INTERNATIONAL STANDARD

ISO/IEC 23736-4

First edition
2020-02

Information technology — Digital publishing — EPUB 3.0.1 —

Part 4: Open container format

Technologies de l'information — Publications numériques — EPUB 3.0.1 —

Partie 4: Format de conteneur ouvert

Reference number
ISO/IEC 23736-4:2020(E)

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by the World Wide Web Consortium (W3C) [as EPUB Open Container Format (OCF) 3.0.1] and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

A list of all parts in the ISO/IEC 23736 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

EPUB Open Container Format (OCF) 3.0.1

Recommended Specification 26 June 2014

THIS VERSION

http://www.idpf.org/epub/301/spec/epub-ocf-20140626.html

LATEST VERSION

http://www.idpf.org/epub3/latest/ocf

PREVIOUS VERSION

http://www.idpf.org/epub/301/spec/epub-ocf-20140228.html

A diff of changes from the previous version is also available.

Please refer to the errata for this document, which may include some normative corrections.

Editors

James Pritchett, Learning Ally (formerly Recording for the Blind & Dyslexic)

Markus Gylling, International Digital Publishing Forum (IDPF)

TABLE OF CONTENTS

› 1 Overview

› 1.1 Purpose and Scope

This section is informative

This specification, EPUB Open Container Format (OCF) 3.0.1, defines a file format and processing model for encapsulating the set of related resources that comprise an EPUB® Publication into a single-file container.

This specification is one of a family of related specifications that compose EPUB 3, the third major revision of an interchange and delivery format for digital publications based on XML and Web Standards. It is meant to be read and understood in concert with the other specifications that make up EPUB 3:

- The EPUB 3 Overview [EPUB3Overview], which provides an informative overview of EPUB and a roadmap to the rest of the EPUB 3 documents. The Overview should be read first.
- EPUB Publications 3.0.1 [Publications301], which defines the semantics and overarching conformance requirements for each Rendition of an EPUB Publication.
- EPUB Content Documents 3.0.1 [ContentDocs301], which defines profiles of XHTML, SVG and CSS for use in the context of EPUB Publications.
- EPUB Media Overlays 3.0.1 [MediaOverlays301], which defines a format and a processing model for synchronization of text and audio.

OCF is the required container technology for EPUB Publications. OCF may play a role in the following workflows:

- During the preparation steps in producing an EPUB Publication, OCF may be used as the container format when exchanging an in-progress EPUB Publication between different individuals and/or different organizations.
- When providing an EPUB Publication from publisher or conversion house to the distribution or sales channel, OCF is the recommended container format to be used as the transport format.

- When delivering the final EPUB Publication to an EPUB Reading System or User, OCF is the required format for the container that holds all of the assets that make up the EPUB Publication.

The OCF specification defines the rules for structuring the file collection in the abstract: the "abstract container". It also defines the rules for the representation of this abstract container within a ZIP archive: the "physical container". The rules for ZIP physical containers build upon the ZIP technologies used by [ODF]. OCF also defines a standard method for obfuscating embedded resources, such as fonts, for those EPUB Publications that require this functionality.

This specification supersedes Open Container Format (OCF) 3.0 [OCF30]. Refer to [EPUB3Changes] for information on differences between this specification and its predecessor.

› 1.2 Terminology

EPUB Publication

A collection of one or more Renditions conforming to this specification and its sibling specifications , packaged in an EPUB Container.

An EPUB Publication typically represents a single intellectual or artistic work, but this specification and its sibling specifications do not circumscribe the nature of the content.

Rendition

A logical document entity consisting of a set of interrelated resources representing one rendering of an EPUB Publication.

Default Rendition

The Rendition listed in the first `rootfile` element in the container.xml file.

Publication Resource

A resource that contains content or instructions that contribute to the logic and rendering of at least one Rendition of an EPUB Publication. In the absence of this resource, the EPUB Publication might not render as intended by the Author. Examples of Publication Resources include a Rendition's Package Document, EPUB Content Document, EPUB Style Sheets, audio, video, images, embedded fonts and scripts.

With the exception of the Package Document itself, the Publication Resources required to render a Rendition are listed in that Rendition's manifest [Publications301] and bundled in the EPUB Container file (unless specified otherwise in Publication Resource Locations [Publications301]).

Examples of resources that are not Publication Resources include those identified by the Package Document link [Publications301] element and those identified in outbound hyperlinks that resolve outside the EPUB Container (e.g., referenced from an [HTML5] `a` element `href` attribute).

EPUB Content Document

A Publication Resource that conforms to one of the EPUB Content Document definitions (XHTML or SVG).

An EPUB Content Document is a Core Media Type, and may therefore be included in the EPUB Publication without the provision of fallbacks [Publications301] .

XHTML Content Document

An EPUB Content Document conforming to the profile of [HTML5] defined in XHTML Content Documents [ContentDocs301] .

XHTML Content Documents use the XHTML syntax of [HTML5].

SVG Content Document

An EPUB Content Document conforming to the constraints expressed in SVG Content Documents [ContentDocs301] .

Core Media Type

A set of Publication Resource types for which no fallback is required. Refer to Publication Resources [Publications301] for more information.

Package Document

A Publication Resource carrying bibliographical and structural metadata about a given Rendition of an EPUB Publication, as defined in Package Documents [Publications301] .

Unique Identifier

The Unique Identifier is the primary identifier for an EPUB Publication, as identified by the `unique-identifier` attribute. The Unique Identifier may be shared by one or many Renditions of the same EPUB Publication that conform to the EPUB standard and embody the same content.

The Unique Identifier is less granular than the ISBN. However, significant revision, abridgement, etc. of the content requires a new Unique Identifier.

EPUB Style Sheet (or Style Sheet)

A CSS Style Sheet conforming to the CSS profile defined in EPUB Style Sheets [ContentDocs301] .

Viewport

The region of an EPUB Reading System in which the content of an EPUB Publication is rendered visually to a User.

EPUB Container (or Container)

The ZIP-based packaging and distribution format for EPUB Publications defined in *OCF ZIP Container* .

OCF Processor

A software application that processes EPUB Containers according to this specification.

Root Directory

The root directory represents the base of the Abstract Container file system. This directory is virtual in nature: a Reading System might or might not generate a physical root directory for the contents of the Abstract Container if the contents are unzipped.

Author

The person(s) or organization responsible for the creation of an EPUB Publication, which is not necessarily the creator of the content and resources it contains.

User

An individual that consumes an EPUB Publication using an EPUB Reading System.

EPUB Reading System (or Reading System)

A system that processes EPUB Publications for presentation to a User in a manner conformant with this specification and its sibling specifications .

› 1.3 Typographic Conventions

The following typographic conventions are used in this specification:

`markup`

All markup (elements, attributes, properties), code (JavaScript, pseudo-code), machine processable values (string, characters, media types) and file names are in red-orange monospace font.

`markup`

Links to markup and code definitions are underlined and in red-orange monospace font. Only the first instance in each section is linked.

`http://www.idpf.org/`

URIs are in navy blue monospace font.

hyperlink

Hyperlinks are underlined and in blue.

[reference]

Normative and informative references are enclosed in square brackets.

Term

Terms defined in the Terminology are in capital case.

Term

Links to term definitions have a dotted blue underline. Only the first instance in each section is linked.

Normative element, attribute and property definitions are in blue boxes.

```
Informative markup examples are in white boxes.
```

NOTE

Informative notes are in yellow boxes with a "Note" header.

CAUTION

Informative cautionary note are in red boxes with a "Caution" header.

› 1.4 Conformance Statements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in [RFC2119].

All sections of this specification are normative except where identified by the informative status label "This section is informative". The application of informative status to sections and appendices applies to all child content and subsections they may contain.

All examples in this specification are informative.

› 1.5 Content Conformance

- An OCF Abstract Container MUST meet the conformance constraints defined in *OCF Abstract Container* .
- An OCF ZIP Container (also referred to as an EPUB Container) MUST meet the conformance constraints defined in *OCF ZIP Container* .

› 1.6 Reading System Conformance

An EPUB Reading System MUST meet all of the following criteria:

- It MUST process the OCF ZIP Container in conformance with all Reading System conformance constraints expressed in *OCF ZIP Container* .
- If it has a Viewport, it MUST support deobfuscation of resources as defined in *Resource Obfuscation* .

› 2 OCF Abstract Container

› 2.1 Overview

This section is informative

An OCF Abstract Container defines a file system model for the contents of the container. The file system model uses a single common Root Directory for all of the contents of the container. All (non-remote) resources for embedded Renditions are located within the directory tree headed by the container's root directory, although no specific file system structure is mandated for this. The file system model also includes a mandatory directory named `META-INF` that is a direct child of the container's root directory and is used to store the following special files:

`container.xml` [required]

Identifies the file that is the point of entry for each embedded Rendition of the EPUB Publication.

`signatures.xml` [optional]

Contains digital signatures for various assets.

`encryption.xml` [optional]

Contains information about the encryption of Publication Resources. (This file is required when obfuscation is used.)

`metadata.xml` [optional]

Used to store metadata about the EPUB Container.

`rights.xml` [optional]

Used to store information about digital rights.

`manifest.xml` [allowed]

A manifest of container contents as allowed by Open Document Format [ODF].

Complete conformance requirements for the various files in `META-INF` are found in META-INF.

› 2.2 File and Directory Structure

The virtual file system for the OCF Abstract Container MUST have a single common Root Directory for all of the contents of the container.

The OCF Abstract Container MUST include a directory named `META-INF` that is a direct child of the container's root directory. Requirements for the contents of this directory are described in META-INF.

The file name `mimetype` in the root directory is reserved for use by OCF ZIP Containers, as explained in *OCF ZIP Container*.

All other files within the OCF Abstract Container MAY be in any location descendant from the container's root directory except within the `META-INF` directory.

It is RECOMMENDED that the contents of the EPUB Publication be stored within its own dedicated directory under the container's root.

› 2.3 Relative IRIs for Referencing Other Components

Files within the OCF Abstract Container MUST reference each other via Relative IRI References ([RFC3987] and [RFC3986]). For example, if a file named `chapter1.html` references an image file named `image1.jpg` that is located in the same directory, then `chapter1.html` might contain the following as part of its content:

```
<img src="image1.jpg" alt="…" />
```

For Relative IRI References, the Base IRI [RFC3986] is determined by the relevant language specifications for the given file formats. For example, the CSS specification defines how relative IRI references work in the context of CSS style sheets and property declarations. Note that some language specifications reference RFCs that preceded RFC3987, in which case the earlier RFC applies for content in that particular language.

Unlike most language specifications, the Base IRIs for all files within the `META-INF` directory use the Root Directory for the Abstract Container as the default Base IRI. For example, if `META-INF/container.xml` has the following content:

```
<?xml version="1.0"?>
<container version="1.0"
xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
        <rootfile full-path="EPUB/Great Expectations.opf"
            media-type="application/oebps-package+xml" />
    </rootfiles>
</container>
```

then the path `EPUB/Great Expectations.opf` is relative to the root directory for the OCF Abstract Container and not relative to the `META-INF` directory.

› 2.4 File Names

The term **File Name** represents the name of any type of file, either a directory or an ordinary file within a directory within an OCF Abstract Container.

For a given directory within the OCF Abstract Container, the **Path Name** is a string holding all directory File Names in the full path concatenated together with a / (`U+002F`) character separating the directory File Names. For a given file within the Abstract Container, the Path Name is the string holding all directory File Names concatenated together with a / character separating the directory File Names, followed by a / character and then the File Name of the file.

The File Name restrictions described below are designed to allow Path Names and File Names to be used without modification on most commonly used operating systems. This specification does not specify how an OCF Processor that is unable to represent OCF File and Path Names would compensate for this incompatibility.

In the context of an OCF Abstract Container, File and Path Names MUST meet all of the following criteria:

- › File Names MUST be UTF-8 [Unicode] encoded.
- › File Names MUST NOT exceed 255 bytes.
- › The Path Name for any directory or file within the Abstract Container MUST NOT exceed 65535 bytes.

› File Names MUST NOT use the following [Unicode] characters, as these characters might not be supported consistently across commonly-used operating systems:

SOLIDUS: `/` (`U+002F`)

QUOTATION MARK: `"` (`U+0022`)

ASTERISK: `*` (`U+002A`)

FULL STOP as the last character: `.` (`U+002E`)

COLON: `:` (`U+003A`)

LESS-THAN SIGN: `<` (`U+003C`)

GREATER-THAN SIGN: `>` (`U+003E`)

QUESTION MARK: `?` (`U+003F`)

REVERSE SOLIDUS: `\` (`U+005C`)

DEL (`U+007F`)

C0 range (`U+0000 … U+001F`)

C1 range (`U+0080 … U+009F`)

Private Use Area (`U+E000 … U+F8FF`)

Non characters in Arabic Presentation Forms-A (`U+FDDO … U+FDEF`)

Specials (`U+FFF0 … U+FFFF`)

Tags and Variation Selectors Supplement (`U+E0000 … U+E0FFF`)

Supplementary Private Use Area-A (`U+F0000 … U+FFFFF`)

Supplementary Private Use Area-B (`U+100000 … U+10FFFF`)

› File Names are case sensitive.

› All File Names within the same directory MUST be unique following case normalization as described in section 3.13 of [Unicode].

› All File Names within the same directory SHOULD be unique following NFC or NFD normalization [TR15].

NOTE

Some commercial ZIP tools do not support the full Unicode range and may support only the ASCII range for File Names. Content creators who want to use ZIP tools that have these restrictions may find it is best to restrict their File Names to the ASCII range. If the names of files cannot be preserved during the unzipping process, it will be necessary to compensate for any name translation which took place when the files are referenced by URI from within the content.

› 2.5 META-INF

All OCF Abstract Containers MUST include a directory called `META-INF`. This directory contains the files specified below. Files other than the ones defined below MAY be included in the `META-INF` directory; OCF Processors MUST NOT fail when encountering such files.

› 2.5.1 Container - META-INF/container.xml

All OCF Containers MUST include a file called `container.xml` within the `META-INF` directory. The `container.xml` file MUST identify the media type of, and path to, the root file for each of the Renditions of the EPUB Publication included within the Container.

The `container.xml` file MUST NOT be encrypted.

The schema for `container.xml` files is available in Schema for `container.xml`. `container.xml` files MUST be valid according to this schema after removing all elements and attributes from other namespaces (including all attributes and contents of such elements).

The `rootfiles` element MUST contain one or more `rootfile` elements, each of which MUST uniquely reference a Package Document representing a single Rendition of an EPUB Publication. If more than one Rendition is stored in an OCF, each MUST be a different rendering of the same EPUB Publication.

An OCF Processor MUST consider the first `rootfile` element within the `rootfiles` element to represent the Default Rendition for the contained EPUB Publication. Reading Systems are NOT REQUIRED to use any Rendition except the default one.

The following example shows a sample `container.xml` *for an EPUB Publication with the root file* `EPUB/My Crazy Life.opf` *(the Package Document):*

```
<?xml version="1.0"?>
<container version="1.0"
xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
        <rootfile full-path="EPUB/My Crazy Life.opf"
            media-type="application/oebps-package+xml" />
    </rootfiles>
</container>
```

The following example shows SVG and XHTML Renditions of The Sandman bundled in the same container:

```
<?xml version="1.0"?>
<container version="1.0"
xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
        <rootfile full-path="SVG/Sandman.opf"
            media-type="application/oebps-package+xml" />
        <rootfile full-path="XHTML/Sandman.opf"
            media-type="application/oebps-package+xml" />
    </rootfiles>
```

```
</container>
```

The `manifest` element contained within the Package Document specifies the one and only manifest used for processing a given Rendition. Ancillary manifest information contained in the ZIP archive or in the optional `manifest.xml` file MUST NOT be used for processing the Rendition. Any extra files in the ZIP archive MUST NOT be used in the processing of the Rendition (i.e., files within the ZIP archive that are not listed within the Package Document's `manifest` element, such as `META-INF` files or or resources specific to other Renditions of the EPUB Publication).

The `container.xml` file MAY include a `links` element following the `rootfiles` element, which, when present, MUST contain one or more `link` elements. Each `link` element MUST include an `href` attribute whose value identifies the location of a resource necessary for the processing of the EPUB Container. Each `link` element also MUST include a `rel` attribute whose value identifies the relationship of the resource, and MAY include a `media-type` attribute whose value MUST be a media type [RFC2046] that specifies the type and format of the resource referenced by the `link`.

The value of the `rootfile` element `full-path` attribute and the `link` element `href` attribute MUST contain a *path component* (as defined by RFC3986) which MUST take the form of a *path-rootless* only (also defined by RFC 3986). The path components are relative to the root of the container in which they are used.

OCF Processors MUST ignore foreign elements and attributes within a `container.xml` file.

› 2.5.2 Encryption - META-INF/encryption.xml

An optional `encryption.xml` file within the `META-INF` directory at the root level of the container file system holds all encryption information on the contents of the container. If any resource within the container is encrypted, `encryption.xml` MUST be present to indicate that the resource is encrypted and provide information on how it is encrypted.

This file is an XML document whose root element is `encryption`. The `encryption` element contains child elements of type `EncryptedKey` and `EncryptedData` as defined by [XML ENC Core]. An `EncryptedKey` element describes each encryption key used in the container, while an `EncryptedData` element describes each encrypted file. Each `EncryptedData` element refers to an `EncryptedKey` element, as described in XML Encryption.

The schema for `encryption.xml` files is available in Schema for `encryption.xml` ; `encryption.xml` files MUST be valid according to this schema.

OCF encrypts individual files independently, trading off some security for improved performance, allowing the container contents to be incrementally decrypted. Encryption in this way exposes the directory structure and file naming of the whole package.

OCF uses XML Encryption [XML ENC Core] to provide a framework for encryption, allowing a variety of algorithms to be used. XML Encryption specifies a process for encrypting arbitrary data and representing the result in XML. Even though an OCF Abstract Container may contain non-XML data, XML Encryption can be used to encrypt all data in an OCF Abstract Container. OCF encryption supports only the encryption of entire files within the container, not parts of files. The `encryption.xml` file, if present, MUST NOT be encrypted.

Encrypted data replaces unencrypted data in an OCF Abstract Container. For example, if an image named `photo.jpeg` is encrypted, the contents of the `photo.jpeg` resource SHOULD be replaced by its encrypted contents. When stored in a ZIP container, streams of data MUST be compressed before they are encrypted and Deflate compression MUST be used. Within the ZIP directory, encrypted files SHOULD be stored rather than Deflate-compressed.

Note that some situations require obfuscating the storage of embedded resources referenced by a Rendition to tie them to the "parent" EPUB Publication and make them more difficult to extract for unrestricted use (e.g., fonts). Although obfuscation is not encryption, the `encryption.xml` file is used in conjunction with the IDPF resource obfuscation algorithm to identify resources that need to be de-obfuscated before they can be used.

The following files MUST NOT be encrypted, regardless of whether default or specific encryption is requested:

`mimetype`

`META-INF/container.xml`

`META-INF/encryption.xml`

`META-INF/manifest.xml`

`META-INF/metadata.xml`

`META-INF/rights.xml`

`META-INF/signatures.xml`

`EPUB rootfile` (the Package Document)

Signed resources MAY subsequently be encrypted using the Decryption Transform for XML Signature [XML SIG Decrypt]. This feature enables an application such as an OCF agent to distinguish data that was encrypted before signing from data that was encrypted after signing. Only data that was encrypted after signing MUST be decrypted before computing the digest used to validate the signature.

In the following example, adapted from Section 2.2.1 of [XML ENC Core] the resource `image.jpeg` is encrypted using a symmetric key algorithm (AES) and the symmetric key is further encrypted using an asymmetric key algorithm (RSA) with a key of John Smith.

```
<encryption
    xmlns ="urn:oasis:names:tc:opendocument:xmlns:container"
    xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <enc:EncryptedKey Id="EK">
        <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <ds:KeyInfo>
            <ds:KeyName>John Smith</ds:KeyName>
        </ds:KeyInfo>
        <enc:CipherData>
            <enc:CipherValue>xyzabc</enc:CipherValue>
        </enc:CipherData>
    </enc:EncryptedKey>
    <enc:EncryptedData Id="ED1">
        <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
        <ds:KeyInfo>
            <ds:RetrievalMethod URI="#EK"
                Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
        </ds:KeyInfo>
        <enc:CipherData>
            <enc:CipherReference URI="image.jpeg"/>
        </enc:CipherData>
    </enc:EncryptedData>
</encryption>
```

› 2.5.3 Manifest - META-INF/manifest.xml

An optional file with the reserved name `manifest.xml` MAY be included within the `META-INF` directory at the root level of the container file system.

The `manifest.xml` file, if present, MUST NOT be encrypted.

› 2.5.4 Metadata - META-INF/metadata.xml

An optional file with the reserved name `metadata.xml` MAY be included within the `META-INF` directory at the root level of the container file system. This file, if present, MUST be used for container-level metadata.

If the `META-INF/metadata.xml` file is present, its contents SHOULD be only namespace-qualified elements [XMLNS]. The file SHOULD contain the root element `metadata` in the namespace `http://www.idpf.org/2013/metadata`, but other root elements are allowed for backwards compatibility. Reading Systems SHOULD ignore `metadata.xml` files with unrecognized root elements.

This version of the OCF specification does not define metadata for use in the `metadata.xml` file. Container-level metadata MAY be defined in future versions of this specification and in IDPF-defined EPUB extension specifications.

The `metadata.xml` file, if present, MUST NOT be encrypted.

› 2.5.5 Rights Management - META-INF/rights.xml

An optional file with the reserved name `rights.xml` MAY be included within the `META-INF` directory at the root level of the container file system. This file is reserved for digital rights management (DRM) information for trusted exchange of EPUB Publications among rights holders, intermediaries, and users. This version of the OCF specification does not specify a required format for DRM information, but a future version may specify a particular format for DRM information.

If the `META-INF/rights.xml` file is present, its contents SHOULD be only namespace-qualified elements [XMLNS] to avoid collision with future versions of OCF that may specify a particular format for this file.

The `rights.xml` file MUST NOT be encrypted.

When the `rights.xml` file is not present, the OCF container provides no information indicating any part of the container is rights governed.

› 2.5.6 Digital Signatures - META-INF/signatures.xml

An optional `signatures.xml` within the `META-INF` directory at the root level of the container file system holds digital signatures of the container and its contents. This file is an XML document whose root element is `signatures`. The `signatures` element contains child elements of type `Signature` as defined by [XML DSIG Core]. Signatures can be applied to any Rendition of an EPUB Publication as a whole or to parts of the Renditions. XML Signature can specify the signing of any kind of data, not just XML.

The `signatures.xml` file MUST NOT be encrypted.

When the `signatures.xml` file is not present, the OCF container provides no information indicating any part of the container is digitally signed at the container level. It is possible that digital signing exists within any of the contained Renditions, however.

The schema for `signatures.xml` files is available in Schema for `signatures.xml` ; `signatures.xml` files MUST be valid according to this schema.

When an OCF agent creates a signature of data in a container, it SHOULD add the new signature as the last child `Signature` element of the `signatures` element in the `signatures.xml` file.

NOTE

Each `Signature` in the `signatures.xml` file identifies by IRI the data to which the signature applies, using the XML Signature `Manifest` element and its `Reference` sub-elements. Individual contained files may be signed separately or together. Separately signing each file creates a digest value for the resource that can be validated independently. This approach may make a Signature element larger. If files are signed together, the set of signed files can be listed in a single XML Signature `Manifest` element and referenced by one or more `Signature` elements.

Any or all files in the container can be signed in their entirety with the exception of the `signatures.xml` file since that file will contain the computed signature information. Whether and how the `signatures.xml` file should be signed depends on the objective of the signer.

If the signer wants to allow signatures to be added or removed from the container without invalidating the signer's signature, the `signatures.xml` file SHOULD NOT be signed.

If the signer wants any addition or removal of a signature to invalidate the signer's signature, the Enveloped Signature transform (defined in Section 6.6.4 of [XML DSIG Core]) can be used to sign the entire preexisting signature file excluding the `Signature` being created. This transform would sign all previous signatures, and it would become invalid if a subsequent signature was added to the package.

If the signer wants the removal of an existing signature to invalidate the signer's signature but also wants to allow the addition of signatures, an XPath transform can be used to sign just the existing signatures. (This is only a suggestion. The particular XPath transform is not a part of the OCF specification.)

XML-Signature does not associate any semantics with a signature; an agent MAY include semantic information, for example, by adding information to the Signature element that describes the signature. XML Signature describes how additional information can be added to a signature (for example, by using the `SignatureProperties` element).

The following XML expression shows the content of an example `signatures.xml` file, and is based on the examples found in Section 2 of [XML DSIG Core]. It contains one signature, and the signature applies *to two resources,* `OEBFPS/book.html` *and* `OEBFPS/images/cover.jpeg`*, in the container.*

```
<signatures xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <Signature Id="sig" 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="#Manifest1">
                <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
            </Reference>
        </SignedInfo>
        <SignatureValue>…</SignatureValue>
        <KeyInfo>
            <KeyValue>
                <DSAKeyValue>
                    <P>…</P><Q>…</Q><G>…</G><Y>…</Y>
                </DSAKeyValue>
            </KeyValue>
        </KeyInfo>
        <Object>
            <Manifest Id="Manifest1">
                <Reference URI="OEBFPS/book.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="OEBFPS/images/cover.jpeg">
                    <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>
            </Manifest>
        </Object>
    </Signature>
</signatures>
```

› 3 OCF ZIP Container

› 3.1 Overview

This section is informative

An OCF ZIP Container is a physical single-file manifestation of an abstract container.

› 3.2 ZIP File Requirements

An OCF ZIP Container uses the ZIP format as specified by [ZIP APPNOTE 6.3.3], but with the following constraints and clarifications:

› The contents of the OCF ZIP Container MUST be a conforming abstract container.

› OCF ZIP Containers MUST NOT use the features in the ZIP application note that allow ZIP files to be split across multiple storage media. OCF Processors MUST treat any OCF files that specify that the ZIP file is split across multiple storage media as being in error.

› OCF ZIP Containers MUST include only stored (uncompressed) and Deflate-compressed ZIP entries within the ZIP archive. OCF Processors MUST treat any OCF Containers that use compression techniques other than Deflate as being in error.

› OCF ZIP Containers MAY use the ZIP64 extensions defined as "Version 1" in section V, subsection G of the application note at [ZIP APPNOTE 6.3.3] and SHOULD use only those extensions when the content requires them. OCF Processors MUST support the ZIP64 extensions defined as "Version 1".

› OCF ZIP Containers MUST NOT use the encryption features defined by the ZIP format; instead, encryption MUST be done using the features described in Encryption – META-INF/encryption.xml. OCF Processors MUST treat any other OCF ZIP Containers that use ZIP encryption features as being in error.

› It is not a requirement that OCF Processors preserve information from an OCF ZIP Container through load and save operations that is not defined within the OCF Abstract Container; in particular, an OCF Processor does not have to preserve CRC values, comment fields or fields that hold file system information corresponding to a particular operating system (e.g., *External file attributes* and *Extra field*).

› OCF ZIP Containers MUST encode File System Names using UTF-8 [Unicode].

The following constraints apply to particular fields in the OCF ZIP Container archive:

› In the local file header table, OCF ZIP Containers MUST set the `version needed to extract` fields to the values `10`, `20` or `45` in order to match the maximum version level needed by the given file (e.g., `20` if Deflate is needed, `45` if ZIP64 is needed). OCF Processors MUST treat any other values as being in error.

› In the local file header table, OCF ZIP Containers MUST set the `compression` method field to the values `0` or `8`. OCF Processors MUST treat any other values as being in error.

› OCF Processors MUST treat OCF ZIP Containers with an `Archive decryption header` or an `Archive extra data record` as being in error.

› 3.3 OCF ZIP Container Media Type Identification

OCF ZIP Containers MUST include a `mimetype` file as the first file in the Container, and the contents of this file MUST be the MIME type string `application/epub+zip` encoded in US-ASCII [US-ASCII].

The contents of the `mimetype` file MUST NOT contain any leading padding or whitespace, MUST NOT begin with the Unicode signature (or Byte Order Mark), and the case of the MIME type string MUST be exactly as presented above. The `mimetype` file additionally MUST NOT be compressed or encrypted, and there MUST NOT be an extra field in its ZIP header.

NOTE

Refer to Appendix C, *The* `application/epub+zip` *Media Type* for further information about the `application/epub+zip` media type.

› 4 Resource Obfuscation

› 4.1 Introduction

This section is informative

Since an OCF Zip Container is fundamentally a ZIP file, commonly available ZIP tools can be used to extract any unencrypted content stream from the package. Moreover, the nature of ZIP files means that their contents may appear like any other native container on some systems (e.g., a folder).

While this simplicity of ZIP files is quite useful, it also poses a problem when ease of extraction of resources is not a desired side-effect of not encrypting them. An Author who wishes to include a third-party font, for example, typically does not want that font extracted and re-used by others. More critically, many commercial fonts allow embedding, but embedding a font implies making it an integral part of the EPUB Publication, not just providing the original font file along with the content.

Since integrated ZIP support is so ubiquitous in modern operating systems, simply placing a font in the ZIP archive is insufficient to signify that it is not intended to be reused in other contexts. This uncertainty can undermine the otherwise very useful font embedding capability of EPUB Publications.

In order to discourage reuse of the font, some font vendors may only allow use of their fonts in EPUB Publications if those fonts are bound in some way to the EPUB Publication. That is, if the font file cannot be installed directly for use on an operating system with the built-in tools of that computing device, and it cannot be directly used by other EPUB Publications.

It is beyond the scope of this specification to provide a digital rights management or enforcement system for such resources. This section instead defines a method of obfuscation that will require additional work on the part of the final OCF recipient to gain general access to any obfuscated resources.

Note that no claim is made in this specification, or by the IDPF, that this constitutes encryption, nor does it guarantee that the resource will be secure from copyright infringement. It is the hope of the

IDPF, however, that this algorithm will meet the requirements of most vendors who require some assurance that their resources cannot simply be extracted by unzipping the Container.

In the case of fonts, the primary use case for obfuscation, the defined mechanism will simply provide a stumbling block for those who are unaware of the license details. It will not prevent a determined user from gaining full access to the font. Given an OCF Container, it is possible to apply the algorithms defined to extract the raw font file. Whether this method of obfuscation satisfies the requirements of individual font licenses remains a question for the licensor and licensee.

› 4.2 Obfuscation Key

The key used in the obfuscation algorithm is derived from the Unique Identifier of the Default Rendition.

All whitespace characters, as defined by the XML 1.0 specification [XML], section 2.3, MUST be removed from this identifier — specifically, the Unicode code points `U+0020`, `U+0009`, `U+000D` and `U+000A`.

A SHA-1 digest of the UTF-8 representation of the resulting string SHOULD be generated as specified by the Secure Hash Standard [SHA-1]. This digest is then directly used as the key for the algorithm.

› 4.3 Obfuscation Algorithm

The algorithm employed to obfuscate resource consists of modifying the first 1040 bytes (~1KB) of the file. In the unlikely event that the file is less than 1040 bytes, then the entire file will be modified.

To obfuscate the original data, the result of performing a logical exclusive or (XOR) on the first byte of the raw file and the first byte of the obfuscation key is stored as the first byte of the embedded resource.

This process is repeated with the next byte of source and key, and continues until all bytes in the key have been used. At this point, the process continues starting with the first byte of the key and 21st byte of the source. Once 1040 bytes have been encoded in this way (or the end of the source is reached), any remaining data in the source is directly copied to the destination.

Obfuscation of resources MUST occur before they are compressed and added to the OCF Container. Note that as obfuscation is not encryption, this requirement is not a violation of the one in Encryption – META-INF/encryption.xml to compress resources before encrypting them.

The following pseudo-code exemplifies the obfuscation algorithm.

```
set ocf to OCF container file
set source to file
set destination to obfuscated file
set keyData to key for file
set outer to 0
while outer < 52 and not (source at EOF)
    set inner to 0
    while inner < 20 and not (source at EOF)
        read 1 byte from source     //Assumes read advances file
position
        set sourceByte to result of read
        set keyByte to byte inner of keyData
        set obfuscatedByte to (sourceByte XOR keyByte)
        write obfuscatedByte to destination
        increment inner
    end while
    increment outer
end while
if not (source at EOF) then
    read source to EOF
    write result of read to destination
end if
Deflate destination
store destination as source in ocf
```

To get the original font data back, the process is simply reversed: the source file becomes the obfuscated data and the destination file will contain the raw data.

NOTE

The obfuscation of fonts was allowed prior to EPUB 3.0.1, but the order of obfuscation and compression was not specified. As a result, invalid fonts might be encountered after decompression and de-obfuscation. In such instances, de-obfuscating the data before inflating it may return a valid font. Supporting this method of retrieval is optional, as it is not compliant with this version of this specification, but needs to be considered when supporting EPUB 3 content generally.

› 4.4 Specifying Obfuscated Resources

Although not technically encrypted data, all obfuscated resources MUST have an entry in the `encryption.xml` file accompanying the EPUB Publication (see Encryption – META-INF/encryption.xml).

An `EncryptionMethod` element MUST be included for each obfuscated resource. Each MUST include a child `EncryptedData` element whose `Algorithm` attribute is set to the value `http://www.idpf.org/2008/embedding`. The presence of this attribute signals the use of the algorithm described in this specification. The path to the obfuscated resource MUST be listed in the `CipherReference` child of the `CipherData` element.

The following example shows an entry for an obfuscated font in the `encryption.xml` *file.*

```
<encryption
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container"
    xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
    <enc:EncryptedData>
        <enc:EncryptionMethod
Algorithm="http://www.idpf.org/2008/embedding"/>
        <enc:CipherData>
            <enc:CipherReference URI="EPUB/Fonts/BKANT.TTF"/>
        </enc:CipherData>
    </enc:EncryptedData>
</encryption>
```

To prevent trivial copying of the embedded resource to other EPUB Publications, the obfuscation key MUST NOT be provided in the `encryption.xml` file.

› Appendix A. Schemas

› A.1 Schema for `container.xml`

The schema for `container.xml` files is available at `http://www.idpf.org/epub/301/schema/ocf-container-30.rnc` .

Validation using this schema requires a processor that supports [RelaxNG] and [XSD-DATATYPES].

› A.2 Schema for `encryption.xml`

The schema for `encryption.xml` files is included in [XML Sec RNG Schemas].

› A.3 Schema for `signatures.xml`

The schema for `signatures.xml` files is included in [XML Sec RNG Schemas].

› Appendix B. Example

This appendix is informative

The following example demonstrates the use of this OCF format to contain a signed and encrypted EPUB Publication within a ZIP Container.

› Example B.1. Ordered list of files in the ZIP Container

```
mimetype
META-INF/container.xml
META-INF/signatures.xml
META-INF/encryption.xml
EPUB/As You Like It.opf
EPUB/book.html
EPUB/nav.html
EPUB/toc.ncx
EPUB/images/cover.png
```

› Example B.2. The contents of the `mimetype` file

```
application/epub+zip
```

› Example B.3. The contents of the `META-INF/container.xml` file

```
<?xml version="1.0"?>
<container version="1.0"
xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
        <rootfile full-path="EPUB/As You Like It.opf"
            media-type="application/oebps-package+xml" />
    </rootfiles>
</container>
```

› Example B.4. The contents of the `META-INF/signatures.xml` file

```
<signatures xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <Signature Id="AsYouLikeItSignature"
xmlns="http://www.w3.org/2000/09/xmldsig#">

        <!-- SignedInfo is the information that is actually signed. In
this case -->
        <!-- the SHA1 algorithm is used to sign the canonical form of
the XML    -->
        <!-- documents enumerated in the Object element below
-->
        <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="#AsYouLikeIt">
                <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <DigestValue>…</DigestValue>
            </Reference>
        </SignedInfo>

        <!-- The signed value of the digest above using the DSA
algorithm -->
        <SignatureValue>…</SignatureValue>

        <!-- The key to use to validate the signature -->
        <KeyInfo>
            <KeyValue>
                <DSAKeyValue>
                    <P>…</P>
                    <Q>…</Q>
                    <G>…</G>
                    <Y>…</Y>
                </DSAKeyValue>
            </KeyValue>
        </KeyInfo>
        <!-- The list documents to sign. Note that the canonical form of
XML   -->
        <!-- documents is signed while the binary form of the other
documents -->
        <!-- is used -->
        <Object>
            <Manifest Id="AsYouLikeIt">
                <Reference URI="EPUB/As You Like It.opf">
                    <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="EPUB/book.html">
                    <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="EPUB/images/cover.png">
                    <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                    <DigestValue></DigestValue>
                </Reference>
            </Manifest>
        </Object>
    </Signature>
</signatures>
```

› Example B.5. The contents of the `META-INF/encryption.xml` file

```
<?xml version="1.0"?>
<encryption xmlns="urn:oasis:names:tc:opendocument:xmlns:container"
    xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

    <!-- The RSA encrypted AES-128 symmetric key used to encrypt the
data -->
    <enc:EncryptedKey Id="EK">
        <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        <ds:KeyInfo>
            <ds:KeyName>John Smith</ds:KeyName>
        </ds:KeyInfo>
        <enc:CipherData>
            <enc:CipherValue>xyzabc…</enc:CipherValue>
        </enc:CipherData>
    </enc:EncryptedKey>
    <!-- Each EncryptedData block identifies a single document that has
been     -->
    <!-- encrypted using the AES-128 algorithm. The data remains stored
in it's -->
    <!-- encrypted form in the original file within the container.
-->
    <enc:EncryptedData Id="ED1">
        <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
        <ds:KeyInfo>
            <ds:RetrievalMethod URI="#EK"
Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
        </ds:KeyInfo>
        <enc:CipherData>
            <enc:CipherReference URI="EPUB/book.html"/>
        </enc:CipherData>
    </enc:EncryptedData>

    <enc:EncryptedData Id="ED2">
        <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
        <ds:KeyInfo>
            <ds:RetrievalMethod URI="#EK"
Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
        </ds:KeyInfo>
        <enc:CipherData>
            <enc:CipherReference URI="EPUB/images/cover.png"/>
        </enc:CipherData>
    </enc:EncryptedData>

</encryption>
```

› **Example B.6. The contents of the `EPUB/As You Like It.opf` file**

```
<?xml version="1.0"?>
<package version="3.0"
         xml:lang="en"
         xmlns="http://www.idpf.org/2007/opf"
         unique-identifier="pub-id">

    <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:identifier
              id="pub-id">urn:uuid:B9B412F2-CAAD-4A44-B91F-
A375068478A0</dc:identifier>
        <meta refines="#pub-id"
              property="identifier-type"
              scheme="xsd:string">uuid</meta>

        <dc:language>en</dc:language>

        <dc:title>As You Like It</dc:title>

        <dc:creator id="creator">William Shakespeare</dc:creator>
        <meta refines="#creator"
```

```
               property="role"
               scheme="marc:relators">aut</meta>

        <meta property="dcterms:modified">2000-03-24T00:00:00Z</meta>

        <dc:publisher>Project Gutenberg</dc:publisher>

        <dc:date>2000-03-24</dc:date>

        <meta property="dcterms:dateCopyrighted">9999-01-01</meta>

        <dc:identifier
               id="isbn13">urn:isbn:9780741014559</dc:identifier>
        <meta refines="#isbn13"
               property="identifier-type"
               scheme="onix:codelist5">15</meta>

        <dc:identifier id="isbn10">0-7410-1455-6</dc:identifier>
        <meta refines="#isbn10"
               property="identifier-type"
               scheme="onix:codelist5">2</meta>

        <link rel="xml-signature"
               href="../META-INF/signatures.xml#AsYouLikeItSignature"/>
    </metadata>

    <manifest>
        <item id="r4915"
               href="book.html"
               media-type="application/xhtml+xml"/>
        <item id="r7184"
               href="images/cover.png"
               media-type="image/png"/>
        <item id="nav"
               href="nav.html"
               media-type="application/xhtml+xml"
               properties="nav"/>
        <item id="ncx"
               href="toc.ncx"
               media-type="application/x-dtbncx+xml"/>
    </manifest>

    <spine toc="ncx">
        <itemref idref="r4915"/>
    </spine>
</package>
```

› Appendix C. The `application/epub+zip` Media Type

This appendix is informative