Document Number: DSP0243

Date: 2008-09-04

Version: 1.0.0d

Open Virtualization Format Specification

Document Type: Specification

Document Status: Preliminary Standard

Document Language: E

CONTENTS

Tables

Foreword

The *Open Virtualization Format Specification* (DSP0243) was prepared by the DMTF System Virtualization, Partitioning, and Clustering Working Group.

This specification has been developed as a result of joint work with many individuals and teams, including:

- Simon Crosby, XenSource
- Ron Doyle, IBM
- Michael Gionfriddo, Sun Microsystems
- Steffen Grarup, VMware (Co-Editor)
- Steve Hand, Symantec
- Mark Hapner, Sun Microsystems
- Daniel Hiltgen, VMware
- Michael Johanssen, IBM
- Lawrence J. Lamers, VMware (Chair)
- Fumio Machida, NEC Corporation
- Andreas Maier, IBM
- Ewan Mellor, XenSource
- John Parchem, Microsoft
- Shishir Pardikar, XenSource
- Stephen J. Schmidt, IBM
- René W. Schmidt, VMware (Co-Editor)
- Andrew Warfield, XenSource
- Mark D. Weitzel, IBM
- John Wilson, Dell

Introduction

The *Open Virtualization Format (OVF) Specification* describes an open, secure, portable, efficient and extensible format for the packaging and distribution of software to be run in virtual machines. The key properties of the format are as follows:

- **Optimized for distribution**

  OVF supports content verification and integrity checking based on industry-standard public key infrastructure, and it provides a basic scheme for management of software licensing.

- **Optimized for a simple, automated user experience**

  OVF supports validation of the entire package and each virtual machine or metadata component of the OVF during the installation phases of the virtual machine (VM) lifecycle management process. It also packages with the package relevant user-readable descriptive information that a virtualization platform can use to streamline the installation experience.

- **Supports both single VM and multiple-VM configurations**

  OVF supports both standard single VM packages and packages containing complex, multi-tier services consisting of multiple interdependent VMs.

- **Portable VM packaging**

  OVF is virtualization platform neutral, while also enabling platform-specific enhancements to be captured. It supports the full range of virtual hard disk formats used for hypervisors today, and it is extensible, which will allow it to accommodate formats that may arise in the future. Virtual machine properties are captured concisely and accurately.

- **Vendor and platform independent**

  OVF does not rely on the use of a specific host platform, virtualization platform, or guest operating system.

- **Extensible**

  OVF is immediately useful — and extensible. It is designed to be extended as the industry moves forward with virtual appliance technology. It also supports and permits the encoding of vendor-specific metadata to support specific vertical markets.

- **Localizable**

  OVF supports user-visible descriptions in multiple locales, and it supports localization of the interactive processes during installation of an appliance. This capability allows a single packaged appliance to serve multiple market opportunities.

- **Open standard**

  OVF has arisen from the collaboration of key vendors in the industry, and it is developed in an accepted industry forum as a future standard for portable virtual machines.

It is not an explicit goal for OVF to be an efficient execution format. A hypervisor is allowed but not required to run software in virtual machines directly out of the Open Virtualization Format.

Open Virtualization Format Specification

1 Scope

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

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 Approved References

ANSI/IEEE Standard 1003.1-2001, *IEEE Standard for Information Technology- Portable Operating System Interface (POSIX)*, Institute of Electrical and Electronics Engineers, August 2001, http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=1316

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

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

DMTF CIM Schema Version 2.19 (MOF files), http://www.dmtf.org/standards/cim/cim_schema_v219

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

DMTF DSP1042, *System Virtualization Profile (SVP),* http://www.dmtf.org/standards/published_documents/DSP1042.pdf

DMTF DSP1057, *Virtual System Profile (VSP),* http://www.dmtf.org/standards/published_documents/DSP1057.pdf

DMTF DSP0230, *WS-CIM Mapping Specification,* http://www.dmtf.org/standards/published_documents/DSP0230.pdf

IETF RFC 1738, T. Berners-Lee, *Uniform Resource Locators (URL)*, December 1994, http://www.ietf.org/rfc/rfc1738.txt

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

IETF RFC 2234, *Augmented BNF (ABNF)*, http://www.ietf.org/rfc/rfc2234.txt

IETF RFC 2616, R. Fielding et al, *Hypertext Transfer Protocol – HTTP/1.1*, June 1999, http://www.ietf.org/rfc/rfc2616.txt

IETF RFC 2818, E. Rescorla, *HTTP over TLS*, May 2000, http://www.ietf.org/rfc/rfc2818.txt

IEFT RFC 3986, *Uniform Resource Identifiers (URI): Generic Syntax,* http://www.ietf.org/rfc/rfc3986.txt

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

2.2 Other References

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

W3C, Y. Savourel et al, *Best Practices for XML Internationalization*, Working Draft, October 2007, http://www.w3.org/TR/2007/WD-xml-i18n-bp-20071031

W3C, S. Gao et al, *XML Schema Definition Language (XSDL) 1.1, Part 1: Structures*, Working Draft, August 2007, http://www.w3.org/TR/xmlschema11-1

W3C, D. Peterson et al, *XML Schema Definition Language (XSDL) 1.1, Part 2: Datatypes*, Working Draft, February 2006, http://www.w3.org/TR/xmlschema11-2

3 Terms and Definitions

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

3.1
can
used for statements of possibility and capability, whether material, physical, or causal

3.2
cannot
used for statements of possibility and capability, whether material, physical, or causal

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

3.4
mandatory
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.5
may
indicates a course of action permissible within the limits of the document

3.6
need not
indicates a course of action permissible within the limits of the document

3.7
optional
indicates a course of action permissible within the limits of the document

3.8
shall
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.9
shall not
indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.10
should
indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.11
should not
indicates that a certain possibility or course of action is deprecated but not prohibited

3.12
appliance
see *virtual appliance*

3.13
deployment platform
the product that installs an OVF package

3.14
guest software
the software, stored on the virtual disks, that runs when a virtual machine is powered on
The guest is typically an operating system and some user-level applications and services.

3.15
OVF package
OVF XML descriptor file accompanied by zero or more files

3.16
platform
see *deployment platform*

3.17
virtual appliance
a service delivered as a complete software stack installed on one or more virtual machines
A virtual appliance is typically expected to be delivered in an OVF package.

3.18
virtual hardware
the hardware (including the CPU, controllers, Ethernet devices, and disks) that is seen by the guest software

3.19
virtual machine
the complete environment that supports the execution of guest software
A virtual machine is a full encapsulation of the virtual hardware, virtual disks, and the metadata associated with it. Virtual machines allow multiplexing of the underlying physical machine through a software layer called a hypervisor.

3.20
virtual machine collection
a service comprised of a set of virtual machines
The service can be a simple set of one or more virtual machines, or it can be a complex service built out of a combination of virtual machines and other virtual machine collections. Because virtual machine collections can be composed, it enables complex nested components.

4 Symbols and Abbreviated Terms

The following symbols and 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
VM
Virtual Machine

5 OVF Packages

5.1 OVF Package Structure

An OVF package shall consist of the following files:

- one OVF descriptor file (descriptor file or .ovf file)
- zero or one OVF manifest file (manifest file or .mf file)
- zero or one OVF certification file (certification file or .cert file)
- zero or more disk image files
- zero or more additional resource files, such as ISO images

The file extensions .ovf, .mf and .cert should be used.

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.vmdk
  resource.iso
```

NOTE: The previous example uses VMDK disk files, but multiple disk formats are supported.

Optionally, an OVF package may have a manifest file with extension .mf containing the SHA-1 digests of individual files in the package. The manifest file shall have the same base name as the .ovf file. If the manifest file is present, a consumer of the OVF package shall verify the digests by computing the actual SHA-1 digests and comparing them with the digests listed in the manifest file.

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

The format of the .mf file is as follows:

```
  manifest_file = *( file_digest )
  file_digest   = algorithm "(" file_name ")" "=" digest nl
  algorithm     = "SHA1"
  digest        = 40( hex-digit ) // 160-bit digest in 40-digit hexadecimal
  hex-digit     = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "a" |
"b" | "c" | "d" | "e" | "f"
  nl            = 0x0a
```

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

```
  SHA1(package.ovf)= 237de026fb285b85528901da058475e56034da95
  SHA1(vmdisk1.vmdk)= 393a66df214e192ffbfedb78528b5be75cc9e1c3
```

An OVF package may be signed by signing the manifest file. The signature of the digest is stored in a .cert file along with the base64-encoded X.509 certificate. The .cert file shall have the same base name as the OVF descriptor file. A consumer of the OVF package shall verify the signature and should validate the certificate. The format of the .cert file shall be:

```
  certificate_file   = signature_part certificate_part
  signature_part     = algorithm "(" file_name ")" "=" signature nl
  algorithm          = "SHA1"
  signature          = 128( hex-digit) // 512-bit signature in 128 digit hexadecimal
  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"
  nl                 = 0x0a
```

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:

```
SHA1(package.mf)= 7f4b8efb8fe20c06df1db68281a63f1b088e19dbf00e5af9db5e8e3e319de
7019db88a3bc699bab6ccd9e09171e21e88ee20b5255cec3fc28350613b2c529089
-----BEGIN CERTIFICATE-----
MIIBgjCCASwCAQQwDQYJKoZIhvcNAQEEBQAwODELMAkGA1UEBhMCQVUxDDAKBgNV
BAgTA1FMRDEbMBkGA1UEAxMSU1NMZWF5L3JzYSB0ZXN0IENBMB4XDTk1MTAwOTIz
MzIwNVoXDTk4MDcwNTIzMzIwNVowYDELMAkGA1UEBhMCQVUxDDAKBgNVBAgTA1FM
RDEZMBcGA1UEChMQTWluY29tIFB0eS4gTHRkLjELMAkGA1UECxMCQ1MxGzAZBgNV
BAMTElNTTGVheSBkZW1vIHNlcnZlcjBcMA0GCSqGSIb3DQEBAQUAA0sAMEgCQQC3
LCXcScWua0PFLkHBLm2VejqpA1F4RQ8q0VjRiPafjx/Z/aWH3ipdMVvuJGa/wFXb
/nDFLDlfWp+oCPwhBtVPAgMBAAEwDQYJKoZIhvcNAQEEBQADQQArNFsihWIjBzb0
DCsU0BvL2bvSwJrPEqFlkDq3F4M6EGutL9axEcANWgbbEdAvNJD1dmEmoWny27Pn
IMs6ZOZB
-----END 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 which 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. It is recommended that these URIs are resolvable.

5.3 Distribution as a Single File

An OVF package can be stored as a single file using the TAR format. The extension of that file should be .ova (open virtual appliance or application).

EXAMPLE: The following example shows a sample filename for an OVF package of this type:

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

Ordinarily, a TAR extraction tool would have to scan the whole archive, even if the file requested is found at the beginning, because replacement files can be appended without modifying the rest of the archive. For OVF TAR files, duplication is not allowed within the archive. In addition, the files shall be in the following order inside the archive:

1) .ovf descriptor file
2) .mf manifest file (optional)
3) .cert certificate file (optional)
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).
5) .mf manifest file (optional)
6) .cert certificate (optional)

Note that the certificate file is optional. If no certificate file is present, the manifest file is also optional. If the manifest or certificate files are present, they shall either both be placed after the OVF descriptor file, or both be placed at the end of the archive.

For deployment, the ordering restriction ensures that it is possible to extract the OVF descriptor from an OVF TAR file without scanning the entire archive. For generation, the ordering restriction ensures that an OVF TAR file can easily be generated on-the-fly. The restrictions do not prevent OVF TAR files from being created using standard TAR packaging tools.

The TAR format used shall comply with the USTAR (Uniform Standard Tape Archive) format as defined by the POSIX IEEE 1003.1 standards group.

5.4 Distribution as a Set of Files

An OVF package can be made available as a set of files — for example on a standard Web server:

```
    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
```

6 OVF Descriptor

All metadata about the package and its contents is stored in the OVF descriptor. This is an extensible XML document for encoding information, such as product details, virtual hardware requirements, and licensing.

The `ovf-envelope.xsd` XML schema definition file for the OVF descriptor contains the elements and attributes.

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

The XML document of an OVF descriptor shall contain one `Envelope` element, which is the only element allowed at the top level.

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/1`
`ovfenv`	`http://schemas.dmtf.org/ovf/environment/1`
`rasd`	`http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_ResourceAllocationSettingData`
`vssd`	`http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_VirtualSystemSettingData`

7 Envelope element

The `Envelope` element describes all metadata for the virtual machines (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. These are typically virtual disk files, ISO images, and internationalization resources.
- A metadata part, defined by section elements, as defined in clause 9.
- A description of the content, either a single virtual machine (`VirtualSystem` element) or a collection of multiple virtual machines (`VirtualSystemCollection` element).
- A specification of message resource bundles for zero or more locales, defined by a `Strings` element for each locale.

EXAMPLE: 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/1"
    xmlns="http://schemas.dmtf.org/ovf/envelope/1"
    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>
```

The optional `xml:lang` attribute on the `Envelope` element specifies the default locale for messages in the descriptor. The optional `Strings` elements contain message resource bundles for different locales. See clause 10 for more details on internationalization support.

7.1 File References

The file reference part defined by the `References` element allows a tool to easily 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. The URL schemes `"file"`, `"http"`, and `"https"` shall be supported. Using other URL schemes is allowed but not recommended. If no URL scheme is specified, the value of the `ovf:href` attribute shall be interpreted as a path name of the referenced file that is relative to the location of the OVF descriptor file itself. The relative path name shall use the syntax of relative-path references in IEFT RFC 3986. 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 can optionally be specified using the `ovf:size` attribute. The unit of this attribute is always bytes.

Each file referenced by a `File` element may be compressed using gzip (see RFC1952), which is indicated using the `ovf:compression="gzip"` attribute. Omitting the compression attribute, or specifying it as `"identity"`, states that no compression is used. Alternatively, if the href is an HTTP or HTTPS URL, then 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.

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

When `ovf:chunkSize` 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 is given below. The syntax uses ABNF with the exceptions listed in ANNEX A.

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

where href-value is the value of the `ovf:href` attribute, and chunk-number is the 0-based position of the chunk starting with the value 0 and increases with increments of 1 for each chunk.

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

7.2 Content Part

The virtual machine configurations required by an OVF package is 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.

The `VirtualSystem` element describes a single virtual machine and is simply a container of section elements. These section elements describe virtual hardware, resources, product information, and so on, and are described in detail in clause 8 and 9.

The structure of a `VirtualSystem` element is as follows:

```
<VirtualSystem ovf:id="Simple Appliance">
    <Info>A virtual machine</Info>
    <SomeSection>
        <!-- Additional section content -->
    </SomeSection>
    <!-- Additional sections can follow -->
 </VirtualSystem>
```

The `VirtualSystemCollection` element is a container of multiple `VirtualSystem` or `VirtualSystemCollection` elements. Thus, arbitrary complex configurations can be described. The section elements at the `VirtualSystemCollection` level describe appliance information, properties, resource requirements, and so on, and are described in detail in clause 9.

The structure of a `VirtualSystemCollection` element is as follows:

```
<VirtualSystemCollection ovf:id="Multi-tier Appliance">
    <Info>A collection of virtual machines</Info>
    <SomeSection>
        <!-- Additional section content -->
    </SomeSection>
    <!-- Additional sections can follow -->
    <VirtualSystem ovf:id="...">
        <!-- Additional sections -->
    </VirtualSystem>
    <!-- Additional VirtualSystem or VirtualSystemCollection elements can follow-->
</VirtualSystemCollection>
```

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. Similarly, 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.

All elements in the `Content` and `Section` substitution groups shall contain an `Info` element which contains a human readable description of the meaning of this entity. See clause 10 for details on how to localize the `Info` element.

7.3 Extensibility

The OVF schemas associated with this specification are expressed in XML Schema 1.0. Extensions that are subtypes of `Section` can be added, but existing types cannot be extended with additional elements. The plan is to add an extension model based on the design of the open content model in XML Schema 1.1.

Custom extensions shall not use XML namespaces defined in this specification.

All subtypes of `Section` contain an `Info` element which 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. See clause 10 for details on how to localize the `Info` element.

7.4 Compatibility

On extensions, a Boolean `ovf:required` attribute specifies whether the information in the element is required for correct behavior or optional. If not specified, the `ovf:required` attribute defaults to FALSE. An OVF application that detects an extension that is required and that it does not understand shall fail.

For known `Section` elements, if additional child elements that are not understood are found and the value of their `ovf:required` attribute is TRUE, the OVF application shall interpret the entire section as one it does not understand. The check is not recursive; it applies only to the direct children of the `Section` element.

This behavior ensures that older parsers will reject newer OVF specifications, unless explicitly instructed not to do so.

EXAMPLE:

```
    <AnnotationSection>
        <Info>Specifies an annotation for this virtual machine</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 -->
        <Author ovf:required="false">John Smith</Author>
    </AnnotationSection>
```

8 Virtual Hardware Description

8.1 VirtualHardware Section

The virtual hardware required by a virtual machine is specified in the `VirtualHardware` section. This specification supports abstract or incomplete hardware descriptions in which only the major devices are described. The hypervisor is allowed to create additional virtual hardware controllers and devices, as long as the required devices listed in the descriptor are realized.

This virtual hardware description is based on the CIM classes `CIM_VirtualSystemSettingData` and `CIM_ResourceAllocationSettingData`. The XML representation of the CIM model is based on the WS-CIM mapping (DSP0230).

EXAMPLE: Example of `VirtualHardware` section:

```
   <VirtualHardwareSection ovf:transport="iso">
      <Info>500Mb, 1 CPU, 1 disk, 1 nic virtual machine</Info>
      <System>
         <vssd:VirtualSystemType>vmx-4</vssd:VirtualSystemType>
      </System>
      <Item>
         <rasd:AllocationUnits>byte * 2^20</rasd:AllocationUnits>
         <rasd:Description>Memory Size</rasd:Description>
         <rasd:ElementName>512 MB of memory</rasd:ElementName>
         <rasd:InstanceID>2</rasd:InstanceID>
         <rasd:ResourceType>4</rasd:ResourceType>
         <rasd:VirtualQuantity>512</rasd:VirtualQuantity>
      </Item>
      <!-- Additional Item elements can follow -->
    </VirtualHardwareSection>
```

`VirtualHardware` is a required child element for a `VirtualSystem` element, and it is disallowed as a direct child element of a `VirtualSystemCollection` element and of an `Envelope` element.

Multiple `VirtualHardware` element occurrences are allowed within a single `VirtualSystem` element. The OVF application can select the most appropriate virtual hardware description, typically based on the family attribute.

The `ovf:transport` attribute specifies the types of transport mechanisms by which properties are passed to the virtual machine in an OVF environment document. This attribute supports a pluggable and extensible architecture for providing guest/platform communication mechanisms. Several transport types can be specified separated by single space character. See 9.5 for a description of properties and clause 11 for a description of transport types and OVF environments.

The `vssd:VirtualSystemType` element specifies a virtual system type identifier, which is an implementation defined string that uniquely identifies the type of the virtual system. For example, a virtual system type identifier could be `vmx-4` for VMware's fourth-generation virtual hardware or `xen-3` for Xen's third-generation virtual hardware. 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 machine on the target platform is required to 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 (see DSP1042).

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 can be specified in an `Item` element, separated by single space character.

EXAMPLE:

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

8.2 Extensibility

The optional `ovf:required` attribute on the `Item` element specifies whether the realization of the element (for example, a CD-rom or USB controller) is required for correct behavior of the guest software. If not specified, `ovf:required` defaults to FALSE.

On child elements of the `Item` element, the optional Boolean attribute `ovf:required` shall be interpreted, even though these elements are in a different RASD WS-CIM namespace. A tool parsing an `Item` element shall 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`
Known	FALSE	Shall interpret `Item`
Unknown	TRUE or not specified	Shall fail `Item`
Unknown	FALSE	Shall ignore `Item`

8.3 Virtual Hardware Elements

The general form of any `Item` element in a `VirtualHardware` element is as follows:

```
   <Item ovf:required="…" ovf:configuration="…" ovf:bound="…">
       <rasd:Address> ... </rasd:Address>
       <rasd:AddressOnParent> ... </rasd:AddressOnParent>
       <rasd:AllocationUnits> ... </rasd:AllocationUnits>
       <rasd:AutomaticAllocation> ... </rasd:AutomaticAllocation>
       <rasd:AutomaticDeallocation> ... </rasd:AutomaticDeallocation>
       <rasd:Caption> ... </rasd:Caption>
       <rasd:Connection> ... </rasd:Connection>
       <!-- multiple connection elements can be specified -->
       <rasd:ConsumerVisibility> ... </rasd:ConsumerVisibility>
       <rasd:Description> ... </rasd:Description>
       <rasd:ElementName> ... </rasd:ElementName>
       <rasd:HostResource> ... </rasd:HostResource>
       <rasd:InstanceID> ... </rasd:InstanceID>
       <rasd:Limit> ... </rasd:Limit>
       <rasd:MappingBehavior> ... </rasd:MappingBehavior>
       <rasd:OtherResourceType> ... </rasd:OtherResourceType>
       <rasd:Parent> ... </rasd:Parent>
       <rasd:PoolID> ... </rasd:PoolID>
       <rasd:Reservation> ... </rasd:Reservation>
       <rasd:ResourceSubType> ... </rasd:ResourceSubType>
       <rasd:ResourceType> ... </rasd:ResourceType>
       <rasd:VirtualQuantity> ... </rasd:VirtualQuantity>
       <rasd:Weight> ... </rasd:Weight>
    </Item>
```

The elements represent the 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 document.

EXAMPLE: The following example shows a description of the number of virtual CPUs:

```
   <Item>
       <rasd:AllocationUnits>hertz * 10^6</rasd:AllocationUnits>
       <rasd:Description>The number of virtual CPUs</rasd:Description>
       <rasd:ElementName>2 virtual CPUs, a 300 MHz reservation</rasd:ElementName>
       <rasd:InstanceID>1</rasd:InstanceID>
       <rasd:Reservation>300</rasd:Reservation>
       <rasd:ResourceType>3</rasd:ResourceType>
       <rasd:VirtualQuantity>2</rasd:VirtualQuantity>
   </Item>
```

The `Description` element is used to provide additional metadata about the element itself. This element enables an OVF application 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 on internationalization support.

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

Devices such as disks, CD-ROMs, and networks need a backing from the deployment platform. The requirements on a backing are either specified using the `HostResource` or the `Connection` element.

For an Ethernet adapter, a logical network name is specified in the `Connection` element. Ethernet adapters that refer to the same logical network name within an OVF package shall be deployed on the same network.

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

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. <id> shall be the value of the `ovf:diskId` attribute of the `Disk` element being referenced.

If no backing is specified for a device that requires a backing, the deployment platform shall make an appropriate choice, for example, by prompting the user. Specifying more than one backing for a device is not allowed.

Table 4 gives a brief overview on how elements are used to describe virtual devices and controllers.

Table 4 – Elements for Virtual Devices and Controllers

Element	Usage
`rasd:Description`	A human-readable description of the meaning of the information. For example, "Specifies the memory size of the virtual machine".
`rasd:ElementName`	A human-readable description of the content. For example, "256MB memory".
`rasd:InstanceID`	A unique instance ID of the element within the section.
`rasd:HostResource`	Abstractly specifies how a device shall connect to a resource on the deployment platform. Not all devices need a backing. See Table 3.
`rasd:ResourceType` `rasd:OtherResourceType` `rasd:ResourceSubtype`	Specifies the kind of device that is being described.
`rasd:AutomaticAllocation`	For devices that are connectable, such as floppies, CD-ROMs, and Ethernet adaptors, this element specifies whether the device should be connected at power on.
`rasd:Parent`	The InstanceID of the parent controller (if any).
`rasd:Connection`	For an Ethernet adapter, this specifies the abstract network connection name for the virtual machine. All Ethernet adapters that specify the same abstract network connection name within an OVF package shall be deployed on the same network. The abstract network connection name shall be listed in the NetworkSection at the outermost envelope level.
`rasd:Address`	Device specific. For an Ethernet adapter, this specifies the MAC address.
`rasd:AddressOnParent`	For a device, this specifies its location on the controller.
`rasd:AllocationUnits`	Specifies the units of allocation used. For example, "byte * 2^20".
`rasd:VirtualQuantity`	Specifies the quantity of resources presented. For example, "256".
`rasd:Reservation`	Specifies the minimum quantity of resources guaranteed to be available.
`rasd:Limit`	Specifies the maximum quantity of resources that will be granted.
`rasd: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.

8.4 Ranges on Elements

The optional `ovf:bound` attribute can 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`.

A platform deploying an OVF package is recommended to start with the normal value and adjust the value within the range for ongoing performance tuning and validation.

For the `Item` elements in `VirtualHardware` and `ResourceAllocation` elements, the following additional semantics is defined:

- Each `Item` element has an optional `ovf:bound` attribute. This value can be specified as `min`, `max`, or `normal`. The value defaults to `normal`. If the attribute is not specified or is specified as `normal`, then the item is interpreted as being part of the regular virtual hardware or resource allocation description.

- 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:

- `InstanceID` and `ResourceType` shall be specified, and the `ResourceType` shall match other `Item` elements with the same `InstanceID`.
- Specifying more than one `min` range marker or more than one `max` range marker for a given RASD (identified with `InstanceID`) is invalid.
- An `Item` element with a range marker shall have a corresponding `Item` element without a range marker, that is, an `Item` element with no `ovf:bound` attribute or `ovf:bound` attribute with value normal. This corresponding item specifies the default value.
- For an `Item` 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` 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.
- The use of non-integer elements in range marker RASDs is invalid.

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>
```

9 Core Metadata Sections

The following core metadata sections are defined:

Section	Locations	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 machine 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 machine collection is powered on	VirtualSystemCollection	Zero or One
`DeploymentOptionSection` Specifies a discrete set of intended resource requirements	Envelope	Zero or One
`OperatingSystemSection` Specifies the installed guest operating system of a virtual machine	VirtualSystem	Zero or One
`InstallSection` Specifies that the virtual machine needs to be initially booted to install and configure the software	VirtualSystem	Zero or One

The following clauses 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 OVF schema for a detailed specification of all attributes and elements.

9.1 DiskSection

A `DiskSection` describes meta-information about virtual disks in the OVF package. Virtual disks and their metadata are described outside the virtual hardware to facilitate sharing between virtual machines within an OVF package.

```
<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/specifications/vmdk.html#sparse">
    </Disk>
    <Disk ovf:diskId="vmdisk2" ovf:capacity="536870912"
          ovf:format="http://www.vmware.com/specifications/vmdk.html#sparse">
    </Disk>
    <Disk ovf:diskId="vmdisk3" ovf:capacity="${disk.size}"
          ovf:capacityAllocationUnits="GigaBytes"
          ovf:format="http://www.vmware.com/specifications/vmdk.html#sparse">
    </Disk>
</DiskSection>
```

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

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

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.

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

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. In this case, the disk shall be created and the entire disk content zeroed at installation time.

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 for an empty disk can be given using an OVF property, for example `ovf:capacity="${disk.size}"`. The OVF property shall resolve to an `xs:long` integer value. See 9.5 for a description of OVF properties. The `ovf:capacityAllocationUnits` attribute is useful when using OVF properties because a user may be prompted and can then enter disk sizing information in e.g. gigabytes.

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

OVF allows a disk image to 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. For a `Disk` element, a parent disk can optionally be specified using the `ovf:parentRef` attribute, which 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.

9.2 NetworkSection

The `NetworkSection` element shall list all logical networks used in the OVF package.

```
<NetworkSection>
    <Info>List of logical networks used in the package</Info>
    <Network ovf:name="red">
        <Description>The network the Red service will be available on</Description>
    </Network>
</NetworkSection>
```

`NetworkSection` is a valid element at the outermost envelope level.

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

9.3 ResourceAllocationSection

The `ResourceAllocationSection` element describes all resource allocation requirements of a `VirtualSystemCollection` entity. These resource allocations shall be performed when deploying the OVF package.

```
<ResourceAllocationSection>
   <Info>Defines reservations for CPU and memory for the collection of VMs</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>
</ResourceAllocationSection>
```

`ResourceAllocationSection` is a valid element for a `VirtualSystemCollection` entity.

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 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. Such annotations may be displayed when deploying the OVF package.

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

`AnnotationSection` is a valid element for a `VirtualSystem` and a `VirtualSystemCollection` entity.

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

9.5 ProductSection

The `ProductSection` element specifies product-information for an appliance, such as product name, version, vendor, and so on.

```
<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>
   <AppUrl>http://${app.ip}/</AppUrl>
   <Icon ovf:height="32" ovf:width="32" ovf:mimeType="image/png" ovf:fileRef="icon">
   <Category>Email properties</Category>   <Property ovf:key="admin.email"
ovf:type="string" ovf:userConfigurable="true">
        <Label>Admin email</Label>
        <Description>Email address of administrator</Description>
   </Property>
   <Category>Admin properties</Category>
   <Property ovf:key="app.log" ovf:type="string" ovf:value="low"
ovf:userConfigurable="true">
        <Description>Loglevel for the service</Description>
   </Property>
   <Property ovf:key="app.ip" ovf:type="string" ovf:qualifiers="ip"
ovf:value="${appserver-vm}">
        <Description>The IP address of the application server virtual
machine</Description>
   </Property>
</ProductSection>
```

`Property` elements specify application-level customization parameters and are particularly 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.

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

`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 on 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.

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 5 and valid qualifiers are listed in Table 6.

The optional attribute `ovf:value` is used to provide a default value for a property. One or more optional `Value` elements may be used to define alternative default values for specific configurations, as defined in clause 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 and IP types, because the empty string is not a valid integer or IP value. For string types, prompting can be forced by using a type for a non-empty string.

Zero or more `ProductSections` can be specified within a `VirtualSystem` or `VirtualSystemCollection`. Typically, a `ProductSection` corresponds to a particular software product that is installed. Each product section at the same entity level shall have a unique `ovf:class` and `ovf:instance` attribute pair. For the common case where only a single `ProductSection` is used, the `ovf:class` and `ovf:instance` attributes are optional and default to the empty string. It is recommended that the `ovf:class` property be used to uniquely 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 is used to identify the different instances.

Property elements are exposed to the guest software through the OVF environment, as described in clause 11. 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]
```

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.

EXAMPLE: The following OVF environment example shows how properties can be propagated to the guest software:

```
<Property ovf:key="com.vmware.tools.logLevel"    ovf:value="none"/>
<Property ovf:key="org.apache.tomcat.logLevel.1" ovf:value="debug"/>
<Property ovf:key="org.apache.tomcat.logLevel.2" ovf:value="normal"/>
```

The consumer of an OVF package should prompt for properties where `ovf:userConfigurable` is TRUE. These properties can be defined in multiple `ProductSections` as well as in sub-entities in the OVF package.

The first `ProductSection` entity defined in the top-level `Content` element of a package shall define summary information that describes the entire package. After installation, an OVF application could choose to make this information available as an instance of the CIM_Product class.

`Property` elements specified on a `VirtualSystemCollection` can also be seen by its immediate children (see clause 11). Children can refer to the properties of a parent `VirtualSystemCollection` using macros on the form `${name}` as value for the `ovf:key` attributes.

Table 5 lists the valid types for properties. These are a subset of CIM intrinsic types defined in DSP0004, which also define the value space and format for each intrinsic type. Each `Property` element in a shall specify a type using the `ovf:type` attribute.

Table 5 – 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 6 lists the supported CIM type qualifiers as defined in DSP0004. Each `Property` element in a may specify type qualifiers using the `ovf:qualifiers` attribute.

Table 6 – Property qualifiers

Type	Description
`string`	`MinLen(min)` `MaxLen(max)` `ValueMap{...}`
`uint8` `sint8` `uint16` `sint16` `uint32` `sint32` `uint64` `sint64`	`ValueMap{...}`

The `MinLen`, `MaxLen` and `ValueMap` qualifiers take values as defined in DSP0004.

9.6 EulaSection

A `EulaSection` contains the legal terms for using its parent `Content` element. This license shall be shown and accepted during deployment of an OVF package. Multiple `EulaSections` can be present in an OVF. If unattended installations are allowed, all embedded license sections are implicitly accepted.

```
<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>
```

`EulaSection` is a valid section for a `VirtualSystem` and a `VirtualSystemCollection` entity.

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

9.7 StartupSection

The `StartupSection` specifies how a virtual machine collection is powered on and off.

```
   <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>
```

Each `Content` element that is a direct child of a `VirtualSystemCollection` may have a corresponding `Item` element in the `StartupSection` entity of the `VirtualSystemCollection` entity. Note that `Item` elements can correspond to both `VirtualSystem` and `VirtualSystemCollection` entities. When a start or stop action is performed on a `VirtualSystemCollection` entity, the respective actions on the `Item` elements of its `StartupSection` entity are invoked in the specified order. Whenever an `Item` element corresponds to a (nested) `VirtualSystemCollection` entity, the actions on the `Item` elements of its `StartupSection` entity shall be invoked before the action on the Item element corresponding to that `VirtualSystemCollection` entity is invoked (i.e., depth-first traversal).

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

- `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 machine or virtual machine collection to which the actions defined in the `Item` element apply.
- `ovf:order` specifies the startup order using non-negative integer values. The order of execution of the start action is the numerical ascending order of the values. `Items` with same order identifier may be started up concurrently. The order of execution of the stop action is the numerical descending order of the values.

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

- `ovf:startDelay` specifies a delay in seconds to wait until proceeding to the next order in the start sequence. The default value is 0.
- `ovf:waitingForGuest` enables the platform to resume the startup sequence after the guest software has reported it is ready. The interpretation of this is deployment 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 deployment platform specific. The default value is `powerOff`.

If not specified, an implicit default `Item` is created for each entity in the collection with `ovf:order="0"`. Thus, for a trivial startup sequence no `StartupSection` needs to be specified.

9.8 DeploymentOptionSection

The `DeploymentOptionSection` specifies a discrete set of intended resource configurations. The author of an OVF package can include sizing metadata for different configurations. A consumer of the OVF shall select a configuration, for example, by prompting the user. The selected configuration will be visible in the OVF environment, enabling guest software to adapt to the selected configuration. See clause 11.

The `DeploymentOptionSection` specifies an ID, label, and description for each configuration.

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

The DeploymentOptionSection has the following semantics:

- If present, the DeploymentOptionSection is valid only at the envelope level, and only one section can be specified 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 package.
- A default `Configuration` element can be specified with the optional `ovf:default` attribute. If no default is specified, the first element in the list is the default. Specifying more than one element as the default is invalid.
- The `Label` and `Description` elements are localizable using the `ovf:msgid` attribute. See clause 10 for more details on internationalization support.

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

Each `Item` 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 an ID that is not specified in the `DeploymentOptionSection`.

- Within a single `VirtualHardwareSection` or `ResourceAllocationSection`, multiple `Item` elements are allowed to refer to the same InstanceID. A single combined `Item` for the given InstanceID shall be constructed by picking up the child elements of each `Item` element, with child elements of a former `Item` element in the OVF descriptor not being picked up if there is a like-named child element in a latter `Item` element. Any attributes specified on child elements of `Item` elements that are not picked up that way, are not part of the combined `Item` element.
- All `Item` elements shall specify ResourceType, and `Item` elements with the same InstanceID shall agree on ResourceType.

EXAMPLE: 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>
```

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

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

- It is possible to use 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.

EXAMPLE: The following shows an example ProductSection:

```
<ProductSection>
   <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>
```

9.9 OperatingSystemSection

An `OperatingSystemSection` specifies the operating system installed on a virtual machine.

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

The valid values for `ovf:id` are defined by the `ValueMap` qualifier in the `CIM_OperatingSystem.OsType` property.

`OperatingSystemSection` is a valid section for a `VirtualSystem` entity only.

9.10 InstallSection

The `InstallSection`, if specified, indicates that the virtual machine 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 machine does not need to be powered on to complete installation of guest software.

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

`InstallSection` is a valid section for a `VirtualSystem` entity only.

The optional `ovf:initialBootStopDelay` attribute specifies a delay in seconds to wait for the virtual machine to power off. If not set, the implementation shall wait for the virtual machine to power off by itself. If the delay expires and the virtual machine has not powered off, the OVF application shall indicate a failure.

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

Several VMs in a virtual machine collection may have an `InstallSection` defined, in which case the above step is done for each VM, potentially concurrently.

10 Internationalization

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

- `Info` 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 `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 `ResourceAllocation`

The `ovf:msgid` attribute contains an identifier that refers to a message that can have different values in different locales.

EXAMPLE 1:

```
<Info ovf:msgid="info.text">Default info.text value if no locale is set or no locale
match</Info>
<License ovf:msgid="license.tomcat-6_0"/>  <!-- No default message -->
```

The `xml:lang` attribute on the `Envelope` element specifies the default locale for messages in the descriptor. If not specified, the locale defaults to the locale of the consumer of the OVF package.

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.

EXAMPLE 2:

```
  <ovf:Envelope xml:lang="en-US">
      ...
      ... sections and content here ...
      ...
      <Info msgid="info.os">Operating System</Info>
      ...
      <Strings xml:lang="da-DA">
          <Msg ovf:msgid="info.os">Operativsystem</Msg>
          ...
      </Strings>
      <Strings xml:lang="de-DE">
          <Msg ovf:msgid="info.os">Betriebssystem</Msg>
          ...
      </Strings>
  </ovf:Envelope>
```

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.

EXAMPLE 3:

```
  <ovf:Envelope xml:lang="en-US">
     <References>
        ...
        <File ovf:id="it-it-resources" ovf:href="resources/it-it-bundle.msg"/>
     </References>
      ... sections and content here ...
      ...
      <Strings xml:lang="it-IT" ovf:fileRef="it-it-resources"/>
          ...
  </ovf:Envelope>
```

EXAMPLE 4: Example content of external resources/it-it-bundle.msg file, which is referenced in previous example:

```
  <Strings
    xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
    xmlns="http://schemas.dmtf.org/ovf/envelope/1"
    xml:lang="it-IT">
        <Msg ovf:msgid="info.os">Sistema operativo</Msg>
        ...
  </Strings>
```

The embedded and external `Strings` elements can 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 occurs, a latter value overwrites a former.

11 OVF Environment

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

The environment specification is split into a *protocol* part and a *transport* part. The *protocol* part defines the format and semantics of an XML document that can be made accessible to the guest software. The *transport* part defines how the information is communicated between the deployment platform and the guest software.

The `ovf-environment.xsd` XML schema definition file for the OVF environment contains the elements and attributes.

11.1 Environment Document

The environment document is an extensible XML document that is provided to the guest software about the environment in which it is being executed. The way that the document is obtained depends on the transport type.

EXAMPLE: An example of the structure of the OVF environment document follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Environment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:ovfenv="http://schemas.dmtf.org/ovf/environment/1"
             xmlns="http://schemas.dmtf.org/ovf/environment/1"
             ovfenv:id="identification of VM from OVF descriptor">
    <!-- Information about virtualization platform -->
    <PlatformSection>
       <Kind>Type of virtualization platform</Kind>
       <Version>Version of virtualization platform</Version>
       <Vendor>Vendor of virtualization platform</Vendor>
       <Locale>Language and country code</Locale>
       <TimeZone>Current timezone offset in minutes from UTC</TimeZone>
    </PlatformSection>
    <!--- Properties defined for this virtual machine -->
    <PropertySection>
       <Property ovfenv:key="key" ovfenv:value="value">
       <!-- More properties -->
    </PropertySection>
    <Entity ovfenv:id="id of sibling virtual system or virtual machine collection">
       <!-- More properties -->
    </Entity>
</Environment>
```

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

The `PropertySection` element contains `Property` elements that correspond to those defined in the OVF descriptor for the current virtual machine. The environment presents properties as a simple list to make it easy for applications to parse. Furthermore, the single list format supports the override semantics where a property on a `VirtualSystem` can override one defined on a parent `VirtualSystemCollection`. The overridden property will not be in the list.

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 machine The `Property` section contains the key/value pairs defined for all the properties specified in the OVF descriptor for the current virtual machine, as well as properties specified for the immediate parent `VirtualSystemCollection`, if one exists.

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 siblings OVF environment documents. This information can be used, for example, to make configuration information such as IP addresses available to `VirtualSystems` being part of a multi-tiered application.

The environment document is extensible by providing new section types. A consumer of the document should ignore unknown section types and elements.

11.2 Transport

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

The transport types define methods by which the environment document is communicated from the deployment platform to the guest software. Standardizing transport types does pose some challenges, since no industry-standard cross-vendor para-virtualized device exists. Possible transports types includes dynamically generated DVD images, dynamically generated floppy images, XenSource XenBus, Microsoft VMBus, VMware VMCI, and so on.

To enable interoperability, OVF requires all implementations that support CD-ROM devices to support the `"iso"` transport type. This transport communicates the environment document by making a dynamically generated ISO image available to the guest software. To support the `iso` transport type, prior to booting a virtual machine, an implementation shall make an ISO 9660 read-only disk image available as backing for a disconnected CD-ROM. If the `iso` transport is selected for a `VirtualHardwareSection`, at least one disconnected CD-ROM device shall be present in this section.

Support for the `"iso"` transport type is not a requirement for virtual hardware architectures or guest operating systems which do not have CD-ROM device support.

The ISO image shall contain the OVF environment for this particular virtual machine, and the environment shall be present in an XML file named `ovf-env.xml` that is contained in the root directory of the ISO image. The guest software can now access the information using standard guest operating system tools.

If the virtual machine 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.

To be compliant with this specification, any transport format other than `iso` shall be given by a URI which identifies an unencumbered specification on how to use the transport. 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 format. The specification shall be sufficient for a skilled person to properly interpret the transport mechanism for implementing the protocols. It is recommended that these URIs are resolvable.

ANNEX A
(informative)

Symbols and Conventions

XML examples use the XML namespace prefixes 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 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 Augmented BNF (ABNF) use ABNF as defined in IETF RFC 2234 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 white space as defined in ABNF.

ANNEX B
(informative)

Change Log

Version	Date	Description
1.0.0a	2008-06-04	Work in progress release
1.0.0b	2008-07-23	Preliminary release
		Revised XML schemas to use substitution groups
1.0.0c	2008-08-13	Preliminary release
		Errata
1.0.0d	2008-08-18	Preliminary release

ANNEX C
(normative)

OVF XSD

A normative copy of the XML schemas for this specification may be retrieved by resolving a URL which consists of the XML namespace URI for the XML schema, followed by `"/<dspnumber>_<dspversion>.xsd"`, e.g. `"/dsp8023_1.0.0.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` and`CIM_VirtualSystemSettingData` may be retrieved by resolving the following XML namespace URIs below. Note that `".xsd"` has to be appended to the URIs.

```
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"
```

This specification is based on the following CIM MOFs:

```
  CIM_VirtualSystemSettingData.mof
  CIM_ResourceAllocationSettingData.mof
  CIM_OperatingSystem.mof
```