INTERNATIONAL STANDARD

ISO/IEC 17203

Second edition
2017-09

Information technology — Open Virtualization Format (OVF) specification

Technologies de l'information — Spécification du format de virtualisation ouvert (OVF)

Reference number
ISO/IEC 17203:2017(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Foreword

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

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. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see http://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).

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

For an explanation on the voluntary nature of Standard, the meaning of the ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: Foreword – Supplementary information

This document was prepared by ANSI (as INCITS 469-2015) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

The list of all currently available parts of ISO/IEC 17203 series, under the general title *Information technology*, can be found on the ISO web site.

CONTENTS

Tables

AMERICAN NATIONAL STANDARD INCITS 469-2015

American National Standard
for Information Technology –

Open Virtualization Format (OVF) Specification

1 Scope

The *Open Virtualization Format (OVF) Specification* describes an open, secure, efficient and extensible format for the packaging and distribution of software to be run in virtual systems.

The OVF package enables the authoring of portable virtual systems and the transport of virtual systems between virtualization platforms. This version of the specification (2.1) is intended to allow OVF 1.x tools to work with OVF 2.x descriptors in the following sense:

- Existing OVF 1.x tools should be able to parse OVF 2.x descriptors.
- Existing OVF 1.x tools should be able to give warnings/errors if dependencies to 2.x features are required for correct operation.

If a conflict arises between the schema, text, or tables, the order of precedence to resolve the conflicts is schema; then text; then tables. Figures are for illustrative purposes only and are not a normative part of the standard.

A table may constrain the text but it shall not conflict with it.

The profile conforms to the cited CIM Schema classes where used. Any requirements contained in the cited CIM Schema classes shall be met. If a conflict arises the CIM Schema takes precedence.

The profile conforms to the cited OVF XML Schema. It may constrain the schema but it shall not conflict with it. If a conflict arises the OVF XML Schema takes precedence.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this American National Standard. All standards are subject to revision, and parties to agreements based on this American National Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

DMTF DSP0004, *Common Information Model (CIM) Infrastructure Specification 2.7*, http://www.dmtf.org/standards/published_documents/DSP0004_2.7.pdf

DMTF DSP0223, *Generic Operations 1.0*, http://www.dmtf.org/standards/published_documents/DSP0223_1.0.pdf

DMTF DSP0230, *WS-CIM Mapping Specification 1.0*, http://www.dmtf.org/sites/default/files/standards/documents/DSP0230_1.0.2.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide 1.1*, http://www.dmtf.org/standards/published_documents/DSP1001_1.1.pdf

DMTF DSP1041, *Resource Allocation Profile (RAP) 1.1*, http://www.dmtf.org/standards/published_documents/DSP1041_1.1.pdf

DMTF DSP1043, *Allocation Capabilities Profile (ACP) 1.0*, http://www.dmtf.org/standards/published_documents/DSP1043_1.0.pdf

DMTF DSP1047, *Storage Resource Virtualization Profile 1.0*, http://www.dmtf.org/standards/published_documents/DSP1047_1.0.pdf

DMTF DSP1050, Ethernet Port Resource Virtualization Profile 1.0, http://www.dmtf.org/standards/published_documents/DSP1050_1.0.pdf

DMTF DSP1057, *Virtual System Profile 1.0*, http://www.dmtf.org/standards/published_documents/DSP1057_1.0.pdf

DMTF DSP8023, *OVF XML Schema Specification for OVF Envelope 2.0*, http://schemas.dmtf.org/ovf/envelope/2/dsp8023_2.0.xsd

DMTF DSP8027, *OVF XML Schema Specification for OVF Environment 1.1*, http://schemas.dmtf.org/ovf/environment/1/dsp8027_1.1.xsd

DMTF DSP8049, *Network Port Profile XML Schema,* http://schemas.dmtf.org/ovf/networkportprofile/1/dsp8049_1.0.xsd

IETF RFC1738, T. Berners-Lee, *Uniform Resource Locators (URL)*, December 1994, http://tools.ietf.org/html/rfc1738

IETF RFC1952, P. Deutsch, *GZIP file format specification version 4.3*, May 1996, http://tools.ietf.org/html/rfc1952

IETF RFC2616, R. Fielding et al, *Hypertext Transfer Protocol – HTTP/1.1*, June 1999, http://tools.ietf.org/html/rfc2616

IETF Standard 66, *Uniform Resource Identifiers (URI): Generic Syntax*, http://tools.ietf.org/html/rfc3986

IETF Standard 68, *Augmented BNF for Syntax Specifications: ABNF*, http://tools.ietf.org/html/rfc5234

ISO 9660, 1988 Information processing-Volume and file structure of CD-ROM for information interchange, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=17505

ISO, ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*, http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

ISO/IEC/IEEE 9945:2009: Information technology -- Portable Operating System Interface (POSIX®) Base Specifications, Issue 7 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50516

W3C, *XML Schema Part 1: Structures Second Edition.* 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

W3C, *XML Schema Part 2: Datatypes Second Edition.* 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

W3C, XML Encryption Syntax and Processing Version 1.1, 13 March 2012, W3C Candidate Recommendation http://www.w3.org/TR/2012/CR-xmlenc-core1-20120313/

FIPS 180-2: Secure Hash Standard (SHS) http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

3 Terms and definitions

In this document, some terms have a specific meaning beyond the normal English meaning. Those terms are defined in this clause.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may," "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Annex H. The terms in parenthesis are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Annex H specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 5.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

The terms defined in DSP0004, DSP0223, and DSP1001 apply to this document. The following additional terms are used in this document.

3.1
authoring function
the creation of the OVF package

3.2
chassis
a placement policy as defined in the class CIM_Chassis

3.3
conditional
indicates requirements to be followed strictly to conform to the document when the specified conditions are met

3.4
deployment function
a function the result of which is a prepared virtual system

3.5
geographic
a placement policy referring to a geographic location (e.g., a country, a state, a province, a latlong)

3.6
guest software
the software that runs inside a virtual system

3.7
mandatory
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.8
optional
indicates a course of action permissible within the limits of the document

3.9
rack
a placement policy as defined in the class CIM_Rack

3.10
site
a placement policy as defined in Access, Terminals, Transmission and Multiplexing (ATTM); Broadband Deployment - Energy Efficiency and Key Performance Indicators; Part 2: Network sites; Sub-part 1: Operator sites, Technical Report, ETSI TR 105 174-2-1 V1.1.1 (2009-10)

3.11
OVF package
a single compressed file or a set of files that contains the OVF descriptor file and may contain associated virtual disks, operational metadata, and other files

3.12
OVF descriptor
an XML file that validates to DSP8023 and provides the information needed to deploy the OVF package

3.13
virtualization platform
the hypervisor on which the virtual systems run

3.14
virtual appliance
a service delivered as a software stack that utilizes one or more virtual systems

3.15
virtual hardware
the processor, memory and I/O resources provided by a virtualization platform that supports a virtual system

3.16
virtual system
as defined in the Virtual System Profile plus the guest software if any

3.17
virtual system collection
a collection of virtual systems

3.18
virtualization management
the software that performs resource allocation and management of virtual systems

4 Symbols and abbreviated terms

The abbreviations defined in DSP0004, DSP0223, and DSP1001 apply to this document. The following additional abbreviations are used in this document.

4.1
CIM
Common Information Model

4.2
IP
Internet Protocol

4.3
OVF
Open Virtualization Format

4.4
VS
virtual system

4.5
VSC
virtual system collection

5 OVF package

5.1 OVF package structure

An OVF package shall consist of the following files:

- one OVF descriptor with extension `.ovf`
- zero or one OVF manifest with extension `.mf`
- zero or one OVF certificate with extension `.cert`
- zero or more disk image files
- zero or more additional resource files, such as ISO images

The file extensions `.ovf`, `.mf` and `.cert` shall be used. See D.1 for an example.

An OVF package can be stored as either a single compressed file (.ova) or a set of files, as described in 5.3 and 5.4. Both modes shall be supported.

An OVF package may have a manifest file containing the SHA digests of individual files in the package. OVF packages authored according to this version of the specification shall use SHA256 digests. The manifest file shall have an extension `.mf` and the same base name as the `.ovf` file and be a sibling of the `.ovf` file. If the manifest file is present, a consumer of the OVF package should verify the digests in the manifest file in the OVF package by computing the actual SHA digests and comparing them with the digests listed in the manifest file. The manifest file shall contain SHA digests for all distinct files referenced in the `References` element of the OVF descriptor and for no other files. See 7.1

The syntax definitions below use ABNF with the exceptions listed in ANNEX A.

The format of the manifest file is as follows:

```
manifest_file = *( file_digest )
```

```
  file_digest    = algorithm "(" file_name ")" "=" sp digest nl
  algorithm      = "SHA1" | "SHA256"
  digest         = *( hex-digit )
  hex-digit      = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "a" |
"b" | "c" | "d" | "e" | "f"
  sp             = %x20
  nl             = %x0A
```

See D.1 for an example.

An OVF package may be signed by signing the manifest file. The digest of the manifest file is stored in a certificate file with extension `.cert` file along with the base64-encoded X.509 certificate. The `.cert` file shall have the same base name as the `.ovf` file and be a sibling of the `.ovf` file.

See ANNEX F for deployment considerations.

The format of the certificate file shall be as follows:

```
  certificate_file   = manifest_digest certificate_part
  manifest_digest    = algorithm "(" file_name ")" "=" sp signed_digest nl
  algorithm          = "SHA1" | "SHA256"
  signed_digest      = *( hex-digit)
  certificate_part   = certificate_header certificate_body certificate_footer
  certificate_header = "-----BEGIN CERTIFICATE-----" nl
  certificate_footer = "-----END CERTIFICATE-----" nl
  certificate_body   = base64-encoded-certificate nl
                       ; base64-encoded-certificate is a base64-encoded X.509
                       ; certificate, which may be split across multiple lines
  hex-digit          = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" |
"a" | "b" | "c" | "d" | "e" | "f"
  sp                 = %x20
  nl                 = %x0A
```

See D.1 for an example.

The manifest and certificate files, when present, shall not be included in the `References` section of the OVF descriptor (see 7.1). This ensures that the OVF descriptor content does not depend on whether the OVF package has a manifest or is signed, and the decision to add a manifest or certificate to a package can be deferred to a later stage.

The file extensions `.mf` and `.cert` may be used for other files in an OVF package, as long as they do not occupy the sibling URLs or path names where they would be interpreted as the package manifest or certificate.

5.2 Virtual disk formats

OVF does not require any specific disk format to be used, but to comply with this specification the disk format shall be given by a URI that identifies an unencumbered specification on how to interpret the disk format. The specification need not be machine readable, but it shall be static and unique so that the URI may be used as a key by software reading an OVF package to uniquely determine the format of the disk. The specification shall provide sufficient information so that a skilled person can properly interpret the disk format for both reading and writing of disk data. The URI should be resolvable.

5.3 OVF package options

An OVF package may be stored as a compressed OVF package or as a set of files in a directory structure. A compressed OVF package is stored as single file. The file extension is `.ova` (open virtual appliance or application). See D.2 for an example.

All file references in the OVF descriptor are relative-path references and are described in section 7.1. Entries in a compressed OVF package shall exist only once.

In addition, the entries shall be in one of the following orders inside the OVF package:

1) OVF descriptor
2) The remaining files shall be in the same order as listed in the References section (see 7.1). Note that any external string resource bundle files for internationalization shall be first in the References section (see clause 10).

or

1) OVF descriptor
2) OVF manifest
3) OVF certificate
4) The remaining files shall be in the same order as listed in the References section (see 7.1). Note that any external string resource bundle files for internationalization shall be first in the References section (see clause 10).

or

1) OVF descriptor
2) The intermediate files shall be in the same order as listed in the References section (see 7.1). Note that any external string resource bundle files for internationalization shall be first in the References section (see clause 10).
3) OVF manifest
4) OVF certificate

The ordering restriction ensures that it is possible to extract the OVF descriptor from a compressed OVF package without scanning the entire archive. The ordering restriction enables the efficient generation of a compressed OVF package-

A compressed OVF package shall be created by using the TAR format that complies with the USTAR (Uniform Standard Tape Archive) format as defined by the ISO/IEC/IEEE 9945:2009.

5.4 Distribution as a set of files

An OVF package may be made available as a set of files. See D.2 for an example.

6 OVF descriptor

The OVF descriptor contains the metadata about the OVF package. This is an extensible XML document for encoding information, such as product details, virtual hardware requirements, and licensing.

DSP8023 is the schema definition file for the OVF descriptor that contains the elements and attributes. The OVF descriptor shall validate against DSP8023.

Clauses 7, 8, and 9, describe the semantics, structure, and extensibility framework of the OVF descriptor. These clauses are not a replacement for reading the schema definitions, but they complement the schema definitions.

The XML namespaces used in this specification are listed in Table 1. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 1 – XML namespace prefixes

Prefix	XML Namespace
ovf	http://schemas.dmtf.org/ovf/envelope/2
ovfenv	http://schemas.dmtf.org/ovf/environment/1
rasd	http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData.xsd
vssd	http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData.xsd
epasd	http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_EthernetPortAllocationSettingData.xsd
sasd	http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_StorageAllocationSettingData.xsd
cim	http://schemas.dmtf.org/wbem/wscim/1/common.xsd

7 Envelope element

The `Envelope` element describes all metadata for the virtual systems (including virtual hardware), as well as the structure of the OVF package itself.

The outermost level of the envelope consists of the following parts:

- A version indication, defined by the XML namespace URIs
- A list of file references to all external files that are part of the OVF package, defined by the `References` element and its `File` child elements, e.g., virtual disk files, ISO images, and internationalization resources
- A metadata part, defined by section elements, defined in clause 9
- A description of the content, either a single virtual system (`VirtualSystem` element) or a collection of multiple virtual systems (`VirtualSystemCollection` element)
- A specification of message resource bundles for zero or more locales, defined by a `Strings` element for each locale

See D.3 for an example.

The `xml:lang` attribute on the `Envelope` element is optional. If present, it shall specify the default locale for messages in the descriptor. The `Strings` element is optional. If present, it shall contain string resource bundles for different locales. See clause 10 for more details about internationalization support.

7.1 File references

The file reference part defined by the `References` element allows a tool to determine the integrity of an OVF package without having to parse or interpret the entire structure of the descriptor. Tools can safely manipulate (for example, copy or archive) OVF packages with no risk of losing files.

External string resource bundle files for internationalization shall be placed first in the `References` element. See clause 10 for details.

Each `File` element in the reference part shall be given an identifier using the `ovf:id` attribute. The identifier shall be unique inside an OVF package. Each `File` element shall be specified using the `ovf:href` attribute, which shall contain a URL. Relative-path references and the URL schemes `"file"`, `"http"`, and `"https"` shall be supported, (see RFC1738 and RFC3986). Relative path references shall not contain “..” dot-segments. Other URL schemes should not be used. If no URL scheme is specified, the value of the `ovf:href` attribute shall be interpreted as a path name of the referenced file relative to the location of the OVF descriptor itself. The relative path name shall use the

syntax of relative-path references in RFC3986. The referenced file shall exist. Two different `File` elements shall not reference the same file with their `ovf:href` attributes.

The size of the referenced file may be specified using the `ovf:size` attribute. The unit of this attribute shall be bytes. If present, the value of the `ovf:size` attribute should match the actual size of the referenced file.

Each file referenced by a `File` element may be compressed using gzip (see RFC1952). When a `File` element is compressed using gzip, the `ovf:compression` attribute shall be set to "`gzip`". Otherwise, the `ovf:compression` attribute shall be set to "`identity`" or the entire attribute omitted. Alternatively, if the href is an HTTP or HTTPS URL, the compression may be specified by the HTTP server by using the HTTP header `Content-Encoding: gzip` (see RFC2616). Using HTTP content encoding in combination with the `ovf:compression` attribute is allowed, but in general does not improve the compression ratio. When compression is used, the `ovf:size` attribute shall specify the size of the actual compressed file.

Files referenced from the reference part may be split into chunks to accommodate file size restrictions on certain file systems. Chunking shall be indicated by the presence of the `ovf:chunkSize` attribute; the value of `ovf:chunkSize` attribute shall be the size of each chunk, except the last chunk, which may be smaller.

If the `ovf:chunkSize` attribute is specified, the `File` element shall reference a chunk file representing a chunk of the entire file. In this case, the value of the `ovf:href` attribute specifies only a part of the URL, and the syntax for the URL resolving to the chunk file shall be as follows:

```
chunk-url     = href-value "." chunk-number
chunk-number  = 9(decimal-digit)
decimal-digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The syntax is defined in ABNF notation with the exceptions listed in ANNEX A. The href-value shall be the value of the `ovf:href` attribute. The chunk-number shall be the 0-based position of the chunk starting with the value 0 and increasing with increments of 1 for each chunk.

If chunking is combined with compression, the entire file shall be compressed before chunking and each chunk shall be an equal slice of the compressed file, except for the last chunk which may be smaller.

If the OVF package has a manifest file, the file name in the manifest entries shall match the value of the `ovf:href` attribute for the file, except if the file is split into multiple chunks, in which case the `chunk-url` shall be used, and the manifest file shall contain an entry for each individual chunk. If chunked files are used, the manifest file may contain an entry for the entire file; and if present, this digest shall also be verified. See D.4 for an example.

7.2 Content element

Virtual system configurations in an OVF package are represented by a `VirtualSystem` or `VirtualSystemCollection` element. These elements shall be given an identifier using the `ovf:id` attribute. Direct child elements of a `VirtualSystemCollection` shall have unique identifiers.

In the OVF Schema, the `VirtualSystem` and `VirtualSystemCollection` elements are part of a substitution group with the `Content` element as head of the substitution group. The `Content` element is abstract and cannot be used directly. The OVF descriptor shall have one or more `Content` elements.

The `VirtualSystem` element describes a single virtual system and is a container of section elements. These section elements describe virtual hardware, resources, and product information as defined in clauses 8 and 9. See D.5 for an example.

The `VirtualSystemCollection` element is a container of zero or more `VirtualSystem` or `VirtualSystemCollection` elements. Thus, arbitrary complex configurations can be described. The section elements at the `VirtualSystemCollection` level describe appliance information, properties, and resource requirements as defined in clause 9. See D.5 for an example.

All elements in the `Content` substitution group shall contain an `Info` element and may contain a `Name` element. The `Info` element contains a human readable description of the meaning of this entity. The `Name` element is a localizable display name of the content. Clause 10 defines how to localize the `Info` and `Name` element.

7.3 Extensibility

Custom metadata may be added to OVF descriptors in several ways:

- New section elements may be defined as part of the `Section` substitution group, and used where the OVF Schemas allow sections to be present. All subtypes of the `Section` element shall contain an `Info` element that contains a human-readable description of the meaning of this entity. The values of `Info` elements can be used, for example, to give meaningful warnings to users when a section is being skipped, even if the parser does not know anything about the section. Clause 10 defines how to localize the `Info` element.
- The OVF Schemas use an open content model, where all existing types may be extended at the end with additional elements. Extension points are declared in the OVF Schemas with `xs:any` declarations with `namespace="##other"`.
- The OVF Schemas allow additional attributes on existing types.

Custom extensions shall not use XML namespaces defined in this specification. This applies to both custom elements and custom attributes.

If custom elements are used, the `ovf:required` attribute specifies whether the information in the element is mandatory or is optional. If not specified, the `ovf:required` attribute defaults to TRUE, i.e., mandatory. A deployment function that detects a custom element that is mandatory and that it does not understand shall fail.

If custom attributes are used, the information contained in them shall not be required for correct behavior.

If a `Section` element defined in the OVF Schema is used and it contains additional child elements that are not understood and the value of their `ovf:required` attribute is TRUE, the deployment function shall fail.

See D.6 for an example.

7.4 Conformance

This standard defines three conformance levels for OVF descriptors, with 1 being the highest level of conformance:

- Conformance Level: 1 - The OVF descriptor uses only sections and elements and attributes that are defined in this specification.
- Conformance Level: 2 - The OVF descriptor uses custom sections or elements or attributes that are not defined in this specification and all such extensions are optional as defined in 7.3.
  Conformance Level: 3 - The OVF descriptor uses custom sections or elements that are not defined in this specification and at least one such extension is required as defined in 7.3. The definition of all required extensions shall be publicly available in an open and unencumbered XML Schema. The complete specification may be inclusive in the XML Schema or available as a separate document.

The use of conformance level 3 should be avoided if the OVF package is intended to be portable.

The conformance level is not specified directly in the OVF descriptor but shall be determined by the above rules.

8 Virtual hardware description

8.1 VirtualHardwareSection

The `VirtualHardwareSection` element can be used to describe the virtual hardware used by the virtual system.

This standard allows incomplete virtual hardware descriptions.

The virtualization platform may create additional virtual hardware devices.

The virtual hardware devices listed in the `VirtualHardwareSection` element shall be realized.

This virtual hardware description is based on the CIM classes `CIM_VirtualSystemSettingData`, `CIM_ResourceAllocationSettingData`, `CIM_EthernetPortAllocationSettingData`, and `CIM_StorageAllocationSettingData`. The XML representation of the CIM model is based on the WS-CIM mapping as defined in DSP0230.

NOTE This means that the XML elements that belong to the class complex type should be ordered by Unicode code point (binary) order of their CIM property name identifiers. See D.7 for an example.

A `VirtualSystem` element shall have a `VirtualHardwareSection` direct child element. The `VirtualHardwareSection` shall not be a direct child element of a `VirtualSystemCollection` element or of an `Envelope` element.

One or more `VirtualHardwareSection` elements may occur within a `VirtualSystem` element. See ANNEX F for virtual hardware deployment considerations. If more than one `VirtualHardwareSection` element occurs, an `ovf:id` attribute shall be used to identify the element. If present, the `ovf:id` attribute value shall be unique within the `VirtualSystem` element.

The `ovf:transport` attribute specifies the transport media type by which `property` elements are passed to the virtual system. See 9.5 for a description of `property` elements. See 11.2 for a description of transport types.

A `VirtualHardwareSection` element contains child elements that describe virtual system and virtual hardware resources (CPU, memory, network, and storage).

A `VirtualHardwareSection` element shall have the following direct child elements:

- zero or one `System` elements
- zero or more `Item` elements
- zero or more `EthernetPortItem` elements
- zero or more `StorageItem` elements.

The `System` element is an XML representation of the values of one or more properties of the CIM class `CIM_VirtualSystemSettingData`. The `vssd:VirtualSystemType`, a direct child element of `System` element, specifies a virtual system type identifier, which is an implementation defined string that uniquely identifies the type of the virtual system. Zero or more virtual system type identifiers may be specified, separated by single space character. In order for the OVF virtual system to be deployable on a target platform, the virtual system on the target platform should support at least one of the virtual system types identified in the `vssd:VirtualSystemType` elements. The virtual system type identifiers specified in `vssd:VirtualSystemType` elements are expected to be matched against the values of property VirtualSystemTypesSupported of CIM class `CIM_VirtualSystemManagementCapabilities`.

The virtual hardware characteristics are described as a sequence of `Item` elements. The `Item` element is an XML representation of an instance of the CIM class `CIM_ResourceAllocationSettingData`. The element can describe all memory and CPU requirements as well as virtual hardware devices.

Multiple device subtypes may be specified in an `Item` element, separated by a single space (0x20) character.

The network hardware characteristics are described as a sequence of `EthernetPortItem` elements. The `EthernetPortItem` element is an XML representation of the values of one or more properties of the CIM class `CIM_EthernetPortAllocationSettingData`.

The storage hardware characteristics are described as a sequence of `StorageItem` elements. The `StorageItem` element is an XML representation of the values of one or more properties of the CIM class `CIM_StorageAllocationSettingData`.

8.2 Extensibility

The `ovf:required` attribute is optional on the `Item, EthernetPortItem, or StorageItem` elements. If used it specifies whether the realization of the element is required for correct behavior of the guest software. If not specified, `ovf:required` defaults to TRUE.

On child elements of the `Item, EthernetPortItem,` or `StorageItem` elements, the `ovf:required` attribute shall be interpreted, even though these elements are in a different RASD-WS-CIM namespace. A tool parsing an `Item` element should act according to Table 2.

Table 2 – Actions for child elements with `ovf:required` attribute

Child Element	`ovf:required` Attribute Value	Action
Known	TRUE or not specified	Shall interpret `Item, EthernetPortItem, or StorageItem`
Known	FALSE	Shall interpret `Item, EthernetPortItem, or StorageItem`
Unknown	TRUE or not specified	Shall fail `Item, EthernetPortItem, or StorageItem`
Unknown	FALSE	Shall ignore Child element

8.3 Virtual hardware elements

The element type of the `Item` element in a `VirtualHardwareSection` element is `CIM_ResourceAllocationSettingData_Type` as defined in `CIM_ResourceAllocationSettingData`. See ANNEX B.

The child elements of `Item` represent the values of one or more properties exposed by the `CIM_ResourceAllocationSettingData` class. They have the semantics of defined settings as defined in DSP1041, any profiles derived from DSP1041 for specific resource types, and this standard. See D.8 for an example.

The element type of the `EthernetPortItem` element in a `VirtualHardwareSection` element is `CIM_EthernetPortAllocationSettingData_Type` as defined in `CIM_EthernetPortAllocationSettingData`. See ANNEX B.

The child elements represent the values of one or more properties exposed by the `CIM_EthernetPortAllocationSettingData` class. They have the semantics of defined resource allocation setting data as defined in DSP1050, any profiles derived from DSP1050 for specific Ethernet port resource types, and this standard. See D.8 for an example.

The element type of the `StorageItem` element in a `VirtualHardwareSection` element is `CIM_StorageAllocationSettingData_Type` as defined in `CIM_StorageAllocationSettingData`. See ANNEX B

The child elements represent the values of one or more properties exposed by the `CIM_StorageAllocationSettingData` class. They have the semantics of defined resource allocation setting data as defined in DSP1047, any profiles derived from DSP1047 for specific storage resource types, and this standard. See D.8 for an example.

The `Description` element is used to provide additional metadata about the `Item`, `EthernetPortItem`, or `StorageItem` element itself. This element enables a consumer of the OVF package to provide descriptive information about all items, including items that were unknown at the time the application was written.

The `Caption`, `Description` and `ElementName` elements are localizable using the `ovf:msgid` attribute from the OVF envelope namespace. See clause 10 for more details about internationalization support.

The optional `ovf:configuration` attribute contains a list of configuration names. See 9.8 on deployment options for semantics of this attribute. The optional `ovf:bound` attribute is used to specify ranges; see 8.4.

All Ethernet adapters in the OVF package that connect to the same network shall have a `Connection` element that contains the same logical network name. If a `Connection` element is used to represent a network, the corresponding network shall be represented as a child element of the `Network Section` element with a name attribute that matches the value of the `Connection` element.

The `HostResource` element is used to refer to resources included in the OVF descriptor as well as logical devices on the deployment function. Values for `HostResource` elements referring to resources included in the OVF descriptor are formatted as URIs as specified in Table 3.

Table 3 – HostResource element

Content	Description
`ovf:/file/<id>`	A reference to a file in the OVF, as specified in the References section. <id> shall be the value of the `ovf:id` attribute of the `File` element being referenced.
`ovf:/disk/<id>`	A reference to a virtual disk, as specified in the DiskSection or SharedDiskSection. <id> shall be the value of the `ovf:diskId` attribute of the `Disk` element being referenced.

See ANNEX F for virtual hardware deployment considerations. More than one backing for a device shall not be specified in a `VirtualHardware` element.

Table 4 gives a brief overview on how elements from RASD, EPASD, and SASD namespaces are used to describe virtual devices and controllers.

Table 4 – Elements for virtual devices and controllers

Element	Usage
`Description`	Is a human-readable description of the meaning of the information. For example, “Specifies the memory size of the virtual system”.
`ElementName`	Is a human-readable description of the content.
`InstanceID`	Specifies a unique instance ID of the element within the section.
`HostResource`	Specifies how a virtual device connects to a resource on the virtualization platform. Not all devices need a backing. See Table 3.

`ResourceType`
`OtherResourceType`
`ResourceSubtype`
	Specifies the kind of device that is being described.

`AutomaticAllocation`	For devices that are connectable, such as floppies, CD-ROMs, and Ethernet adaptors, specifies whether the device should be connected at power on.
`Parent`	Specifies the InstanceID of the parent controller (if any).
`Connection`	Used with Ethernet adapters to specify the network connection name for the virtual system.
`Address`	Is device specific.
`AddressOnParent`	For a device, specifies its location on the controller.
`AllocationUnits`	Specifies the unit of allocation used.
`VirtualQuantity`	Specifies the quantity of a resource presented.
`Reservation`	Specifies the minimum quantity of a resource guaranteed to be available.
`Limit`	Specifies the maximum quantity of a resource that is granted.
`Weight`	Specifies a relative priority for this allocation in relation to other allocations.

Only fields directly related to describing devices are mentioned. Refer to the CIM MOF for a complete description of all fields, each field corresponds to the identically named property in the `CIM_ResourceAllocationSettingData` class or a class derived from it.

8.4 Ranges on elements

The optional `ovf:bound` attribute may be used to specify ranges for the `Item` elements. A range has a minimum, normal, and maximum value, denoted by `min`, `normal`, and `max`, where `min <= normal <= max`. The default values for `min` and `max` are those specified for `normal`.

See ANNEX F for virtual hardware deployment considerations.

For the `Item`, `EthernetPortItem`, and `StorageItem` elements in the `VirtualHardwareSection` and the `ResourceAllocationSection` elements, the following additional semantics are defined:

- Each `Item`, `EthernetPortItem`, or `StorageItem` element has an optional `ovf:bound` attribute. This value may be specified as `min`, `max`, or `normal`. The value defaults to `normal`.
- If the `ovf:bound` value is specified as either `min` or `max`, the item is used to specify the upper or lower bound for one or more values for a given InstanceID. Such an item is called a range marker.

The semantics of range markers are as follows:

- `InstanceID` and `ResourceType` shall be specified, and the `ResourceType` shall match other `Item` elements with the same `InstanceID`.
- No more than one `min` range marker and no more than one `max` range marker for a given RASD, EPASD, or SASD (identified with `InstanceID`) shall be specified.

- An `Item`, `EthernetPortItem`, or `StorageItem` element with a range marker shall have a corresponding `Item`, `EthernetPortItem`, or `StorageItem` element without a range marker; that is, an `Item`, `EthernetPortItem`, and `StorageItem` element with no `ovf:bound` attribute or `ovf:bound` attribute with value normal. This corresponding item specifies the default value.
- For an `Item`, `EthernetPortItem`, and `StorageItem` element where only a `min` range marker is specified, the `max` value is unbounded upwards within the set of valid values for the property.
- For an `Item`, `EthernetPortItem`, and `StorageItem` where only a `max` range marker is specified, the `min` value is unbounded downwards within the set of valid values for the property.
- The default value shall be inside the range.
- Non-integer elements shall not be used in the range markers for RASD, EPASD, or SASD.

See D.9 for an example.

9 Core metadata sections

Table 5 shows the core metadata sections that are defined in the `ovf` namespace.

Table 5 – Core metadata sections

Section element	Parent element	Multiplicity
`DiskSection` Describes meta-information about all virtual disks in the package	Envelope	Zero or one
`NetworkSection` Describes logical networks used in the package	Envelope	Zero or one
`ResourceAllocationSection` Specifies reservations, limits, and shares on a given resource, such as memory or CPU for a virtual system collection	VirtualSystemCollection	Zero or one
`AnnotationSection` Specifies a free-form annotation on an entity	VirtualSystem VirtualSystemCollection	Zero or one
`ProductSection` Specifies product-information for a package, such as product name and version, along with a set of properties that can be configured	VirtualSystem VirtualSystemCollection	Zero or more
`EulaSection` Specifies a license agreement for the software in the package	VirtualSystem VirtualSystemCollection	Zero or more
`StartupSection` Specifies how a virtual system collection is powered on	VirtualSystemCollection	Zero or one
`DeploymentOptionSection` Specifies a discrete set of intended resource requirements	Envelope	Zero or one
`OperatingSystemSection` Specifies the guest software installed in a virtual system	VirtualSystem	Zero or one
`InstallSection` Specifies that the virtual system needs to be initially booted to install and configure the software	VirtualSystem	Zero or one
`EnvironmentFilesSection` Specifies additional files from an OVF package to be included in the OVF environment	VirtualSystem	Zero or one
`BootDeviceSection` Specifies boot device order to be used by a virtual system	VirtualSystem	Zero or more
`SharedDiskSection` Specifies virtual disks shared by more than one VirtualSystems at runtime	Envelope	Zero or one
`ScaleOutSection` Specifies that a VirtualSystemCollection contain a set of children that are homogeneous with respect to a prototype	VirtualSystemCollection	Zero or more
`PlacementGroupSection` Specifies a placement policy for a group of VirtualSystems or VirtualSystemCollections	Envelope	Zero or more
`PlacementSection` Specifies membership of a particular placement policy group	VirtualSystem VirtualSystemCollection	Zero or one

Section element	Parent element	Multiplicity
`EncryptionSection` Specifies encryption scheme for encrypting parts of an OVF descriptor or files to which it refers.	Envelope	Zero or one

The following subclauses describe the semantics of the core sections and provide some examples. The sections are used in several places of an OVF envelope; the description of each section defines where it may be used. See the DSP8023 schema for a detailed specification of all attributes and elements.

In the OVF Schema, all sections are part of a substitution group with the `Section` element as head of the substitution group. The `Section` element is abstract and cannot be used directly.

9.1 DiskSection

The `DiskSection` element describes meta-information about the virtual disks in the OVF package. The virtual disks and associated metadata are described outside of the `VirtualHardwareSection` element to facilitate sharing between the virtual systems within an OVF package.

The virtual disks in the `DiskSection` element may be referenced by one or more virtual systems. However, as seen from the guest software, each virtual system gets individual private disks. Any level of sharing done at runtime is virtualization platform specific and not visible to the guest software. See 9.13 for details about how to configure sharing of a virtual disk at runtime with concurrent access. See D.10 for an example.

The `DiskSection` element is only valid as a direct child element of the Envelope element.

Each virtual disk represented by a `Disk` element shall be given an identifier using the `ovf:diskId` attribute; the identifier shall be unique within the `DiskSection` element.

The capacity of a virtual disk shall be specified by the `ovf:capacity` attribute with an `xs:long` integer value. The default unit of allocation shall be bytes. The optional string attribute `ovf:capacityAllocationUnits` may be used to specify a particular unit of allocation. Values for `ovf:capacityAllocationUnits` shall match the format for programmatic units defined in DSP0004 with the restriction that the base unit shall be `"byte"`.

The `ovf:fileRef` attribute denotes the virtual disk content by identifying an existing `File` element in the `References` element. The `File` element is identified by matching its `ovf:id` attribute value with the `ovf:fileRef` attribute value. Omitting the `ovf:fileRef` attribute shall indicate an empty disk. If an empty disk is indicated, the virtual disk shall be created and the content zeroed at deployment.

The format URI (see 5.2) of a non-empty virtual disk shall be specified by the `ovf:format` attribute.

Different `Disk` elements shall not contain `ovf:fileRef` attributes with identical values. `Disk` elements shall be ordered such that they identify any `File` elements in the same order as these are defined in the `References` element.

For empty disks, rather than specifying a fixed virtual disk capacity, the capacity may be given using a reference to a `Property` element in a `ProductSection` element. This is done by setting `ovf:capacity="${<id>}"` where `<id>` shall be the identifier of a `Property` element in the `ProductSection` element. The `Property` element value shall resolve to an `xs:long` integer value. See 9.5 for a description of `Property` elements. The `ovf:capacityAllocationUnits` attribute is useful when using `Property` elements because a user may be prompted and can then enter disk sizing information in appropriate units, for example gigabytes.

For non-empty disks, the actual used size of the disk may be specified using the `ovf:populatedSize` attribute. The unit of this attribute shall be bytes. The `ovf:populatedSize` attribute may be an estimate of used disk size but shall not be larger than `ovf:capacity`.

In VirtualHardwareSection, virtual disk devices may have a rasd:HostResource element referring to a Disk element in DiskSection; see 8.3. The virtual disk capacity shall be defined by the ovf:capacity attribute on the Disk element. If a rasd:VirtualQuantity element is specified along with the rasd:HostResource element, the virtual quantity value shall not be considered and may have any value.

A disk image may be represented as a set of modified blocks in comparison to a parent image. The use of parent disks can often significantly reduce the size of an OVF package if it contains multiple disks with similar content, such as a common base operating system. See ANNEX F for deployment considerations.

For the Disk element, a parent disk may be specified using the ovf:parentRef attribute that shall contain a valid ovf:diskId reference to a different Disk element. If a disk block does not exist locally, lookup for that disk block then occurs in the parent disk. In DiskSection, parent Disk elements shall occur before child Disk elements that refer to them. Similarly, in References element, the File elements referred from these Disk elements shall respect the same ordering. The ordering restriction ensures that parent disks always occur before child disks, making it possible for a tool to consume the OVF package in a streaming mode; see also clause 5.3.

9.2 NetworkSection

The NetworkSection element shall list all logical networks used in the OVF package. See D.11 for an example.

The NetworkSection is only valid as a direct child element of the Envelope element. A Network element is a child element of NetworkSection. Each Network element in the NetworkSection shall be given a unique name using the ovf:name attribute. The name shall be unique within an OVF envelope.

All networks referred to from Connection elements in all VirtualHardwareSection elements shall be defined in the NetworkSection.

Each logical network may contain a set of networking attributes that should be applied when mapping the logical network at deployment time to a physical or virtual network. Networking attributes are specified by zero or more instances of NetworkPortProfile child element or NetworkPortProfileURI child element of the Network element.

The NetworkPortProfile element shall contain zero or more instances of Item elements of type epasd:CIM_EthernetPortAllocationSettingData_Type that define the contents of zero or more network port profiles. The NetworkPortProfileURI shall be a URI reference to a network port profile.

Examples of using the network port profiles are in ANNEX E.

9.3 ResourceAllocationSection

The ResourceAllocationSection element describes all resource allocation requirements of a VirtualSystemCollection entity and applies only to the direct child VirtualSystem elements that do not contain a VirtualHardwareSection element. It does not apply to a child VirtualSystemCollection elements.

See ANNEX F for deployment considerations. See D.12 for an example.

The ResourceAllocationSection is a valid element for a VirtualSystemCollection entity.

The ovf:configuration attribute is optional and contains a list of configuration names. See 9.8 on deployment options for semantics of this attribute.

The ovf:bound attribute is optional and contains a value of min, max, or normal. See 8.4 for semantics of this attribute.

9.4 AnnotationSection

The `AnnotationSection` element is a user-defined annotation on an entity. See ANNEX F for deployment considerations. See D.13 for an example.

The `AnnotationSection` element is a valid element for the `VirtualSystem` and the `VirtualSystemCollection` entities.

See clause 10 for details about how to localize the `Annotation` element.

9.5 ProductSection

The `ProductSection` element specifies product-information for an appliance, such as product name, version, and vendor. Typically it corresponds to a particular software product that is installed.

Zero or more elements may be specified within a `VirtualSystem` element or `VirtualSystemCollection` element.

Each `ProductSection` element with the same parent element shall have a unique `ovf:class` and `ovf:instance` attribute pair. If there is only one `ProductSection` element, the `ovf:class` and `ovf:instance` attributes are optional and default to an empty string.

The `ovf:class` attribute should be used to identify the software product using the reverse domain name convention. Examples of values are com.vmware.tools and org.apache.tomcat. If multiple instances of the same product are installed, the `ovf:instance` attribute shall be used to identify the different instances.

If a `ProductSection` element exists, the first `ProductSection` element defined in the `VirtualSystem` element or `VirtualSystemCollection` element that is the direct child element of the root element of an OVF package shall define summary information that describes the entire package. This information may be mapped into an instance of the `CIM_Product` class.

See D.14 for an example.

The `Product` element is optional and specifies the name of the product.

The `Vendor` element is optional and specifies the name of the product vendor.

The `Version` element is optional and specifies the product version in short form.

The `FullVersion` element is optional and describes the product version in long form.

The `ProductUrl` element is optional and specifies a URL that shall resolve to a human-readable description of the product.

The `VendorUrl` element is optional and specifies a URL that shall resolve to a human-readable description of the vendor.

The `AppUrl` element is optional and specifies a URL resolving to the deployed product instance.

The `Icon` element is optional and specifies display icons for the product.

9.5.1 Property elements

The `Property` elements specify customization parameters and are relevant to appliances that need to be customized during deployment with specific settings such as network identity, the IP addresses of DNS servers, gateways, and others.

The `ProductSection` is a valid section for a `VirtualSystem` and a `VirtualSystemCollection` entity.

The `Property` elements may be grouped by using `Category` elements. The set of `Property` elements grouped by a `Category` element is the sequence of `Property` elements following the `Category`

element, until but not including an element that is not a `Property` element. For OVF packages containing a large number of `Property` elements, this may provide a simpler installation experience. Similarly, each `Property` element may have a short label defined by its `Label` child element in addition to a description defined by its `Description` child element. See clause 10 for details about how to localize the `Category` element and the `Description` and `Label` child elements of the `Property` element.

Each `Property` element in a `ProductSection` shall be given an identifier that is unique within the `ProductSection` using the `ovf:key` attribute. The `ovf:key` attribute shall not contain the period character ('.') or the colon character (':')

Each `Property` element in a `ProductSection` shall be given a type using the `ovf:type` attribute and optionally type qualifiers using the `ovf:qualifiers` attribute. Valid types are listed in Table 6, and valid qualifiers are listed in Table 7.

The optional attribute `ovf:value` is used to provide a default value for a `Property` element. One or more optional `Value` elements may be used to define alternative default values for different configurations, as defined in 9.8.

The optional attribute `ovf:userConfigurable` determines whether the property value is configurable during the installation phase. If `ovf:userConfigurable` is FALSE or omitted, the `ovf:value` attribute specifies the value to be used for that customization parameter during installation. If `ovf:userConfigurable` is TRUE, the `ovf:value` attribute specifies a default value for that customization parameter, which may be changed during installation.

A simple OVF implementation, such as a command-line installer, typically uses default values for properties and does not prompt even though `ovf:userConfigurable` is set to `TRUE`. To force prompting at startup time, omitting the `ovf:value` attribute is sufficient for integer types, because the empty string is not a valid integer value. For string types, prompting may be forced by adding a qualifier requiring a non-empty string; see Table 7.

The `ovf:password` attribute indicates that the property value may contain sensitive information. The default value is `FALSE`. OVF implementations prompting for property values are advised to obscure these values when the `ovf:password` attribute is set to `TRUE`. Note that this mechanism affords limited security protection only. Although sensitive values are masked from casual observers, default values in the OVF descriptor and assigned values in the OVF environment are still passed in clear text.

The ID and the value of the `Property` elements are exposed to the guest software using the OVF environment file. The ovf:class and ovf:instance attributes shall not contain the colon character (':'). If only one instance of a product is installed, the `ovf:instance` attribute should not be set. The value of the `ovfenv:key` attribute of a `Property` element exposed in the OVF environment shall be constructed from the value of the `ovf:key` attribute of the corresponding `Property` element defined in a `ProductSection` entity of an OVF descriptor as follows:

```
key-value-env = [class-value "."] key-value-prod ["." instance-value]
```

The syntax definition above use ABNF with the exceptions listed in ANNEX A, where:

- `class-value` is the value of the `ovf:class` attribute of the `Property` element defined in the `ProductSection` entity. The production `[class-value "."]` shall be present if and only if `class-value` is not the empty string.
- `key-value-prod` is the value of the `ovf:key` attribute of the `Property` element defined in the `ProductSection` entity.
- `instance-value` is the value of the `ovf:instance` attribute of the `Property` element defined in the `ProductSection` entity. The production `["." instance-value]` shall be present if and only if `instance-value` is not the empty string.

If the `ovf:userConfigurable` attribute is TRUE, the deployment function should prompt for values of the `Property` elements. These `Property` elements may be defined in multiple `ProductSection` elements.

`Property` elements specified on a `VirtualSystemCollection` element are also seen by its immediate child elements. Child elements may refer to the properties of a parent `VirtualSystemCollection` element using macros on the form `${name}` as value for `ovf:value` attributes.

Table 6 lists the valid types for properties. These are a subset of CIM intrinsic types defined in DSP0004 that also define the value space and format for each intrinsic type. Each `Property` element shall specify a type using the `ovf:type` attribute.

Table 6 – Property types

Type	Description
`uint8`	Unsigned 8-bit integer
`sint8`	Signed 8-bit integer
`uint16`	Unsigned 16-bit integer
`sint16`	Signed 16-bit integer
`uint32`	Unsigned 32-bit integer
`sint32`	Signed 32-bit integer
`uint64`	Unsigned 64-bit integer
`sint64`	Signed 64-bit integer
`String`	String
`Boolean`	Boolean
`real32`	IEEE 4-byte floating point
`real64`	IEEE 8-byte floating point

Table 7 lists the supported CIM type qualifiers as defined in DSP0004. Each `Property` element may optionally specify type qualifiers using the `ovf:qualifiers` attribute with multiple qualifiers separated by commas; see production `qualifierList` in ANNEX A “MOF Syntax Grammar Description” in DSP0004.

Table 7 – Property qualifiers

Property Type	Property Qualifier

`String`	`MinLen(min)`
	`MaxLen(max)`
	`ValueMap{...}`

`uint8`
`sint8`
`uint16`
`sint16`
`uint32`
`sint32`
`uint64`
`sint64`
	`ValueMap{...}`

9.6 EulaSection

A `EulaSection` contains the legal terms for using its parent `Content` element. Multiple `EulaSections` may be present in an OVF. See ANNEX F for deployment considerations. See D.15 for an example. The `EulaSection` is a valid section for a `VirtualSystem` and a `VirtualSystemCollection` entity.

See clause 10 for details about how to localize the `License` element.

See also clause 10 for a description of storing EULA license contents in an external file without any XML header or footer. This allows inclusion of standard license or copyright text files in unaltered form.

9.7 StartupSection

The `StartupSection` element specifies how a collection of virtual systems identified by a `VirtualSystemCollection` element is powered on and off. The `StartupSection` element shall not be part of a `VirtualSystem` element. See D.16 for an example.

If a `VirtualSystemCollection` element has a `StartupSection` element then each `VirtualSystem` element or `VirtualSystemCollection` element that is a direct child element shall have a corresponding `Item` element in the `StartupSection` element.

When a start or stop action is performed on a `VirtualSystemCollection` element, the respective actions on the `Item` elements of its `StartupSection` element are invoked in the specified order. Whenever an `Item` element corresponds to a nested `VirtualSystemCollection` element, the actions on the `Item` elements of its `StartupSection` element shall be invoked before the action on the `Item` element corresponding to that `VirtualSystemCollection` element is invoked (i.e., depth-first traversal).

The following required attributes on `Item` element are supported for a `VirtualSystem` and `VirtualSystemCollection` elements:

- `ovf:id` shall match the value of the `ovf:id` attribute of a `Content` element which is a direct child of this `VirtualSystemCollection`. That `Content` element describes the virtual system or virtual system collection to which the actions defined in the `Item` element apply.
- `ovf:order` specifies the startup order of the item using non-negative integer values. If the `ovf:order ="0"`, the order is not specified. If the `ovf:order` is non-zero, the order of execution of the start action shall be the numerical ascending order of the values. The `Items` with same order identifier may be started concurrently.

The order of execution of the stop action should be the numerical descending order of the values if the `ovf:shutdownorder` attribute is not specified. In implementation-specific scenarios, the order of execution of the stop action may be non-descending.

The following optional attributes on the `Item` element are supported for a `VirtualSystem` element.

- `ovf:shutdownorder` specifies the shutdown order using non-negative integer values. If the `ovf:shutdownorder ="0"`, the shutdown order is not specified. If the `ovf:shutdownorder` is non-zero, the order of execution of the stop action shall be the numerical descending order of the values. The `Items` with same order identifier may be stopped concurrently.
- `ovf:startDelay` specifies a delay in seconds to wait until proceeding to the next virtual system in the start sequence. The default value is 0.
- `ovf:waitingForGuest` enables the virtualization platform to resume the startup sequence after the guest software has reported it is ready. The interpretation of this is virtualization platform specific. The default value is FALSE.
- `ovf:startAction` specifies the start action to use. Valid values are `powerOn` and `none`. The default value is `powerOn`.
- `ovf:stopDelay` specifies a delay in seconds to wait until proceeding to the previous order in the stop sequence. The default value is `0`.
- `ovf:stopAction` specifies the stop action to use. Valid values are `powerOff`, `guestShutdown`, and `none`. The interpretation of `guestShutdown` is virtualization platform specific. The default value is `powerOff`.

If the `StartupSection` element is not specified, an `ovf:order="0"` attribute is implied.

9.8 DeploymentOptionSection

The `DeploymentOptionSection` element specifies a discrete set of intended resource configurations. The author of an OVF package can include sizing metadata for different configurations. The deployment shall select one of the configurations, e.g., by prompting the user. The selected configuration shall be available in the OVF environment file. See ANNEX F.

The `DeploymentOptionSection` specifies an ID, label, and description for each configuration. See D.17 for an example.

The `DeploymentOptionSection` has the following semantics:

- If present, the `DeploymentOptionSection` is valid only as a direct child element of the root element. Only one `DeploymentOptionSection` section shall be present in an OVF descriptor.
- The discrete set of configurations is described with `Configuration` elements, which shall have identifiers specified by the `ovf:id` attribute that are unique in the OVF package.
- A default `Configuration` element may be specified with the optional `ovf:default` attribute. Only one default `Configuration` element shall be specified. If no default is specified, the first element in the list is the default.
- The `Label` and `Description` elements are localizable using the `ovf:msgid` attribute. See clause 10 for more details about internationalization support.

Configurations may be used to control resources for virtual hardware and for virtual system collections. The `Item`, `EthernetPortItem`, and `StorageItem` elements in `VirtualHardwareSection` elements describe resources for VirtualSystem entities, while the `Item`, `EthernetPortItem`, and `StorageItem` elements in `ResourceAllocationSection` elements describe resources for virtual system collections. For these two `Item`, `EthernetPortItem`, or `StorageItem` types, the following additional semantics are defined:

- Each `Item`, `EthernetPortItem`, and `StorageItem` has an optional `ovf:configuration` attribute, containing a list of configurations separated by a single space character. If not specified, the item shall be selected for any configuration. If specified, the item shall be selected only if the chosen configuration ID is in the list. A configuration attribute shall not contain a configuration ID that is not specified in the `DeploymentOptionSection`.
- Within a single `VirtualHardwareSection` or `ResourceAllocationSection`, multiple `Item`, `EthernetPortItem`, and `StorageItem` elements are allowed to refer to the same InstanceID. A single combined `Item`, `EthernetPortItem`, or `StorageItem` for the given InstanceID shall be constructed by picking up the child elements of each `Item`, `EthernetPortItem`, or `StorageItem` element, with child elements of a former `Item`, `EthernetPortItem`, or `StorageItem` element in the OVF descriptor not being picked up if there is a like-named child element in a latter `Item`, `EthernetPortItem`, or `StorageItem` element. Any attributes specified on child elements of `Item`, `EthernetPortItem`, or `StorageItem` elements that are not picked up that way, are not part of the combined `Item`, `EthernetPortItem`, or `StorageItem` element.
- All `Item`, `EthernetPortItem`, `StorageItem` elements shall specify `ResourceType`, and `Item`, `EthernetPortItem`, and `StorageItem` elements with the same InstanceID shall agree on ResourceType.

Note that the attributes `ovf:configuration` and `ovf:bound` on `Item` may be used in combination to provide flexible configuration options.

Configurations can further be used to control default values for properties and whether properties are user configurable. For `Property` elements inside a `ProductSection`, the following additional semantic is defined:

- It is possible to specify alternative default property values for different configurations in a `DeploymentOptionSection`. In addition to a `Label` and `Description` element, each

`Property` element may optionally contain `Value` elements. The `Value` element shall have an `ovf:value` attribute specifying the alternative default and an `ovf:configuration` attribute specifying the configuration in which this new default value should be used. Multiple `Value` elements shall not refer to the same configuration.

- A `Property` element may optionally have an `ovf:configuration` attribute specifying the configuration in which this property should be user configurable. The value of `ovf:userConfigurable` is implicitly set to FALSE for all other configurations, in which case the default value of the property may not be changed during installation.

9.9 OperatingSystemSection

An `OperatingSystemSection` specifies the operating system installed on a virtual system. See D.18 for an example.

The values for `ovf:id` should be taken from the `ValueMap` of the `CIM_OperatingSystem.OsType` property. The description should be taken from the corresponding `Values` of the `CIM_OperatingSystem.OsType` property.

The `OperatingSystemSection` element is a valid section for a `VirtualSystem` element only.

9.10 InstallSection

The `InstallSection` element, if specified, indicates that the virtual system needs to be booted once in order to install and/or configure the guest software. The guest software is expected to access the OVF environment during that boot, and to shut down after having completed the installation and/or configuration of the software, powering off the guest.

If the `InstallSection` is not specified, this indicates that the virtual system does not need to be powered on to complete installation of guest software. See D.19 for an example.

The `InstallSection` element shall be valid only for a `VirtualSystem` element.

The `ovf:initialBootStopDelay` attribute specifies a delay in seconds to wait for the virtual system to power off.

If the delay expires and the virtual system has not powered off, the deployment function shall indicate a failure.

An `ovf:initialBootStopDelay` attribute value of zero indicates that the boot stop delay is not specified.

Note that the guest software in the virtual system can do multiple reboots before powering off.

Several virtual systems in a virtual system collection may have an `InstallSection` element defined, in which case the above step is done for each virtual system that has an `InstallSection` element.

9.11 EnvironmentFilesSection

The `EnvironmentFilesSection` enables the OVF package to specify additional environment file(s) (AEF) besides the virtual disks. These AEFs enable increased flexibility in image customization outside of virtual disk capture, allowing an OVF package to provide customized solutions by combining existing virtual disks without modifying them.

The AEF contents are neither generated nor validated by the deployment function.

The AEFs are included in the transport media generated by the deployment function.

The AEFs are conveyed to the guest software using the indicated transport media type. The AEFs and OVF environment files are intended to use same transport media and transport media type

The `EnvironmentFilesSection` shall contain a `File` element with the attributes `ovf:fileRef` and `ovf:path` for each AEF provided to the guest software.

The `ovf:fileRef` attribute shall specify an existing `File` element in the `References` element. The `File` element is identified by matching its `ovf:id` attribute value with the `ovf:fileRef` attribute value.

The `ovf:path` attribute specifies the relative location in the transport media (see 11.1) where the file should be placed, using the syntax of relative-path references in RFC3986.

The referenced `File` element in the `References` element identifies the content using one of the URL schemes `"file"`, `"http"`, or `"https"`. For the `"file"` scheme, the content is static and included in the OVF package. See ANNEX F for deployment considerations

For details about transport media type, see 11.2.

9.12 BootDeviceSection

Individual virtual systems use the default device boot order provided by the virtualization platform's virtual BIOS. The `BootDeviceSection` allows the OVF package author to specify particular boot configurations and boot order settings. This enables booting from non-default devices, such as a NIC using PXE, a USB device, or a secondary disk. Moreover, there could be multiple boot configurations with different boot orders. For example, a virtual disk may need to be patched before it is bootable and a patch ISO image could be included in the OVF package.

The Common Information Model (CIM) defines artifacts to deal with boot order use cases prevalent in the industry for BIOSes found in desktops and servers. The boot configuration is defined by the class `CIM_BootConfigSetting` that in turn contains one or more `CIM_BootSourceSetting` classes as defined in the CIM Schema. Each class representing the boot source in turn has either the specific device or a "device type", such as disk or CD/DVD, as a boot source.

In the context of this specification, the `InstanceID` property of `CIM_BootSourceSetting` is used for identifying a specific device as the boot source. The `InstanceID` property of the device as specified in the `Item` description of the device in the `VirtualHardwareSection` element is used to specify the device as a boot source. In case the source is desired to be a device type, the `StructuredBootString` field is used to denote the type of device with values defined by the CIM boot control profile. See ANNEX F for deployment considerations.

See D.21 for an example.

9.13 SharedDiskSection

The existing `DiskSection` element in 9.1 describes virtual disks in the OVF package. Virtual disks in the `DiskSection` element can be referenced by multiple virtual systems, but seen from the guest software, each virtual system gets individual private disks. Any level of sharing done at runtime is virtualization platform specific and not visible to the guest software.

Certain applications, such as clustered databases, rely on multiple virtual systems sharing the same virtual disk at runtime. `SharedDiskSection` allows the OVF package to specify `Disk` elements shared by more than one virtual system at runtime. These virtual disks may be backed by an external `File` reference, or may be empty virtual disks without backing. It is recommended that the guest software use cluster-aware file system technology to be able to handle concurrent access. See D.22 for an example.

The `SharedDiskSection` is a valid section at the outermost envelope level only.

Each virtual disk is represented by a `SharedDisk` element that shall be given an identifier using the `ovf:diskId` attribute; the identifier shall be unique within the combined content of `DiskSection` and `SharedDiskSection` element. The `SharedDisk` element has the same structure as the `Disk` element in the `DiskSection` element, with the addition of an `ovf:readOnly` attribute. The `ovf:readOnly` is optional and states whether shared disk access is read-write, i.e., FALSE, or read-only, i.e., TRUE.

Shared virtual disks are referenced from virtual hardware by using the `HostResource` element as described in 8.3.

Support of the `SharedDiskSection` element is optional. The virtualization platform should give an appropriate error message based on the value of the `ovf:required` attribute on the `SharedDiskSection` element.

9.14 ScaleOutSection

The number of virtual systems or collections of virtual system contained in an OVF package is fixed and determined by the structure inside the `Envelope` element. The `ScaleOutSection` element allows a `VirtualSystemCollection` element to contain a set of children that are homogeneous with respect to a prototypical `VirtualSystem` or `VirtualSystemCollection` element. The `ScaleOutSection` element shall cause the deployment function to replicate the prototype a number of times, thus allowing the number of instantiated virtual systems to be configured dynamically at deployment time. See D.23 for an example.

This mechanism enables scaling of virtual system instances at deployment time. Scaling at runtime is not within the scope of this specification.

The `ScaleOutSection` element is a valid section inside `VirtualSystemCollection` element only.

The `ovf:id` attribute on `ScaleOutSection` element identifies the virtual system or collection of virtual systems prototype to be replicated.

For the `InstanceCount` element, the `ovf:minimum` and `ovf:maximum` attribute values shall be non-negative integers and `ovf:minimum` shall be less than or equal to the value of `ovf:maximum`. The `ovf:minimum` value may be zero in which case the `VirtualSystemCollection` may contain zero instances of the prototype. If the `ovf:minimum` attribute is not present, it shall be assumed to have a value of one. If the `ovf:maximum` attribute is not present, it shall be assumed to have a value of unbounded. The `ovf:default` attribute is required and shall contain a value within the range defined by `ovf:minimum` and `ovf:maximum`.

Each replicated instance shall be assigned a unique `ovf:id` value within the `VirtualSystemCollection` element. The unique `ovf:id` value shall be constructed from the prototype `ovf:id` value with a sequence number appended as follows:

```
replica-ovf-id = prototype-ovf-id "-" decimal-number
decimal-number = decimal-digit | (decimal-digit decimal-number)
decimal-digit  = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

The syntax definitions above use ABNF with the exceptions listed in ANNEX A. The first replica shall have sequence number one and following sequence numbers shall be incremented by one for each replica. Note that after deployment, no `VirtualSystem` will have the prototype `ovf:id` value itself.

If the prototype being replicated has a starting order in the `StartupSection`, all replicas shall share this value. It is not possible to specify a particular starting sequence among replicas.

Property values for `Property` elements in the prototype are prompted once per replica created. If the OVF package author requires a property value to be shared among instances, that `Property` may be declared at the containing `VirtualSystemCollection` level and referenced by replicas as described in 9.5.

Configurations from the `DeploymentOptionSection` element may be used to control values for `InstanceCount` element. The `InstanceCount` element may have an `ovf:configuration` attribute specifying the configuration in which this element should be used. Multiple elements shall not refer to the same configuration, and a configuration attribute shall not contain an `ovf:id` value that is not specified in the `DeploymentOptionSection`. See D.23 for an example.

9.15 PlacementGroupSection and PlacementSection

Guest software may require the deployment of virtual systems with specific proximity needs. There are two use cases:

1) the ability to specify that two or more virtual systems should be deployed closely together because they rely on fast communication or have a common dependency
2) the ability to specify that two or more virtual systems should be deployed on different platforms or locations because of high-availability or disaster recovery considerations

The `PlacementGroupSection` element allows an OVF package to define a placement policy for a group of VirtualSystems. The `PlacementSection` element allows the annotation of the elements with membership of a particular placement policy group.

Zero or more `PlacementGroupSections` may be defined at the `Envelope` level. The `PlacementSection` element may be declared at the `VirtualSystem` or `VirtualSystemCollection` level.

Declaring a `VirtualSystemCollection` a member of a placement policy group applies transitively to all child `VirtualSystem` and child `Virtual System Collections` elements provided that no placement policies are specified for the child `VirtualSystem` or `VirtualSystemCollection`.

If a parent `VirtualSystemCollection` and child `VirtualSystem(s)` and/or `VirtualSystemCollection(s)` both have placement policies, the placement policies of the child `VirtualSystems` and/or child `VirtualSystemCollections` should be applied first. Then placement policy of the parent `VirtualSystemCollection` should be applied.

In the event that there is a conflict in the placement policy, the availability policy should override the affinity policy

The `ovf:id` attribute in `PlacementGroupSection` is used to identify a placement policy. The value of the `ovf:id` attribute shall be unique within the OVF package.

Placement policy group membership is specified using the `ovf:group` attribute in the `PlacementSection`. The value of the `ovf:group` attribute shall match the value of an `ovf:id` attribute in a `PlacementGroupSection`. The value of the `ovf:group` attribute shall be a comma-separated text string of placement policy attributes.

This standard defines the placement policies "affinity" and "availability", specified with the required `ovf:policy` attribute on `PlacementGroupSection`.

The set of attributes used for availability and affinity are defined in Table 8 and Table 9.

Table 8 – Availability attributes

Attribute	Description
availability	The virtual systems should be placed on different virtualization platforms.
availability-geographic	The virtual systems should be placed in different geographical areas.
availability-site	The virtual systems should be placed on different operator sites.
availability-rack	The virtual systems should be placed on different physical racks.
availability-chassis	The virtual systems should be placed on different physical chassis.
availability-host	The virtual systems should be placed on different physical hosts.

Table 9 – Affinity Attributes

Attribute	Description
affinity	The virtual systems should be placed on the same virtualization platform.
affinity-geographic	The virtual systems should be placed in the same geographical area.
affinity-site	The virtual systems should be placed on the same operator site.
affinity-rack	The virtual systems should be placed on the same physical rack.
affinity-chassis	The virtual systems should be placed on the same physical chassis.
affinity-host	The virtual systems should be placed in close proximity, i.e., on the same physical host or on hosts that have low latency and high bandwidth network connectivity

The placement policies that can be declared within a `PlacementGroupSection` are combinations of the availability and affinity attributes defined in Table 8 and Table 9. The placement policy is a single string represented by concatenating the valid placement policy combinations using commas as separators. Allowed combinations of affinity and availability attributes is defined in Table 10.

Table 10 – Allowed combinations of scoped affinity and availability

Valid Combinations	availability order	affinity	affinity-geographic	affinity-site	affinity-rack	affinity-chassis	affinity-host
availability		No	Yes	Yes	Yes	Yes	No
availability-geographic	5	Yes	No	No	No	No	No
availability-site	4	Yes	Yes	No	No	No	No
availability-rack	3	Yes	Yes	Yes	No	No	No
availability-chassis	2	Yes	Yes	Yes	Yes	No	No
availability-host	1	No	Yes	Yes	Yes	Yes	No

The availability of the parent shall be higher availability order than the availability of the child.

If the placement policy is 'availability' without scoping, no availability order is specified.

See D.24 for an example.

9.16 EncryptionSection

It is desirable for licensing and other reasons to have an encryption scheme enabling free exchange of OVF appliances while ensuring that only the intended parties can use them. The XML Encryption Syntax and Processing standard is utilized to encrypt either the files in the reference section or any parts of the XML markup of an OVF document.

The various aspects of OVF encryption are as shown below:

1) Block Encryption

   The OVF package shall utilize block encryption algorithms as specified in *XML Encryption Syntax and Processing, Version 1.1* for this purpose.

2) Key Derivation

The OVF package may use the appropriate key for this purpose. If the key is derived using a passphrase, the author shall use one of the key derivations specified in *XML Encryption Syntax and Processing, Version 1.1*.

3) Key Transport

If the encryption key is embedded in the OVF package, the specified key transport mechanisms shall be used.

This standard defines a section called the `EncryptionSection` as a focal point for the encryption functionality. This section provides a single location for placing the encryption-algorithm-related markup and the corresponding reference list to point to the OVF content that has been encrypted.

Note that depending on the parts of the OVF package that has been encrypted, an OVF descriptor may not validate against the DSP8023 until decrypted. See D.25 for an example.

10 Internationalization

The following elements support localizable messages using the optional `ovf:msgid` attribute:

- `Info` element on `Content`
- `Name` element on `Content`
- `Info` element on `Section`
- `Annotation` element on `AnnotationSection`
- `License` element on `EulaSection`
- `Description` element on `NetworkSection`
- `Description` element on `OperatingSystemSection`
- `Description`, `Product`, `Vendor`, `Label`, and `Category` elements on `ProductSection`
- `Description` and `Label` elements on `Property`
- `Description` and `Label` elements on `DeploymentOptionSection`
- `ElementName`, `Caption` and `Description` subelements on the `System` element in `VirtualHardwareSection`
- `ElementName`, `Caption` and `Description` subelements on `Item` elements in `VirtualHardwareSection`
- `ElementName`, `Caption` and `Description` subelements on `Item` elements in `ResourceAllocationSection`

The `ovf:msgid` attribute contains an identifier that refers to a message that may have different values in different locales. See D.26 for an example.

The `xml:lang` attribute on the `Envelope` element shall specify the default locale for messages in the descriptor. The attribute is optional with a default value of `"en-US"`.

10.1 Internal resource bundles

Message resource bundles can be internal or external to the OVF descriptor. Internal resource bundles are represented as `Strings` elements at the end of the `Envelope` element. See D.26 for an example.

10.2 External resource bundles

External resource bundles shall be listed first in the `References` section and referred to from `Strings` elements. An external message bundle follows the same schema as the embedded one. Exactly one

`Strings` element shall be present in an external message bundle, and that `Strings` element shall not have an `ovf:fileRef` attribute specified. See D.26 for an example.

The embedded and external `Strings` elements may be interleaved, but they shall be placed at the end of the `Envelope` element. If multiple occurrences of a `msg:id` attribute with a given locale occur, a latter value overwrites a former.

10.3 Message content in external file

The content of all localizable messages may be stored in an external file using the optional `ovf:fileRef` attribute on the `Msg` element. For the `License` element on `EulaSection` in particular, this allows inclusion of a standard license text file in unaltered form without any XML header or footer.

The `ovf:fileRef` attribute denotes the message content by identifying an existing `File` element in the `References` element; the `File` element is identified by matching its `ovf:id` attribute value with the `ovf:fileRef` attribute value. The content of an external file referenced using `ovf:fileRef` shall be interpreted as plain text in UTF-8 Unicode.

If the referenced file is not available, the embedded content of the `Msg` element shall be used.

The optional `ovf:fileRef` attribute may appear on `Msg` elements in both internal and external `Strings` resource bundles. See D.27 for an example.

11 OVF environment and OVF environment file

The OVF environment defines how the guest software and the virtualization platform interact. The OVF environment enables the guest software to access information about the virtualization platform, such as the user-specified values for the properties defined in the OVF descriptor.

DSP8027 is the XML Schema definition file that contains the elements and attributes defining the format and semantics of an XML document that constitutes the OVF environment file (OEF). The OEF shall validate against DSP8027.

The OEF is created on a per virtual system basis by the deployment function. The basis of the OEF is the OVF descriptor, OVF operational metadata, OVF property values, policy metadata, and other user-provided values.

The OEF provides the guest software information about the environment that it is being executed in. The way that the OEF is conveyed depends on the transport media type. See D.28 for an example.

The value of the `ovfenv:id` attribute of the `Environment` element shall match the value of the `ovf:id` attribute of the `VirtualSystem` entity describing this virtual system.

The `PlatformSection` element contains optional information provided by the deployment function. The `Kind`, `Version`, and `Vendor` elements describe the virtualization platform. The `Locale` and `TimeZone` elements describe the current locale and time zone.

The `PropertySection` element contains `Property` elements with key/value pairs corresponding to all properties specified in the OVF descriptor for the current virtual system, as well as properties specified for the immediate parent `VirtualSystemCollection`, if one exists. The environment presents properties as a single list to make it easy for applications to parse. Furthermore, the single list format supports the override semantics that enables a property on a `VirtualSystem` to override a property defined on a parent `VirtualSystemCollection`. The property that is overridden shall not be in the list. If a property in a virtual system and a property in the parent `VirtualSystemCollection` have identical `ovf:key`, `ovf:class`, and `ovf:instance` attribute values the value of the parent property is overridden by the value of the child property; see 9.5. The value of the parent property may be obtained by adding a child property with a different name referencing the parent property with a macro; see 9.5.

An `Entity` element shall exist for each sibling `VirtualSystem` and `VirtualSystemCollection`, if any are present. The value of the `ovfenv:id` attribute of the `Entity` element shall match the value of

the `ovf:id` attribute of the sibling entity. The `Entity` elements contain the property key/value pairs in the sibling's OVF environment documents, so the content of an `Entity` element for a particular sibling shall contain the exact `PropertySection` seen by that sibling. This information can be used, for example, to make configuration information, such as IP addresses, available to `VirtualSystems` that are a part of a multitiered application.

Table 11 shows the core sections that are defined.

Table 11 – Core sections for OEF

Section	Location	Multiplicity
`PlatformSection` Provides information from the deployment platform	Environment	Zero or one
`PropertySection` Contains key/value pairs corresponding to properties defined in the OVF descriptor	Environment Entity	Zero or one

The OEF is extensible by providing new section types. The deployment function should ignore unknown section types and elements specified in OEF.

11.1 Transport media

The transport media refers to the format used to convey the information to the guest software. The transport media (e.g., ISO image) is generated by the deployment function.

If the transport media type is 'iso', the generated ISO image shall comply with the ISO 9660 specification with support for Joliet extensions.

The transport media shall contain the OVF environment file and any additional environment file(s) for this particular virtual system. The OEF shall be presented as an XML file named `ovf-env.xml` that is contained in the root directory of the transport media. The guest software is now able to access the information.

For additional environment files, the transport media shall have the root location relative to the `ovf:path` attribute in a directory named "ovffiles" contained in the root directory. This provides an access mechanism for the guest software.

Other custom transport media may support this mechanism. Custom transport medium shall specify how to access multiple data sources from a root location. See D.20 for an example. The access mechanism for the guest software is not specified.

11.2 Transport media type

The transport media type refers to a mechanism to convey transport media over a data link or removable storage medium (e.g., CD/DVD-ROM) from deployment functions to guest software.

The `iso` transport media type shall support this mechanism.

This standard defines the "iso" transport type to meet the need for interoperability.

The transport media can be communicated in a number of ways to the guest software. These ways are called transport media types. The transport media types are specified in the OVF descriptor by the `ovf:transport` attribute of `VirtualHardwareSection`. Several transport media types may be specified, separated by a single space character, in which case an implementation is free to use any of them.

To enable interoperability, this specification defines an `iso` transport media type, which all implementations that support CD-ROM devices are required to support. The `iso` transport media type communicates the environment document by making a dynamically generated ISO image available to the guest software.

To support the `iso` transport media type, prior to booting a virtual system, an implementation shall make an ISO read-only disk image available as backing for a disconnected CD-ROM. If the `iso` transport media type is selected for a `VirtualHardwareSection`, at least one disconnected CD-ROM device shall be present in this section.

If the virtual system prior to booting had more than one disconnected CD-ROM, the guest software may have to scan connected CD-ROM devices in order to locate the ISO image containing the `ovf-env.xml` file.

The transport media containing the OVF environment file shall be made available to the guest software on every boot of the virtual system.

Support for the `iso` transport media type is not a requirement for virtual hardware architectures or guest software that do not have CD-ROM device support.

To be conformant with this specification, any transport media type other than `iso` shall be given by a URI that identifies an unencumbered specification on how to use the transport media type. The specification need not be machine readable, but it shall be static and unique so that it may be used as a key by software reading an OVF descriptor to uniquely determine the transport media type. The specification shall be sufficient for a skilled person to properly interpret the transport media type mechanism for implementing the protocols. The URIs should be resolvable.

ANNEX A
(informative)

Symbols and conventions

XML examples use the XML namespace prefixes that are defined in Table 1. The XML examples use a style to not specify namespace prefixes on child elements. Note that XML rules define that child elements specified without a namespace prefix are from the namespace of the parent element, and not from the default namespace of the XML document. Throughout the document, whitespace within XML element values is used for readability. In practice, a service can accept and strip leading and trailing whitespace within element values as if whitespace had not been used.

Syntax definitions in this document use Augmented BNF (ABNF) as defined in IETF RFC5234 with the following exceptions:

- Rules separated by a bar (|) represent choices, instead of using a forward slash (/) as defined in ABNF.
- Any characters must be processed case sensitively, instead of case-insensitively as defined in ABNF.
- Whitespace (i.e., the space character U+0020 and the tab character U+0009) is allowed between syntactical elements, instead of assembling elements without whitespace as defined in ABNF.

ANNEX B
(normative)

OVF XSD

Normative copies of the XML Schemas for this specification may be retrieved by resolving the following URLs:

```
http://schemas.dmtf.org/ovf/envelope/2/dsp8023.xsd
http://schemas.dmtf.org/ovf/environment/1/dsp8027.xsd
```

Any `xs:documentation` content in XML Schemas for this specification is informative and provided only for convenience.

Normative copies of the XML Schemas for the WS-CIM mapping (DSP0230) of `CIM_ResourceAllocationSystemSettingsData`, `CIM_VirtualSystemSettingData`, `CIM_EthernetPortAllocationSettingData`, `CIM_StorageAllocationSettingData` and `CIM_OperatingSystem`, may be retrieved by resolving the following URLs:

```
http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData.xsd
http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_ResourceAllocationSettingData.xsd
http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_EthernetPortAllocationSettingData.xsd
http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_StorageAllocationSettingData.xsd
```

This specification is based on the following CIM MOFs:

```
CIM_VirtualSystemSettingData.mof
CIM_ResourceAllocationSettingData.mof
CIM_EthernetPortAllocationSettingData.mof
CIM_StorageAllocationSettingData.mof
CIM_OperatingSystem.mof
```

ANNEX C
(informative)

OVF mime type registration template

Registration Template

To: ietf-types@iana.org

Subject: Registration of media type Application/OVF

Type name: Application

Subtype name: OVF

Required parameters: none

Optional parameters: none

Encoding considerations: binary

Security considerations:

- An OVF package contains active content that is expected to be launched in a virtual system. The OVF standard, section 5.1 states: “An OVF package may be signed by signing the manifest file. The digest of the manifest file is stored in a certificate file with extension .cert file along with the base64-encoded X.509 certificate. The .cert file shall have the same base name as the .ovf file and be a sibling of the .ovf file. A consumer of the OVF package shall verify the signature and should validate the certificate.”
- An OVF package may contain passwords as part of the configuration information. The OVF standard, section 9.5 states: “The optional Boolean attribute `ovf:password` indicates that the property value may contain sensitive information. The default value is FALSE. OVF implementations prompting for property values are advised to obscure these values when `ovf:password` is set to TRUE. This is similar to HTML text input of type password. Note that this mechanism affords limited security protection only. Although sensitive values are masked from casual observers, default values in the OVF descriptor and assigned values in the OVF environment are still passed in clear text.”

Interoperability considerations:

- OVF has demonstrated interoperability via multiple, interoperating implementations in the market.

Published specification:

- DSP0243_2.0.0.pdf

Applications that use this media type:

- Implementations of the DMTF Standard: Cloud Infrastructure Management Interface (CIMI) (DSP0263_1.0.0.pdf)
- Implementations of the SNIA Cloud Data Management Interface (CDMI) – OVF Extension

Additional information:

- Magic number(s): none
- File extension(s): .ova
- Macintosh file type code(s): none
- Person & email address to contact for further information:
- Intended usage: (One of COMMON, LIMITED USE or OBSOLETE.)
- Restrictions on usage: (Any restrictions on where the media type can be used go here.)

- Author:
- Change controller:

ANNEX D
(informative)

OVF examples

D.1 Examples of OVF package structure

EXAMPLE 1:
The following list of files is an example of an OVF package:

```
  package.ovf
  package.mf
  de-DE-resources.xml
  vmdisk1.vmdk
  vmdisk2.vhd
  resource.iso
```

EXAMPLE 2:
The following example show the partial contents of a manifest file:

```
 SHA256(package.ovf)=
9902cc5ec4f4a00cabbff7b60d039263587ab430d5fbdbc5cd5e8707391c90a4
 SHA256(vmdisk.vmdk)=
aab66c4d70e17cec2236a651a3fc618cafc5ec6424122904dc0b9c286fce40c2
```

EXAMPLE 3:
The following list of files is an example of a signed OVF package:

```
  package.ovf
  package.mf
  package.cert
  de-DE-resources.xml
  vmdisk1.vmdk
  vmdisk2.vmdk
  resource.iso
```

EXAMPLE 4:
The following example shows the contents of a sample OVF certification file, where the SHA1 digest of the manifest file has been signed with a 512 bit key:

```
SHA1(package.mf)= 7f4b8efb8fe20c06df1db68281a63f1b088e19dbf00e5af9db5e8e3e319de
7019db88a3bc699bab6ccd9e09171e21e88ee20b5255cec3fc28350613b2c529089
-----BEGIN CERTIFICATE-----
MIIBgjCCASwCAQQwDQYJKoZIhvcNAQEEBQAwODELMAkGA1UEBhMCQVUxDDAKBgNV
BAgTA1FMRDEbMBkGA1UEAxMSU1NMZWF5L3JzYSB0ZXN0IENBMB4XDTk1MTAwOTIz
MzIwNVoXDTk4MDcwNTIzMzIwNVowYDELMAkGA1UEBhMCQVUxDDAKBgNVBAgTA1FM
RDEZMBcGA1UEChMQTWluY29tIFB0eS4gTHRkLjELMAkGA1UECxMCQ1MxGzAZBgNV
BAMTElNTTGVheSBkZW1vIHNlcnZlcjBcMA0GCSqGSIb3DQEBAQUAA0sAMEgCQQC3
LCXcScWua0PFLkHBLm2VejqpA1F4RQ8q0VjRiPafjx/Z/aWH3ipdMVvuJGa/wFXb
/nDFLDlfWp+oCPwhBtVPAgMBAAEwDQYJKoZIhvcNAQEEBQADQQArNFsihWIjBzb0
DcsfOBvL2bvSwJrPEqFlkDq3F4M6EgutL9axEcANWgbbEdAvNJD1dmEmoWny27Pn
Ims6ZOZB
-----END CERTIFICATE-----
```

D.2 Examples of distribution of files

EXAMPLE 1:
An example of an OVF package as a compressed archive:

```
    D:\virtualappliances\myapp.ova
```

EXAMPLE 2:
An example of an OVF package as a set of files on Web server follows:

```
    http://mywebsite/virtualappliances/package.ovf
```

```
    http://mywebsite/virtualappliances/vmdisk1.vmdk
    http://mywebsite/virtualappliances/vmdisk2.vmdk
    http://mywebsite/virtualappliances/resource.iso
    http://mywebsite/virtualappliances/de-DE-resources.xml
```

D.3 Example of envelope element

An example of the structure of an OVF descriptor with the top-level Envelope element follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:vssd="http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_VirtualSystemSettingData"
    xmlns:rasd="http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_ResourceAllocationSettingData"
    xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/2"
    xmlns="http://schemas.dmtf.org/ovf/envelope/2"
    xml:lang="en-US">
    <References>
      <File ovf:id="de-DE-resources.xml" ovf:size="15240"
            ovf:href="http://mywebsite/virtualappliances/de-DE-resources.xml"/>
      <File ovf:id="file1" ovf:href="vmdisk1.vmdk" ovf:size="180114671"/>
      <File ovf:id="file2" ovf:href="vmdisk2.vmdk" ovf:size="4882023564"
ovf:chunkSize="2147483648"/>
      <File ovf:id="file3" ovf:href="resource.iso" ovf:size="212148764"
ovf:compression="gzip"/>
      <File ovf:id="icon" ovf:href="icon.png" ovf:size="1360"/>
    </References>
    <!-- Describes meta-information about all virtual disks in the package -->
    <DiskSection>
        <Info>Describes the set of virtual disks</Info>
        <!-- Additional section content -->
    </DiskSection>
    <!-- Describes all networks used in the package -->
    <NetworkSection>
         <Info>List of logical networks used in the package</Info>
        <!-- Additional section content -->
    </NetworkSection>
    <SomeSection ovf:required="false">
        <Info>A plain-text description of the content</Info>
        <!-- Additional section content -->
    </SomeSection>
    <!-- Additional sections can follow -->
    <VirtualSystemCollection ovf:id="Some Product">
        <!-- Additional sections including VirtualSystem or
VirtualSystemCollection-->
    </VirtualSystemCollection >
    <Strings xml:lang="de-DE">
      <!-- Specification of message resource bundles for de-DE locale -->
    </Strings>
</Envelope>
```

D.4 Example of file references

EXAMPLE 1:

The following example shows different types of file references:

```
  <File ovf:id="disk1" ovf:href="disk1.vmdk"/>
  <File ovf:id="disk2" ovf:href="disk2.vmdk" ovf:size="5368709120"
                                             ovf:chunkSize="2147483648"/>
  <File ovf:id="iso1" ovf:href="resources/image1.iso"/>
  <File ovf:id="iso2" ovf:href="http://mywebsite/resources/image2.iso"/>
```

```
EXAMPLE 2:
The following example shows manifest entries corresponding to the file references
above:
  SHA1(disk1.vmdk)= 3e19644ec2e806f38951789c76f43e4a0ec7e233
  SHA1(disk2.vmdk.000000000)= 4f7158731ff434380bf217da248d47a2478e79d8
  SHA1(disk2.vmdk.000000001)= 12849daeeaf43e7a89550384d26bd437bb8defaf
  SHA1(disk2.vmdk.000000002)= 4cdd21424bd9eeafa4c42112876217de2ee5556d
  SHA1(resources/image1.iso)= 72b37ff3fdd09f2a93f1b8395654649b6d06b5b3
  SHA1(http://mywebsite/resources/image2.iso)=
d3c2d179011c970615c5cf10b30957d1c4c968ad
```

D.5 Example of content element

```
An example of a VirtualSystem element structure follows:
   <VirtualSystem ovf:id="simple-app">
       <Info>A virtual system</Info>
       <Name>Simple Appliance</Name>
       <SomeSection>
           <!-- Additional section content -->
       </SomeSection>
       <!-- Additional sections can follow -->
    </VirtualSystem>

An example of a VirtualSystemCollection element structure follows:
   <VirtualSystemCollection ovf:id="multi-tier-app">
       <Info>A collection of virtual systems</Info>
       <Name>Multi-tiered Appliance</Name>
       <SomeSection>
           <!-- Additional section content -->
       </SomeSection>
       <!-- Additional sections can follow -->
       <VirtualSystem ovf:id="...">
           <!-- Additional sections -->
       </VirtualSystem>
       <!-- Additional VirtualSystem or VirtualSystemCollection elements can
follow-->
   </VirtualSystemCollection>
```

D.6 Examples of extensibility

```
EXAMPLE 1:
    <!-- Optional custom section example -->
    <otherns:IncidentTrackingSection ovf:required="false">
        <Info>Specifies information useful for incident tracking purposes</Info>
        <BuildSystem>Acme Corporation Official Build System</BuildSystem>
        <BuildNumber>102876</BuildNumber>
        <BuildDate>10-10-2008</BuildDate>
    </otherns:IncidentTrackingSection>

EXAMPLE 2:
    <!-- Open content example (extension of existing type) -->
    <AnnotationSection>
        <Info>Specifies an annotation for this virtual system</Info>
        <Annotation>This is an example of how a future element (Author) can still
be
           parsed by older clients</Annotation>
        <!-- AnnotationSection extended with Author element -->
        <otherns:Author ovf:required="false">John Smith</otherns:Author>
    </AnnotationSection>
```

```
EXAMPLE 3:
    <!-- Optional custom attribute example -->
    <Network ovf:name="VS network" otherns:desiredCapacity="1 Gbit/s">
        <Description>The main network for VSs</Description>
    </Network>
```

D.7 Examples of VirtualHardwareSection

```
EXAMPLE 1:
Example of VirtualHardwareSection:
<VirtualHardwareSection>
    <Info>Memory = 4 GB, CPU = 1 GHz, Disk = 100 GB, 1 Ethernet nic</Info>
    <Item>
        <rasd:AllocationUnits>Hertz*10^9</rasd:AllocationUnits>
        <rasd:Description>Virtual CPU</rasd:Description>
        <rasd:ElementName>1 GHz virtual CPU</rasd:ElementName>
        <rasd:InstanceID>1</rasd:InstanceID>
        <rasd:Reservation>1</rasd:Reservation>
        <rasd:ResourceType>3</rasd:ResourceType>
        <rasd:VirtualQuantity>1</rasd:VirtualQuantity>
        <rasd:VirtualQuantityUnit>Count</ rasd:VirtualQuantityUnit>
    </Item>
    <Item>
        <rasd:AllocationUnits>byte*2^30</rasd:AllocationUnits>
        <rasd:Description>Memory</rasd:Description>
        <rasd:ElementName>1 GByte of memory</rasd:ElementName>
        <rasd:InstanceID>2</rasd:InstanceID>
        <rasd:Limit>4</rasd:Limit>
        <rasd:Reservation>4</rasd:Reservation>
        <rasd:ResourceType>4</rasd:ResourceType>
    </Item>
    <EthernetPortItem>
        <rasd:AllocationUnits>bit / second *2^30 </rasd:AllocationUnits>
        <epasd:Connection>VS Network</epasd:Connection>
        <epasd:Description>Virtual NIC</epasd:Description>
        <epasd:ElementName>Ethernet Port</epasd:ElementName>
        <epasd:NetworkPortProfileID>1</epasd:NetworkPortProfileID>
        <epasd:NetworkPortProfileIDType>4</epasd:NetworkPortProfileIDType>
        <epasd:ResourceType>10</epasd:ResourceType>
        <epasd:VirtualQuantity>1</epasd:VirtualQuantity>
        <epasd:VirtualQuantityUnits>Count</epasd:VirtualQuantityUnits>
    </EthernetPortItem>
    <StorageItem>
        <sasd:AllocationUnits>byte*2^30</sasd:AllocationUnits>
        <sasd:Description>Virtual Disk</sasd:Description>
        <sasd:ElementName>100 GByte Virtual Disk</sasd:ElementName>
        <sasd:Reservation>100</sasd:Reservation>
        <sasd:ResourceType>31</sasd:ResourceType>
        <sasd:VirtualQuantity>1</sasd:VirtualQuantity>
        <sasd:VirtualQuantityUnit>Count</sasd:VirtualQuantityUnit>
    </StorageItem>
</VirtualHardwareSection>

EXAMPLE 2:
        <rasd:ResourceSubType>buslogic lsilogic</rasd:ResourceSubType>
```

D.8 Examples of virtual hardware elements

```
EXAMPLE 1:
The following example shows a description of memory size:
   <Item>
```



```
        <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
        <rasd:Description>Memory Size</rasd:Description>
        <rasd:ElementName>256 MB of memory</rasd:ElementName>
        <rasd:InstanceID>2</rasd:InstanceID>
        <rasd:ResourceType>4</rasd:ResourceType>
        <rasd:VirtualQuantity>256</rasd:VirtualQuantity>
    </Item>
```

EXAMPLE 2:
The following example shows a description of a virtual Ethernet adapter:

```
    <EthernetPortItem>
              <epasd:Address>00-16-8B-DB-00-5E</epasd:Address>
              <epasd:Connection>VS Network</epasd:Connection>
              <epasd:Description>Virtual NIC</epasd:Description>

              <epasd:ElementName>Ethernet Port 1</epasd:ElementName>
              <epasd:InstanceID>3</epasd:InstanceID>
              <epasd:NetworkPortProfileID>1</epasd:NetworkPortProfileID>
              <epasd:NetworkPortProfileIDType>4</epasd:NetworkPortProfileIDType>
              <epasd:VirtualQuantityUnits>1</epasd:VirtualQuantityUnits>
        </EthernetPortItem>
```

EXAMPLE 3:
The following example shows a description of a virtual storage:

```
        <StorageItem>
              <sasd:AllocationUnits>byte*2^30</sasd:AllocationUnits>
              <sasd:Description>Virtual Disk</sasd:Description>
              <sasd:ElementName>100 GByte Virtual Disk</sasd:ElementName>
              <sasd:InstanceID>4</sasd:InstanceID>
              <sasd:Reservation>100</sasd:Reservation>

              <sasd:ResourceType>31</sasd:ResourceType>
              <sasd:VirtualQuantity>1</sasd:VirtualQuantity>
        </StorageItem>
```

D.9 Example of ranges on elements

EXAMPLE:
The following example shows the use of range markers:

```
<VirtualHardwareSection>
  <Info>...</Info>
  <Item>
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>512 MB memory size</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:ResourceType>4</rasd:ResourceType>
      <rasd:VirtualQuantity>512</rasd:VirtualQuantity>
   </Item>
   <Item ovf:bound="min">
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>384 MB minimum memory size</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:Reservation>384</rasd:Reservation>
      <rasd:ResourceType>4</rasd:ResourceType>
   </Item>
   <Item ovf:bound="max">
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>1024 MB maximum memory size</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:Reservation>1024</rasd:Reservation>
```

```
      <rasd:ResourceType>4</rasd:ResourceType>
   </Item>
</VirtualHardwareSection>
```

D.10 Example of DiskSection

EXAMPLE: The following example shows a description of virtual disks:

```
<DiskSection>
    <Info>Describes the set of virtual disks</Info>
    <Disk ovf:diskId="vmdisk1" ovf:fileRef="file1" ovf:capacity="8589934592"
          ovf:populatedSize="3549324972"
          ovf:format=
              "http://www.vmware.com/interfaces/specifications/vmdk.html#sparse">
    </Disk>
    <Disk ovf:diskId="vmdisk2" ovf:capacity="536870912"
    </Disk>
    <Disk ovf:diskId="vmdisk3" ovf:capacity="${disk.size}"
          ovf:capacityAllocationUnits="byte * 2^30"
    </Disk>
</DiskSection>
```

D.11 Example of NetworkSection

```
<NetworkSection>
    <Info>List of logical networks used in the package</Info>
    <Network ovf:name="VS Network">
        <Description>The network that the service will be available
on</Description>
        <NetworkPortProfile>
            <Item>
                <epasd:AllocationUnits>GigaBits per Second</epasd:AllocationUnits>
                <epasd:ElementName>Network Port Profile 1</epasd:ElementName>
                <epasd:InstanceID>1</epasd:InstanceID>
                <epasd:NetworkPortProfileID>1</epasd:NetworkPortProfileID>
                <epasd:NetworkPortProfileIDType>4</epasd:NetworkPortProfileIDType>
                <epasd:Reservation>1</epasd:Reservation>
            </Item>
        </NetworkPortProfile>
    </Network>
</NetworkSection>
```

D.12 Example of ResourceAllocationSection

```
<ResourceAllocationSection>
    <Info>Defines reservations for CPU and memory for the collection of VSs</Info>
    <Item>
        <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
        <rasd:ElementName>300 MB reservation</rasd:ElementName>
        <rasd:InstanceID>0</rasd:InstanceID>
        <rasd:Reservation>300</rasd:Reservation>
        <rasd:ResourceType>4</rasd:ResourceType>
    </Item>
    <Item ovf:configuration="..." ovf:bound="...">
        <rasd:AllocationUnits>hertz * 10^6</rasd:AllocationUnits>
        <rasd:ElementName>500 MHz reservation</rasd:ElementName>
        <rasd:InstanceID>0</rasd:InstanceID>
        <rasd:Reservation>500</rasd:Reservation>
        <rasd:ResourceType>3</rasd:ResourceType>
    </Item>
    <EthernetPortItem>
        <epasd:Address>00-16-8B-DB-00-5E</epasd:Address>
        <epasd:Connection>VS Network</epasd:Connection>
```

```
        <epasd:Description>Virtual NIC</epasd:Description>
        <epasd:ElementName>Ethernet Port 1</epasd:ElementName>
        <epasd:InstanceID>3</epasd:InstanceID>
        <epasd:NetworkPortProfileID>1</epasd:NetworkPortProfileID>
        <epasd:NetworkPortProfileIDType>4</epasd:NetworkPortProfileIDType>
        <epasd:VirtualQuantityUnits>1</epasd:VirtualQuantityUnits>
    </EthernetPortItem>
    <StorageItem>
        <sasd:AllocationUnits>byte*2^30</sasd:AllocationUnits>
        <sasd:Description>Virtual Disk</sasd:Description>
        <sasd:ElementName>100 GByte Virtual Disk</sasd:ElementName>
        <sasd:InstanceID>4</sasd:InstanceID>
        <sasd:Reservation>100</sasd:Reservation>
        <sasd:ResourceType>31</sasd:ResourceType>
        <sasd:VirtualQuantity>1</sasd:VirtualQuantity>
    </StorageItem>
</ResourceAllocationSection>
```

D.13 Example of annotation

```
<AnnotationSection>
    <Info>An annotation on this service. It can be ignored</Info>
    <Annotation>Contact customer support if you have any problems</Annotation>
</AnnotationSection >
```

D.14 Example of Product section

```
<ProductSection ovf:class="com.mycrm.myservice" ovf:instance="1">
    <Info>Describes product information for the service</Info>
    <Product>MyCRM Enterprise</Product>
    <Vendor>MyCRM Corporation</Vendor>
    <Version>4.5</Version>
    <FullVersion>4.5-b4523</FullVersion>
    <ProductUrl>http://www.mycrm.com/enterprise</ProductUrl>
    <VendorUrl>http://www.mycrm.com</VendorUrl>
    <Icon ovf:height="32" ovf:width="32" ovf:mimeType="image/png"
ovf:fileRef="icon">
    <Category>Email properties</Category>
    <Property ovf:key="adminemail" ovf:type="string" ovf:userConfigurable="true">
        <Label>Admin email</Label>
        <Description>Email address of administrator</Description>
    </Property>
    <Category>Admin properties</Category>
    <Property ovf:key="app_log" ovf:type="string" ovf:value="low"
ovf:userConfigurable="true">
        <Description>Loglevel for the service</Description>
    </Property>
    <Property ovf:key="app_isSecondary" ovf:value="false" ovf:type="boolean">
        <Description>Cluster setup for application server</Description>
    </Property>
    <Property ovf:key="app_ip" ovf:type="string" ovf:value="${appserver-vm}">
        <Description>IP address of the application server VS</Description>
    </Property>
</ProductSection>
```

D.15 Example of EULA section

```
<EulaSection>
    <Info>Licensing agreement</Info>
    <License>
Lorem ipsum dolor sit amet, ligula suspendisse nulla pretium, rhoncus tempor
placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est,
```

```
vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliquam
maecenas ligula nostra, accumsan taciti. Sociis mauris in integer, a dolor netus
non dui aliquet, sagittis felis sodales, dolor sociis mauris, vel eu libero cras.
Interdum at. Eget habitasse elementum est, ipsum purus pede porttitor class, ut
adipiscing, aliquet sed auctor, imperdiet arcu per diam dapibus libero duis. Enim
eros in vel, volutpat nec pellentesque leo, scelerisque.
    </License>
</EulaSection>
```

D.16 Example of StartupSection

```
<StartupSection>
    <Item ovf:id="vm1" ovf:order="0" ovf:startDelay="30" ovf:stopDelay="0"
          ovf:startAction="powerOn" ovf:waitingForGuest="true"
ovf:stopAction="powerOff"/>
    <Item ovf:id="teamA" ovf:order="0"/>
    <Item ovf:id="vm2" ovf:order="1" ovf:startDelay="0" ovf:stopDelay="20"
          ovf:startAction="powerOn" ovf:stopAction="guestShutdown"/>
</StartupSection>
```

D.17 Example of DeploymentOptionSection

```
<DeploymentOptionSection>
    <Configuration ovf:id="minimal">
        <Label>Minimal</Label>
        <Description>Some description</Description>
    </Configuration>
    <Configuration ovf:id="normal" ovf:default="true">
        <Label>Typical</Label>
        <Description>Some description</Description>
    </Configuration>
    <!-- Additional configurations -->
</DeploymentOptionSection>

EXAMPLE 1:  The following example shows a VirtualHardwareSection:
<VirtualHardwareSection>
  <Info>...</Info>
  <Item>
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>512 MB memory size and 256 MB
reservation</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:Reservation>256</rasd:Reservation>
      <rasd:ResourceType>4</rasd:ResourceType>
      <rasd:VirtualQuantity>512</rasd:VirtualQuantity>
   </Item>
   ...
   <Item ovf:configuration="big">
      <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
      <rasd:ElementName>1024 MB memory size and 512 MB
reservation</rasd:ElementName>
      <rasd:InstanceID>0</rasd:InstanceID>
      <rasd:Reservation>512</rasd:Reservation>
      <rasd:ResourceType>4</rasd:ResourceType>
      <rasd:VirtualQuantity>1024</rasd:VirtualQuantity>
   </Item>
</VirtualHardwareSection>

EXAMPLE 2:  The following shows an example ProductSection:
<ProductSection>
   <Property ovf:key="app_adminEmail" ovf:type="string" ovf:userConfigurable="true"
```



```
            ovf:configuration="standard">
        <Label>Admin email</Label>
        <Description>Email address of service administrator</Description>
    </Property>
    <Property ovf:key="app_log" ovf:type="string" ovf:value="low"
            ovf:userConfigurable="true">
        <Label>Loglevel</Label>
        <Description>Loglevel for the service</Description>
        <Value ovf:value="none" ovf:configuration="minimal">
    </Property>
</ProductSection>
```

In the example above, the app_adminEmail property is only user configurable in the standard configuration, while the default value for the app_log property is changed from low to none in the minimal configuration.

D.18 Example of OperatingSystemSection

```
<OperatingSystemSection ovf:id="76">
    <Info>Specifies the operating system installed</Info>
    <Description>Microsoft Windows Server 2008</Description>
</OperatingSystemSection>
```

D.19 Example of InstallSection

```
<InstallSection ovf:initialBootStopDelay="300">
    <Info>Specifies that the virtual system needs to be booted once after having
created the guest software in order to install and/or configure the software
    </Info>
</InstallSection>
```

D.20 Example of EnvironmentFilesSection

EXAMPLE:

```
<Envelope>
    <References>
    ...
    <File ovf:id="config" ovf:href="config.xml" ovf:size="4332"/>
    <File ovf:id="resources" ovf:href="http://mywebsite/resources/resources.zip"/>
    </References>
    ...
    <VirtualSystem ovf:id="...">
    ...
    <ovf:EnvironmentFilesSection ovf:required="false" ovf:transport="iso">
        <Info>Config files to be included in OVF environment</Info>
        <ovf:File ovf:fileRef="config" ovf:path="setup/cfg.xml"/>
        <ovf:File ovf:fileRef="resources" ovf:path="setup/resources.zip"/>
    </ovf:EnvironmentFilesSection>
    ...
    </VirtualSystem>
    ...
</Envelope>
```

In the example above, the file config.xml in the OVF package will be copied to the OVF environment ISO image and be accessible to the guest software in location /ovffiles/setup/cfg.xml, while the file resources.zip will be accessible in location /ovffiles/setup/resources.zip.

D.21 Example of BootDeviceSection

In the example below, the Pre-Install configuration specifies the boot source as a specific device (network), while the Post-Install configuration specifies a device type (hard disk).

EXAMPLE:

```
<Envelope>
...
    <VirtualSystem ovf:id="...">
...
    <ovf:BootDeviceSection>
      <Info>Boot device order specification</Info>
      <bootc:CIM_BootConfigSetting>
        <bootc:Caption>Pre-Install</bootc:Caption>
        <bootc:Description>Boot Sequence for fixup of disk</bootc:Description>
        <boots:CIM_BootSourceSetting>
          <boots:Caption>Fix-up DVD on the network</boots:Caption>
          <boots:InstanceID>3</boots:InstanceID>          <!— Network device-->
        </boots:CIM_BootSourceSetting>
        <boots:CIM_BootSourceSetting>
          <boots:Caption>Boot virtual disk</boots:Caption>
          <boots:StructuredBootString>CIM:Hard-Disk</boots:StructuredBootString>
        </boots:CIM_BootSourceSetting>
      </bootc:CIM_BootConfigSetting>
    </ovf:BootDeviceSection>
...
    </VirtualSystem>
</Envelope>
```

D.22 Example of SharedDiskSection

```
EXAMPLE:
<ovf:SharedDiskSection>
    <Info>Describes the set of virtual disks shared between VSs</Info>
    <ovf:SharedDisk ovf:diskId="datadisk" ovf:fileRef="data"
                    ovf:capacity="8589934592" ovf:populatedSize="3549324972"
          ovf:format=
              "http://www.vmware.com/interfaces/specifications/vmdk.html#sparse"/>
    <ovf:SharedDisk ovf:diskId="transientdisk" ovf:capacity="536870912"/>
</ovf:SharedDiskSection>
```

D.23 Example of ScaleOutSection

```
EXAMPLE:
<VirtualSystemCollection ovf:id="web-tier">
...
    <ovf:ScaleOutSection ovf:id="web-server">
        <Info>Web tier</Info>
        <ovf:Description>Number of web server instances in web
tier</ovf:Description>
        <ovf:InstanceCount ovf:default="4" ovf:minimum="2" ovf:maximum="8"/>
    </ovf:ScaleOutSection>
...
    <VirtualSystem ovf:id="web-server">
        <Info>Prototype web server</Info>
...
    </VirtualSystem>
</VirtualSystemCollection>
```

In the example above, the deployment platform creates a web tier containing between two and eight web server virtual system instances, with a default instance count of four. The deployment platform makes an appropriate choice (e.g., by prompting the user). Assuming three replicas were created, the OVF environment available to the guest software in the first replica has the following content structure:

```
EXAMPLE:
<Environment ... ovfenv:id=”web-server-1”>
```



```
  ...
  <Entity ovfenv:id=”web-server-2”>
    ...
  </Entity>
  <Entity ovfenv:id=”web-server-3”>
    ...
  </Entity>
</Environment>
```

```
EXAMPLE:
<VirtualSystemCollection ovf:id="web-tier">
  ...
  <DeploymentOptionSection>
    <Info>Deployment size options</Info>
    <Configuration ovf:id="minimal">
      <Label>Minimal</Label>
      <Description>Minimal deployment scenario</Description>
    </Configuration>
    <Configuration ovf:id="common" ovf:default="true">
      <Label>Typical</Label>
      <Description>Common deployment scenario</Description>
    </Configuration>
    ...
  </DeploymentOptionSection>
  ...
  <ovf:ScaleOutSection ovf:id="web-server">
    <Info>Web tier</Info>
    <ovf:Description>Number of web server instances in web tier</ovf:Description>
      <ovf:InstanceCount ovf:default="4"/>
      <ovf:InstanceCount ovf:default="1" ovf:configuration="minimal"/>
  </ovf:ScaleOutSection>
...
</VirtualSystemCollection>
```

In the example above, the default replica count is four, unless the minimal deployment scenario is chosen, in which case the default is one.

D.24 Example of PlcementGroupSection

```
EXAMPLE:
<Envelope ...>
  ...
  <ovf:PlacementGroupSection ovf:id="web" ovf:policy="availability">
    <Info>Placement policy for group of VSs</Info>
    <ovf:Description>Placement policy for web tier</ovf:Description>
  </ovf:PlacementGroupSection>
    ...
  <VirtualSystemCollection ovf:id="web-tier">
    ...
    <ovf:ScaleOutSection ovf:id="web-node">
      <Info>Web tier</Info>
      ...
    </ovf:ScaleOutSection>
    ...
    <VirtualSystem ovf:id="web-node">
      <Info>Web server</Info>
      ...
      <ovf:PlacementSection ovf:group="web">
        <Info>Placement policy group reference</Info>
      </ovf:PlacementSection>
```

```
      ...
    </VirtualSystem>
  </VirtualSystemCollection>
</Envelope>
```

In the example above, all virtual systems in the compute tier should be placed separately for high availability. This example also use the ScaleOutSection defined in clause 9.14, in which case each replica get the policy assigned.

D.25 Example of EncryptionSection

Below is an example of an OVF encryption section with encryption methods utilized in the OVF document, and the corresponding reference list pointing to the items that have been encrypted.

```
EXAMPLE:
  <ovf:EncryptionSection>
<!--- This section contains two different methods of encryption and the
corresponding back pointers to the data that is encrypted ->
  <!--- Method#1: Pass phrase based Key derivation ->
<!--- The following derived key block defines PBKDF2 and the corresponding back
pointers to the encrypted data elements -->
  <!--- Use a salt value "ovfpassword" and iteration count of 4096 --->
 <xenc11:DerivedKey>
             <xenc11:KeyDerivationMethod
Algorithm="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2"/>
<pkcs-5:PBKDF2-params>
                    <Salt>
                          <Specified>ovfpassword</Specified>
                    </Salt>
                    <IterationCount>4096</IterationCount>
                    <KeyLength>16</KeyLength>
                    <PRF Algorithm="http://www.w3.org/2001/04/xmldsig-more#hmac-
sha256"/>
             </pkcs-5:PBKDF2-params>
…
<!-- The ReferenceList element below contains references to the file Ref-109.vhd
via the URI syntax which is specified by XML Encryption.
--->
<xenc:ReferenceList>
       <xenc:DataReference URI="#first.vhd" />
<xenc:DataReference URI=… />
<xenc:DataReference URI=… />
</xenc:ReferenceList>
       </xenc11:DerivedKey>
    <!-- Method#2: The following example illustrates use of a symmetric key
transported using the public key within a certificate ->
<xenc:EncryptedKey>
             <xenc:EncryptionMethod
   Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                    <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'
                          <ds:X509Data>
                    <ds:X509Certificate> … </ds:X509Certificate>
             </ds:X509Data>
             </ds:KeyInfo>
       <xenc:CipherData>
       <xenc:CipherValue> … </xenc:CipherValue>
       </xenc:CipherData>
<!-- The ReferenceList element below contains reference #second-xml-fragment" to
the XML fragment that has been encrypted using the above method --->
       <xenc:ReferenceList>
```