.26] The Id type is xsd:ID and it shall conform to the naming restrictions for xsd:ID as specified in the W3C Recommendation “XML Schema Part 2: Datatypes.” The value of the Id attribute shall be unique within the Relationships part.
annotation	Represents a single relationship.					

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

9.3.4 Representing Relationships

Relationships are represented in XML in a Relationships part. Each part in the package that is the source of one or more relationships can have an associated Relationships part. This part holds the list of relationships for the source part. For more information on the Relationships namespace and relationship types, see Annex H, “Standard Namespaces and Content Types.”

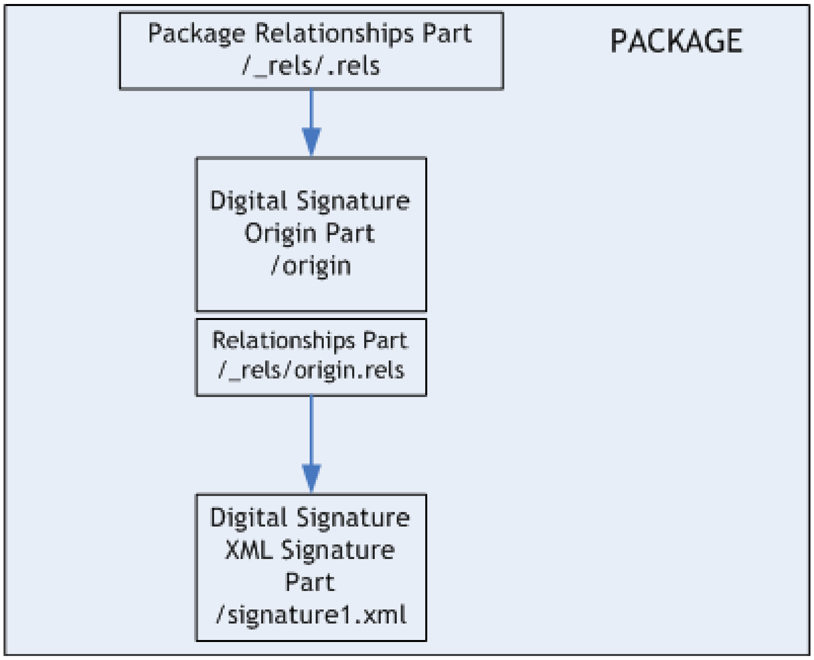
A special naming convention is used for the Relationships part. First, the Relationships part for a part in a given folder in the name hierarchy is stored in a sub-folder called “_rels”. Second, the name of the Relationships part is formed by appending “.rels” to the name of the original part. Package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

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

[Example:

Example 9–3. Sample relationships and associated markup

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



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

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

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

end example]

[Example:

Example 9–4. Targeting resources

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

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

end example]

[Example:

Example 9–5. Re-using attribute values

The following Relationships part contains two relationships, each using unique Id values. The relationships share the same Target.

```
<Relationships
```

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

end example]

9.3.5 Support for Versioning and Extensibility

Producers might generate relationship markup that uses the versioning and extensibility mechanisms defined in Part 5: "Markup Compatibility" to incorporate elements and attributes drawn from other XML namespaces. [O1.7]

Consumers shall process relationship markup in a manner that conforms to Part 5: "Markup Compatibility". Producers editing relationships based on this version of the relationship markup specification shall not preserve any ignored content, regardless of the presence of any preservation attributes as defined in Part 5: "Markup Compatibility". [M1.31]

10. Physical Package

In contrast to the package model that describes the contents of a package in an abstract way, the physical package refers to a package that is stored in a particular physical file format. This includes the physical model and physical mapping considerations.

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

[Note: See Annex I, “Physical Model Design Considerations” for additional discussion of the physical model. *end note]*

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

10.1 Physical Mapping Guidelines

Whereas the package model defines a package abstraction, an *instance* of a package must be based on a physical representation. A *physical package format* is a mapping from the components of the packaging model to the features of a particular physical format.

Producers and consumers use a combination of access styles, layout styles, and communication styles. At least one style from each category is used.

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

[Note: Part names using characters that are not valid for the file system require an escaping mechanism. *end note]*

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

10.1.1 Mapped Components

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

Table 10–1. Mapped components

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

10.1.2 Mapping Content Types

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

10.1.2.1 Identifying the Part Content Type

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

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

For all other physical package formats, the package implementer should include a specially-named XML stream in the package called the *Content Types stream.* [S2.2] The Content Types stream shall not be mapped to a part by the package implementer. [M2.1] This stream is therefore not URI-addressable. However, it can be interleaved in the physical package using the same mechanisms used for interleaving parts.

10.1.2.2 Content Types Stream Markup

The Content Types stream identifies the content type for each package part. The Content Types stream contains XML with a top-level <Types> element, and one or more <Default> and <Override> child elements. <Default> elements define default mappings from the extensions of part names to content types. <Override> elements are used to specify content types on parts that are not covered by, or are not consistent with, the default mappings. Package producers can use pre-defined <Default> elements to reduce the number of <Override> elements on a part, but are not required to do so. [O2.4]

The package implementer shall require that the Content Types stream contain one of the following for every part in the package:

- One matching <Default> element
- One matching <Override> element
- Both a matching <Default> element and a matching <Override> element, in which case the <Override> element takes precedence. [M2.4]

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

The order of <Default> and <Override> elements in the Content Types stream is not significant.

If the package is intended for streaming consumption:

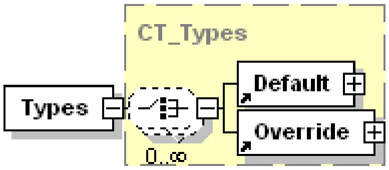
- The package implementer should not allow <Default> elements; as a consequence, there should be one <Override> element for each part in the package.
- The format producer should write the <Override> elements to the package so they appear before the parts to which they correspond, or in close proximity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

[S2.3]

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

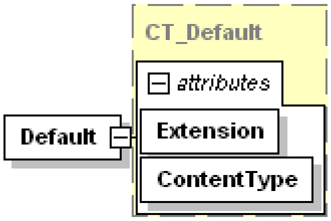
10.1.2.2.1 <Types>Element

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

diagram	
annotation	The root element of the Content Types stream.

10.1.2.2.2 <Default>Element

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

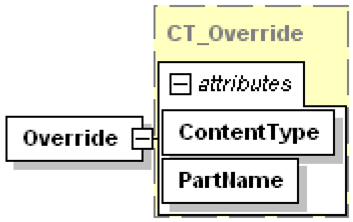
<table>
<tr><td>diagram</td><td>

CT_Default
attributes
Default
Extension
ContentType

</td></tr>
<tr><td>attributes</td><td><table>
<tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr>
<tr><td>Extension</td><td>ST_Extension</td><td>required</td><td></td><td></td><td>A part name extension. A <Default> element matches any part whose name ends with a period followed by the value of this attribute. The package implementer shall require a non-empty extension in a <Default> element. [M2.6]</td></tr>
<tr><td>ContentType</td><td>ST_ContentType</td><td>required</td><td></td><td></td><td>A content type as defined in RFC 2616. Indicates the content type of any matching parts (unless overridden). The package implementer shall require a content type in a <Default> element and the format designer shall specify the content type. [M2.6]</td></tr>
</table></td></tr>
<tr><td>annotation</td><td>Defines default mappings from the extensions of part names to content types.</td></tr>
</table>

10.1.2.2.3 <Override>Element

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

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>attributes</td><td><table>
<tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr>
<tr><td>ContentType</td><td>ST_ContentType</td><td>required</td><td></td><td></td><td>A content type as defined in RFC 2616. Indicates the content type of the matching part. The package implementer shall require a content type and the format designer shall specify the content type in an <Override> element. [M2.7]</td></tr>
<tr><td>PartName</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>A part name. An <Override> element matches the part whose name is equal to the value of this attribute. The package implementer shall require a part name. [M2.7]</td></tr>
</table></td></tr>
<tr><td>annotation</td><td>Specifies content types on parts that are not covered by, or are not consistent with, the default mappings.</td></tr>
</table>

10.1.2.2.4 Content Types Stream Markup Example

[Example:

Example 10–6. Content Types stream markup

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

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

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

end example]

10.1.2.3 Setting the Content Type of a Part

When adding a new part to a package, the package implementer shall ensure that a content type for that part is specified in the Content Types stream; the package implementer shall perform the following steps to do so [M2.8]:

1. Get the extension from the part name by taking the substring to the right of the rightmost occurrence of the dot character (.) from the rightmost segment.
2. If a part name has no extension, a corresponding <Override> element shall be added to the Content Types stream.
3. Compare the resulting extension with the values specified for the Extension attributes of the <Default> elements in the Content Types stream. The comparison shall be case-insensitive ASCII.
4. If there is a <Default> element with a matching Extension attribute, then the content type of the new part shall be compared with the value of the ContentType attribute. The comparison might be case-sensitive and include every character regardless of the role it plays in the content-type grammar of RFC 2616, or it might follow the grammar of RFC 2616.

   a. If the content types match, no further action is required.
   b. If the content types do not match, a new <Override> element shall be added to the Content Types stream. .

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

10.1.2.4 Getting the Content Type of a Part

To get the content type of a part, the package implementer shall perform the following steps [M2.9]:

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

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

4. If there is a <Default> element with a matching Extension attribute, return the value of its ContentType attribute. No further action is required.
5. If neither <Override> nor <Default> elements with matching attributes are found for the specified part, the implementation shall consider this an error.

10.1.2.5 Support for Versioning and Extensibility

The package implementer shall not use the versioning and extensibility mechanisms defined in Part 5: "Markup Compatibility" to incorporate elements and attributes drawn from other XML-namespaces into the Content Types stream markup. [M2.10]

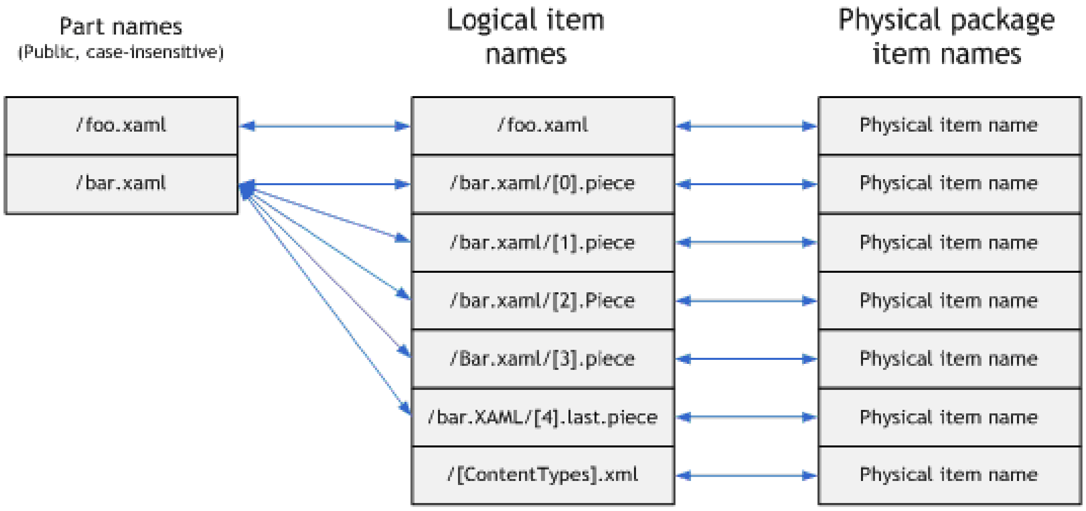
10.1.3 Mapping Part Names to Physical Package Item Names

The mapping of part names to the names of items in the physical package uses an intermediate *logical item name* abstraction. This logical item name abstraction allows package implementers to manipulate physical data items consistently regardless of whether those data items can be mapped to parts or not or whether the package is laid out with simple ordering or interleaved ordering. See §10.1.4, “Interleaving” for interleaving details.

[Example:

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

Figure 10–1. Part names and logical item names



end example]

10.1.3.1 Logical Item Names

Logical item names have the following syntax:

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

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

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

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

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

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

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

10.1.3.2 Mapping Part Names to Logical Item Names

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

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

10.1.3.3 Mapping Logical Item Names and Physical Package Item Names

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

10.1.3.4 Mapping Logical Item Names to Part Names

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

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

The package implementer might allow a package that contains logical item names and complete sequences of logical item names that cannot be mapped to a part name. [O2.7] The package implementer shall not map logical items to parts if the logical item names violate the part naming rules, even if a third-party format includes these logical items in the package. [M2.16]

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

10.1.4 Interleaving

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

The interleaving mechanism breaks the data stream of a part into *pieces*, which can be interleaved with pieces of other parts or with whole parts. Pieces are named using a unique mapping from the part name, defined in §10.1.3, "Mapping Part Names to Physical Package Item Names". This enables a consumer to join the pieces together in their original order, forming the data stream of the part.

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

An individual part shall be stored either in an interleaved or non-interleaved fashion. The package implementer shall not mix interleaving and non-interleaving for an individual part. [M2.11] The format designer specifies whether that format might use interleaving. [O2.1]

The grammar for deriving piece names from a given part name is defined by the logical item name grammar as defined in §10.1.3.1, "Logical Item Names." A suffix name is mandatory.

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

[Example:

Example 10–7. ZIP archive contents

A ZIP archive might contain the following item names mapped to part pieces and whole parts:

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

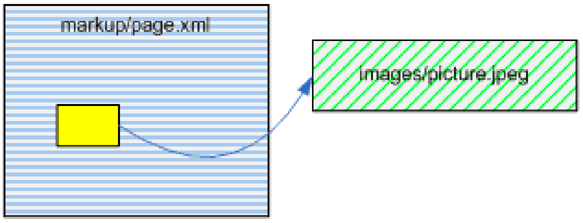
end example]

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

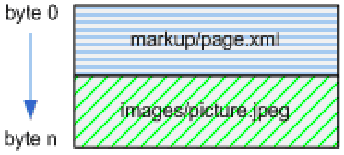
[Example:

Example 10–8. Performance benefits with interleaved ordering

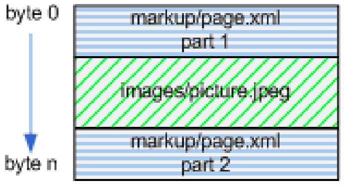
The figure below contains two parts: a page part (markup/page.xml) describing the contents of a page, and an image part (images/picture.jpg) referring to an image that appears on the page.



With simple ordering, *all* of the bytes of the page part are delivered before the bytes of the image part. The figure below illustrates this scenario. The consumer is unable to display the image until it has received *all* of the page part *and* the image part. In some circumstances, such as small packages on a high-speed network, this may be acceptable. In others, having to read through all of markup/page.xml to get to the image results in unacceptable performance or places unreasonable memory demands on the consumer's system.



With interleaved ordering, performance is improved by splitting the page part into pieces and inserting the image part immediately following the reference to the image. This allows the consumer to begin processing the image as soon as it encounters the reference.



end example]

10.2 Mapping to a ZIP Archive

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

A *ZIP archive* is a ZIP file as defined in the ZIP file format specification excluding all elements of that specification related to encryption or decryption. A ZIP archive contains *ZIP items*. ZIP items become files when the archive is unzipped. When users unzip a ZIP-based package, they see a set of files and folders that reflects the parts in the package and their hierarchical naming structure.

Table 10–2, Package model components and their physical representations, shows the various components of the package model and their corresponding physical representation in a ZIP archive.

Table 10–2. Package model components and their physical representations

Package model component	Physical representation
Package	ZIP archive file
Part	ZIP item
Part name	Stored in item header (and ZIP central directory as appropriate). See §10.2.3, “Mapping Part Names to ZIP Item Names,” for conversion rules.
Part content type	ZIP item containing XML that identifies the content types for each part according to the pattern described in §10.1.2.1, “Identifying the Part Content Type”.
Growth hint	Padding reserved in the ZIP Extra field in the local header that precedes the item. See §10.2.7, “Mapping the Growth Hint,” for a detailed description of the data structure.

10.2.1 Mapping Part Data

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

A package implementer shall store a non-interleaved part as a single ZIP item. [M3.1] When interleaved, a package implementer shall represent a part as one or more pieces, using the method described in §10.1.4, “Interleaving”. [M2.18] Pieces are named using the specified pattern, making it possible to rebuild the entire part from its constituent pieces. Each piece is stored within a ZIP archive as a single ZIP item.

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

10.2.2 ZIP Item Names

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

10.2.3 Mapping Part Names to ZIP Item Names

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

1. Convert the part name to a logical item name or, in the case of interleaved parts, to a complete sequence of logical item names.
2. Remove the leading forward slash (/) from the logical item name or, in the case of interleaved parts, from each of the logical item names within the complete sequence.

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

10.2.4 Mapping ZIP Item Names to Part Names

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

1. Map the ZIP item names to logical item names by adding a forward slash (/) to each of the ZIP item names.
2. Map the obtained logical item names to part names. For more information, see §10.1.3.4, "Mapping Logical Item Names to Part Names".

10.2.5 ZIP Package Limitations

The package implementer shall only find ZIP items that are recognized as MS-DOS files according to the ZIP specification as eligible to be mapped to parts; all other ZIP items shall not be mapped to parts. [M3.7]

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

In ZIP archives, the package implementer shall not exceed 65,515 bytes for the combined length of the item name, Extra field, and Comment fields. [M3.8] Accordingly, part names stored in ZIP archives are limited to some length less than 65,535 characters, depending on the size of the Extra and Comment fields.

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

[Example:

Examples of these limitations are:

- On Windows file systems, the asterisk ("*") and colon (":") are not valid, so parts named with this character will not unzip successfully.
- On Windows file systems, many programs can handle only file names that are less than 256 characters *including* the full path; parts with longer names might not behave properly once unzipped.
- On Unix file systems, the semicolon (";") has a special meaning, so parts with this character might not be processed as expected.

end example]

ZIP-based packages shall not include encryption as described in the ZIP specification. Package implementers shall enforce this restriction. [M3.9]

10.2.6 Mapping Part Content Type

Part content types are used for associating content types with part data within a package. In ZIP archives, content type information is stored using the common mapping pattern that stores this information in a single XML stream as follows:

- Package implementers shall store content type data in an item(s) mapped to the logical item name with the prefix_name equal to “/[Content_Types].xml” or in the interleaved case to the complete sequence of logical item names with that prefix_name. [M3.10]
- Package implementers shall not map logical item name(s) mapped to the Content Types stream in a ZIP archive to a part name. [M3.11]

10.2.7 Mapping the Growth Hint

In a ZIP archive, the growth hint is used to reserve additional bytes that can be used to allow an item to grow in-place. The padding is stored in the Extra field, as defined in the ZIP file format specification. If a growth hint is used for an interleaved part, the package implementer shall store the Extra field containing the growth hint padding with the item that represents the first piece of the part. [M3.12]

The ZIP file format specification defines the format of the Extra field to be as shown in Table 10–3, Structure of the Extra field.

Table 10–3. Structure of the Extra field

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

10.2.8 Late Detection of ZIP Items Unfit for Streaming Consumption

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

- A duplicate ZIP item name is detected the moment the second ZIP item with that name is encountered. Duplicate ZIP item names are not allowed. [M3.3]

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

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

10.2.9 ZIP Format Clarifications for Packages

The ZIP format includes a number of features that packages do not support. Some ZIP features are clarified in the package context. See Annex C, “ZIP Appnote.txt Clarifications,” for package-specific ZIP information.

11. Core Properties

Core properties enable users to get and set well-known and common sets of property metadata within packages. The core properties and the Standard that describes them are shown in Table 11–1, “Core properties”. The namespace for the properties in this table in the Open Packaging Conventions domain are defined in Annex H, “Standard Namespaces and Content Types.”

Table 11–1. Core properties

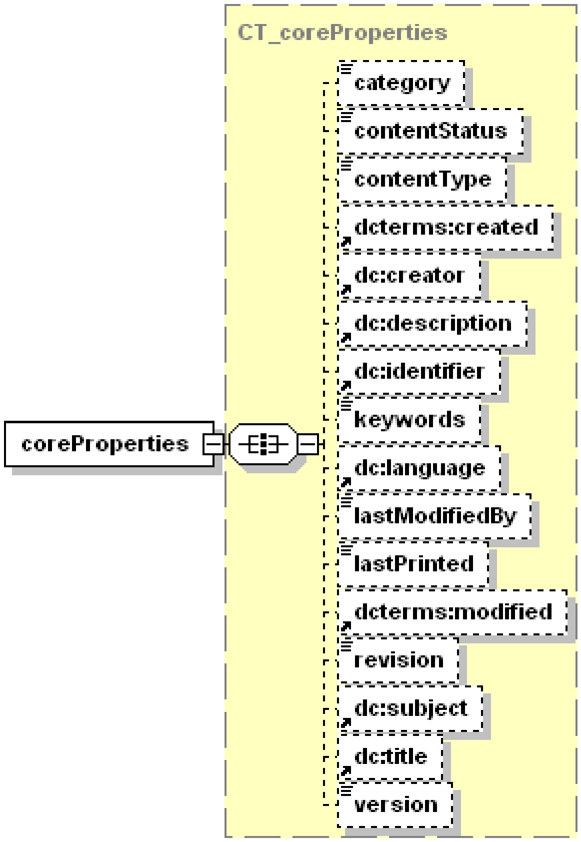
Property	Domain	Description
category	Open Packaging Conventions	The category. This value is typically used by UI applications to create navigation controls.
contentStatus	Open Packaging Conventions	The status of the content. *[Example*: Values might include “Draft”, “Reviewed”, and “Final”. *end example]*
contentType	Open Packaging Conventions	The type of content represented, generally defined by a specific use and intended audience. *[Example*: Values might include “Whitepaper”, “Security Bulletin”, and “Exam”. *end example]* *[Note*: This property is distinct from MIME content types as defined in RFC 2616. *end note]*
created	Dublin Core	Date of creation of the resource.
creator	Dublin Core	An entity primarily responsible for making the content of the resource.
description	Dublin Core	An explanation of the content of the resource. *[Example*: Values might include an abstract, table of contents, reference to a graphical representation of content, and a free-text account of the content. *end example]*
identifier	Dublin Core	An unambiguous reference to the resource within a given context.
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	The language of the intellectual content of the resource.
lastModifiedBy	Open Packaging Conventions	The user who performed the last modification. The identification is environment-specific. *[Example*: A name, email address, or employee ID. *end example]* It is recommended that this value be as concise as possible.

Property	Domain	Description
lastPrinted	Open Packaging Conventions	The date and time of the last printing.
modified	Dublin Core	Date on which the resource was changed.
revision	Open Packaging Conventions	The revision number. *[Example:* This value might indicate the number of saves or revisions, provided the application updates it after each revision. *end example]*
subject	Dublin Core	The topic of the content of the resource.
title	Dublin Core	The name given to the resource.
version	Open Packaging Conventions	The version number. This value is set by the user or by the application.

11.1.1 Core Properties Part

Core properties are stored in XML in the Core Properties part. The Core Properties part content type is defined in Annex H, “Standard Namespaces and Content Types.”

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

diagram	

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

[Example:

Example 11–1. Core properties markup

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

```
<coreProperties
   xmlns="http://schemas.openxmlformats.org/package/2006/metadata/
      core-properties"
   xmlns:dcterms="http://purl.org/dc/terms/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <dc:creator>Alan Shen</dc:creator>
   <dcterms:created xsi:type="dcterms:W3CDTF">
      2005-06-12
   </dcterms:created>
   <contentType>Functional Specification</contentType>
   <dc:title>OPC Core Properties</dc:title>
   <dc:subject>Spec defines the schema for OPC Core Properties and their
      location within the package</dc:subject>
   <dc:language>eng</dc:language>
   <version>1.0</version>
   <lastModifiedBy>Alan Shen</lastModifiedBy>
   <dcterms:modified xsi:type="dcterms:W3CDTF">
      2005-11-23
   </dcterms:modified>
   <contentStatus>Reviewed</contentStatus>
</coreProperties>
```

end example]

11.1.2 Discoverability of Core Properties

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

11.1.3 Support for Versioning and Extensibility

The format designer shall not specify and the format producer shall not create Core Properties that use the Markup Compatibility namespace as defined in Annex H, “Standard Namespaces and Content Types”. A format consumer shall consider the use of the Markup Compatibility namespace to be an error. [M4.2] Instead, versioning and extensibility functionality is accomplished by creating a new part and using a relationship with a new type to point from the Core Properties part to the new part. This specification does not provide any requirements or guidelines for new parts or relationship types that are used to extend core properties.

12. Thumbnails

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

12.1 Thumbnail Parts

The format designer shall specify thumbnail parts that are identified by either a part relationship or a package relationship. The producer shall build the package accordingly. [M5.1] For information about the relationship type for Thumbnail parts, see Annex H, “Standard Namespaces and Content Types.”

13. Digital Signatures

Format designers might allow a package to include digital signatures to enable consumers to validate the integrity of the contents. The producer might include the digital signature when allowed by the format designer. [O6.1] Consumers can identify the parts of a package that have been signed and the process for validating the signatures. Digital signatures do not protect data from being changed. However, consumers can detect whether signed data has been altered and notify the end-user, restrict the display of altered content, or take other actions.

Producers incorporate digital signatures using a specified configuration of parts and relationships. This clause describes how the package digital signature framework applies the W3C Recommendation “XML-Signature Syntax and Processing” (referred to here as the “XML Digital Signature specification”). In addition to complying with the XML Digital Signature specification, producers and consumers also apply the modifications specified in §13.2.4.1, “Modifications to the XML Digital Signature Specification”.

13.1 Choosing Content to Sign

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

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

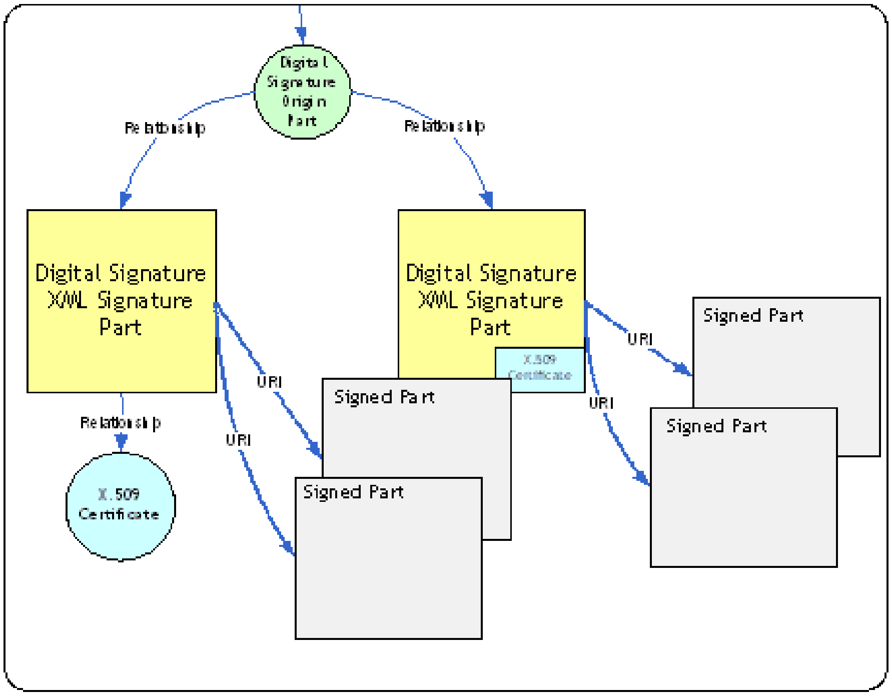
13.2 Digital Signature Parts

The digital signature parts consist of the Digital Signature Origin part, Digital Signature XML Signature parts, and Digital Signature Certificate parts. Relationship names and content types relating to the use of digital signatures in packages are defined in Annex H, “Standard Namespaces and Content Types.”

[Example:

Figure 13–1 shows a signed package with signatures, signed parts, and an X.509 certificate. The example Digital Signature Origin part references two Digital Signature XML Signature parts, each containing a signature. The signatures relate to the signed parts.

Figure 13–1. A signed package



end example]

13.2.1 Digital Signature Origin Part

The Digital Signature Origin part is the starting point for navigating through the signatures in a package. The package implementer shall include only one Digital Signature Origin part in a package and it shall be targeted from the package root using the well-defined relationship type specified in Annex H, “Standard Namespaces and Content Types”. [M6.1] When creating the first Digital Signature XML Signature part, the package implementer shall create the Digital Signature Origin part, if it does not exist, in order to specify a relationship to that Digital Signature XML Signature part. [M6.2] If there are no Digital Signature XML Signature parts in the package, the Digital Signature Origin part is optional. [O6.2] Relationships to the Digital Signature XML Signature parts are defined in the Relationships part. The producer should create empty contents in the Digital Signature Origin part itself. [S6.1]

The presence of the Digital Signature Origin part indicates to a consumer that at least one digital signature is present in the package, stored in a Digital Signature XML Signature part. The producer shall create Digital Signature XML Signature parts that have a relationship from the Digital Signature Origin part and the consumer shall use that relationship to locate signature information within the package. [M6.3]

13.2.2 Digital Signature XML Signature Part

Digital Signature XML Signature parts are targeted from the Digital Signature Origin part by a relationship that uses the well-defined relationship type specified in Annex H, “Standard Namespaces and Content Types”. The Digital Signature XML Signature part contains digital signature markup. The producer might create each Digital Signature XML Signature part with a different form of signature, which can be handled by the consumer with different signature processors. [O6.3] The producer might create zero or many Digital Signature XML Signature parts in a package. [O6.4]

13.2.3 Digital Signature Certificate Part

The Digital Signature Certificate part contains an optional X.509 certificate for validating the signature. The producer might store the certificate as a separate part in the package, might embed it within the Digital Signature XML Signature part itself, or might not include it in the package if certificate data is known or can be obtained from a local or remote certificate store. [O6.5]

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

If the certificate is represented as a separate part within the package, the producer shall target that certificate from the appropriate Digital Signature XML Signature part by a Digital Signature Certificate relationship as specified in Annex H, “Standard Namespaces and Content Types” and the consumer shall use that relationship to locate the certificate. [M6.4] The producer might sign the part holding the certificate. [O6.6] The content types of the Digital Signature Certificate part and the relationship targeting it from the Digital Signature XML Signature part are defined in Annex H, “Standard Namespaces and Content Types”, Producers might share Digital Signature Certificate parts by using the same certificate to create more than one signature. [O6.7] Producers generating digital signatures should not create Digital Signature Certificate parts that are not the target of at least one Digital Signature Certificate relationship from a Digital Signature XML Signature part. In addition, producers should remove a Digital Signature Certificate part if removing the last Digital Signature XML Signature part that has a Digital Signature Certificate relationship to it. [S6.2]

13.2.4 Digital Signature Markup

The markup described here includes a subset of elements and attributes from the XML Digital Signature specification and some package-specific markup. For a complete example of a digital signature, see §13.3, “Digital Signature Example”.

13.2.4.1 Modifications to the XML Digital Signature Specification

The package modifications to the XML Digital Signature specification are summarized as follows:

1. The producer shall create <Reference> elements within a <SignedInfo> element that reference elements within the same <Signature> element. The consumer shall consider <Reference> elements within a <SignedInfo> element that reference any resources outside the same <Signature> element to be in error. [M6.5] The producer should only create <Reference> elements within a <SignedInfo> element that reference an <Object> element. [S6.5] The producer shall not create a reference to a package-specific <Object> element that contains a transform other than a canonicalization transform. The consumer shall consider a reference to a package-specific <Object> element that contains a transform other than a canonical transform to be an error. [M6.6]
2. The producer shall create one and only one package-specific <Object> element in the <Signature> element. The consumer shall consider zero or more than one package-specific <Object> element in the <Signature> element to be an error. [M6.7]

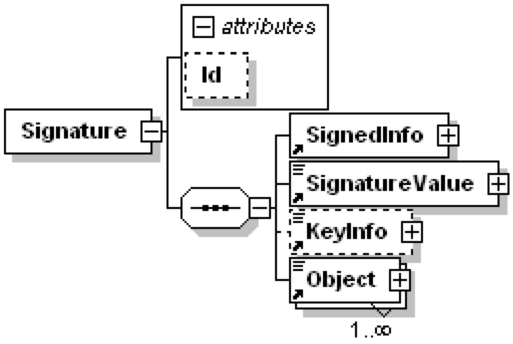
3. The producer shall create package-specific <Object> elements that contain only <Manifest> and <SignatureProperties> elements. The consumer shall consider package-specific <Object> elements that contain other types of elements to be an error. [M6.8] *[Note:* A signature may contain other <Object> elements that are not package-specific. *end note]*

    a. The producer shall create <Reference> elements within a <Manifest> element that reference with their URI attribute only parts within the package. The consumer shall consider <Reference> elements within a <Manifest> element that reference resources outside the package to be an error. [M6.9] The producer shall create relative references to the local parts that have query components that specifies the part content type as described in §13.2.4.6, "<Reference>Element". The consumer shall consider a relative reference to a local part that has a query component that incorrectly specifies the part content type to be an error. [M6.10] The producer shall create <Reference> elements with a query component that specifies the content type that matches the content type of the referenced part. The consumer shall consider signature validation to fail if the part content type does not match the content type specified in the query component of the part reference. [M6.11]
    b. The producer shall not create <Reference> elements within a <Manifest> element that contain transforms other than the canonicalization transform and relationships transform. The consumer shall consider <Reference> elements within a <Manifest> element that contain transforms other than the canonicalization transform and relationships transform to be in error. [M6.12]
    c. A producer that uses an optional relationships transform shall follow it by a canonicalization transform. The consumer shall consider any relationships transform that is not followed by a canonicalization transform to be an error. [M6.13]
    d. The producer shall create only <SignatureProperty> elements that contain a <SignatureTime> element. The consumer shall consider a <SignatureProperty> element that does not contain a <SignatureTime> element to be in error. [M6.14].

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

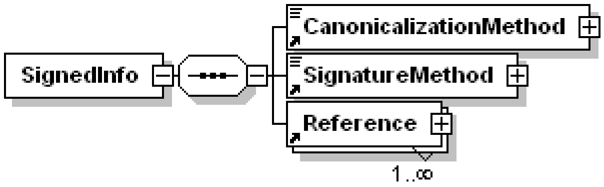
13.2.4.2 <Signature>Element

The structure of a <Signature> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr><tr><td>Id</td><td>xs:ID</td><td>optional</td><td></td><td></td><td>A unique identifier of the signature xml document.</td></tr></table></td></tr>
<tr><td>annotation</td><td>The root element of the signature xml document stored in a signature part. The producer shall create a <Signature> element that contains one local-data, package-specific <Object> element and zero or more application-specific <Object> elements. If a <Signature> element violates this constraint, a consumer shall consider this to be an error. [M6.15]</td></tr>
</table>

13.2.4.3 <SignedInfo>Element

The structure of a <SignedInfo> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>annotation</td><td>Specifies the data in the package that is signed. Holds one or more references to <Object> elements within the same Digital Signature XML Signature part. The producer shall create a <SignedInfo> element that contains a reference to the package-specific <Object> element. The consumer shall consider it an error if a <SignedInfo> element does not contain a reference to the package-specific <Object> element. [M6.16]</td></tr>
</table>

13.2.4.4 <CanonicalizationMethod>Element

The structure of a <CanonicalizationMethod> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
CanonicalizationMethod
attributes
Algorithm

</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table>
<tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr>
<tr><td>Algorithm</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>Contains a URI that identifies the particular canonicalization algorithm.</td></tr>
</table></td></tr>
<tr><td>annotation</td><td>Specifies the canonicalization algorithm applied to the <SignedInfo> element prior to performing signature calculations.</td></tr>
</table>

Since XML allows equivalent content to be represented differently, a producer should apply a canonicalization transform to the <SignedInfo> element when it generates it, and a consumer should apply the canonicalization transform to the <SignedInfo> element when validating it. [S6.3]

[Note: Performing a canonicalization transform ensures that <SignedInfo> content can be validated even if the content has been regenerated using, for example, different entity structures, attribute ordering, or character encoding.

Producers and consumers should also use canonicalization transforms for references to parts that hold XML documents. These transforms are defined using the <Transform> element. *end note]*

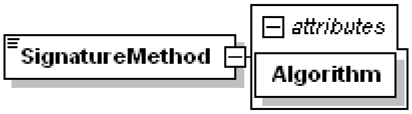
The following canonicalization methods shall be supported by producers and consumers of packages with digital signatures:

1. XML Canonicalization (c14n)
2. XML Canonicalization with Comments (c14n with comments)0.

Consumers validating signed packages shall fail the validation if other canonicalization methods are encountered. [M6.34]

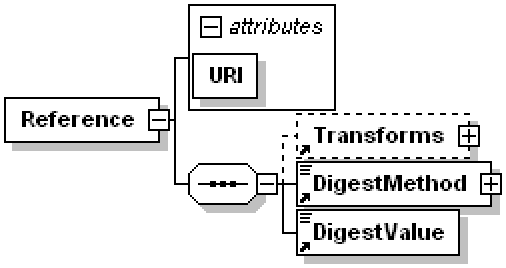
13.2.4.5 <SignatureMethod>Element

The structure of a <SignatureMethod> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr><tr><td>Algorithm</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>Contains a URI that identifies the particular algorithm for the signature method.</td></tr></table></td></tr>
<tr><td>annotation</td><td>Defines the algorithm that is used to convert the <SignedInfo> element into a hashed value contained in the <SignatureValue> element. Producers shall support DSA and RSA algorithms to produce signatures. Consumers shall support DSA and RSA algorithms to validate signatures. [M6.17]</td></tr>
</table>

13.2.4.6 <Reference>Element

The structure of a <Reference> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr><tr><td>URI</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>Contains a URI that identifies an element within the signature xml document.</td></tr></table></td></tr>
<tr><td>annotation</td><td>Specifies the object being signed, a digest algorithm, a digest value, and a list of transforms to be applied prior to digesting.</td></tr>
</table>

[Note: <Reference> elements within a <SignedInfo> element should reference an <Object> element, but might not. *end note]*

The producer shall create a <Reference> element within a <Manifest> element with a URI attribute and that attribute shall contain a part name, without a fragment identifier. The consumer shall consider a <Reference> element with a URI attribute that does not contain a part name to be an error. [M6.18]

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

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

where value is the content type of the targeted part.

[Note: See §13.2.4.1, "Modifications to the XML Digital Signature Specification" for additional requirements on <Reference> elements. *end note]*

[Example:

Example 13–2. Part reference with query component

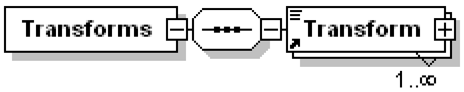
In the following example, the content type is "application/vnd.ms-package.relationships+xml".

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

end example]

13.2.4.7 <Transforms>Element

The structure of a <Transforms> element is shown in the following diagram:

diagram	
namespace	http://www.w3.org/2000/09/xmldsig#
annotation	Contains an ordered list of <Transform> elements that describe how the producer digested the <Object> data before signing it.

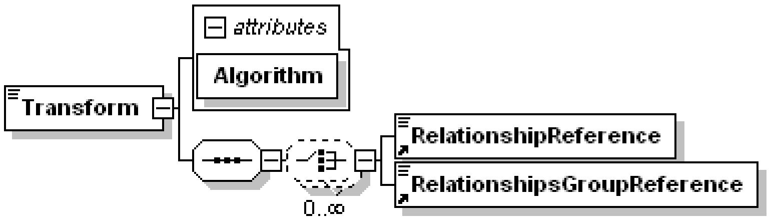
The following transforms shall be supported by producers and consumers of packages with digital signatures:

1. XML Canonicalization (c14n)
2. XML Canonicalization with Comments (c14n with comments)
3. Relationships transform (package-specific)

Consumers validating signed packages shall fail the validation if other transforms are encountered. [M6.19]

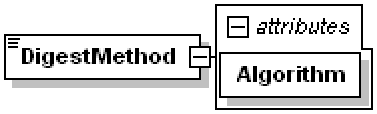
13.2.4.8 <Transform>Element

The structure of a <Transform> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr><tr><td>Algorithm</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>Contains a URI that identifies the particular transformation algorithm.</td></tr></table></td></tr>
<tr><td>annotation</td><td>Describes how the signer obtained the <Object> data that was digested.</td></tr>
</table>

13.2.4.9 <DigestMethod>Element

The structure of a <DigestMethod> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td>
</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr><tr><td>Algorithm</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>Contains a URI that identifies the particular digest method.</td></tr></table></td></tr>
<tr><td>annotation</td><td>Defines the algorithm that yields the <DigestValue> from the object data after transforms are applied. Package producers and consumers shall support RSA-SHA1 algorithms to produce or validate signatures. [M6.17]</td></tr>
</table>

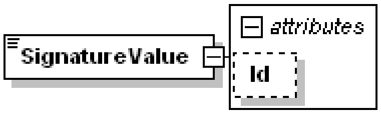
13.2.4.10 <DigestValue>Element

The structure of a <DigestValue> element is shown in the following diagram:

diagram	
namespace	http://www.w3.org/2000/09/xmldsig#
annotation	Contains the encoded value of the digest in base64.

13.2.4.11 <SignatureValue>Element

The structure of a <SignatureValue> element is shown in the following diagram:

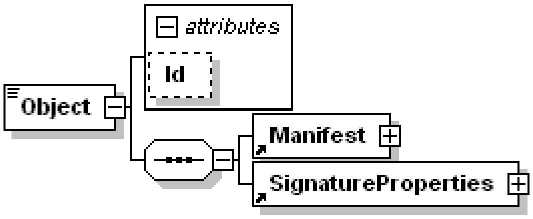
<table>
<tr><td>diagram</td><td> </td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table>
<tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr>
<tr><td>Id</td><td>xs:ID</td><td>optional</td><td></td><td></td><td>Contains a URI that identifies the <SignatureValue> element within the signature xml document.</td></tr>
</table></td></tr>
<tr><td>annotation</td><td>Contains the actual value of the digital signature in base64.</td></tr>
</table>

13.2.4.12 <Object>Element

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

13.2.4.13 Package-Specific <Object> Element

The structure of a <Object> element is shown in the following diagram:

diagram	 Object attributes Id Manifest SignatureProperties

<table>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td><table><tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr><tr><td>Id</td><td>xs:ID</td><td></td><td></td><td></td><td>Shall have value of "idPackageObject".</td></tr></table></td></tr>
<tr><td>annotation</td><td>Holds the Manifest and SignatureProperties elements that are package-specific.</td></tr>
</table>

[Note: Although the diagram above shows use of the Id attribute as optional, as does the XML Digital Signature schema, for package-specific <Object> elements, the Id attribute shall be specified and have the value of “idPackageObject”. This is a package-specific restriction over and above the XML Digital Signature schema. *end note]*

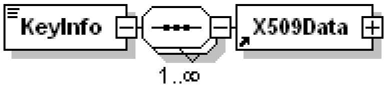
The producer shall create each <Signature> element with exactly one package-specific <Object>. For a signed package, consumers shall treat the absence of a package-specific <Object>, or the presence of multiple package-specific <Object> elements, as an invalid signature. [M6.15]

13.2.4.14 Application-Specific <Object> Element

The application-specific <Object> element specifies application-specific information. The format designer might permit one or more application-specific <Object> elements. If allowed by the format designer, format producers can create one or more application-specific <Object> elements. [O6.8] Producers shall create application-specific <Object> elements that contain XML-compliant data; consumers shall treat data that is not XML-compliant as an error. [M6.20] Format designers and producers might not apply package-specific restrictions regarding URIs and <Transform> elements to application-specific <Object> element. [O6.9]

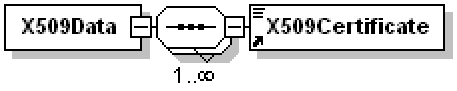
13.2.4.15 <KeyInfo>Element

The structure of a <KeyInfo> element is shown in the following diagram:

diagram	
namespace	http://www.w3.org/2000/09/xmldsig#
annotation	Enables recipients to obtain the key needed to validate the signature. Can contain keys, names, certificates, and other public key management information. Producers and consumers shall use the certificate embedded in the Digital Signature XML Signature part when it is specified. [M6.21]

13.2.4.16 <X509Data>Element

The structure of a <X509Data> element is shown in the following diagram:

diagram	
namespace	http://www.w3.org/2000/09/xmldsig#
annotation	Contains one or more identifiers of X509 certificates.

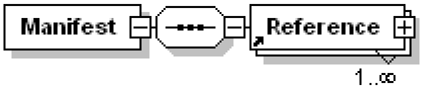
13.2.4.17 <X509Certificate>Element

The structure of a <X509Certificate> element is shown in the following diagram:

diagram	X509Certificate
namespace	http://www.w3.org/2000/09/xmldsig#
annotation	Contains a base64-encoded X509 certificate.

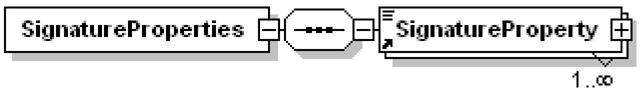
13.2.4.18 <Manifest>Element

The structure of a <Manifest> element is shown in the following diagram:

diagram	
namespace	http://www.w3.org/2000/09/xmldsig#
annotation	Contains references to the signed parts of the package. The producer shall not create a <Manifest> element that references any data outside of the package. The consumer shall consider a <Manifest> element that references data outside of the package to be in error. [M6.22]

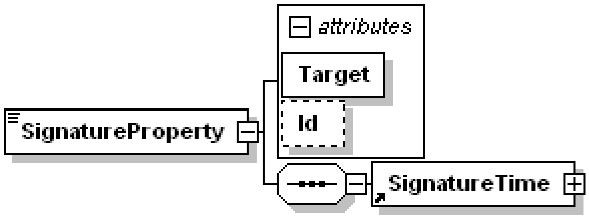
13.2.4.19 <SignatureProperties>Element

The structure of a <SignatureProperties> element is shown in the following diagram:

diagram	
namespace	http://www.w3.org/2000/09/xmldsig#
Annotation	Contains additional information items concerning the generation of signatures placed in <SignatureProperty> elements.

13.2.4.20 <SignatureProperty>Element

The structure of a <SignatureProperty> element is shown in the following diagram:

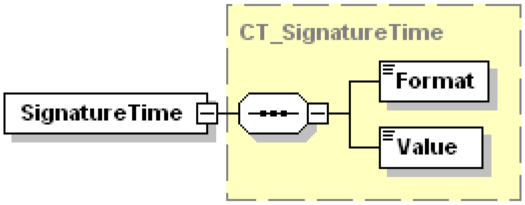
<table>
<tr><td>diagram</td><td>

attributes
Target
Id
SignatureProperty
SignatureTime

</td></tr>
<tr><td>namespace</td><td>http://www.w3.org/2000/09/xmldsig#</td></tr>
<tr><td>attributes</td><td>
<table>
<tr><th>Name</th><th>Type</th><th>Use</th><th>Default</th><th>Fixed</th><th>Annotation</th></tr>
<tr><td>Target</td><td>xs:anyURI</td><td>required</td><td></td><td></td><td>Contains a unique identifier of the Signature element.</td></tr>
<tr><td>Id</td><td>xs:ID</td><td>optional</td><td></td><td></td><td>Contains signature property’s unique identifier.</td></tr>
</table>
</td></tr>
<tr><td>annotation</td><td>Contains additional information concerning the generation of signatures.</td></tr>
</table>

13.2.4.21 <SignatureTime>Element

The structure of a <SignatureTime> element is shown in the following diagram:

diagram	
namespace	http://schemas.openxmlformats.org/package/2006/digital-signature
annotation	Holds the date/time stamp for the signature.

13.2.4.22 <Format>Element

The structure of a <Format> element is shown in the following diagram:

diagram	
namespace	http://schemas.openxmlformats.org/package/2006/digital-signature
annotation	Specifies the format of the date/time stamp. The producer shall create a data/time format that conforms to the syntax described in the W3C Note "Date and Time Formats". The consumer shall consider a format that does not conform to the syntax described in that WC3 note to be in error. [M6.23]

The date and time format definition conforms to the syntax described in the W3C Note “Date and Time Formats.”

13.2.4.23 <Value>Element

The structure of a <Value> element is shown in the following diagram:

diagram	
namespace	http://schemas.openxmlformats.org/package/2006/digital-signature
annotation	Holds the value of the date/time stamp. The producer shall create a value that conforms to the format specified in the <Format> element. The consumer shall consider a value that does not conform to that format to be in error. [M6.24]

13.2.4.24 <RelationshipReference>Element

The structure of a <RelationshipReference> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td> </td></tr>
<tr><td>namespace</td><td>http://schemas.openxmlformats.org/package/2006/digital-signature</td></tr>
<tr><td>attributes</td><td><table>
<tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr>
<tr><td>SourceId</td><td>xsd:string</td><td>required</td><td></td><td></td><td>Specifies the value of the Id attribute of the <Relationship> element.</td></tr>
</table></td></tr>
<tr><td>annotation</td><td>Specifies the <Relationship> element to be signed.</td></tr>
</table>

13.2.4.25 <RelationshipsGroupReference>Element

The structure of a <RelationshipsGroupReference> element is shown in the following diagram:

<table>
<tr><td>diagram</td><td> </td></tr>
<tr><td>namespace</td><td>http://schemas.openxmlformats.org/package/2006/digital-signature</td></tr>
<tr><td>attributes</td><td><table>
<tr><td>Name</td><td>Type</td><td>Use</td><td>Default</td><td>Fixed</td><td>Annotation</td></tr>
<tr><td>SourceType</td><td>xsd:anyURI</td><td>required</td><td></td><td></td><td>Specifies the value of the Type attribute of <Relationship> elements.</td></tr>
</table></td></tr>
<tr><td>annotation</td><td>Specifies that the group of <Relationship> elements with the specified Type value is to be signed.</td></tr>
</table>

Format designers might permit producers to sign individual relationships in a package or the Relationships part as a whole. [O6.10] To sign a subset of relationships, the producer shall use the package-specific relationships transform. The consumer shall use the package-specific relationships transform to validate the signature when a subset of relationships are signed. [M6.25] The transform filters the contents of the Relationships part to include only relationships that have Id values matching the specified SourceId values or Type values matching the specified SourceType values. A producer shall not specify more than one relationship transform for a particular relationships part. A consumer shall treat the presence of more than one relationship transform for a particular relationships part as an error. [M6.35]

Producers and consumers shall perform a canonicalization transform following the relationships transform. [M6.26]

13.2.4.26 Relationships Transform Algorithm

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

The package implementer might create relationships XML that contains content from several namespaces, along with versioning instructions as defined in Part 4: "Markup Compatibility". [O6.11]

The relationships transform algorithm is as follows:

Step 1: Process versioning instructions

1. The package implementer shall process the versioning instructions, considering that the only known namespace is the Relationships namespace.
2. The package implementer shall remove all ignorable content, ignoring preservation attributes.
3. The package implementer shall remove all versioning instructions.

Step 2: Sort and filter relationships

1. The package implementer shall remove all namespace declarations except the Relationships namespace declaration.
2. The package implementer shall remove the Relationships namespace prefix, if it is present.
3. The package implementer shall sort relationship elements by Id value in increasing order, considering Id values as case-sensitive Unicode strings.
4. The package implementer shall remove all <Relationship> elements with an Id value that does not match any SourceId values *and* with a Type value that does not match any SourceType values specified in the transform definition. Producers and consumers shall compare values byte-for-byte as case-sensitive Unicode strings. [M6.27] The resulting XML document holds all <Relationship> elements that have an Id value that matches a SourceId value *or* a Type value that matches a SourceType value specified in the transform definition.

Step 3: Prepare for canonicalization

1. The package implementer shall remove all characters between the <Relationships> start tag and the first <Relationship> start tag.
2. The package implementer shall remove all characters between each pair of <Relationship> start and end tags.
3. The package implementer shall remove all characters between the last <Relationship> end tag and the <Relationships> end tag.

4. If there are no <Relationship> elements, the package implementer shall remove all characters between the <Relationships> start tag and the <Relationships> end tag.

13.3 Digital Signature Example

The contents of digital signature parts are defined by the W3C Recommendation “XML-Signature Syntax and Processing” with some package-specific modifications specified in §13.2.4.1, “Modifications to the XML Digital Signature Specification”.

[Example:

Digital signature markup for packages is illustrated in this example. For information about namespaces used in this example, see Annex H, “Standard Namespaces and Content Types.”

```
<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.ms-package.relationships+xml">
            <Transforms>
               <Transform Algorithm="http://schemas.microsoft.com/
                  package/2005/06/RelationshipTransform">
                  <pds:RelationshipReference SourceId="B1"/>
                  <pds:RelationshipReference SourceId="A1"/>
                  <pds:RelationshipReference SourceId="A11"/>
                  <pds:RelationshipsGroupReference SourceType=
                     "http://schemas.custom.com/required-resource"/>
               </Transform>
               <Transform Algorithm="http://www.w3.org/TR/2001/
                  REC-xml-c14n-20010315"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/
               xmldsig#sha1"/>
            <DigestValue>...</DigestValue>
         </Reference>
      </Manifest>
      <SignatureProperties>
         <SignatureProperty Id="idSignatureTime" Target="#SignatureId">
            <pds:SignatureTime>
               <pds:Format>YYYY-MM-DDThh:mmTZD</pds:Format>
               <pds:Value>2003-07-16T19:20+01:00</pds:Value>
            </pds:SignatureTime>
         </SignatureProperty>
      </SignatureProperties>
   </Object>
   <Object Id="Application">...</Object>
</Signature>
```

end example]

13.4 Generating Signatures

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

The steps below might not be sufficient for signing or validating signatures that contain application-specific <Object> elements. Format designers that utilize application-specific <Object> elements shall also define the additional steps that shall be performed to sign or validate the application-specific <Object> elements.

To generate references:

1. For each package part being signed:
    a. The package implementer shall apply the transforms, as determined by the producer, to the contents of the part.
    b. The package implementer shall calculate the digest value using the resulting contents of the part.
2. The package implementer shall create a <Reference> element that includes the reference of the part with the query component matching the content type of the target part, necessary transform elements, the digest algorithm, and the <DigestValue>element.
3. The package implementer shall construct the package-specific <Object> element containing a <Manifest> element with child <Reference> elements and a <SignatureProperties> element with a child <SignatureTime> element.
4. The package implementer shall create a reference to the resulting package-specific <Object> element.

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

To generate signatures:

1. The package implementer shall create the <SignedInfo> element with a <SignatureMethod>element, a <CanonicalizationMethod> element, and at least one <Reference> element.
2. The package implementer shall canonicalize the data and then calculate the <SignatureValue> element using the <SignedInfo>element based on the algorithms specified in the <SignedInfo>element.
3. The package implementer shall construct a <Signature> element that includes <SignedInfo>, <Object>, and <SignatureValue> elements. If a certificate is embedded in the signature, the package implementer shall also include the <KeyInfo>element.

13.5 Validating Signatures

Consumers validate signatures following the steps described in §3.2, "Core Validation," of the W3C Recommendation "XML-Signature Syntax and Processing." When validating digital signatures, consumers shall verify the content type and the digest contained in each <Reference> descendant element of the <SignedInfo> element, and validate the signature calculated using the <SignedInfo> element. [M6.29]

To validate references:

1. The package implementer shall canonicalize the <SignedInfo> element based on the <CanonicalizationMethod> element specified in the <SignedInfo> element.
2. For each <Reference> element in the <SignedInfo> element:
   a. The package implementer shall obtain the <Object> element to be digested.
   b. For the package-specific <Object> element, the package implementer shall validate references to signed parts stored in the <Manifest> element. The package implementer shall consider references invalid if there is a missing part. [M6.9]
   c. For the package-specific <Object> element, validation of <Reference> elements includes verifying the content type of the referenced part and the content type specified in the reference query component. Package implementers shall consider references invalid if these two values are different. The string comparison shall be case-sensitive and locale-invariant. [M6.11]
   d. The package implementer shall digest the obtained <Object> element using the <DigestMethod> element specified in the <Reference> element.
   e. The package implementer shall compare the generated digest value against the <DigestValue> element in the <Reference> element of the <SignedInfo> element. Package implementers shall consider references invalid if there is any mismatch. [M6.30]

To validate signatures:

1. The package implementer shall obtain the public key information from the <KeyInfo> element or from an external source.
2. The package implementer shall obtain the canonical form of the <SignatureMethod> element using the <CanonicalizationMethod> element. The package implementer shall use the result and the previously obtained <KeyInfo> element to confirm the <SignatureValue> element stored in the <SignedInfo> element. The package implementer shall decrypt the <SignatureValue> element using the public key prior to comparison.

13.5.1 Signature Validation and Streaming Consumption

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

13.6 Support for Versioning and Extensibility

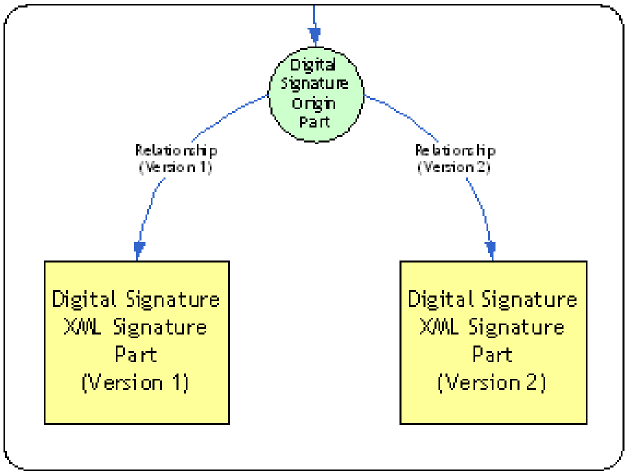
The package digital signature infrastructure supports the exchange of signed packages between current and future package clients.

13.6.1 Using Relationship Types

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

Figure 13–2, “Part names and logical item names”, illustrates this versioning capability that will be available in future versions of the package format.

Figure 13–2. A package containing versioned signatures


Digital Signature Origin Part
Relationship (Version 1)
Relationship (Version 2)
Digital Signature XML Signature Part (Version 1)
Digital Signature XML Signature Part (Version 2)

13.6.2 Markup Compatibility Namespace for Package Digital Signatures

The package implementer shall not use the Markup Compatibility namespace, as specified in Annex H, “Standard Namespaces and Content Types,” within the package-specific <Object> element. The package implementer shall consider the use of the Markup Compatibility namespace within the package-specific <Object> element to be an error. [M6.32]

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

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

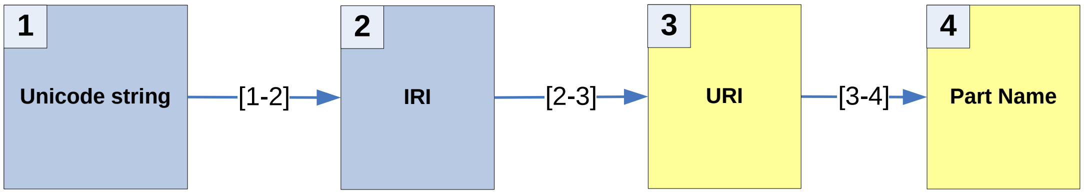
Annex A. Resolving Unicode Strings to Part Names

Package clients might use Unicode strings for referencing parts in a package. *[Example*: Values of xsd:anyURI data type within XML markup are Unicode strings. *end example]*

This annex specifies how such Unicode strings shall be resolved to part names.

The diagram below illustrates the conversion path from the Unicode string to a part name. The numbered arcs identify string transformations.

Figure A–1. Strings are converted to part names for referencing parts



A Unicode string representing a URI can be passed to the producer or consumer. The producing or consuming application shall convert the Unicode string to a URI. If the URI is a relative reference, the application shall resolve it using the base URI of the part, which is expressed using the pack scheme, to the URI of the referenced part. [M1.33]

The process for resolving a Unicode string to a part name follows Arcs [1-2], [2-3], and [3-4].

A.1 Creating an IRI from a Unicode String

With reference to Arc [1-2] in Figure A–1, a Unicode string is converted to an IRI by percent-encoding each ASCII character that does not belong to the set of reserved or unreserved characters as defined in RFC 3986.

A.2 Creating a URI from an IRI

With reference to Arc [2-3] in Figure A–1, an IRI is converted to a URI by converting non-ASCII characters as defined in Step 2 in §3.1 of RFC 3987

If a consumer converts the URI back into an IRI, the conversion shall be performed as specified in §3.2 of RFC 3987. [M1.34]

A.3 Resolving a Relative Reference to a Part Name

If the URI reference obtained in §A.2 is a URI, it is resolved in the regular way, that is, with no package-specific considerations. Otherwise, if the URI reference is a relative reference, it is resolved (with reference to Arc [3-4] in Figure A–1) as follows:

1. Percent-encode each open bracket ([) and close bracket (]).
2. Percent-encode each percent (%) character that is not followed by a hexadecimal notation of an octet value.
3. Un-percent-encode each percent-encoded unreserved character.
4. Un-percent-encode each forward slash (/) and back slash (\).
5. Convert all back slashes to forward slashes.
6. If present in a segment containing non-dot (".") characters, remove trailing dot (".") characters from each segment.
7. If a single trailing forward slash (/) is present, remove that trailing forward slash.
8. Remove complete segments that consist of three or more dots.
9. Resolve the relative reference against the base URI of the part holding the Unicode string, as it is defined in §5.2 of RFC 3986.

The producer shall not create a relative reference that would resolve to a pack URI that does not have a path component that conforms to the part name grammar, and a consumer shall issue an error if it encounters such a relative reference. [M1.35]

A.4 String Conversion Examples

[Example:

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

Unicode string	IRI	URI	Part name
/a/b.xml	/a/b.xml	/a/b.xml	/a/b.xml
/a/ц.xml	/a/ц.xml	/a/%D1%86.xml	/a/%D1%86.xml
/%41/%61.xml	/%41/%61.xml	/%41/%61.xml	/A/a.xml
/%25XY.xml	/%25XY.xml	/%25XY.xml	/%25XY.xml
/%XY.xml	/%XY.xml	/%25XY.xml	/%25XY.xml
/%2541.xml	/%2541.xml	/%2541.xml	/%2541.xml
/../a.xml	/../a.xml	/../a.xml	/a.xml
/./ц.xml	/./ц.xml	/./%D1%86.xml	/%D1%86.xml
/%2e/%2e/a.xml	/%2e/%2e/a.xml	/%2e/%2e/a.xml	/a.xml
\a.xml	%5Ca.xml	%5Ca.xml	/a.xml
\%41.xml	%5C%41.xml	%5C%41.xml	/A.xml
/%D1%86.xml	/%D1%86.xml	/%D1%86.xml	/%D1%86.xml

Unicode string	IRI	URI	Part name
\%2e/a.xml	%5C%2e/a.xml	%5C%2e/a.xml	/a.xml

end example]

Annex B. Pack URI

A package is a logical entity that holds a collection of parts. This specification defines a way to use URIs to reference part resources inside a package. This approach defines a new scheme in accordance with the guidelines in RFC 3986.

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

B.1 Pack URI Scheme

RFC 3986 provides an extensible mechanism for defining new kinds of URIs based on new schemes. Schemes are the prefix in a URI before the colon. [*Example*: "http", "ftp", "file" *end example*] This specification defines a specific URI scheme used to refer to parts in a package: the pack scheme. A URI that uses the pack scheme is called a *pack URI*.

The pack URI grammar is defined as follows:

```
pack_URI      = "pack://" authority [ "/" | path ]
authority     = *( unreserved | sub-delims | pct-encoded )
path     = 1*( "/" segment )
segment  = 1*( pchar )
```

unreserved, sub-delims, pchar and pct-encoded are defined in RFC 3986

The authority component contains an embedded URI that points to a package. The package implementer shall create an embedded URI that meets the requirements defined in RFC 3986 for a valid URI. [M7.1] §B.3, "Composing a Pack URI" describes the rules for composing pack URIs by combining the URI of an entire package resource with a part name.

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

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

In order to be able to embed the URI of the package in the pack URI, it is necessary to replace or percent-encode occurrences of certain characters in the embedded URI. For example, forward slashes (/) are replaced with commas (,). The rules for these substitutions are described in §B.3, "Composing a Pack URI".

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

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

[Example:

Example B–1. Using the pack URI to identify a part

The following URI identifies the “/a/b/foo.xml” part within the “http://www.microsoft.com/my.container” package resource:

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

end example]

[Example:

Example B–2. Equivalent pack URIs

The following pack URIs are equivalent:

```
pack://http%3c,,www.microsoft.com,my.container
pack://http%3c,,www.microsoft.com,my.container/
```

end example]

[Example:

Example B–3. A pack URI with percent-encoded characters

The following URI 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
```

end example]

B.2 Resolving a Pack URI to a Resource

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

1. Parse the pack URI into the potential three components: scheme, authority, path, as well as any fragment identifier.
2. In the authority component, replace all commas (,) with forward slashes (/).
3. Un-percent-encode ASCII characters in the resulting authority component.
4. The URI for the package identified by the pack URI is resulting authority component
5. If the path component is empty, the pack URI resolves to the package as a whole and the resolution process is complete.

6. A non-empty path component shall be a valid part name. If it is not, the pack URI is invalid.
7. The pack URI resolves to the part with this part name in the package identified by the authority component.

[Example:

Example B–4. Resolving a pack URI to a resource

Given the pack URI:

```
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 URI:

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

And the path:

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

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

end example]

B.3 Composing a Pack URI

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

In order to be suitable for creating a pack URI, the URI reference of a package resource shall conform to RFC 3986 requirements for valid absolute URIs.

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

1. Remove the fragment identifier from the package URI, if present.
2. Percent-encode all percent signs (%), question marks (?), at signs (@), colons (:) and commas (,) in the package URI.
3. Replace all forward slashes (/) with commas (,) in the resulting string.
4. Append the resulting string to the string “pack://”.
5. Append a forward slash (/) to the resulting string. The constructed string represents a pack URI with a blank path component.

6. Using this constructed string as a base URI and the part name as a relative reference, apply the rules defined in RFC 3986 for resolving relative references against the base URI.

The result of this operation will be the pack URI that refers to the resource specified by the part name.

[Example:

Example B–5. Composing a pack URI

Given the package URI:

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

And the part name:

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

The pack URI is:

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

end example]

B.4 Equivalence

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

The package implementer shall consider pack URIs equivalent if:

1. The scheme components are octet-by-octet identical after they are both converted to lowercase; *and*
2. The authority components are equivalent (the rules for which will vary by scheme, as per RFC 3986); *and*
3. The path components are equivalent when compared as case-insensitive ASCII strings.

[M7.3]

Annex C. ZIP Appnote.txt Clarifications

The ZIP specification includes a number of features that packages do not support. Some ZIP features are clarified in the context of this specification. Package producers and consumers shall adhere to the requirements noted below.

C.1 Archive File Header Consistency

Data describing files stored in the archive is substantially duplicated in the Local File Headers and Data Descriptors, and in the File headers within the Central Directory Record. For a ZIP archive to be a valid physical layer for a package, the package implementer shall ensure that the ZIP archive holds equal values in the appropriate fields of every File Header within the Central Directory and the corresponding Local File Header and Data Descriptor pair. [M3.14]

C.2 Table Key

- "Yes" — During consumption of a package, a "Yes" value for a field in a table in Annex C indicates a package implementer shall support reading the ZIP archive containing this record or field, however, support may mean ignoring. [M3.15] During production of a package, a "Yes" value for a field in a table in Annex C indicates that the package implementer shall write out this record or field. [M3.16]
- "No" — A "No" value for a field in a table in Annex C indicates the package implementer shall not use this record or field during consumption or production of packages. [M3.17]
- "Optional" — An "Optional" value for a record in a table in Annex C indicates that package implementers might write this record during production. [O3.2]
- "Partially, details below" — A "Partially, details below" value for a record in a table in Annex C indicates that the record contains fields that might not be supported by package implementers during production or consumption. See the details in the corresponding table to determine requirements. [M3.18]
- "Only used when needed" — The value "Only used when needed" associated with a record in a table in Annex C indicates that the package implementer shall use the record only when needed to store data in the ZIP archive. [M3.19]

Table C–1,"Support for records", specifies the requirements for package production, consumption, and editing in regard to particular top-level records or fields described in the ZIP Appnote.txt.

Table C–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

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

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

Table C–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 C–3)	Yes (partially, see Table C–3)	Yes (partially, see Table C–3)
	General purpose bit flag	Yes (partially, see Table C–5)	Yes (partially, see Table C–5)	Yes (partially, see Table C–5)

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	Compression method	Yes (partially, see Table C–4)	Yes (partially, see Table C–4)	Yes (partially, see Table C–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 C–6)	Yes (partially, see Table C–6)	Yes (partially, see Table C–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 (partial, always 4.5)	Yes
	Version needed to extract (see Table C–3 for details)	Yes (partially, see Table C–3)	Yes (1.0, 1.1, 2.1, 4.5)	Yes
	General purpose bit flag	Yes (partially, see Table C–5)	Yes (partially, see Table C–5)	Yes (partially, see Table C–5)
	Compression method	Yes (partially, see Table C–4)	Yes (partially, see Table C–4)	Yes (partially, see Table C–4)
	Last mod file time (Pass through, no interpretation)	Yes	Yes	Yes
	Last mod file date (Pass through, in 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

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	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 C–6)	Yes (partially, see Table C–6)	Yes (partially, see Table C–6)
	File comment (variable size)	Yes	Yes (always set to empty)	Yes
Zip64 end of central directory V1 (from spec version 4.5, only used when needed)	Zip64 end of central directory signature	Yes	Yes	Yes
	Size of zip64 end of central directory	Yes	Yes	Yes
	Version made by: high byte (Pass through, no interpretation)	Yes	Yes (0 = MS-DOS is default publishing value)	Yes
	Version made by: low byte	Yes	Yes (always 4.5)	Yes
	Version needed to extract (see Table C–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

Record	Field	Supported on Consumption	Supported on Production	Pass through on editing
	Offset of start of central directory with respect to the starting disk number	Yes	Yes	Yes
	Zip64 extensible data sector	Yes	No	Yes
Zip64 end of central directory locator (only used when needed)	Zip64 end of central dir locator signature	Yes	Yes	Yes
	Number of the disk with the start of the zip64 end of central directory	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Relative offset of the zip64 end of central directory record	Yes	Yes	Yes
	Total number of disks	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
End of central directory record	End of central dir signature	Yes	Yes	Yes
	Number of this disk	Yes (partial — no multi disk archives)	Yes (always 1 disk)	Yes (partial — no multi disk archives)
	Number of the disk with the start of the central directory	Yes (partial — no multi disk archive)	Yes (always 1 disk)	Yes (partial — no multi disk archive)
	Total number of entries in the central directory on this disk	Yes	Yes	Yes
	Total number of entries in the central directory	Yes	Yes	Yes
	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 C–3, “Support for Version Needed to Extract field”, specifies the detailed production, consumption, and editing requirements for the Extract field, which is fully described in the ZIP Appnote.txt.

Table C–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	Ignore	No	(rewrite/remove)
2.0	File is a folder (directory)	Ignore	No	(rewrite/remove)
2.0	File is compressed using Deflate compression	Yes	Yes	Yes
2.0	File is encrypted using traditional PKWARE encryption	No	No	No
2.1	File is compressed using Deflate64(tm)	No	No	No
2.5	File is compressed using PKWARE DCL Implode	No	No	No
2.7	File is a patch data set	No	No	No
4.5	File uses ZIP64 format extensions	Yes	Yes	Yes
4.6	File is compressed using BZIP2 compression	No	No	No
5.0	File is encrypted using DES	No	No	No
5.0	File is encrypted using 3DES	No	No	No
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 C–4, “Support for Compression Method field”, specifies the detailed production, consumption, and editing requirements for the Compression Method field, which is fully described in the ZIP Appnote.txt.

Table C–4. Support for Compression Method field

Code	Method	Supported on Consumption	Supported on Production	Pass through on editing
0	The file is stored (no compression)	Yes	Yes	Yes
1	The file is Shrunk	No	No	No
2	The file is Reduced with compression factor 1	No	No	No
3	The file is Reduced with compression factor 2	No	No	No
4	The file is Reduced with compression factor 3	No	No	No
5	The file is Reduced with compression factor 4	No	No	No
6	The file is Imploded	No	No	No
7	Reserved for Tokenizing compression algorithm	No	No	No
8	The file is Deflated	Yes	Yes	Yes
9	Enhanced Deflating using Deflate64™	No	No	No
10	PKWARE Data Compression Library Imploding	No	No	No
11	Reserved by PKWARE	No	No	

Table C–5, “Support for modes/structures defined by general purpose bit flags”, specifies the detailed production, consumption, and editing requirements when utilizing these general-purpose bit flags within records.

Table C–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

<table>
<tr><th>Bit</th><th>Feature</th><th>Supported on Consumption</th><th>Supported on Production</th><th>Pass through on editing</th></tr>
<tr><td>1, 2</td><td><table><tr><th>Bit 2</th><th>Bit 1</th><th></th></tr><tr><td>0</td><td>0</td><td>Normal (-en) compression option was used.</td></tr><tr><td>0</td><td>1</td><td>Maximum (-exx/-ex) compression option was used.</td></tr><tr><td>1</td><td>0</td><td>Fast (-ef) compression option was used.</td></tr><tr><td>1</td><td>1</td><td>Super Fast (-es) compression option was used.</td></tr></table></td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>3</td><td>If this bit is set, the fields crc-32, compressed size and uncompressed size are set to zero in the local header. The correct values are put in the data descriptor immediately following the compressed data. (PKZIP version 2.04g for DOS only recognizes this bit for method 8 compression, newer versions of PKZIP recognize this bit for any compression method.)</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>4</td><td>Reserved for use with method 8, for enhanced deflating</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>5</td><td>If this bit is set, this indicates that the file is compressed patched data. (Requires PKZIP version 2.70 or greater.)</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>6</td><td>Strong encryption. If this bit is set, you should set the version needed to extract value to at least 50 and you must also set bit 0. If AES encryption is used, the version needed to extract value must be at least 51.</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>7</td><td>Currently unused</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>8</td><td>Currently unused</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>9</td><td>Currently unused</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>10</td><td>Currently unused</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
<tr><td>11</td><td>Currently unused</td><td>Ignore</td><td>Bits set to 0</td><td>Yes</td></tr>
</table>

Bit	Feature	Supported on Consumption	Supported on Production	Pass through on editing
12	Reserved by PKWARE for enhanced compression	Ignore	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.	Ignore	Bits set to 0	Yes
14	Reserved by PKWARE	Ignore	Bits set to 0	Yes
15	Reserved by PKWARE	Ignore	Bits set to 0	Yes

Table C–6, “Support for Extra field (variable size), PKWARE-reserved”, specifies the detailed production, consumption, and editing requirements for the Extra field entries reserved by PKWARE and described in the ZIP Appnote.txt.

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

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x0001	ZIP64 extended information extra field	Yes	Yes	Optional
0x0007	AV Info	Ignore	No	Yes
0x0008	Reserved for future Unicode file name data (PFS)	Ignore	No	Yes
0x0009	OS/2	Ignore	No	Yes
0x000a	NTFS	Ignore	No	Yes
0x000c	OpenVMS	Ignore	No	Yes
0x000d	Unix	Ignore	No	Yes
0x000e	Reserved for file stream and fork descriptors	Ignore	No	Yes
0x000f	Patch Descriptor	Ignore	No	Yes
0x0014	PKCS#7 Store for X.509 Certificates	Ignore	No	Yes
0x0015	X.509 Certificate ID and Signature for individual file	Ignore	No	Yes

Field ID	Field description	Supported on Consumption	Supported on Production	Pass through on editing
0x0016	X.509 Certificate ID for Central Directory	Ignore	No	Yes
0x0017	Strong Encryption Header	Ignore	No	Yes
0x0018	Record Management Controls	Ignore	No	Yes
0x0019	PKCS#7 Encryption Recipient Certificate List	Ignore	No	Yes
0x0065	IBM S/390 (Z390), AS/400 (I400) attributes — uncompressed	Ignore	No	Yes
0x0066	Reserved for IBM S/390 (Z390), AS/400 (I400) attributes — compressed	Ignore	No	Yes
0x4690	POSZIP 4690 (reserved)	Ignore	No	Yes

Table C–7, “Support for Extra field (variable size), third-party extensions”, specifies the detailed production, consumption, and editing requirements for the Extra field entries reserved by third parties and described in the ZIP Appnote.txt.

Table C–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	Ignore	No	Yes
0x2605	ZipIt Macintosh	Ignore	No	Yes
0x2705	ZipIt Macintosh 1.3.5+	Ignore	No	Yes
0x2805	ZipIt Macintosh 1.3.5+	Ignore	No	Yes
0x334d	Info-ZIP Macintosh	Ignore	No	Yes
0x4341	Acorn/SparkFS	Ignore	No	Yes
0x4453	Windows NT security descriptor (binary ACL)	Ignore	No	Yes
0x4704	VM/CMS	Ignore	No	Yes
0x470f	MVS	Ignore	No	Yes
0x4b46	FWKCS MD5 (see below)	Ignore	No	Yes

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

The package implementer shall ensure that all 64-bit stream record sizes and offsets have the high-order bit = 0. [M3.20]

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

Annex D. Relationships Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema
xmlns="http://schemas.openxmlformats.org/package/2006/relationships"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://schemas.openxmlformats.org/package/2006/relations
hips" elementFormDefault="qualified" attributeFormDefault="unqualified"
blockDefault="#all">
  <xsd:element name="Relationships" type="CT_Relationships"/>
  <xsd:element name="Relationship" type="CT_Relationship"/>
  <xsd:complexType name="CT_Relationships">
    <xsd:sequence>
      <xsd:element ref="Relationship" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CT_Relationship">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="TargetMode" type="ST_TargetMode"
use="optional"/>
        <xsd:attribute name="Target" type="xsd:anyURI" use="required"/>
        <xsd:attribute name="Type" type="xsd:anyURI" use="required"/>
        <xsd:attribute name="Id" type="xsd:ID" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:simpleType name="ST_TargetMode">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="External"/>
      <xsd:enumeration value="Internal"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Annex E. Package Digital Signature Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema
xmlns="http://schemas.openxmlformats.org/package/2006/digital-signature"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:pds="http://schemas.openxmlformats.org/package/2006/digital-
signature"
targetNamespace="http://schemas.openxmlformats.org/package/2006/digital-
signature" elementFormDefault="qualified"
attributeFormDefault="unqualified" blockDefault="#all">

<!-- Individual date and time patterns
For better readability, each pattern using numbers is also described
  in a comment using one of the following pattern designators.

The actual patterns are generated by replacement by the schema
  publication process.
-->
<!--DEFINE [_y]     "([0-9][0-9][0-9][0-9])" -->
<!--DEFINE [_mo]    "((0[1-9])|(1(0|1|2)))" -->
<!--DEFINE [_d]     "((0[1-9])|(1[0-9])|(2[0-9])|(3(0|1)))" -->
<!--DEFINE [_h]     "((0[0-9])|(1[0-9])|(2(0|1|2|3)))" -->
<!--DEFINE [_ms]    "((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
9]))" -->
<!--DEFINE [_mss]   "(((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
9]))\.[0-9])" -->
<!--DEFINE [_TZD]   "(((\+|-)((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-
9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9])))|Z)" -->

  <xsd:element name="SignatureTime" type="CT_SignatureTime"/>
  <xsd:element name="RelationshipReference"
type="pds:CT_RelationshipReference"/>
  <xsd:element name="RelationshipsGroupReference"
type="pds:CT_RelationshipsGroupReference"/>
  <xsd:complexType name="CT_SignatureTime">
    <xsd:sequence>
      <xsd:element name="Format">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="(YYYY)|(YYYY-MM)|(YYYY-MM-DD)|(YYYY-MM-
DDThh:mmTZD)|(YYYY-MM-DDThh:mm:ssTZD)|(YYYY-MM-DDThh:mm:ss.sTZD)"/>
            </xsd:restriction>
          </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Value">
        <xsd:simpleType>
```

```
          <xsd:restriction base="xsd:string">
            <xsd:pattern value="(([0-9][0-9][0-9][0-9]))|(([0-9][0-9][0-
9][0-9])-((0[1-9])|(1(0|1|2))))|(([0-9][0-9][0-9][0-9])-((0[1-
9])|(1(0|1|2)))-((0[1-9])|(1[0-9])|(2[0-9])|(3(0|1))))|(([0-9][0-9][0-
9][0-9])-((0[1-9])|(1(0|1|2)))-((0[1-9])|(1[0-9])|(2[0-
9])|(3(0|1)))T((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-
9])|(3[0-9])|(4[0-9])|(5[0-9]))(((\+|-)((0[0-9])|(1[0-
9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
9])))|Z))|(([0-9][0-9][0-9][0-9])-((0[1-9])|(1(0|1|2)))-((0[1-9])|(1[0-
9])|(2[0-9])|(3(0|1)))T((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-
9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9])):((0[0-9])|(1[0-9])|(2[0-
9])|(3[0-9])|(4[0-9])|(5[0-9]))(((\+|-)((0[0-9])|(1[0-
9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
9])))|Z))|(([0-9][0-9][0-9][0-9])-((0[1-9])|(1(0|1|2)))-((0[1-9])|(1[0-
9])|(2[0-9])|(3(0|1)))T((0[0-9])|(1[0-9])|(2(0|1|2|3))):((0[0-9])|(1[0-
9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-9])):(((0[0-9])|(1[0-9])|(2[0-
9])|(3[0-9])|(4[0-9])|(5[0-9]))\.[0-9])(((\+|-)((0[0-9])|(1[0-
9])|(2(0|1|2|3))):((0[0-9])|(1[0-9])|(2[0-9])|(3[0-9])|(4[0-9])|(5[0-
9])))|Z))"/>
            <!--
            <xsd:pattern value="([_y])|\
              ([_y]-[_mo])|\
              ([_y]-[_mo]-[_d])|\
              ([_y]-[_mo]-[_d]T[_h]:[_ms][_TZD])|\
              ([_y]-[_mo]-[_d]T[_h]:[_ms]:[_ms][_TZD])|\
              ([_y]-[_mo]-[_d]T[_h]:[_ms]:[_mss][_TZD])"/>
            -->
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CT_RelationshipReference">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="SourceId" type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="CT_RelationshipsGroupReference">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="SourceType" type="xsd:anyURI"
use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

Annex F. Core Properties Schema

F.1 Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://schemas.openxmlformats.org/package/2006/metadata/
core-properties"
xmlns="http://schemas.openxmlformats.org/package/2006/metadata/core-
properties" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcterms="http://purl.org/dc/terms/" elementFormDefault="qualified"
blockDefault="#all">

<!--
schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dc.xsd"
-->
  <xs:import namespace="http://purl.org/dc/elements/1.1/" />

<!--
schemaLocation="http://dublincore.org/schemas/xmls/qdc/2003/04/02/dcterms
.xsd" -->
  <xs:import namespace="http://purl.org/dc/terms/" />

  <xs:element name="coreProperties" type="CT_coreProperties" />

  <xs:complexType name="CT_coreProperties">
    <xs:all>
      <xs:element name="category" minOccurs="0" maxOccurs="1"
type="xs:string" />
      <xs:element name="contentStatus" minOccurs="0" maxOccurs="1"
type="xs:string" />
      <xs:element name="contentType" minOccurs="0" maxOccurs="1"
type="xs:string" />
      <xs:element ref="dcterms:created" minOccurs="0" maxOccurs="1" />
      <xs:element ref="dc:creator" minOccurs="0" maxOccurs="1" />
      <xs:element ref="dc:description" minOccurs="0" maxOccurs="1" />
      <xs:element ref="dc:identifier" minOccurs="0" maxOccurs="1" />
      <xs:element name="keywords" minOccurs="0" maxOccurs="1"
type="xs:string" />
      <xs:element ref="dc:language" minOccurs="0" maxOccurs="1" />
      <xs:element name="lastModifiedBy" minOccurs="0" maxOccurs="1"
type="xs:string" />
      <xs:element name="lastPrinted" minOccurs="0" maxOccurs="1"
type="xs:dateTime" />
      <xs:element ref="dcterms:modified" minOccurs="0" maxOccurs="1" />
      <xs:element name="revision" minOccurs="0" maxOccurs="1"
type="xs:string" />
      <xs:element ref="dc:subject" minOccurs="0" maxOccurs="1" />
      <xs:element ref="dc:title" minOccurs="0" maxOccurs="1" />
      <xs:element name="version" minOccurs="0" maxOccurs="1"
type="xs:string" />
    </xs:all>
  </xs:complexType>
</xs:schema>
```

F.2 Restrictions

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

1. Producers shall not create a document element that contains refinements to the Dublin Core elements, except for the two specified in the schema: <dcterms:created> and <dcterms:modified> Consumers shall consider a document element that violates this constraint to be an error. [M4.3]
2. Producers shall not create a document element that contains the xml:lang attribute. Consumers shall consider a document element that violates this constraint to be an error. [M4.4] For Dublin Core elements, this restriction is enforced by applications.
3. Producers shall not create a document element that contains the xsi:type attribute, except for a <dcterms:created> or <dcterms:modified> element where the xsi:type attribute shall be present and shall hold the value “dcterms:W3CDTF.” Consumers shall consider a document element that violates this constraint to be an error. [M4.5]

Annex G. Content Types Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema
targetNamespace="http://schemas.openxmlformats.org/package/2006/content-
types" elementFormDefault="qualified" attributeFormDefault="unqualified"
blockDefault="#all"
xmlns="http://schemas.openxmlformats.org/package/2006/content-types"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Individual patterns for content type grammar

For better readability, each pattern using numbers is also described in
  a comment using one of the following pattern designators.

  The actual patterns are generated by replacement by the schema
  publication process.
-->

  <!--DEFINE [media-type]
"([type]/[subtype]([LWS]*;[LWS]*[parameter])*)" -->
  <!--DEFINE [parameter]     "([attribute]=[value])" -->
  <!--DEFINE [attribute]     "([token])" -->
  <!--DEFINE [value]         "([token]|[quoted-string])" -->
  <!--DEFINE [subtype]       "([token])" -->
  <!--DEFINE [type]          "([token])" -->
  <!--DEFINE [quoted-string] "(&quot;([qdtext]|[quoted-pair])*&quot;)" --
>
  <!--DEFINE [qdtext]        "([^\p{Cc}[DEL]&quot;\n\r]|[LWS])" -->
  <!--DEFINE [quoted-pair]   "(\\[CHAR])" -->

  <!--DEFINE [LWS]           "([SP]+)" -->
  <!--DEFINE [CHAR]          "[\p{IsBasicLatin}]" -->
  <!--DEFINE [TEXT]          "[^\p{Cc}[DEL]]" -->
  <!--DEFINE [separators]    "[\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\} \t]"
-->
  <!--DEFINE [token]
"([^\p{Cc}[DEL]\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]+)" -->
  <!--DEFINE [SP]            "\s" -->
  <!--DEFINE [DEL]           "&#127;" -->

  <xs:element name="Types" type="CT_Types" />

  <xs:complexType name="CT_Types">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Default" type="CT_Default" />
      <xs:element name="Override" type="CT_Override" />
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="CT_Default">
    <xs:attribute name="Extension" type="ST_Extension" use="required" />
    <xs:attribute name="ContentType" type="ST_ContentType"
use="required"/>
  </xs:complexType>

  <xs:complexType name="CT_Override">
    <xs:attribute name="ContentType" type="ST_ContentType"
use="required"/>
    <xs:attribute name="PartName" type="xs:anyURI" use="required" />
  </xs:complexType>

  <xs:simpleType name="ST_ContentType">
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse" />
      <!--
        <xs:pattern value="[media-type]"/>
      -->
      <xs:pattern
value="((([^\p{Cc}□\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]+))/(([^\p{Cc
}□\(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]+))((\s+)*;(\s+)*((([^\p{Cc}□\
(\)&lt;&gt;@,;:\\&quot;/\[\]\?=\{\}\s\t]+))=(([^\p{Cc}□\(\)&lt;&gt;@,;:\\
&quot;/\[\]\?=\{\}\s\t]+)|(&quot;(([^\p{Cc}□&quot;\n\r]|(\s+))|(\\[\p{IsB
asicLatin}]))*&quot;))))*)" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_Extension">
    <xs:restriction base="xs:string">
      <xs:pattern value="([!$&amp;'\(\)\*\+,:=]|(%[0-9a-fA-F][0-9a-fA-
F])|[:@]|[a-zA-Z0-9\-_~])*" />
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Annex H. Standard Namespaces and Content Types

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

Table H–1. Package-wide namespaces

Description	Namespace URI
Content Types	http://schemas.openxmlformats.org/package/2006/content-types
Core Properties	http://schemas.openxmlformats.org/package/2006/metadata/core-properties
Digital Signatures	http://schemas.openxmlformats.org/package/2006/digital-signature
Relationships	http://schemas.openxmlformats.org/package/2006/relationships
Markup Compatibility	http://schemas.openxmlformats.org/markup-compatibility/2006

The content types available for use in a package are listed in Table H–2, Package-wide content types

Table H–2. Package-wide content types

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

Package implementers and format designers shall not allow content types for the package-specific parts defined in this specification to include parameters. [M1.22]

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

Table H–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

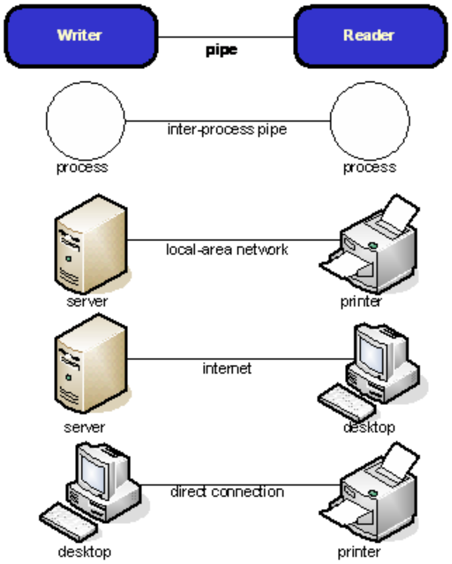
Description	Relationship Type
Digital Signature Certificate	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/certificate
Digital Signature Origin	http://schemas.openxmlformats.org/package/2006/relationships/digital-signature/origin
Thumbnail	http://schemas.openxmlformats.org/package/2006/relationships/metadata/thumbnail

Annex I. Physical Model Design Considerations

This annex is informative.

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

Figure I–1. Components of the physical model



A *producer* is a piece of software or a device that *writes* packages. A *consumer* is a piece of software or a device that *reads* packages. A *device* is a piece of 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 is simply the API calls that the consumer makes to read the package from the local file system.

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

In order to maximize performance, physical package designs consider access style, layout style, and communication style.

I.1 Access Styles

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

I.1.1 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.

I.1.2 Streaming Consumption

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

I.1.3 Streaming Creation

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

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.

I.1.4 Simultaneous Creation and Consumption

Simultaneous creation and consumption allows streaming creation and streaming consumption to happen at the same time on a package. Supporting this access style can push the design of a physical format in opposing directions. However, 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.

I.2 Layout Styles

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 or interleaved.

I.2.1 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 part *n* appear in the package before the bytes for part *n*+1.

I.2.2 Interleaved Ordering

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

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

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

I.3 Communication Styles

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

I.3.1 Sequential Delivery

With *sequential delivery*, all of part *n* is delivered to a consumer before part *n*+1. Generally, all pipes support sequential delivery.

I.3.2 Random Access

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

End of informative text.

Annex J. Conformance Requirements

This annex is informative.

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

Conformance requirements are divided into tables based on their general topic below. The tables contain the requirements that producers and consumers shall follow, those that they should follow, and those that are optional. Each conformance requirement is given a unique ID comprised of a letter (M – MANDATORY; S – SHOULD; O – OPTIONAL), an identifier for the topic it relates to, and a unique ID within that topic. Producers and consumers might use these IDs to report error conditions.

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

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

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

J.1 Package Model

Table J–1. Package model conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.1	The package implementer shall require a part name.	9.1, 9.1.1	×			
M1.2	The package implementer shall require a content type and the format designer shall specify the content type.	9.1	×	×		
M1.3	A part name shall not have empty segments.	9.1.1.1	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.4	A part name shall start with a forward slash (“/”) character.	9.1.1.1	×			
M1.5	A part name shall not have a forward slash as the last character.	9.1.1.1	×			
M1.6	A segment shall not hold any characters other than pchar characters. .	9.1.1.1	×			
M1.7	A segment shall not contain percent-encoded forward slash (“/”), or backward slash (“\”) characters.	9.1.1.1	×			
M1.8	A segment shall not contain percent-encoded unreserved characters.	9.1.1.1	×			
M1.9	A segment shall not end with a dot (“.”) character.	9.1.1.1	×			
M1.10	A segment shall include at least one non-dot character	9.1.1.1	×			
M1.11	A package implementer shall neither create nor recognize a part with a part name derived from another part name by appending segments to it.	9.1.1.2	×			
M1.12	Part name equivalence is determined by comparing part names as case-insensitive ASCII strings. Packages shall not contain equivalent part names and package implementers shall neither create nor recognize packages with equivalent part names.	9.1.1.3	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.13	Package implementers shall only create and only recognize parts with a content type; format designers shall specify a content type for each part included in the format. Content types for package parts shall fit the definition and syntax for media types as specified in RFC 2616, §3.7.	9.1.2	×	×		
M1.14	Content types shall not use linear white space either between the type and subtype or between an attribute and its value. Content types also shall not have leading or trailing white spaces. Package implementers shall create only such content types and shall require such content types when retrieving a part from a package; format designers shall specify only such content types for inclusion in the format.	9.1.2	×	×		
M1.15	The package implementer shall require a content type that does not include comments and the format designer shall specify such a content type.	9.1.2	×	×		
M1.16	If the package implementer specifies a growth hint, it is set when a part is created and the package implementer shall not change the growth hint after the part has been created.	9.1.3	×		×	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.17	XML content shall be encoded using either UTF-8 or UTF-16. If any part includes an encoding declaration (as defined in §4.3.3 of the XML specification), that declaration shall not name any encoding other than UTF-8 or UTF-16. Package implementers shall enforce this requirement upon creation and retrieval of the XML content.	9.1.4	×			
M1.18	DTD content shall not be used in the XML markup defined in this specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content and shall treat the presence of DTD content as an error.	9.1.4	×			
M1.19	If the XML content contains the Markup Compatibility namespace, as described in Part 4: "Markup Compatibility", it shall be processed by the package implementer to remove Markup Compatibility elements and attributes and ignorable namespaces before applying further validation rules below.	9.1.4	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.20	XML content shall be valid according to the corresponding XSD schema defined in this specification. In particular, the XML content shall not contain namespaces that are not explicitly defined in the corresponding XSD unless the XSD allows any namespace to be present in particular locations in the XML markup. Package implementers shall enforce this requirement upon creation and retrieval of the XML content.	9.1.4	×			
M1.21	XML content shall not contain elements or attributes drawn from “xml” or “xsi” namespaces unless they are explicitly defined in the XSD schema or by other means described in the specification. Package implementers shall enforce this requirement upon creation and retrieval of the XML content.	9.1.4	×			
M1.22	Package implementers and format designers shall not allow content types for the package-specific parts defined in this specification to include parameters.	Annex H	×	×		
M1.23	XML markup might contain Unicode strings referencing other parts as a value of the xsd:anyURI data type. Format consumers shall convert these Unicode strings to URIs, as defined in Annex A, “Resolving Unicode Strings to Part Names,” before resolving them relative to the base URI of the part containing the Unicode string.	9.2.1				×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.24	Some types of content provide a way to override the default base URI by specifying a different base in the content. In the presence of one of these overrides, format consumers shall use the specified base URI instead of the default.	9.2.1				×
M1.25	The Relationships part shall not have relationships to any other part. Package implementers shall enforce this requirement upon the attempt to create such a relationship and shall treat any such relationship as invalid.	9.3.1	×			
M1.26	The package implementer shall require that every <Relationship> element has an Id attribute, the value of which is unique within the Relationships part, and that the Id type is xsd:ID, the value of which conforms to the naming restrictions for xsd:ID as described in the W3C Recommendation "XML Schema Part 2: Datatypes."	9.3.3	×			
M1.27	The package implementer shall require the Type attribute to be a URI that defines the role of the relationship and the format designer shall specify such a Type.	9.3.3.2	×	×		
M1.28	The package implementer shall require the Target attribute to be a URI reference pointing to a target resource. The URI reference shall be a URI or a relative reference.	9.3.3.2	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.29	When set to Internal, the Target attribute shall be a relative reference and that reference is interpreted relative to the “parent” part. For package relationships, the package implementer shall resolve relative references in the Target attribute against the pack URI that identifies the entire package resource.	9.3.3.2	×			
M1.30	The package implementer shall name relationship parts according to the special relationships part naming convention and require that parts with names that conform to this naming convention have the content type for a Relationships part	9.3.4	×			
M1.31	Consumers shall process relationship markup in a manner that conforms to Part 5: "Markup Compatibility". Producers editing relationships based on this version of the relationship markup specification shall not preserve any ignored content, regardless of the presence of any preservation attributes as defined in Part 5: "Markup Compatibility".	9.3.5			×	×
M1.32	If a fragment identifier is allowed in the Target attribute of the <Relationship> element, a package implementer shall not resolve the URI to a scope less than an entire part.	9.3.3.2	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M1.33	A Unicode string representing a URI can be passed to the producer or consumer. The producing or consuming application shall convert the Unicode string to a URI. If the URI is a relative reference, the application shall resolve it using the base URI of the part, which is expressed using the pack scheme, to the URI of the referenced part.	Annex A			×	×
M1.34	If a consumer converts the URI back into an IRI, the conversion shall be performed as specified in §3.2 of RFC 3987.	A.2				×
M1.35	The producer shall not create a relative reference that would resolve to a pack URI that does not have a path component that conforms to the part name grammar, and a consumer shall issue an error if it encounters such a relative reference.	A.3			×	×

Table J–2. Package model optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O1.1	The package implementer might allow a growth hint to be provided by a producer.	9.1, 9.1.3	×			
O1.2	Format designers might restrict the usage of parameters for content types.	9.1.2		×		
O1.3	The package implementer might ignore the growth hint or adhere only loosely to it when specifying the physical mapping.	9.1.3	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O1.4	If the format designer permits it, parts can contain Unicode strings representing references to other parts. If allowed by the format designer, format producers can create such parts and format consumers shall consume them.	9.2.1		×	×	×
O1.5	The package implementer might allow a TargetMode to be provided by a producer.	9.3.3.2	×			
O1.6	A format designer might allow fragment identifiers in the value of the Target attribute of the <Relationship> element.	9.3.3.2		×		
O1.7	Producers might generate relationship markup that uses the versioning and extensibility mechanisms defined in Part 5: "Markup Compatibility" to incorporate elements and attributes drawn from other XML namespaces.	9.3.5			×	

1 ## J.2 Physical Packages

2 Table J–3. Physical packages conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.1	The Content Types stream shall not be mapped to a part by the package implementer.	10.1.2.1	×A			
M2.2	The package implementer shall define a physical package format with a mapping for the required components package, part name, part content type and part contents.	10.1.1	×			
M2.3	The package implementer shall define a physical package format mapping with a mechanism for associating content types with parts.	10.1.2.1	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.4	The package implementer shall require that the Content Types stream contain one of the following for every part in the package: One matching <Default> element One matching <Override> element Both a matching <Default> element and a matching <Override> element, in which case the <Override> element takes precedence.	10.1.2.2	×A			
M2.5	The package implementer shall require that there not be more than one <Default> element for any given extension, and there not be more than one <Override> element for any given part name.	10.1.2.2	×A			
M2.6	The package implementer shall require a non-empty extension in a <Default> element. The package implementer shall require a content type in a <Default> element and the format designer shall specify the content type.	10.1.2.2.2	×A	×A		
M2.7	The package implementer shall require a content type and the format designer shall specify the content type in an <Override> element. The package implementer shall require a part name.	10.1.2.2.3	×A	×A		
M2.8	When adding a new part to a package, the package implementer shall ensure that a content type for that part is specified in the Content Types stream; the package implementer shall perform the steps described in §10.1.2.3, “Setting the Content Type of a Part”.	10.1.2.3	×A			
M2.9	To get the content type of a part, the package implementer shall perform the steps described in §10.1.2.4, “Getting the Content Type of a Part”.	10.1.2.4	×A			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M2.10	The package implementer shall not use the versioning and extensibility mechanisms defined in Part 5: "Markup Compatibility" to incorporate elements and attributes drawn from other XML-namespaces into the Content Types stream markup.	10.1.2.5	×A			
M2.11	The package implementer shall not mix interleaving and non-interleaving for an individual part.	10.1.4	×B			
M2.12	The package implementer shall compare prefix names as case-insensitive ASCII strings.	10.1.3.1	×			
M2.13	The package implementer shall compare suffix names as case-insensitive ASCII strings.	10.1.3.1	×B			
M2.14	The package implementer shall not allow packages that contain equivalent logical item names.	10.1.3.1	×			
M2.15	The package implementer shall not allow packages that contain logical items with equivalent prefix names and with equal piece numbers, where piece numbers are treated as integer decimal values.	10.1.3.1	×B			
M2.16	The package implementer shall not map logical items to parts if the logical item names violate the part naming rules, even if a third-party format includes these logical items in the package.	10.1.3.4	×			
M2.17	The package implementer shall consider naming collisions within the set of part names mapped from logical item names to be an error.	10.1.3.4	×			
M2.18	When interleaved, a package implementer shall represent a part as one or more pieces, using the method described in §10.1.4, “Interleaving”.	10.2.1	×B			

Notes:

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

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

Table J–4. Physical packages recommendations

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S2.1	Some physical package formats have a native mechanism for representing content types. For such packages, the package implementer should use the native mechanism to map the content type for a part.	10.1.2.1	×			
S2.2	If no native method of mapping a content type to a part exists, the package implementer should include a specially-named XML stream in the package called the Content Types stream	10.1.2.1	×			
S2.3	If the package is intended for streaming consumption: The package implementer should not allow <Default> elements; as a consequence, there should be one <Override> element for each part in the package. The format producer should write the <Override> elements to the package so they appear before the parts to which they correspond, or in close proximity to the part to which they correspond.	10.1.2.2	×A		×A	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S2.4	The package implementer should use the mechanism described in this Standard to allow interleaving when mapping to the physical package for layout scenarios that support streaming consumption.	10.1.4	×B			
S2.5	The package implementer should store pieces in their natural order for optimal efficiency.	10.1.4	×B			

Notes:

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

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

Table J–5. Physical packages optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O2.1	The format designer specifies whether that format might use interleaving.	10.1.4		×		
O2.2	Optional. The package implementer might provide a physical mapping for a growth hint that might be specified by a producer.	10.1.1	×			
O2.3	Package implementers might use the common mapping solutions defined in this Standard.	10.1	×			
O2.4	Package producers can use pre-defined <Default> elements to reduce the number of <Override> elements on a part, but are not required to do so.	10.1.2.2			×A	
O2.5	The package implementer can define <Default> content type mappings even though no parts use them.	10.1.2.2	×A			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O2.6	The package implementer might create a physical package containing interleaved parts and non-interleaved parts.	10.1.4	×			
O2.7	The package implementer might allow a package that contains logical item names and complete sequences of logical item names that cannot be mapped to a part name.	10.1.3.4	×B			

Notes:

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

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

J.3 ZIP Physical Mapping

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

Table J–6. ZIP physical mapping conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.1	A package implementer shall store a non-interleaved part as a single ZIP item.	10.2.1	×			
M3.2	ZIP item names are case-sensitive ASCII strings. Package implementers shall create ZIP item names that conform to ZIP archive file name grammar.	10.2.2	×			
M3.3	Package implementers shall create item names that are unique within a given archive.	10.2.2	×			
M3.4	To map part names to ZIP item names the package implementer shall perform, in order, the steps described in §10.2.3, “Mapping Part Names to ZIP Item Names”.	10.2.3	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.5	The package implementer shall not map a logical item name or complete sequence of logical item names sharing a common prefix to a part name if the logical item prefix has no corresponding content type.	10.2.3	×			
M3.6	To map ZIP item names to part names, the package implementer shall perform, in order, the steps described in §10.2.4, "Mapping ZIP Item Names to Part Names".	10.2.4	×			
M3.7	The package implementer shall only find ZIP items that are recognized as MS-DOS files according to the ZIP specification as eligible to be mapped to parts; all other ZIP items shall not be mapped to parts.	10.2.5	×			
M3.8	The package implementer shall only find ZIP items that are recognized as MS-DOS files according to the ZIP specification as eligible to be mapped to parts; all other ZIP items shall not be mapped to parts. [M3.7] *[Note:* The ZIP specification specifies that ZIP items recognized as MS-DOS files are those with a "version made by" field and an "external file attributes" field in the "file header" record in the central directory that have a value of 0. *end note]* In ZIP archives, the package implementer shall not exceed 65,515 bytes for the combined length of the item name, Extra field, and Comment fields.	10.2.5	×			
M3.9	ZIP-based packages shall not include encryption as described in the ZIP specification. Package implementers shall enforce this restriction.	10.2.5	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.10	Package implementers shall store content type data in an item(s) mapped to the logical item name with the prefix_name equal to "/[Content_Types].xml" or in the interleaved case to the complete sequence of logical item names with that prefix_name.	10.2.6	×			
M3.11	Package implementers shall not map logical item name(s) mapped to the Content Types stream in a ZIP archive to a part name.	10.2.6	×			
M3.12	If a growth hint is used for an interleaved part, the package implementer shall store the Extra field containing the growth hint padding with the item that represents the first piece of the part.	10.2.7	×			
M3.13	Several substantial conditions that represent a package unfit for streaming consumption may be detected mid-processing by a streaming package implementer, described in §10.2.8, "Late Detection of ZIP Items Unfit for Streaming Consumption". When any of these conditions are detected, the streaming package implementer shall generate an error, regardless of any processing that has already taken place. Package implementers shall not generate a package containing any of these conditions when generating a package intended for streaming consumption.	10.2.8	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.14	For a ZIP archive to be a valid physical layer for a package, the package implementer shall ensure that the ZIP archive holds equal values in the appropriate fields of every File Header within the Central Directory and the corresponding Local File Header and Data Descriptor pair.	Annex C	×			
M3.15	During consumption of a package, a "Yes" value for a field in a table in Annex C indicates a package implementer shall support reading the ZIP archive containing this record or field, however, support may mean ignoring.	Annex C	×			
M3.16	During production of a package, a “Yes” value for a field in a table in Annex C indicates that the package implementer shall write out this record or field.	Annex C	×			
M3.17	A “No” value for a field in a table in Annex C indicates the package implementer shall not use this record or field during consumption or production of packages.	Annex C	×			
M3.18	A “Partially, details below” value for a record in a table in Annex C indicates that the record contains fields that might not be supported by package implementers during production or consumption. See the details in the corresponding table to determine requirements.	Annex C	×			
M3.19	The value “Only used when needed” associated with a record in a table in Annex C indicates that the package implementer shall use the record only when needed to store data in the ZIP archive.	Annex C	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M3.20	The value “Only used when needed” associated with a record in a table in Annex C indicates that the package implementer shall use the record only when needed to store data in the ZIP archive.	Annex C	×			
M3.21	The package implementer shall ensure that all 64-bit stream record sizes and offsets have the high-order bit = 0.	Annex C	×			

Notes:

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

Table J–7. ZIP physical mapping recommendations

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S3.1	Package implementers should restrict part naming to accommodate file system limitations when naming parts to be stored as ZIP items.	10.2.5	×			

Table J–8. ZIP physical mapping optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O3.1	A package implementer might intentionally order the sequence of ZIP items in the archive to enable an efficient organization of the part data in order to achieve correct and optimal interleaving.	10.2.1	×			
O3.2	An “Optional” value for a record in a table in Annex C indicates that package implementers might write this record during production.	Annex C	×			

J.4 Core Properties

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

Table J–9. Core properties conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M4.1	The format designer shall specify and the format producer shall create at most one core properties relationship for a package. A format consumer shall consider more than one core properties relationship for a package to be an error. If present, the relationship shall target the Core Properties part.	11.1.2		×	×	×
M4.2	The format designer shall not specify and the format producer shall not create Core Properties that use the Markup Compatibility namespace as defined in Annex H, "Standard Namespaces and Content Types". A format consumer shall consider the use of the Markup Compatibility namespace to be an error.	11.1.3		×	×	×
M4.3	Producers shall not create a document element that contains refinements to the Dublin Core elements, except for the two specified in the schema: <dcterms:created> and <dcterms:modified> Consumers shall consider a document element that violates this constraint to be an error.	F.2			×	×
M4.4	Producers shall not create a document element that contains the xml:lang attribute. Consumers shall consider a document element that violates this constraint to be an error.	F.2			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M4.5	Producers shall not create a document element that contains the xsi:type attribute, except for a <dcterms:created> or <dcterms:modified> element where the xsi:type attribute shall be present and shall hold the value "dcterms:W3CDTF." Consumers shall consider a document element that violates this constraint to be an error.	F.2			×	×

J.5 Thumbnail

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

Table J–10. Thumbnail conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M5.1	The format designer shall specify thumbnail parts that are identified by either a part relationship or a package relationship. The producer shall build the package accordingly.	12.1		×	×	

Table J–11. Thumbnail optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O5.1	The format designer might allow images, called thumbnails, to be used to help end-users identify parts of a package or a package as a whole. These images are generated by the producer and stored as parts.	12		×	×	

J.6 Digital Signatures

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

Table J–12. Digital Signatures conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.1	The package implementer shall include only one Digital Signature Origin part in a package and it shall be targeted from the package root using the well-defined relationship type specified in Annex H, “Standard Namespaces and Content Types”.	13.2.1	×			
M6.2	When creating the first Digital Signature XML Signature part, the package implementer shall create the Digital Signature Origin part, if it does not exist, in order to specify a relationship to that Digital Signature XML Signature part.	13.2.1	×			
M6.3	The producer shall create Digital Signature XML Signature parts that have a relationship from the Digital Signature Origin part and the consumer shall use that relationship to locate signature information within the package.	13.2.1			×	×
M6.4	If the certificate is represented as a separate part within the package, the producer shall target that certificate from the appropriate Digital Signature XML Signature part by a Digital Signature Certificate relationship as specified in Annex H, “Standard Namespaces and Content Types” and the consumer shall use that relationship to locate the certificate.	13.2.3			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.5	The producer shall create <Reference> elements within a <SignedInfo> element that reference elements within the same <Signature> element. The consumer shall consider <Reference> elements within a <SignedInfo> element that reference any resources outside the same <Signature> element to be in error.	13.2.4.1			×	×
M6.6	The producer shall not create a reference to a package-specific <Object> element that contains a transform other than a canonicalization transform. The consumer shall consider a reference to a package-specific <Object> element that contains a transform other than a canonical transform to be an error.	13.2.4.1			×	×
M6.7	The producer shall create one and only one package-specific <Object> element in the <Signature> element. The consumer shall consider zero or more than one package-specific <Object> element in the <Signature> element to be an error.	13.2.4.1			×	×
M6.8	The producer shall create package-specific <Object> elements that contain only <Manifest> and <SignatureProperties> elements. The consumer shall consider package-specific <Object> elements that contain other types of elements to be an error.	13.2.4.1			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.9	The producer shall create <Reference> elements within a <Manifest> element that reference with their URI attribute only parts within the package. The consumer shall consider <Reference> elements within a <Manifest> element that reference resources outside the package to be an error.	13.2.4.1			×	×
M6.10	The producer shall create relative references to the local parts that have query components that specifies the part content type as described in §13.2.4.6, "<Reference>Element". The consumer shall consider a relative reference to a local part that has a query component that incorrectly specifies the part content type to be an error.	13.2.4.1			×	×
M6.11	The producer shall create <Reference> elements with a query component that specifies the content type that matches the content type of the referenced part. The consumer shall consider signature validation to fail if the part content type does not match the content type specified in the query component of the part reference.	13.2.4.1			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.12	The producer shall not create <Reference> elements within a <Manifest> element that contain transforms other than the canonicalization transform and relationships transform. The consumer shall consider <Reference> elements within a <Manifest> element that contain transforms other than the canonicalization transform and relationships transform to be in error.	13.2.4.1			×	×
M6.13	A producer that uses an optional relationships transform shall follow it by a canonicalization transform. The consumer shall consider any relationships transform that is not followed by a canonicalization transform to be an error.	13.2.4.1			×	×
M6.14	The producer shall create only <SignatureProperty> elements that contain a <SignatureTime> element. The consumer shall consider a <SignatureProperty> element that does not contain a <SignatureTime> element to be in error.	13.2.4.1			×	×
M6.15	The producer shall create a <Signature> element that contains one local-data, package-specific <Object> element and zero or more application-specific <Object> elements. If a <Signature> element violates this constraint, a consumer shall consider this to be an error.	13.2.4.2			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.16	The producer shall create a <SignedInfo> element that contains a reference to the package-specific <Object> element. The consumer shall consider it an error if a <SignedInfo> element does not contain a reference to the package-specific <Object> element.	13.2.4.3			×	×
M6.17	Producers shall support DSA and RSA algorithms to produce signatures. Consumers shall support DSA and RSA algorithms to validate signatures.	13.2.4.5			×	×
M6.18	The producer shall create a <Reference> element within a <Manifest> element with a URI attribute and that attribute shall contain a part name, without a fragment identifier. The consumer shall consider a <Reference> element with a URI attribute that does not contain a part name to be an error.	13.2.4.6			×	×
M6.19	The following transforms shall be supported by producers and consumers of packages with digital signatures: 8. XML Canonicalization (c14n) 9. XML Canonicalization with Comments (c14n with comments) 10. Relationships transform (package-specific) Consumers validating signed packages shall fail the validation if other transforms are encountered.	13.2.4.7			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.20	Producers shall create application-specific <Object> elements that contain XML-compliant data; consumers shall treat data that is not XML-compliant as an error.	13.2.4.14			×	×
M6.21	Producers and consumers shall use the certificate embedded in the Digital Signature XML Signature part when it is specified.	13.2.4.15			×	×
M6.22	The producer shall not create a <Manifest> element that references any data outside of the package. The consumer shall consider a <Manifest> element that references data outside of the package to be in error.	13.2.4.18			×	×
M6.23	The producer shall create a data/time format that conforms to the syntax described in the W3C Note "Date and Time Formats". The consumer shall consider a format that does not conform to the syntax described in that WC3 note to be in error.	13.2.4.22			×	×
M6.24	The producer shall create a value that conforms to the format specified in the <Format> element. The consumer shall consider a value that does not conform to that format to be in error.	13.2.4.23			×	×
M6.25	To sign a subset of relationships, the producer shall use the package-specific relationships transform. The consumer shall use the package-specific relationships transform to validate the signature when a subset of relationships are signed.	13.2.4.25			×	×
M6.26	Producers and consumers shall perform a canonicalization transform following the relationships transform.	13.2.4.25			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.27	When applying a relationships transform for digital signatures, the package implementer shall remove all <Relationship> elements with an Id value that does not match any SourceId values *and* with a Type value that does not match any SourceType values specified in the transform definition. Producers and consumers shall compare values byte-for-byte as case-sensitive Unicode strings.	13.2.4.26			×	×
M6.28	When signing <Object> element data, package implementers shall follow the generic reference creation algorithm described in §3.1, “Core Generation,” of the W3C Recommendation “XML-Signature Syntax and Processing”.	13.4	×			
M6.29	When validating digital signatures, consumers shall verify the content type and the digest contained in each <Reference> descendant element of the <SignedInfo> element, and validate the signature calculated using the <SignedInfo> element.	13.5				×
M6.30	The package implementer shall compare the generated digest value against the <DigestValue> element in the <Reference> element of the <SignedInfo> element. Package implementers shall consider references invalid if there is any mismatch.	13.5	×			
M6.31	Streaming consumers that maintain signatures shall be able to cache the parts necessary for detecting and processing signatures.	13.5.1				×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.32	The package implementer shall not use the Markup Compatibility namespace, as specified in Annex H, “Standard Namespaces and Content Types,” within the package-specific <Object> element. The package implementer shall consider the use of the Markup Compatibility namespace within the package-specific <Object> element to be an error.	13.6.2	×			
M6.33	If an application allows for a single part to contain information that might not be fully understood by all implementations, then the format designer shall carefully design the signing and verification policies to account for the possibility of different implementations being used for each action in the sequence of content creation, content signing, and signature verification. Producers and consumers shall account for this possibility in their signing and verification processing.	13.6.2		×	×	×
M6.34	The following canonicalization methods shall be supported by producers and consumers of packages with digital signatures: XML Canonicalization (c14n) XML Canonicalization with Comments (c14n with comments)0. Consumers validating signed packages shall fail the validation if other canonicalization methods are encountered.	13.2.4.4			×	×

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M6.35	A producer shall not specify more than one relationship transform for a particular relationships part. A consumer shall treat the presence of more than one relationship transform for a particular relationships part as an error.	13.2.4.25			×	×

1 Table J–13. Digital signatures recommendations

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
S6.1	The producer should create empty contents in the Digital Signature Origin part itself.	13.2.1			×	
S6.2	Producers generating digital signatures should not create Digital Signature Certificate parts that are not the target of at least one Digital Signature Certificate relationship from a Digital Signature XML Signature part. In addition, producers should remove a Digital Signature Certificate part if removing the last Digital Signature XML Signature part that has a Digital Signature Certificate relationship to it.	13.2.3			×	
S6.3	For digital signatures, a producer should apply a canonicalization transform to the <SignedInfo> element when it generates it, and a consumer should apply the canonicalization transform to the <SignedInfo> element when validating it.	13.2.4.4			×	×
S6.4	Producers and consumers should also use canonicalization transforms for references to parts that hold XML documents.	13.2.4.4			×	×
S6.5	The producer should only create <Reference> elements within a <SignedInfo> element that reference an <Object> element.	13.2.4.1			×	

2 Table J–14. Digital signatures optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.1	Format designers might allow a package to include digital signatures to enable consumers to validate the integrity of the contents. The producer might include the digital signature when allowed by the format designer.	13		×	×	
O6.2	If there are no Digital Signature XML Signature parts in the package, the Digital Signature Origin part is optional.	13.2.1			×	
O6.3	The producer might create each Digital Signature XML Signature part with a different form of signature, which can be handled by the consumer with different signature processors.	13.2.2			×	×
O6.4	The producer might create zero or many Digital Signature XML Signature parts in a package.	13.2.2			×	
O6.5	The producer might store the certificate as a separate part in the package, might embed it within the Digital Signature XML Signature part itself, or might not include it in the package if certificate data is known or can be obtained from a local or remote certificate store.	13.2.3			×	
O6.6	The producer might sign the part holding the certificate.	13.2.3			×	
O6.7	Producers might share Digital Signature Certificate parts by using the same certificate to create more than one signature.	13.2.3			×	
O6.8	The format designer might permit one or more application-specific <Object> elements. If allowed by the format designer, format producers can create one or more application-specific <Object> elements.	13.2.4.14		×	×	

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O6.9	Format designers and producers might not apply package-specific restrictions regarding URIs and <Transform> elements to application-specific <Object> element.	13.2.4.14		×	×	
O6.10	Format designers might permit producers to sign individual relationships in a package or the Relationships part as a whole.	13.2.4.25		×	×	
O6.11	The package implementer might create relationships XML that contains content from several namespaces, along with versioning instructions as defined in Part 4: "Markup Compatibility".	13.2.4.26	×			
O6.12	Format designers might specify an application-specific package part format that allows for the embedding of versioned or extended content that might not be fully understood by all present and future implementations. Producers might create such embedded versioned or extended content and consumers might encounter such content.	13.6.2		×	×	×

J.7 Pack URI

Table J–15. Pack URI conformance requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M7.1	The authority component contains an embedded URI that points to a package. The package implementer shall create an embedded URI that meets the requirements defined in RFC 3986 for a valid URI.	B.1	×			

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
M7.2	The optional path component identifies a particular part within the package. The package implementer shall only create path components that conform to the part naming rules. When the path component is missing, the resource identified by the pack URI is the package as a whole.	B.1	×			
M7.3	The package implementer shall consider pack URIs equivalent if: The scheme components are octet-by-octet identical after they are both converted to lowercase; *and* The authority components are equivalent (the rules for which will vary by scheme, as per RFC 3986); *and* The path components are equivalent when compared as case-insensitive ASCII strings.	B.4	×			
M7.4	The package implementer shall not create an authority component with an unescaped colon (:) character.	B.1	×			

Table J–16. Pack URI optional requirements

ID	Rule	Reference	Package Implementer	Format Designer	Format Producer	Format Consumer
O7.1	Consumer applications, based on the obsolete URI specification RFC 2396, might tolerate the presence of an unescaped colon character in an authority component.	B.1				×

End of informative text.

Annex K. Bibliography

The bibliography is informative.

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

[1] *ISO/IEC Directives Part 2, Rules for the structure and drafting of International Standards, Fourth edition*, 2001, ISBN 92-67-01070-0.

[2] *The Unicode Standard, Version 3.0*, by the Unicode Consortium; Addison-Wesley Publishing Co, ISBN 0-201-61633-5, February 2000. The latest version can be found at the Unicode Consortium's web site, www.unicode.org, at this writing.

Dublin Core Metadata Initiative, http://dublincore.org.

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

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

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

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

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

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

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

XML Base, W3C Recommendation, 27 June 2001.

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

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

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

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

XSL Transformations (XSLT), Version 1.0, W3C Recommendation, 16 November 1999.

ZIP File Format Specification, Version 6.2.1, PKWARE Inc., 2005.

End of informative text.

Annex L. Index

This annex is informative.

End of informative text.